Breaking the $\mathcal{O}(n)$-Barrier in the Construction of Compressed Suffix Arrays and Suffix Trees*

Dominik Kempa† Tomasz Kociumaka‡

Abstract

The suffix array, describing the lexicographical order of suffixes of a given text, and the suffix tree, a path-compressed trie of all suffixes, are the two most fundamental data structures for string processing, with plethora of applications in data compression, bioinformatics, and information retrieval. For a length-n text, however, they use $\Theta(n \log n)$ bits of space, which is often too costly. To address this, Grossi and Vitter [STOC 2000] and, independently, Ferragina and Manzini [FOCS 2000] introduced space-efficient versions of the suffix array, known as the *compressed suffix array* (CSA) and the *FM-index*. Sadakane [SODA 2002] then showed how to augment them to obtain the *compressed suffix tree* (CST). For a length-n text over an alphabet of size σ, these structures use only $\mathcal{O}(n \log \sigma)$ bits. Nowadays, these structures are part of the standard toolbox: modern textbooks spend dozens of pages describing their applications, and they almost completely replaced suffix arrays and suffix trees in space-critical applications. The biggest remaining open question is how efficiently they can be constructed. After two decades, the fastest algorithms still run in $\mathcal{O}(n)$ time [Hon et al., FOCS 2003], which is $\Theta(\log_\sigma n)$ factor away from the lower bound of $\Omega(n/\log_\sigma n)$ (following from the necessity to read the input).

In this paper, we make the first in 20 years improvement in n for this problem by proposing a new compressed suffix array and a new compressed suffix tree which admit $o(n)$-time construction algorithms while matching the space bounds and the query times of the original CSA/CST and the FM-index. More precisely, our structures take $\mathcal{O}(n \log \sigma)$ bits, support SA queries and full suffix tree functionality in $\mathcal{O}(\log^\epsilon n)$ time per operation, and can be constructed in $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ time using $\mathcal{O}(n \log \sigma)$ bits of working space. (For example, if $\sigma = 2$, the construction time is $\mathcal{O}(n/\sqrt{\log n}) = o(n)$.) We derive this result as a corollary from a much more general reduction: We prove that all parameters of a compressed suffix array/tree (query time, space, construction time, and construction working space) can essentially be reduced to those of a data structure answering new query types that we call *prefix rank* and *prefix selection*. Using the novel techniques, we also develop a new index for pattern matching.

1 Introduction

Let T be a text of length n. A *suffix tree* [81] of T is a trie of all suffixes of T, in which every unary path has been replaced with a single edge labeled by a text substring. The resulting tree has less than $2n$ nodes and thus can be encoded in $\mathcal{O}(n \log n)$ bits. Related to suffix trees are *suffix arrays* [60]. The suffix array $\mathrm{SA}[1\,..\,n]$ of T stores the permutation of $\{1, \ldots, n\}$ such that $\mathrm{SA}[i]$ is the starting position of the ith lexicographically smallest suffix of T. Consider now the following problem: Construct a data structure that, given any length-m pattern P, counts the number of occurrences of P in T. To solve it using a suffix tree, it suffices to descend the tree in $\mathcal{O}(m)$ time and report the precomputed number of leaves below the reached node. Using a suffix array, it suffices to perform an $\mathcal{O}(m \log n)$-time binary search resulting in the range $\mathrm{SA}[b\,..\,e)$ of suffixes of T having P as a prefix. Then, $e - b$ is the number of occurrences of P in T (and $\mathrm{SA}[b\,..\,e)$ contains their starting positions). The advantage of suffix array is that it is more space efficient: it only needs $n\lceil \log n \rceil$ bits. The queries, however, are usually slightly slower.

The above is a canonical application of suffix arrays/trees. It is, however, only the tip of the iceberg. Suffix trees and suffix arrays are widely considered to be the two most fundamental data structures for string processing.

*The full version of the paper can be accessed at https://arxiv.org/abs/2106.12725.

†Stony Brook University, NY, USA. Email: kempa@cs.stonybrook.edu. Work in part done while at University of California, Berkeley and Johns Hopkins University. Supported by NIH HG011392, NSF DBI-2029552, 1652303, 1934846 grants, an Alfred P. Sloan Fellowship, and a Simons Foundation Junior Faculty Fellowship.

‡Max Planck Institute for Informatics, Saarland Informatics Campus, Germany. Email: tomasz.kociumaka@mpi-inf.mpg.de. Work mostly done while at the University of California, Berkeley, partly supported by NSF 1652303, 1909046, and HDR TRIPODS 1934846 grants, and an Alfred P. Sloan Fellowship.

As written by Gusfield in his classical textbook [44]: *"Suffix trees can be used to solve the exact matching problem in linear time (...), but their real virtue comes from their use in linear-time solutions to many string problems more complex than exact matching"*. This includes well-studied problems like Maximal Repeats, Longest Repeated Factor, Minimal Absent Word, Longest Common Substring, Matching Statistics, Maximal Unique Matches, LZ77 Factorization, BWT Compression, and many more (see, e.g., [1, 44, 58, 65, 72]).

With the increasing size of datasets that need processing, plain suffix arrays and suffix trees, however, have become expensive to use, particularly in applications where the input text is over a small alphabet $[0\,..\,\sigma)$. Such text requires $n\lceil\log\sigma\rceil$ bits, whereas the suffix array/tree uses at least $n\lceil\log n\rceil$ bits of space. In some applications, the gap $\frac{\log n}{\log\sigma}$ can be quite large, e.g., in computational biology, where we usually have $\sigma = 4$, the gap is typically between 16 and 32. This shortcoming was addressed by Grossi and Vitter and, independently, Ferragina and Manzini at the turn of the millennium. They introduced space-efficient versions of the suffix array, known as the *compressed suffix array (CSA)* [42, 43] and the *FM-index* [27, 28]. For a length-n text over an alphabet of size σ, these data structures use only $\mathcal{O}(n\log\sigma)$ bits, and they can answer SA queries (asking for $\mathrm{SA}[i]$ given $i \in [1\,..\,n]$) in $\mathcal{O}(\log^{\epsilon} n)$ time, where $\epsilon > 0$ is an arbitrary predefined constant. With such data structure, one can execute any algorithm that uses the suffix array, but consuming less space and only incurring a factor of $\mathcal{O}(\log^{\epsilon} n)$ penalty in the runtime.[1] Shortly after these discoveries, Sadakane [79] extended CSA/FM-index into a *compressed suffix tree (CST)*, supporting all suffix tree operations in $\mathcal{O}(\log^{\epsilon} n)$ time (while still using $\mathcal{O}(n\log\sigma)$ bits of space). This powerful structure can be plugged into an even larger set of algorithms [37].

Nowadays, CSAs and CSTs are widely used in practice. Modern string algorithms textbooks focus on the use and applications of CSAs/CSTs and related data structures [1, 58], or even entirely on the emerging notion of *compressed data structures* [65]. The FM-index occupies the central role in some of the most commonly used bioinformatics tools, like `Bowtie` [55], `BWA` [56], and `Soap2` [57], and mature and highly engineered implementations of CSAs and CSTs are available through the `sdsl` library[2] of Gog et al. [37, 38]. Despite these developments in functionality and practical adoption of CSAs/CSTs, the time complexity of their construction remains an open problem. The original paper of Grossi and Vitter [42], describes a method that, given a length-n text over alphabet $\Sigma = [0\,..\,\sigma)$, constructs the CSA in $\mathcal{O}(n\log\sigma)$ time and using $\mathcal{O}(n\log n)$ bits of working space. In 2003, a celebrated result of Hon et al. [46] lowered the time complexity to $\mathcal{O}(n\log\log\sigma)$ and the space to the optimal $\mathcal{O}(n\log\sigma)$ bits. Note, however, that, e.g., for $\sigma = 2$, this algorithm still runs in $\Theta(n)$ time, which is slower by a $\Theta(\log n)$ factor than the lower bound of $\Omega(n/\log n)$, following simply from the necessity to read the entire input. Recently, Belazzougui [5] improved the time complexity of the CSA/CST construction to randomized $\mathcal{O}(n)$ (while using the optimal space of $\mathcal{O}(n\log\sigma)$ bits), making it independent of the alphabet size σ. Shortly after, Munro, Navarro, and Nekrich [61] proposed a deterministic solution. Despite these advances, 20 years after the result of Hon et al. [46], the bound of $\Omega(n)$ still stands on the construction of CSAs/CSTs. Given the fundamental role of CSAs and CSTs, we thus ask:

Given a text over alphabet $\Sigma = [0\,..\,\sigma)$ represented using $\mathcal{O}(n\log\sigma)$ bits, can we construct a compressed suffix array/tree of T in $o(n)$ time?

Our Results We answer the above question affirmatively by describing a new data structure that takes $\mathcal{O}(n\log\sigma)$ bits, supports all operations of CSA and CST in $\mathcal{O}(\log^{\epsilon} n)$ time, and can be constructed in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time using $\mathcal{O}(n\log\sigma)$ bits of space (Theorems 5.1 and 7.1). Thus, our solution matches the size and the query time of [27, 42, 79] (as well as more recent CSTs [14, 16, 31, 34, 75, 77]) but, unlike those, admits a sublinear-time construction for small σ. In particular, we achieve $\mathcal{O}(n/\sqrt{\log n}) = o(n)$ time for $\sigma = 2$, constituting the first improvement in n since 2003 [46].

In addition to a new CSA/CST, we also present a new pattern matching index. We show (in Theorem 6.1) how, given a length-n text T stored using $\mathcal{O}(n\log\sigma)$ bits, to construct in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time an index of size $\mathcal{O}(n\log\sigma)$ bits that, given the packed representation (i.e., using $\mathcal{O}(m\log\sigma)$ bits) of any pattern $P[1\,..\,m]$, counts the occurrences of P in T in $\mathcal{O}(m/\log_\sigma n + \log^{\epsilon} n)$ time (where $\epsilon > 0$ is an arbitrary predefined constant). The best previous solutions using compact space (i.e., $\mathcal{O}(n\log\sigma)$ bits) achieve $\mathcal{O}(n\log\sigma/\sqrt{\log n})$-time[3] construction and $\mathcal{O}(m/\log_\sigma n + \log n\cdot\log_\sigma n)$-time queries [63], or $\mathcal{O}(n)$-time construction and $\mathcal{O}(m/\log_\sigma n + \log^{\epsilon}_\sigma n)$-time

[1] This is often acceptable: a slower algorithm remains usable, but insufficient memory can thwart it entirely.

[2] The library is available at https://github.com/simongog/sdsl-lite.

[3] Although CSA lets us implement pattern counting queries, an index implementing pattern counting queries does not let us implement SA queries; thus, although built in $o(n)$ time, [63] cannot be used to answer SA queries.

queries [62]. Thus, for the most difficult case of $\sigma = 2^{\mathcal{O}(\sqrt{\log n})}$, our construction subsumes both these indexes in both aspects.[4] Since our pattern-matching index not only returns the number of occurrences but also the range in SA containing all suffixes prefixed with P, combining the above result with our CSA yields the structure that can additionally *report* all occurrences of P in $\mathcal{O}(\log^{\epsilon} n)$ time per occurrence (Theorem 6.2).

The query times of all our data structures (i.e., both the CSA/CST and the pattern matching index) are worst-case, and all our algorithms are deterministic.

Our data structures differ significantly from the CSA of Grossi and Vitter [42], the FM-index of Ferragina and Manzini [27], and the CST of Sadakane [79], which are based on the so-called Ψ function [42] or the Burrows–Wheeler transform [15]. We instead rely on the combination of the recently developed notion of *string synchronizing sets (SSS)* [50] and the new type of queries we call *prefix rank* and *prefix selection* queries. Although the prior work on SSS [50, 52] laid out its basic properties, it cannot be turned into an efficient CSA, because it heavily relies on *orthogonal range counting* queries [18], which are *provably incapable* of supporting SA queries as fast as the CSA or FM-index: Pătraşcu [76] showed a lower bound $\Omega(\frac{\log n}{\log\log n})$ on the query time of any structure using near-linear space. On the other hand, the $\mathcal{O}(n \log \sigma/\sqrt{\log n})$-time BWT construction from [50] is not sufficient to obtain an implementation of CSA since the classical BWT-based CSA, in addition to BWT, requires SA samples, i.e., a set containing all pairs $(\mathrm{SA}^{-1}[j], j)$ such that j is a multiple of $\log n$, and it is not known how to obtain such a sequence using prior techniques. The key difficulty is computing the (global) rank $\mathrm{SA}^{-1}[j]$ of each sampled suffix; an easy application of sparse suffix sorting gives, in $\mathcal{O}(n/\log_\sigma n)$ time, the lexicographic order of the sampled suffixes, but this is insufficient for placing each sampled suffix among the n suffixes of the original string.

We sidestep these obstacles and demonstrate that *general* orthogonal range counting queries [19, 18] are in fact not needed at all, and each of their uses can either be: (1) eliminated completely (see the proof in Section 5.3.6), (2) replaced with prefix rank/selection queries (see Section 4.3), or (3) improved, utilizing the fact that the instances arising in our construction have properties that permit a fast custom solution (see Section 4.4). More details are provided in the Technical Overview (Section 3). As a result, we obtain a general set of reductions for the construction of CSA/CST and pattern-matching indexes, stated in Theorems 5.2, 6.3, and 7.2. In a single theorem, we can summarize it as follows; note that our reduction achieves *near-perfect efficiency*, i.e., it incurs no overhead (compared to the optimal solution) in space, preprocessing time, and preprocessing space, and only has an extra $\mathcal{O}(\log\log n)$ term in the query time. Everything else depends entirely on prefix rank and selection queries.

THEOREM 1.1. (MAIN RESULT OF THIS PAPER) *Consider a data structure answering prefix rank and selection queries (Section 2.1) that, for any string of length m over alphabet $[0\,..\,\sigma)^{\ell}$ (or equivalently, a sequence of m length-ℓ strings over alphabet $[0\,..\,\sigma)$), achieves:*

1. *Space usage $S(m, \ell, \sigma)$ (measured in $\Theta(\log m)$-bit machine words),*
2. *Preprocessing time $P_t(m, \ell, \sigma)$,*
3. *Preprocessing space $P_s(m, \ell, \sigma)$,*
4. *Query time $Q(m, \ell, \sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/7}$, there exist $m = \mathcal{O}(n/\log_\sigma n)$ and $\ell = \mathcal{O}(\log_\sigma n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(m, \ell, \sigma))$ time and $\mathcal{O}(n/\log_\sigma n + P_s(m, \ell, \sigma))$ working space build a structure of size $\mathcal{O}(n/\log_\sigma n + S(m, \ell, \sigma))$ that:

- *Supports* SA *and inverse* SA *queries in $\mathcal{O}(\log\log n + Q(m, \ell, \sigma))$ time;*
- *Supports all suffix tree operations (Table 1) in $\mathcal{O}(\log\log n + Q(m, \ell, \sigma))$ time;*
- *Given the packed representation of any pattern $P \in [0\,..\,\sigma)^m$, returns:*
  - *The range $\mathrm{SA}[b\,..\,e)$ of suffixes of T with prefix P in $\mathcal{O}(m/\log_\sigma n + \log\log n + Q(m, \ell, \sigma))$ time;*
  - *All occ starting positions of P in T in $\mathcal{O}(m/\log_\sigma n + (occ + 1)(\log\log n + Q(m, \ell, \sigma)))$ time.*

Using this general reduction, we obtain the specific tradeoffs for CSA/CST and pattern matching queries we announced earlier by plugging in the data structure for prefix rank/selection queries from Theorem 2.2.

Related Work In parallel to efforts to improve the complexity of CSA/CST construction, were the efforts to make it more practical [38, 39, 40, 48, 73, 75]. This resulted in libraries of compressed data structures such as `sdsl` [38], `sux`, and `libcds`. More recently, some of these data structures have been extended to the dynamic setting, e.g., in the `DYNAMIC` [74] library.

[4] Note that if $\log\sigma = \mathcal{O}(\sqrt{\log n})$, then $\log^{\epsilon}_{\sigma} n = \Theta(\log^{\epsilon'} n)$ holds for $\epsilon' = \frac{\epsilon}{2}$.

In addition to CSA/CST and indexes using the optimal space of $\mathcal{O}(n \log \sigma)$ bits, previous work addressed the problem of designing structures using $\omega(n \log \sigma)$ but still $o(n \log n)$ bits [28, 35, 43]. We formulated our main result (Theorem 1.1) as a general reduction so that techniques from these and similar future studies could be easily combined with ours, potentially yielding new tradeoffs for pattern matching and CSA/CST queries.

In recent years, there has also been progress in the query time of $\mathcal{O}(n \log n)$-bit pattern matching indexes. The suffix trees support $\mathcal{O}(m)$-time pattern search after $\mathcal{O}(n)$-time randomized or $\mathcal{O}(n \log\log \sigma)$-time deterministic construction [26, 78]. Fischer and Gawrychowski [30] achieved $\mathcal{O}(m + \log\log \sigma)$-time queries after $\mathcal{O}(n)$-time deterministic construction, improving upon [22, 60]. If the pattern is given using $\mathcal{O}(m \log \sigma)$ bits, Bille et al. [12] achieved $\mathcal{O}(m/\log_\sigma n + \log m + \log\log \sigma)$ time, which Navarro and Nekrich [69] improved to $\mathcal{O}(m/\log_\sigma n + 1)$.

Surprisingly, the size of some CSAs, CSTs, and compact indexes can be reduced *below* $n\lceil \log \sigma \rceil$ bits for statistically compressible texts. For example, already the original FM-index [27] takes only $\mathcal{O}(nH_k(T)) + o(n \log \sigma)$ bits, where $H_k(T)$ denotes the *empirical kth-order entropy* of the text T [23]. Currently, the smallest indexes reach $nH_k(T) + o(n(H_k(T) + 1))$ bits [4, 8]. Navarro and Mäkinen [68], and Belazzougui and Navarro [7] survey the achievable tradeoffs for such *fully compressed* indexes. Chan et al. [17], and Mäkinen and Navarro [59] describe *dynamic* compressed pattern-matching indexes maintaining a collection of texts supporting insertions/deletions.

Compressed indexes based on LZ77 [82] and run-length BWT [15] rapidly gain popularity. The early indexes [6, 11, 13, 32, 33] support only pattern search and random-access operations. Subsequent works generalized them to other dictionary compressors [20, 53, 70] and added dynamism [36, 71]. Support for SA queries is a recent addition of Gagie et al. [34]. Navarro surveys these indexes [67] and the intricate network of the underlying compressibility measures [66]. Interestingly, some of these pattern matching indexes can be constructed in compressed time. For example, the index of [36] can be constructed in $\mathcal{O}(z \log^3 n)$ time from the LZ77 representation of T (with z phrases), and then it locates pattern occurrences in $\mathcal{O}(m + occ \log n)$ time. On the other hand, the only compressed index supporting SA queries [34] is only constructible in $\Omega(n)$ time, but it can be built in compressed space $\mathcal{O}(r \log(n/r))$ given the run-length BWT of T (with r runs).

Organization of the Paper After introducing the basic notation and tools in Section 2, we give a technical overview of the paper in Section 3. In Section 4, we then introduce some auxiliary tools utilized in our data structures. Section 5 describes our data structure answering SA and SA^{-1} queries. In Section 6, we present our index for counting and reporting occurrences of patterns given using packed representation. Finally, in Section 7, we extend the functionality of our CSA into that of a CST.

2 Preliminaries

A *string* is a finite sequence of characters from a given *alphabet*. The length of a string S is denoted $|S|$. For $i \in [1\,..\,|S|]$,[5] the ith character of S is denoted $S[i]$. A *substring* of S is a string of the form $S[i\,..\,j) = S[i]S[i+1]\cdots S[j-1]$ for some $1 \le i \le j \le |S|+1$. *Prefixes* and *suffixes* are substrings of the form $S[1\,..\,j)$ and $S[i\,..\,|S|]$, respectively. We use $\overline{S}$ to denote the *reverse* of S, i.e., $S[|S|]\cdots S[2]S[1]$. We denote the *concatenation* of two strings U and V, that is, $U[1]\cdots U[|U|]V[1]\cdots V[|V|]$, by UV or $U \cdot V$. Furthermore, $S^k = \bigodot_{i=1}^{k} S$ is the concatenation of $k \in \mathbb{Z}_{\ge 0}$ copies of S; note that $S^0 = \varepsilon$ is the *empty string*. For a non-empty string $S \in \Sigma^+$, we define the special infinite string S^∞ such that $S^\infty[i] = S[1 + (i-1) \bmod |S|]$ holds for every $i \in \mathbb{Z}$; in particular, $S^\infty[1\,..\,|S|] = S[1\,..\,|S|]$. An integer $p \in [1\,..\,|S|]$ is a *period* of S if $S[i] = S[i+p]$ holds for every $i \in [1\,..\,|S| - p]$. We denote the shortest period of S as $\mathrm{per}(S)$.

Throughout the paper, we consider a string (called the *text*) T of length $n \ge 2$ over an integer alphabet $\Sigma = [0\,..\,\sigma)$, where $\sigma = n^{\mathcal{O}(1)}$. We assume $T[n] = 0$, and that 0 (also denoted with \$) does not appear elsewhere in T. We use $\preceq$ to denote the order on Σ, extended to the *lexicographic* order on Σ^* (the set of strings over Σ) so that $U, V \in \Sigma^*$ satisfy $U \preceq V$ if and only if either U is a prefix of V, or $U[1\,..\,i) = V[1\,..\,i)$ and $U[i] \prec V[i]$ holds for some $i \in [1\,..\,\min(|U|,|V|)]$. The *suffix array* $\mathrm{SA}[1\,..\,n]$ of T is a permutation

i	SA[i]	T[SA[i]..n]
1	18	\$
2	17	a\$
3	12	aababa\$
4	3	aababababaababa\$
5	15	aba\$
6	10	abaababa\$
7	1	abaababababaababa\$
8	13	ababa\$
9	8	abababaababa\$
10	6	ababababaababa\$
11	4	abababababaababa\$
12	16	ba\$
13	11	baababa\$
14	2	baababababaababa\$
15	14	baba\$
16	9	babaababa\$
17	7	bababaababa\$
18	5	bababababaababa\$

Figure 1: A list of all sorted suffixes of T = abaababababaababa\$ along with the suffix array of T.

[5] For $i, j \in \mathbb{Z}$, denote $[i\,..\,j] = \{k \in \mathbb{Z} : i \le k \le j\}$, $[i\,..\,j) = \{k \in \mathbb{Z} : i \le k < j\}$, and $(i\,..\,j] = \{k \in \mathbb{Z} : i < k \le j\}$.

of $[1\,..\,n]$ such that $T[\mathrm{SA}[1]\,..\,n] \prec T[\mathrm{SA}[2]\,..\,n] \prec \cdots \prec T[\mathrm{SA}[n]\,..\,n]$, i.e., $\mathrm{SA}[i]$ is the starting position of the lexicographically ith suffix of T; see Fig. 1 for an example. The *inverse suffix array* $\mathrm{ISA}[1\,..\,n]$ (also denoted $\mathrm{SA}^{-1}[1\,..\,n]$) is the inverse permutation of SA, i.e., $\mathrm{ISA}[j] = i$ holds if and only if $\mathrm{SA}[i] = j$. Intuitively, $\mathrm{ISA}[j]$ stores the lexicographic *rank* of a suffix $T[j\,..\,n]$ among the suffixes of T. By $\mathrm{lcp}(U, V)$ we denote the length of the longest common prefix of U and V. For $j_1, j_2 \in [1\,..\,n]$, we let $\mathrm{LCE}(j_1, j_2) = \mathrm{lcp}(T[j_1\,..], T[j_2\,..])$. For any $P, S \in \Sigma^*$, we let

$$\begin{aligned}\mathrm{Occ}(P, S) &= \{j \in [1\,..\,|S|] : j + |P| \le |S| + 1 \text{ and } S[j\,..\,j+|P|) = P\},\\ \mathrm{RangeBeg}(P, S) &= |\{i \in [1\,..\,|S|] : S[i\,..\,|S|] \prec P\}|,\\ \mathrm{RangeEnd}(P, S) &= \mathrm{RangeBeg}(P, S) + |\mathrm{Occ}(P, S)|.\end{aligned}$$

Observe that the following equality holds for every $P \in \Sigma^*$:

$$\mathrm{Occ}(P, T) = \{\mathrm{SA}[i] : i \in (\mathrm{RangeBeg}(P, T)\,..\,\mathrm{RangeEnd}(P, T))\}.$$

We use the word RAM model of computation [45] with w-bit *machine words*, where $w \ge \log n$. In this model, strings are typically represented as arrays, with each character occupying one memory cell. A single character, however, only needs $\lceil \log \sigma \rceil$ bits, which might be much less than w. We can therefore store (the *packed representation* of) a text $T \in [0\,..\,\sigma)^n$ using $\mathcal{O}\big(\lceil \frac{n \log \sigma}{w} \rceil\big)$ memory cells.

2.1 (Prefix) Rank and Selection Queries

Let us recall the (ordinary) rank and selection queries on a string $S \in \Sigma^n$:

Rank query $\mathsf{rank}_{S,a}(j)$**:** Given $a \in \Sigma$ and $j \in [0\,..\,n]$, compute $|\{i \in [1\,..\,j] : S[i] = a\}|$.

Selection query $\mathsf{select}_{S,a}(r)$**:** Given $a \in \Sigma$ and $r \in [1\,..\,\mathsf{rank}_{S,a}(n)]$, find the rth smallest element of $\{i \in [1\,..\,n] : S[i] = a\}$.

Theorem 2.1. (Rank and selection queries in bitvectors [3, 21, 47, 64]) *For every string $S \in \{0,1\}^*$, there exists a data structure of $\mathcal{O}(|S|)$ bits answering rank and selection queries in $\mathcal{O}(1)$ time. Moreover, given the packed representations of m binary strings of total length n, the data structures for all these strings can be constructed in $\mathcal{O}(m + n/\log n)$ time.*

Next, we provide a generalization of rank and selection queries specific to sequences of strings (strings whose characters are strings themselves). Let $W \in (\Sigma^*)^m$ be a sequence of m strings.

Prefix rank query $\mathsf{rank}_{W,X}(j)$**:** Given $X \in \Sigma^*$ and $j \in [0\,..\,m]$, compute $|\{i \in [1\,..\,j] : X \text{ is a prefix of } W[i]\}|$.

Prefix selection query $\mathsf{select}_{W,X}(r)$**:** Given $X \in \Sigma^*$ and $r \in [1\,..\,\mathsf{rank}_{W,X}(m)]$, find the rth smallest element of $\{i \in [1\,..\,m] : X \text{ is a prefix of } W[i]\}$.

The following result, proved in Section 4.3 by building on the results of Belazzougui and Puglisi [9], provides an efficient implementation of prefix rank and selection queries. Note that we require W to consist of same-length strings over an integer alphabet.

Theorem 2.2. *For all integers $m, \ell, \sigma \in \mathbb{Z}_{\ge 1}$ satisfying $m \ge \sigma^\ell \ge 2$, every constant $\epsilon > 0$, and every string $W \in ([0\,..\,\sigma)^\ell)^{\le m}$, there exists a data structure of size $\mathcal{O}(m)$ answering prefix rank queries in $\mathcal{O}(\ell^{\epsilon/2} \log\log m) = \mathcal{O}(\log^\epsilon m)$ time and prefix selection queries in $\mathcal{O}(\ell^{\epsilon/2}) = \mathcal{O}(\log^\epsilon m)$ time. Moreover, it can be constructed in $\mathcal{O}(m \min(\ell, \sqrt{\log m}))$ time using $\mathcal{O}(m)$ working space given the packed representation of W and the constant parameter $\epsilon > 0$.*

2.2 Range Counting and Selection

Let $A[1\,..\,m]$ be an array of nonnegative integers. We define the following queries on A:

Range counting query $\mathsf{rcount}_A(v, j)$**:** Given an integer $v \ge 0$ and a position $j \in [0\,..\,m]$, compute $|\{i \in [1\,..\,j] : A[i] \ge v\}|$.

Range selection query $\mathsf{rselect}_A(v, r)$**:** Given integers $v \geq 0$ and $r \in [1\,..\,\mathsf{rcount}_A(v, m)]$, find the rth smallest element of $\{i \in [1\,..\,m] : A[i] \geq v\}$.

The currently fastest general-purpose data structure for range counting/selection queries is described in [18, Theorems 2.3 and 3.3]. The instances in our construction, however, satisfy an additional property, namely, that the sum $\sum_{i=1}^{m} A[i]$ is bounded. This lets us obtain a solution with faster queries and smaller construction time; see Section 4.4.

PROPOSITION 2.1. *An array $A[1..m']$ of $m' \in [2..m]$ nonnegative integers satisfying $\sum_{i=1}^{m'} A[i] = \mathcal{O}(m \log m)$ can be preprocessed in $\mathcal{O}(m)$ time so that range counting and selection queries can be answered in $\mathcal{O}(\log \log m)$ time and $\mathcal{O}(1)$ time, respectively.*

2.3 String Synchronizing Sets

DEFINITION 2.1. (τ-SYNCHRONIZING SET [50]) *Let $T \in \Sigma^n$ be a string and let $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$ be a parameter. A set $\mathsf{S} \subseteq [1\,..\,n - 2\tau + 1]$ is called a* τ-synchronizing set *of T if it satisfies the following* consistency *and* density *conditions:*

1. *If $T[i\,..\,i + 2\tau) = T[j\,..\,j + 2\tau)$, then $i \in \mathsf{S}$ holds if and only if $j \in \mathsf{S}$ (for $i, j \in [1\,..\,n - 2\tau + 1]$),*
2. *$\mathsf{S} \cap [i\,..\,i + \tau) = \emptyset$ if and only if $i \in \mathsf{R}(\tau, T)$ (for $i \in [1\,..\,n - 3\tau + 2]$), where*

$$\mathsf{R}(\tau, T) := \{i \in [1\,..\,|T| - 3\tau + 2] : \mathrm{per}(T[i\,..\,i + 3\tau - 2]) \leq \tfrac{1}{3}\tau\}.$$

In most applications, we want to minimize $|\mathsf{S}|$. Note, however, that the density condition imposes a lower bound $|\mathsf{S}| = \Omega(\frac{n}{\tau})$ for strings of length $n \geq 3\tau - 1$ that do not contain substrings of length $3\tau - 1$ which are periodic with period $\leq \frac{1}{3}\tau$. Thus, we cannot hope to achieve an upper bound improving in the worst case upon the following one.

THEOREM 2.3. ([50, PROPOSITION 8.10]) *For any string T of length n and parameter $\tau \in [1\,..\,\lfloor \frac{n}{2} \rfloor]$, there exists a τ-synchronizing set S of size $|\mathsf{S}| = \mathcal{O}\left(\frac{n}{\tau}\right)$. Moreover, if $T \in [0\,..\,\sigma)^n$, where $\sigma = n^{\mathcal{O}(1)}$, such S can be deterministically constructed in $\mathcal{O}(n)$ time.*

Note that when $\tau = \omega(1) \cap \mathcal{O}(\log_\sigma n)$ and $T \in [0\,..\,\sigma)^n$ is given in the packed representation, the first part of Theorem 2.3 opens the possibility of an algorithm running in $\mathcal{O}(\frac{n}{\tau}) = o(n)$ time. In [50], it was shown that this lower bound is achievable (the upper bound $\tau = \mathcal{O}(\log_\sigma n)$ follows from the fact that every algorithm needs to at least read the input, which takes $\Theta(n/\log_\sigma n)$ time; thus, for larger τ, the algorithm cannot run in $\mathcal{O}(\frac{n}{\tau})$ time).

THEOREM 2.4. ([50, THEOREM 8.11]) *For every constant $\mu < \frac{1}{5}$, given the packed representation of a text $T \in [0\,..\,\sigma)^n$ and a positive integer $\tau \leq \mu \log_\sigma n$, one can deterministically construct in $\mathcal{O}(\frac{n}{\tau})$ time a τ-synchronizing set of size $\mathcal{O}(\frac{n}{\tau})$.*

3 Technical Overview

In this section, we give an overview of our data structures to answer SA and ISA queries (Section 3.1), pattern matching queries (Section 3.2), and suffix tree queries (Section 3.3). Each subsection contains a summary of the key new techniques.

3.1 SA and ISA Queries

Let $\epsilon \in (0, 1)$ and $T \in [0\,..\,\sigma)^n$, where $2 \leq \sigma < n^{1/7}$. In this section, we give an overview of our data structure to compute the value of $\mathrm{SA}[i]$ (resp. $\mathrm{ISA}[j]$) given any $i \in [1\,..\,n]$ (resp. $j \in [1\,..\,n]$) in $\mathcal{O}(\log^\epsilon n)$ time. The data structure uses $\mathcal{O}(n/\log_\sigma n)$ space. We assume $\sigma < n^{1/7}$ since for larger σ the plain representations of SA and ISA use $\mathcal{O}(n \log n) = \mathcal{O}(n \log \sigma)$ bits and can be constructed in $\mathcal{O}(n)$ time [49].

Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where $\mu < \frac{1}{6}$ is a positive constant chosen so that $\tau \geq 1$ (such μ exists by $\sigma < n^{1/7}$). We use R as a shorthand for $\mathsf{R}(\tau, T)$ (see Definition 2.1). Our data structure to compute $\mathrm{SA}[i]$ (resp. $\mathrm{ISA}[j]$) works differently depending on whether $\mathrm{SA}[i] \in \mathsf{R}$ (resp. $j \in \mathsf{R}$). To check if $\mathrm{SA}[i] \in \mathsf{R}$, we store a bitvector

$B_{3\tau-1}$ marking boundaries between the blocks of suffixes in SA sharing the length-$(3\tau-1)$ prefix. We also store the sequence A_{short} of those prefixes (by $\mu < \frac{1}{6}$, it needs $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n^{3\mu}) = o(n/\log_\sigma n)$ space). Given any $i \in [1\,..\,n]$, we can then check if $\mathrm{SA}[i] \in \mathsf{R}$ by first computing the block k containing position i using a rank query on $B_{3\tau-1}$, and then checking (using a lookup table) if $X = A_{\text{short}}[k]$ satisfies $\mathrm{per}(X) \le \frac{1}{3}\tau$. As for an ISA query, checking if $j \in \mathsf{R}$ only needs the lookup table (since we store T).

The Nonperiodic Positions We first focus on computing $\mathrm{ISA}[j]$. Let $j \notin \mathsf{R}$ and let S be a τ-synchronizing set of T of size $n' := |\mathsf{S}| = \mathcal{O}(n/\tau)$ (such S exists and can be quickly constructed using Theorem 2.4). The query algorithm relies on the following two observations:

Observation 1: S induces a partitioning of SA *into blocks.* The density condition of S implies $\mathsf{S} \cap [j\,..\,j+\tau) \neq \emptyset$, i.e., the successor of j in S, denoted $s := \mathrm{succ}_{\mathsf{S}}(j)$, satisfies $s < j + \tau$. Hence, the string $X := T[j\,..\,s+2\tau)$, called the *distinguishing prefix* of $T[j\,..\,n]$, is of length $|X| \le 3\tau - 1$. By the local consistency of S, the set $\mathcal{D}$ of distinguishing prefixes of all suffixes $T[j\,..\,n]$ with $j \notin \mathsf{R}$ is prefix-free (i.e., no string in $\mathcal{D}$ is a prefix of another). All positions in $[1\,..\,n] \setminus \mathsf{R}$ in the SA of T can thus be partitioned into disjoint blocks according to distinguishing prefixes. Since $\mathcal{D} \subseteq [0\,..\,\sigma)^{\le 3\tau-1}$, the number of blocks is $\mathcal{O}(\sigma^{3\tau-1})$, so we can store their boundaries in a lookup table of size $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n^{3\mu}) = o(n/\log_\sigma n)$. To efficiently determine $\mathrm{succ}_{\mathsf{S}}(j)$, we store a bitvector marking positions in S, augmented with $\mathcal{O}(1)$-time rank and select queries. Once the block $\mathrm{SA}(b\,..\,e]$ for $X = T[j\,..\,\mathrm{succ}_{\mathsf{S}}(j)+2\tau)$ is found, it remains to locate j within that block.

Observation 2: The order in each block is consistent with S. Assume $\mathrm{SA}(b\,..\,e]$ represents all suffixes of T having $X = T[j\,..\,\mathrm{succ}_{\mathsf{S}}(j)+2\tau)$ as a prefix. By the consistency condition, letting $\delta_{\text{text}} = |X| - 2\tau$, for every $i \in (b\,..\,e]$, we have $\mathrm{succ}_{\mathsf{S}}(\mathrm{SA}[i]) = \mathrm{SA}[i] + \delta_{\text{text}}$. Thus, letting $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$ contain S sorted by the corresponding suffixes $T[s_i^{\text{lex}}\,..\,n]$, positions in $\mathrm{SA}(b\,..\,e]$ increased by δ_{text} occur in $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$ in the same relative order. Hence, if we define $W[i] = \overline{X_i}$, where $X_i = T^\infty[s_i^{\text{lex}} - \tau\,..\,s_i^{\text{lex}} + 2\tau)$, and select $W[y]$ as the kth string in W having $\overline{X}$ as a prefix, then $s_y^{\text{lex}} - \delta_{\text{text}} = \mathrm{SA}[b+k]$ is the kth position in $\mathrm{SA}(b\,..\,e]$. Thus, to obtain $\mathrm{ISA}[j]$, it suffices to find the index y such that $s_y^{\text{lex}} = \mathrm{succ}_{\mathsf{S}}(j)$. Then, the offset of j in the block $\mathrm{SA}(b\,..\,e]$ is $\mathsf{rank}_{W,\overline{X}}(y)$. To efficiently determine y, we store the permutation that maps elements of S sorted left-to-right to elements of $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$. This lets us determine y in $\mathcal{O}(1)$ time. We then compute $\mathsf{rank}_{W,\overline{X}}(y)$ in $\mathcal{O}(\log^\epsilon n)$ time using Theorem 2.2; see Proposition 5.4.

Let us now turn to the computation of $\mathrm{SA}[i]$ when $\mathrm{SA}[i] \notin \mathsf{R}$. Using a rank query on $B_{3\tau-1}$ and an access to A_{short}, we first determine the length-$(3\tau-1)$ prefix of $T[\mathrm{SA}[i]\,..\,n]$. A lookup table lets us retrieve the prefix $X \in \mathcal{D}$ of $T[\mathrm{SA}[i]\,..\,n]$ and the boundaries of the corresponding block $\mathrm{SA}(b\,..\,e]$. By Observation 2 above, it remains to determine the index y of the $(i-b)$th leftmost string in W having $\overline{X}$ as a prefix, which we compute in $\mathcal{O}(\log^\epsilon n)$ time as $y = \mathsf{select}_{W,\overline{X}}(i-b)$ using Theorem 2.2. We then have $\mathrm{SA}[i] = s_y^{\text{lex}} - \delta_{\text{text}}$, where $\delta_{\text{text}} = |X| - 2\tau$. See Proposition 5.5 for details.

The Periodic Positions Let $j \in \mathsf{R}$. We again first focus on computing $\mathrm{ISA}[j]$. As before, we aim to find the location of j in the block $\mathrm{SA}(b\,..\,e]$ containing all suffixes of T prefixed with $X = T[j\,..\,j+3\tau-1)$. Note that all positions in $\mathrm{SA}(b\,..\,e]$ are in R. The challenge is thus to devise a way to compare suffixes starting in R. The problem with applying a similar approach as before is that the size of R can reach $\Theta(n)$. There exists, however, a subset of R that, when combined with a bitvector representing remaining positions in R, can be applied here. We derive it as follows:

Structure of R *in the left-to-right (text) order:* The gap between $|X| = 3\tau - 1$ and $\mathrm{per}(X) \le \frac{1}{3}\tau$ ensures that every maximal block of positions in R corresponds to a *τ-run*, i.e., a maximal substring of T of length $\ge 3\tau - 1$ whose shortest period is $\le \frac{1}{3}\tau$ (Lemma 5.3). Since any two τ-runs overlap by at most $\frac{2}{3}\tau$ positions (see the proof of Lemma 5.6), their number is $\mathcal{O}(n/\tau)$. We can thus succinctly encode R by storing the set R' of τ-run starting positions.

Structure of R *in the lexicographic order:* For $x \in \mathsf{R}$, let $e(x)$ denote the position following the τ-run containing x. Observe that, for every $x \in \mathsf{R}$, we can uniquely write $T[x\,..\,e(x)) = H'H^kH''$, where H is the lexicographically smallest rotation of $T[x\,..\,x+p)$, $p = \mathrm{per}(T[x\,..\,e(x)))$, and H' (resp. H'') is a proper suffix (resp. prefix) of H (Section 5.3.1). Denote $\text{L-root}(x) = H$, $\text{L-head}(x) = |H'|$, $\text{L-exp}(x) = k$, and $\text{L-tail}(x) = |H''|$. Let also $\mathrm{type}(x) = -1$ if $T[e(x)] \prec T[e(x) - |H|]$ and $\mathrm{type}(x) = +1$ otherwise. Then, in $\mathrm{SA}(b\,..\,e]$, all positions x with $\mathrm{type}(x) = -1$ precede all x with $\mathrm{type}(x) = +1$. Moreover, the value of $e(x) - x$ is non-decreasing (resp.

non-increasing) among the positions x with $\mathrm{type}(x) = -1$ (resp. $\mathrm{type}(x) = +1$); see Lemma 5.4.

Denote $\mathsf{R}^- = \{x \in \mathsf{R} : \mathrm{type}(x) = -1\}$, $\mathsf{R}'^- = \mathsf{R}' \cap \mathsf{R}^-$, $\mathsf{R}_H = \{x \in \mathsf{R} : \text{L-root}(x) = H\}$, $\mathsf{R}'^-_H = \mathsf{R}'^- \cap \mathsf{R}_H$, $\mathsf{R}_{s,H} = \{x \in \mathsf{R}_H : \text{L-head}(x) = s\}$, and $\mathsf{R}^-_{s,H} = \mathsf{R}^- \cap \mathsf{R}_{s,H}$. Assume $\mathrm{type}(j) = -1$ (the case of $\mathrm{type}(j) = +1$ is symmetric), $\text{L-head}(j) = s$, and $\text{L-root}(j) = H$. Given the above structural insights, we can phrase locating j in $\mathrm{SA}(b\,..\,e]$ as counting the positions $x \in \mathsf{R}^-_{s,H}$ satisfying $T[x\,..\,n] \preceq T[j\,..\,n]$. By the analysis above, all such positions satisfy $e(x) - x \le e(j) - j$ and hence $\text{L-exp}(x) \le \text{L-exp}(j)$. We first compute the size of $\mathrm{Pos}^{\mathsf{a}}(j) := \{x \in \mathsf{R}^-_{s,H} : \text{L-exp}(x) \le \text{L-exp}(j)\}$ and then subtract the size of $\mathrm{Pos}^{\mathsf{s}}(j) := \{x \in \mathsf{R}^-_{s,H} : \text{L-exp}(x) = \text{L-exp}(j) \text{ and } T[x\,..\,n] \succ T[j\,..\,n]\}$ as follows:[6]

- Since $|\mathrm{Pos}^{\mathsf{a}}(j)|$ only depends on $\text{L-exp}(j)$, it suffices to store a bitvector B_{exp} marking the boundaries in SA between blocks of positions with subsequent values of L-exp. Computing $|\mathrm{Pos}^{\mathsf{a}}(j)|$ then reduces to $\mathcal{O}(1)$-time rank and selection queries on B_{exp} (Proposition 5.9).
- As for $|\mathrm{Pos}^{\mathsf{s}}(j)|$, we observe that $\mathrm{Pos}^{\mathsf{s}}(j)$ contains at most one position in each τ-run (Lemma 5.10). We store all $x \in \mathsf{R}'^-_H$ sorted by the suffix starting right after the last occurrence of H, i.e., $T[e^{\mathrm{full}}(x)\,..\,n]$, where $e^{\mathrm{full}}(x) := e(x) - \text{L-tail}(x)$. In a separate array, we also record $e^{\mathrm{full}}(x) - x$ at the corresponding position. This lets us compute $|\mathrm{Pos}^{\mathsf{s}}(j)|$ by first locating the block of positions $x \in \mathsf{R}'^-_H$ for which $T[e^{\mathrm{full}}(x)\,..\,n] \succ T[e^{\mathrm{full}}(j)\,..\,n]$, and then counting the ones with $e^{\mathrm{full}}(x) - x \ge e^{\mathrm{full}}(j) - j$ (Proposition 5.10). Since the sum of $e^{\mathrm{full}}(x) - x$ over all $x \in \mathsf{R}'$ is $\mathcal{O}(n)$ (Section 5.3.2), we use a specialized structure for range counting (Proposition 2.1), bypassing the general $\Omega(\frac{\log n}{\log\log n})$-time lower bound [76].

Let us now turn to an $\mathrm{SA}[i]$ query with $\mathrm{SA}[i] \in \mathsf{R}^-$. First, we determine $T[\mathrm{SA}[i]\,..\,\mathrm{SA}[i] + 3\tau - 1)$ using $B_{3\tau-1}$ and A_{short}, as well as $s = \text{L-head}(\mathrm{SA}[i])$ and $H = \text{L-root}(\mathrm{SA}[i])$ using a lookup table (Proposition 5.8). Then, rank and selection queries on B_{exp} let us easily determine $\text{L-exp}(\mathrm{SA}[i])$ and $|\mathrm{Pos}^{\mathsf{s}}(\mathrm{SA}[i])|$ (Proposition 5.12). As explained above, to compute $\mathrm{SA}[i]$, it remains to first select the kth (where $k = |\mathrm{Pos}^{\mathsf{s}}(\mathrm{SA}[i])| + 1$) largest element $j \in \mathsf{R}'^-_H$ according to the string $T[e^{\mathrm{full}}(j)\,..\,n]$, among positions $j'' \in \mathsf{R}'^-_H$ satisfying $e^{\mathrm{full}}(j'') - j'' \ge \text{L-head}(\mathrm{SA}[i]) + \text{L-exp}(\mathrm{SA}[i]) \cdot |H|$. The position $j' \in [j\,..\,e(j) - 3\tau + 2)$ with $\text{L-exp}(j') = \text{L-exp}(\mathrm{SA}[i])$ and $\text{L-head}(j') = \text{L-head}(\mathrm{SA}[i])$ must then satisfy $\mathrm{SA}[i] = j'$. To compute j, we use our specialized data structure for range queries (Proposition 2.1). Position j' is then obtained by subtracting $\text{L-head}(\mathrm{SA}[i]) + \text{L-exp}(\mathrm{SA}[i]) \cdot |H|$ from $e^{\mathrm{full}}(j)$ (Proposition 5.13).

In total, the query time for periodic positions is $\mathcal{O}(\log\log n)$ (Propositions 5.11 and 5.14).

Summary of New Techniques The key distinctive feature of our technique is the use of local consistency without general orthogonal range queries, present in prior approaches [20, 50, 51, 52]. This lets us sidestep Pătraşcu's $\Omega(\frac{\log n}{\log\log n})$ lower bound [76], which is achieved in three steps:

- We replace range counting/selection in the nonperiodic case with *prefix rank* and *prefix selection*, for which we propose a new tradeoff by plugging in the technique of Belazzougui and Puglisi [9]. This reveals the power of our reduction: we achieve a non-trivial tradeoff for complex queries by solving a simple bit-permuting problem (note that the tradeoff behind Theorem 2.2, e.g., occupies only 1.5 pages in Section 4.3; the bulk of our paper is the reduction).
- We replace range counting/selection in the periodic case by observing that the sum of coordinates is small (Section 5.3.2). This lets us use a specialized solution (Section 4.4).
- The above two cases occur at query time. The third case concerns the construction of the structure. More precisely, we completely eliminate range queries naturally occurring during the construction of components for periodic positions [50, 51] by a complex bit-optimal algorithm for the construction of the bitvector B_{exp} (see Section 5.3.6).

As a result, we obtain a very general reduction stated in Theorem 5.2.

3.2 Pattern Matching Queries

Let $\epsilon \in (0, 1)$ and $T \in [0\,..\,\sigma)^n$ be as in Section 3.1. We now give an overview of our data structure that, given a packed representation of any pattern $P \in [0\,..\,\sigma)^m$, returns the pair of indexes $(b, e) = (\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T))$, i.e., the boundaries of the SA block containing all suffixes having P as

[6] Note that here we first overestimate the number of smaller suffixes in $\mathrm{SA}(b\,..\,e]$ and then subtract the larger suffixes. We explain the reason for this counterintuitive approach in Remark 5.1.

a prefix, in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time. Note that having this range immediately gives us $|\mathrm{Occ}(P,T)| = e - b$, i.e., it implements *pattern counting*. Moreover, combined with the result from Section 3.1, we obtain *pattern reporting*, i.e., we can enumerate $\mathrm{Occ}(P,T)$ in $\mathcal{O}(m/\log_\sigma n + (|\mathrm{Occ}(P,T)|+1)\log^\epsilon n)$ time.

Let τ be as in Section 3.1. Our structure to compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ works differently depending on whether $m \geq 3\tau - 1$ and whether $\mathrm{per}(P[1\,..\,3\tau-1]) \leq \frac{1}{3}\tau$ (such P is called *periodic*) or not. Checking if P is periodic is easily implemented via a lookup table.

The Nonperiodic Patterns Let us assume $m \geq 3\tau - 1$ (shorter patterns are handled using a precomputed array) and let S be as in Section 3.1. The basic idea of the pattern matching query is to decompose $P = XY$, where $X \in \mathcal{D}$, and then utilize the following observation about S (generalizing the second observation in Section 3.1):

Observation: The order in the suffix array range corresponding to $\mathrm{Occ}(P,T)$ *is consistent with* S. Let $(b,e) = (\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$. By the consistency condition, for every $i \in (b\,..\,e]$, we have $\mathrm{succ}_\mathsf{S}(\mathrm{SA}[i]) = \mathrm{SA}[i] + \delta_{\mathrm{text}}$, where $\delta_{\mathrm{text}} = |X| - 2\tau$. Thus, letting $(s_i^{\mathrm{lex}})_{i\in[1\,..\,n']}$ be defined as in Section 3.1, the positions in $\mathrm{SA}(b\,..\,e]$ increased by δ_{text} occur in $(s_i^{\mathrm{lex}})_{i\in[1\,..\,n']}$ in the same relative order. Therefore, to compute $|\mathrm{Occ}(P,T)|$, it suffices to first locate a range of $(s_i^{\mathrm{lex}})_{i\in[1\,..\,n']}$ consisting of positions followed by $P(\delta_{\mathrm{text}}\,..\,m]$ in T, and then count those which are additionally preceded with $X[1\,..\,\delta_{\mathrm{text}}]$. The first goal is implemented in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time via a compact trie over the set of strings $\{T[s_i^{\mathrm{lex}}\,..\,n]\}_{i\in[1\,..\,n']}$, reinterpreted as strings over alphabet of size $n^{\Theta(1)}$. The second step reduces to a prefix rank query over $W[1\,..\,n']$ (Section 3.1). This approach easily generalizes to return (b,e) instead of $|\mathrm{Occ}(P,T)|$; see Lemma 6.1.

The Periodic Patterns Let us now assume that $m \geq 3\tau - 1$ and $\mathrm{per}(P[1\,..\,3\tau-1]) \leq \frac{1}{3}\tau$. We first generalize the notion of L-root(x), $e(x)$, and all other functions from positions to strings (Section 6.3.1). The main idea is to decompose P into the periodic prefix $P[1\,..\,e(P))$ and the remaining suffix $P[e(P)\,..\,|P|]$. Let us consider the harder case when $e(P) = |P|+1$ (see Lemma 6.6). We define $\mathrm{Occ}^{\mathsf{a}}(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) > \text{L-exp}(P)\}$ and $\mathrm{Occ}^{\mathsf{s}}(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) = \text{L-exp}(P)\}$, where $H = \text{L-root}(P)$ and $s = \text{L-head}(P)$. The value $|\mathrm{Occ}(P,T)|$ is determined in two steps:

- First, we compute the size of $\mathrm{Occ}^{\mathsf{a}-}(P,T) := \mathrm{Occ}^{\mathsf{a}}(P,T) \cap \mathsf{R}^-$ (the size of $\mathrm{Occ}^{\mathsf{a}+}(P,T)$ is computed symmetrically) as in Section 3.1 by utilizing rank and selection queries on B_{exp} (Proposition 6.7). This only requires knowing L-exp(P), which can be retrieved in $\mathcal{O}(1 + m/\log_\sigma n)$ time (Proposition 6.6).
- Next, we compute the size of $\mathrm{Occ}^{\mathsf{s}-}(P,T) := \mathrm{Occ}^{\mathsf{s}}(P,T) \cap \mathsf{R}^-$. We first show that $\mathrm{Occ}^{\mathsf{s}-}(P,T)$ contains at most one position in every τ-run (Lemma 6.7), i.e., an analogue of Lemma 5.10. The computation is similar as in Section 3.1, except that we use a trie over meta-symbols to find the range of positions $x \in \mathsf{R}'^-_H$ with $T[e^{\mathrm{full}}(x)\,..\,n]$ prefixed by $P[e^{\mathrm{full}}(P)\,..\,m]$. We then perform an $\mathcal{O}(\log\log n)$-time range query (Proposition 6.8). A small complication is to separate positions $x \in \mathsf{R}'^-$ with different L-root(x) in the trie. For this, we insert into the trie suffixes starting slightly earlier than $e^{\mathrm{full}}(x)$; see Section 6.3.2.

The above algorithm generalizes to the computation of (b,e), rather than $|\mathrm{Occ}(P,T)|$, except that handling "fully periodic" patterns (with $e(P) = |P|+1$) requires some care (see Remark 6.2).

Summary of New Techniques Our key technical contributions are as follows:

- We show how to directly apply string synchronizing sets [50] to the problem of pattern matching (not by simply using SA/ISA queries) and consequently obtain a very efficient reduction from pattern matching to prefix rank and prefix selection queries.
- To achieve this, we prove several new combinatorial results for periodic patterns (Section 6.3.1), and then show for efficiently apply them (Sections 6.3.2 to 6.3.4).
- As a result, we obtain the first optimal-size pattern matching index that is constructible in $o(n)$ time and supports:
  - pattern occurrence counting in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time, and
  - pattern occurrence reporting in $\mathcal{O}(m/\log_\sigma n + (|\mathrm{Occ}(P,T)|+1)\log^\epsilon n)$ time.

  This improves over [62, 63] in either construction or query time and, perhaps more importantly, provides a very general reduction that enables achieving further time-space tradeoffs much easier (Theorem 6.3).

3.3 Suffix Tree Queries

Let $\epsilon \in (0,1)$ and $T \in [0\,..\,\sigma)^n$ be as in Section 3.1. In this section, we outline how to extend the techniques presented in Sections 3.1 and 3.2 to obtain the full suffix tree functionality in optimal $\mathcal{O}(n/\log_\sigma n)$ space. All operations (see Table 1) are supported in $\mathcal{O}(\log^\epsilon n)$ time, and the data structure can be constructed in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space. Thus, e.g., for $\sigma = 2$, our construction takes $\mathcal{O}(n/\sqrt{\log n}) = o(n)$ time.

Each node v of the suffix tree of T, denoted $\mathcal{T}_{\text{st}}$, is encoded either as (j, ℓ) such that $T[j\,..\,j+\ell) = \text{str}(v)$ or as $(b, e) = (\text{RangeBeg}(\text{str}(v), T), \text{RangeEnd}(\text{str}(v), T))$, where $\text{str}(v)$ is the string represented by v. Since the latter representation is more common (e.g. [31]), we adopt it as the default interface and denote $\text{repr}(v)$.

In this overview, we focus on the child operation, which illustrates some of our key techniques. We remark, however, that other operations require different combinatorial insights.

Let τ be as in Section 3.1. Our structure works differently depending on whether the operation is performed on a node v such that $\text{str}(v)$ is periodic or not (see Section 3.2).

The Nonperiodic Nodes Let v be a node of $\mathcal{T}_{\text{st}}$ such that $\text{str}(v)$ is nonperiodic and let $c \in \Sigma$. Our goal is to compute $\text{repr}(\text{child}(v, c))$ given $\text{repr}(v)$. Let S again be as in Section 3.1. The basic idea is to reduce the computation concerning the SA-interval for string $\text{str}(v)$ to the computation involving only positions in S. For this purpose, we store the compact trie $\mathcal{T}_{\mathsf{S}}$ for $\{T[s_i^{\text{lex}}\,..\,n]\}_{i\in[1\,..\,n']}$. Typically, each operation on $\mathcal{T}_{\text{st}}$ then involves the following steps:

1. Map the input node v of $\mathcal{T}_{\text{st}}$ (given as $\text{repr}(v)$) to some node u of $\mathcal{T}_{\mathsf{S}}$,
2. Perform some operation in $\mathcal{T}_{\mathsf{S}}$ resulting in a node u' (in our case, $u' = \text{child}(u, c)$),
3. Map u' back to some node v' of $\mathcal{T}_{\text{st}}$ (producing $\text{repr}(v')$ as output).

Mapping from $\mathcal{T}_{\text{st}}$ to $\mathcal{T}_{\mathsf{S}}$: Let $(b, e) = \text{repr}(v)$ and let $X \in \mathcal{D}$ be the distinguishing prefix of $\text{str}(v)$. By the observation for nonperiodic patterns in Section 3.2, the left-to-right order of leaves of $\mathcal{T}_{\mathsf{S}}$ corresponding to suffixes in $\text{SA}(b\,..\,e]$ shifted by $\delta_{\text{text}} = |X| - 2\tau$ is consistent with their order in $\text{SA}(b\,..\,e]$. Moreover, since we know how to compute the position in $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$ corresponding to suffix $T[\text{SA}[i]\,..\,n]$ for any $i \in [1\,..\,n]$ such that $\text{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$ (see Section 3.1), we can compute the pointer to the leaf of $\mathcal{T}_{\mathsf{S}}$ corresponding to any suffix in $\text{SA}(b\,..\,e]$. This implies that: (1) there exists a node $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) := u$ in $\mathcal{T}_{\mathsf{S}}$ such that $\text{str}(v) = X[1\,..\,\delta_{\text{text}}] \cdot \text{str}(u)$, and (2) a pointer to u can be computed via a lowest common ancestor (LCA) query from the leaves of $\mathcal{T}_{\mathsf{S}}$ corresponding to first and last suffix in $\text{SA}(b\,..\,e]$ (see Section 7.2.2).

Mapping from $\mathcal{T}_{\mathsf{S}}$ to $\mathcal{T}_{\text{st}}$: After computing $u' = \text{child}(u, c)$, the next step is to go back to $\mathcal{T}_{\text{st}}$. Our approach exploits a similar principle as when computing $\text{ISA}[j]$: knowing $X \in \mathcal{D}$ and a position in $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$ lets us determine the corresponding position in SA. More generally, given $X \in \mathcal{D}$ and an interval of $(s_i^{\text{lex}})_{i\in[1\,..\,n']}$, we can retrieve the corresponding interval in SA. The former is precomputed and stored with each node of $\mathcal{T}_{\mathsf{S}}$. Note, however, the following complication: $\mathcal{T}_{\mathsf{S}}$ may have extra nodes between $u = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v)$ and $\widehat{u} = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(\text{child}(v, c))$, and then $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(\text{child}(v, c)) \neq \text{child}(\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v), c)$ (see also Remark 7.3). We thus need to first prove that applying the inverse mapping to *any* node between u and $\widehat{u}$ (in particular, to u') yields $\text{repr}(\text{child}(v, c))$ (Lemma 7.9). This exploits properties of S specific to the child operation; we omit the details here but remark that this step differs among core operations (see, e.g., Lemmas 7.8 and 7.11).

The Periodic Nodes Let us now assume that $\text{str}(v)$ is periodic. The basic idea is similar as above: we keep a compact trie (denoted $\mathcal{T}_{\mathsf{Z}}$) letting us search the suffixes in the set $\{T[e^{\text{full}}(j)\,..\,n]\}_{j\in\mathsf{R}'-}$. Although the implementation of mapping and combinatorial proofs are more technical, establishing these higher-level navigation primitives results in simpler and more concise implementation of queries (see, e.g., Lemma 7.20).

Summary of New Techniques Our key technical contributions are as follows:

- We show how to reduce all operations of a suffix tree to prefix rank and selection queries, resulting in the first $o(n)$-time construction of optimal-size compressed suffix tree, with all operations simultaneously matching the state-of-the-art [14, 16, 31, 34, 75, 77, 80].
- To achieve this, we first define and efficiently implement mappings between the nodes of $\mathcal{T}_{\text{st}}$ and of two auxiliary tries $\mathcal{T}_{\mathsf{S}}$ and $\mathcal{T}_{\mathsf{Z}}$. We then prove new combinatorial results (see, e.g., Lemmas 7.8 to 7.11, 7.16, and 7.18 to 7.21) showing that these high-level navigation primitives correctly handle all suffix tree operations.

4 Auxiliary Tools

4.1 Weighted Ancestors

Consider a rooted tree $\mathcal{T}$. Let $\mathrm{root}(\mathcal{T})$ denote the node at depth 0 and let $\mathrm{parent}(v)$ denote the immediate ancestor of each node $v \neq \mathrm{root}(\mathcal{T})$. We let $\mathrm{parent}(\mathrm{root}(\mathcal{T})) = \bot$. Assume that each node v has an associated weight $w(v)$ such that $w(\mathrm{root}(\mathcal{T})) = 0$ and, for every $v \neq \mathrm{root}(\mathcal{T})$, it holds $w(\mathrm{parent}(v)) < w(v)$. We then say that the weight function w is *monotone*. Given any node v of $\mathcal{T}$ and an integer $0 \leq d \leq w(v)$, we define $v' = \mathrm{WA}(v, d)$ (the *weighted ancestor* [25]) as the (unique) ancestor of v in $\mathcal{T}$ that satisfies $w(v') \geq d$ and for which $w(v')$ is minimized.

THEOREM 4.1. ([2, SECTION 6.2.1]) *Let $\mathcal{T}$ be a rooted tree with $n \leq N$ nodes and a monotone weight function w mapping nodes to $[0\,..\,N)$. There exists a data structure of size $\mathcal{O}(n)$ that answers weighted ancestor queries in $\mathcal{T}$ in $\mathcal{O}(\log\log N)$ time after $\mathcal{O}(n \log_n N)$-time preprocessing.*

Proof. A solution for $N = n$ was presented in [2, Section 6.2.1]. To generalize it to the case of $N \geq n$, it suffices to map all node weights to their ranks. For this, we sort all the node weights in $\mathcal{O}(n \log_n N)$ time (radix sort) and build a deterministic predecessor structure [30, Proposition 2] in $\mathcal{O}(n)$ time so that the rank of a query threshold can be computed in $\mathcal{O}(\log\log N)$ time. □

4.2 Tries and Compact Tries

A set of strings $\mathcal{S} \subseteq \Sigma^+$ is *prefix-free* if there are no $S, S' \in \mathcal{S}$ such that S is a proper prefix of S'. For any prefix-free set of string $\mathcal{S} \subseteq \Sigma^+$, its *trie* is a minimal rooted tree $\mathcal{T}$, with each edge labelled by some $c \in \Sigma$, such that: (1) no two edges outgoing from the same node have the same label, (2) for each $S \in \mathcal{S}$ there exists a path from $\mathrm{root}(\mathcal{T})$ to some node such that the concatenation of edge-labels on that path is equal to S, and (3) children of every node are ordered according to the lexicographical rank of the connecting edge. A *compact trie* of $\mathcal{S}$ is a trie of $\mathcal{S}$ in which all maximal unary paths have been replaced with edges labelled by substrings of elements of $\mathcal{S}$. The nodes of the trie omitted in the compact trie are referred to as *implicit*. All other nodes are *explicit*. Unless explicitly stated otherwise, by *node* we always mean an explicit node.

For any node v of a (compact) trie $\mathcal{T}$, by $\mathrm{str}(v)$ we denote the *label* of v, i.e., the string obtained by concatenating the labels of all edges on the path from $\mathrm{root}(\mathcal{T})$ to v. We denote $\mathrm{sdepth}(v) = |\mathrm{str}(v)|$. The parent of v in denoted $\mathrm{parent}(v)$. For any $c \in \Sigma$, we define $\mathrm{child}(v, c)$ as a child v' of v such that $\mathrm{str}(v')[|\mathrm{str}(v)| + 1] = c$, or $\bot$ if no such node exists. For any $c \in \Sigma$, we also define $\mathrm{pred}(v, c)$ as follows:

- If there exists $c' < c$ such that $\mathrm{child}(v, c') \neq \bot$, then we let $\mathrm{pred}(v, c) = \mathrm{child}(v, c_{\max})$, where $c_{\max} = \max\{c' \in [0\,..\,c) : \mathrm{child}(v, c') \neq \bot\}$.
- Otherwise, we let $\mathrm{pred}(v, c) = \bot$.

We define $(\mathrm{lrank}(v), \mathrm{rrank}(v))$ as a pair of integers satisfying $\mathrm{lrank}(v) = |\{S \in \mathcal{S} : S \prec \mathrm{str}(v)\}|$ and $\mathrm{rrank}(v) - \mathrm{lrank}(v) = |\{S \in \mathcal{S} : \mathrm{str}(v) \text{ is a prefix of } S\}|$. Observe that then collecting every ith leftmost leaf of $\mathcal{T}$, where $i \in (\mathrm{lrank}(v)\,..\,\mathrm{rrank}(v)]$ results in precisely the set of leaves in the subtree rooted in v. Given any node v of $\mathcal{T}$ and an integer $0 \leq d \leq |\mathrm{str}(v)|$, we let $v' = \mathrm{WA}(v, d)$ to be the weighted ancestor of v assuming the weight of each node is defined as $w(v) = \mathrm{sdepth}(v)$. Thus, v' is the (unique) ancestor of v in $\mathcal{T}$ that satisfies $\mathrm{sdepth}(v') \geq d$ and for which $\mathrm{sdepth}(v')$ is minimized. For any two nodes v_1 and v_2, the node $v = \mathrm{LCA}(v_1, v_2)$ (the *lowest common ancestor*) is defined as the (unique) ancestor of both v_1 and v_2 with the maximal depth.

OBSERVATION 4.1. *If v_1 and v_2 are nodes of a (compact) trie, then letting $v = \mathrm{LCA}(v_1, v_2)$ and $\ell = \mathrm{lcp}(\mathrm{str}(v_1), \mathrm{str}(v_2))$, it holds $\mathrm{sdepth}(v) = \ell$ and $\mathrm{str}(v) = \mathrm{str}(v_1)[1\,..\,\ell] = \mathrm{str}(v_2)[1\,..\,\ell]$.*

4.2.1 Small Alphabet

PROPOSITION 4.1. *Given a packed representation of $T \in [0\,..\,\sigma)^n$ with $2 \leq \sigma \leq n$ and an array $A[1\,..\,q]$ of q positions in T such that, for any $1 \leq i < j \leq q$, it holds $T[A[i]\,..\,n] \prec T[A[j]\,..\,n]$, we can in $\mathcal{O}(q + n/\log_\sigma n)$ time construct a representation of the compact trie $\mathcal{T}$ of the set $\{T[A[i]\,..\,n] : i \in [1\,..\,q]\}$, augmented with auxiliary data structures to support the following operations on $\mathcal{T}$ in $\mathcal{O}(1)$ time:*

- *Given $i \in [1\,..\,q]$ return the ith leftmost leaf of $\mathcal{T}$,*

- *Given pointers to nodes* v_1 *and* v_2 *return a pointer to* $\mathrm{LCA}(v_1, v_2)$*,*
- *Given a pointer to node* v*, return* $(\mathrm{lrank}(v), \mathrm{rrank}(v))$ *and* $\mathrm{sdepth}(v)$*.*

It also supports the following operations in $\mathcal{O}(\log \log n)$ *time:*

- *Given a pointer to node* v *and* d *such that* $0 \leq d \leq |\mathrm{str}(v)|$*, return the pointer to* $\mathrm{WA}(v, d)$*,*
- *Given a pointer to node* v *and* $c \in \Sigma$*, check if* $\mathrm{pred}(v, c) \neq \bot$ *(resp.* $\mathrm{child}(v, c) \neq \bot$*), and if so, return the pointer to* $\mathrm{pred}(v, c)$ *(resp.* $\mathrm{child}(v, c)$*).*

Proof. The data structure consists of five components:

1. The packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The compact trie $\mathcal{T}$. Since we assumed that $T[n]$ is unique in T (see Section 2), the set $\{T[A[i]\,..\,n]\}_{i\in[1\,..\,q]}$ is prefix-free, and hence $\mathcal{T}$ has exactly q leaves. Each node v of $\mathcal{T}$ stores the precomputed values $\mathrm{sdepth}(v)$, $(\mathrm{lrank}(v), \mathrm{rrank}(v))$, and a predecessor data structure that, given any $c \in [0\,..\,\sigma)$, returns a pointer to $\mathrm{pred}(v, c)$. Using the structure from [30, Proposition 2], we achieve linear space and $\mathcal{O}(\log \log n)$ query time.
3. The array of pointers $L[1\,..\,q]$ such that $L[i]$ is the pointer to the ith leftmost leaf of $\mathcal{T}$.
4. The data structure of Bender and Farach-Colton [10] that augments $\mathcal{T}$ with support for LCA queries. The structure needs $\mathcal{O}(q)$ space and answers queries in $\mathcal{O}(1)$ time.
5. The data structure from Theorem 4.1 for $\mathcal{T}$ with the weight function $\mathrm{sdepth}(v) \in [0\,..\,n]$. The structure needs $\mathcal{O}(q)$ space and answers queries in $\mathcal{O}(\log \log n)$ time.

In total, the data structure takes $\mathcal{O}(q + n/\log_\sigma n)$ space.

Using the above structures, the implementation of all queries in the claim follows immediately (note that $\mathrm{child}(v, c)$ can be determined using $\mathrm{pred}(v, c+1)$ and the packed representation of T).

Construction algorithm The data structure is constructed as follows. We start by building a data structure that supports LCE queries for suffixes of T. Using [50, Theorem 5.4], the construction takes $\mathcal{O}(n/\log_\sigma n)$ time, and the resulting data structure answers queries in $\mathcal{O}(1)$ time. We then construct $\mathcal{T}$ by inserting elements of $\{T[A[i]\,..\,n]\}_{i\in[1\,..\,q]}$ in the order given by A. We maintain a stack containing the internal nodes on the rightmost path, with the deepest node on top. When inserting each string, we first determine the depth at which that string branches from the rightmost path using LCE queries on T. We then update the rightmost path of the trie. Adding each string first removes some elements from the stack, and then adds at most two new elements. Since the total number of elements pushed on stack is $\mathcal{O}(q)$, the construction of $\mathcal{T}$ takes $\mathcal{O}(q)$ time. During the construction, we record the value $\mathrm{sdepth}(v)$ in each node and collect pointers to consecutive leaves in the $L[1\,..\,q]$ array. With the single traversal of $\mathcal{T}$, we then precompute in $\mathcal{O}(q)$ time the values $\mathrm{lrank}(v)$ and $\mathrm{rrank}(v)$ for every node v (note that at this point, pointers to all children of each node are stored simply using a list, since this traversal does not require fast lookups or predecessor queries). Next, we construct the predecessor data structure for every node. Since the keys in every node are sorted, using [30, Proposition 2], over all nodes of $\mathcal{T}$, the construction takes $\mathcal{O}(q)$ time. Finally, we construct the data structures supporting LCA and WA queries on $\mathcal{T}$. Using [10] and Theorem 4.1, this takes $\mathcal{O}(q)$ and $\mathcal{O}(q \log_q n) = \mathcal{O}((q+\sqrt{n}) \log_{q+\sqrt{n}} n) = \mathcal{O}(q + \sqrt{n}) = \mathcal{O}(q + n/\log_\sigma n)$ time, respectively. □

4.2.2 Large Alphabet

Proposition 4.2. *Given a packed representation of* $T \in [0\,..\,\sigma)^n$ *with* $2 \leq \sigma < n^{1/7}$ *and an array* $A[1\,..\,q]$ *of* q *positions in* T *such that, for any* $1 \leq i < j \leq q$*, it holds* $T[A[i]\,..\,n] \prec T[A[j]\,..\,n]$*, we can in* $\mathcal{O}(q + n/\log_\sigma n)$ *time construct a data structure that, given the packed representation of any* $P \in [0\,..\,\sigma)^m$*, returns in* $\mathcal{O}(m/\log_\sigma n + \log \log n)$ *time a pair of integers* $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ *satisfying:*

- $b_{\mathrm{pre}} = |\{i \in [1\,..\,q] : T[A[i]\,..\,n] \prec P\}|$*, and*
- $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,q] : P$ *is a prefix of* $T[A[i]\,..\,n]\}$*.*

Proof. The basic idea is to construct the compact trie of strings in $\{T[A[i]\,..\,n]\}_{i\in[1\,..\,q]}$ converted into strings over the alphabet of metasymbols (of $\Theta(\log_\sigma n)$ original symbols each). Our mapping of symbols to metasymbols does not, however, simply group symbols into blocks. We instead introduce a special mapping that will allow us to deduce the output range $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ using two predecessor queries in the image of the set $\{T[A[i]\,..\,n]\}_{i\in[1\,..\,q]}$ and

some carefully crafted patterns. This will allow us to use the augmentation of tries of Fischer and Gawrychowski [30, Theorem 1] in a black-box manner.

Definitions First, we introduce the mapping of strings over $[0\,..\,\sigma)$ into strings over metasymbols. Let $\tau = \lfloor \frac{1}{7}\log_\sigma n \rfloor$ and $\kappa = 3\tau - 1$. For any $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, let $\mathrm{int}(X)$ denote an integer constructed by appending $6\tau - 2|X|$ zeros and $|X|$ cs (where $c = \sigma - 1$) to X, and then interpreting the resulting string as a base-σ representation of a number in $[0\,..\,\sigma^{6\tau})$. Note that $X \neq X'$ implies $\mathrm{int}(X) \neq \mathrm{int}(X')$. Let also $\mathrm{int}'(X)$ denote an integer constructed by appending $6\tau - 2|X|$ cs (where $c = \sigma - 1$) and $|X|$ zeros to X, and then interpreting the resulting string as a base-σ representation of a number in $[0\,..\,\sigma^{6\tau})$. For any string $S \in [0\,..\,\sigma)^*$ of length $\ell \geq 0$, we define $\mathrm{mstr}(S)$ as a string of length $\ell' = \lceil \frac{\ell+1}{\kappa} \rceil > 0$ over alphabet $[0\,..\,\sigma^{6\tau})$ such that, for any $i \in [1\,..\,\ell']$, it holds $\mathrm{mstr}(S)[i] = \mathrm{int}(S((i-1)\cdot\kappa\,..\,\min(\ell, i\cdot\kappa))])$. For $S \in [0\,..\,\sigma)^*$ of length $\ell \geq 0$, we define $\mathrm{mstr}'(S)$ as a string of length $\ell' = \lceil \frac{\ell+1}{\kappa} \rceil > 0$ over alphabet $[0\,..\,\sigma^{6\tau})$ such that $\mathrm{mstr}'(S)[1\,..\,\ell') = \mathrm{mstr}(S)[1\,..\,\ell')$ and $\mathrm{mstr}'(S)[\ell'] = \mathrm{int}'(S((\ell'-1)\cdot\kappa\,..\,\min(\ell, \ell'\cdot\kappa))])$. Note that if ℓ is a multiple of κ, then the last symbol of $\mathrm{mstr}(S)$ is $\mathrm{int}(\varepsilon) = 0$, whereas the last symbol of $\mathrm{mstr}'(S)$ is $\mathrm{int}'(\varepsilon) = \sigma^{6\tau} - 1$. Observe that

- For any set of strings $\mathcal{S} \subseteq [0\,..\,\sigma)^*$, the set $\{\mathrm{mstr}(X) : X \in \mathcal{S}\}$ is prefix-free.
- For any strings $X, Y \in [0\,..\,\sigma)^*$, $X \prec Y$ holds if and only if $\mathrm{mstr}(X) \prec \mathrm{mstr}(Y)$.
- A string $P \in [0\,..\,\sigma)^*$ is a prefix of $X \in [0\,..\,\sigma)^*$ if and only if $\mathrm{mstr}(P) \preceq \mathrm{mstr}(X) \prec \mathrm{mstr}'(P)$.

For any set of strings $\mathcal{S}$ and any string Y, denote $\mathrm{rank}_{\mathcal{S}}(Y) := |\{X \in \mathcal{S} : X \prec Y\}|$. Observe that by the above properties, letting $P_1 = \mathrm{mstr}(P)$, $P_2 = \mathrm{mstr}'(P)$, and $\mathcal{A} = \{\mathrm{mstr}(T[A[i]\,..\,n])\}_{i\in[1\,..\,q]}$, we have $(b_{\mathrm{pre}}, e_{\mathrm{pre}}) = (\mathrm{rank}_{\mathcal{A}}(P_1), \mathrm{rank}_{\mathcal{A}}(P_2))$. Let $\mathcal{T}$ denote the compact trie of the set $\mathcal{A}$.

Components The data structure consists of two components:

1. The packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The trie $\mathcal{T}$ augmented using [30, Theorem 1]. Note that this result requires that the alphabet of strings in $\mathcal{A}$ is of size $|\mathcal{A}|^{\mathcal{O}(1)}$, which may be violated for $q = n^{o(1)}$. Thus, we actually define the alphabet to be $[0\,..\,\sigma')$, where $\sigma' = \sigma^{6\tau} + \lceil\sqrt{n}\rceil$, and insert to $\mathcal{A}$ additional $\lceil\sqrt{n}\rceil$ dummy length-1 strings corresponding to the $\lceil\sqrt{n}\rceil$ largest characters in $[0\,..\,\sigma')$. As a result, we must have $\sigma' = \mathcal{O}(n) = \mathcal{O}(|\mathcal{A}|^2)$. At the same time, the dummy strings do not change $\mathrm{rank}_{\mathcal{A}}(Q)$ for any $Q \in [0\,..\,\sigma^{6\tau})^*$. By [30, Theorem 1], such augmented $\mathcal{T}$ needs $\mathcal{O}(q + \sqrt{n})$ space.

In total, the data structure takes $\mathcal{O}(q + n/\log_\sigma n)$ space.

Implementation of queries Using T and $\mathcal{T}$, given the packed representation of $P \in [0\,..\,\sigma)^m$, we compute the output pair $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ as follows. First, note that, given the packed representation of P and T, we can in $\mathcal{O}(1)$ time access any symbol of $\mathrm{mstr}(P)$, $\mathrm{mstr}'(P)$, and $\mathrm{mstr}(T[i\,..\,n])$ for any $i \in [1\,..\,n]$. We start by computing P_1 and P_2 from P in $\mathcal{O}(m/\log_\sigma n + 1)$ time. Then, using [30, Theorem 1], we compute $\mathrm{rank}_{\mathcal{A}}(P_1)$ and $\mathrm{rank}_{\mathcal{A}}(P_2)$ in $\mathcal{O}(|P_1| + \log\log\sigma') = \mathcal{O}(m/\log_\sigma n + \log\log n)$ and $\mathcal{O}(|P_2| + \log\log\sigma') = \mathcal{O}(m/\log_\sigma n + \log\log n)$ time, respectively. By the above discussion, this gives us the output pair $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$. During the query, the algorithm may want to access symbols of strings from $\mathcal{A}$. We do not store them explicitly (note that storing $\mathrm{mstr}(T)$ would not be enough), but instead perform the mapping on-the-fly.

Construction algorithm We start by building a data structure that supports LCE queries for suffixes of T. Using [50, Theorem 5.4], the construction takes $\mathcal{O}(n/\log_\sigma n)$ time, and the resulting data structure answers queries in $\mathcal{O}(1)$ time. Denote the length of the longest common prefix between suffixes $T[i\,..\,n]$ and $T[j\,..\,n]$ as $\mathrm{LCE}(i, j)$. Observe that for any $i, j \in [1\,..\,n]$ such that $i \neq j$, the longest common prefix of $\mathrm{mstr}(T[i\,..\,n])$ and $\mathrm{mstr}(T[j\,..\,n])$ has length $\lfloor \frac{\mathrm{LCE}(i,j)}{\kappa} \rfloor$, which can be computed in $\mathcal{O}(1)$ time. We construct $\mathcal{T}$ by inserting elements of $\{\mathrm{mstr}(T[A[i]\,..\,n])\}_{i\in[1\,..\,q]}$ in the order given by A. The construction proceeds as in the proof of Proposition 4.1 and takes $\mathcal{O}(q)$ time. Once the trie is constructed, we add the $\lceil\sqrt{n}\rceil$ dummy length-1 strings and augment $\mathcal{T}$ using [30, Theorem 1] in $\mathcal{O}(q + \sqrt{n})$ time. In total, the construction takes $\mathcal{O}(q + n/\log_\sigma n)$ time.

4.3 (Prefix) Rank and Selection Queries

We start with an implementation of rank and selection queries for larger alphabets.

LEMMA 4.1. (BELAZZOUGUI AND PUGLISI [9]) *For all integers $N \geq n \geq \sigma \geq 2$ and every string $S \in [0\,..\,\sigma)^{\leq n}$,*

there exists a data structure of $\mathcal{O}(n \log \sigma)$ bits that answers rank queries in $\mathcal{O}(\log \log N)$ time and selection queries in $\mathcal{O}(1)$ time. Moreover, given a table precomputed in $\mathcal{O}(N)$ time (shareable across all instances with common parameter N) and the packed representation of S, the data structure can be constructed in $\mathcal{O}(\min(n, \sigma + n \log \sigma / \sqrt{\log N}))$ time using $\mathcal{O}(n \log \sigma)$ bits of space.

Proof. If $\log^2 \sigma \geq \log N$, we use the data structure of [9, Lemma C.2], which occupies $\mathcal{O}(n \log \sigma)$ bits, answers rank queries in $\mathcal{O}(\log \log n)$ time and selection queries in $\mathcal{O}(1)$ time, and can be constructed in $\mathcal{O}(n)$ time using $\mathcal{O}(n \log \sigma)$ bits of space.[7] Otherwise, we use the data structure of [9, Lemma C.3], which occupies $\mathcal{O}(n \log \sigma)$ bits, answers rank queries in $\mathcal{O}(\log \log N)$ time and selection queries in $\mathcal{O}(1)$ time, and can be constructed in $\mathcal{O}(\sigma + n \log^2 \sigma / \log N)$ time using $\mathcal{O}(n \log \sigma)$ bits of space. □

The following proposition, instantiated with $h = \lceil \ell^{\epsilon/2} \rceil$, immediately yields Theorem 2.2.

PROPOSITION 4.3. *For all integers $h, m, \ell, \sigma \in \mathbb{Z}_{\geq 1}$ satisfying $h \geq 2$ and $m \geq \sigma^\ell \geq 2$, and every string $W \in ([0\,..\,\sigma)^\ell)^{\leq m}$, there exists a data structure of size $\mathcal{O}(m \log_h(h\ell))$ that answers prefix rank queries in $\mathcal{O}(h \log \log m \log_h(h\ell))$ time and prefix selection queries in $\mathcal{O}(h \log_h(h\ell))$ time. Moreover, it can be constructed in $\mathcal{O}(m \min(\ell, \sqrt{\log m}) \log_h(h\ell))$ time using $\mathcal{O}(m \log_h(h\ell))$ space given the packed representation of W and the parameter h.*

Proof. The data structure consists in the wavelet tree of W and, when $h \leq \ell$, an instance constructed recursively for an auxiliary string $\tilde{W}$ defined below.

Wavelet tree Let $\Sigma = [0\,..\,\sigma)$ so that the alphabet of W is Σ^ℓ. The wavelet tree of W [41] is the trie of Σ^ℓ with each internal node v_X (representing a string $X \in \Sigma^{\leq \ell - 1}$) associated to a string $B_X[1\,..\,\mathsf{rank}_{W,X}(|W|)] \in \Sigma^*$ such that $B_X[r] = W[\mathsf{select}_{W,X}(r)][|X| + 1]$ for $r \in [1\,..\,\mathsf{rank}_{W,X}(|W|)]$. The strings B_X are augmented with the component of Lemma 4.1 (for rank and selection queries) with parameter $N := m$.

Recursive instance We shall define $\tilde{W}$ as a string of length $|W|$ over the alphabet $\tilde{\Sigma}^{\tilde{\ell}}$, where $\tilde{\ell} := \lfloor \ell / h \rfloor$, $\tilde{\sigma} = \sigma^h$, and $\tilde{\Sigma} := [0\,..\,\tilde{\sigma})$. We identify $\tilde{\Sigma}$ with Σ^h, treating each string in Σ^h as the h-digit base-σ representation of an integer in $\tilde{\Sigma}$. For every string $X \in \Sigma^*$, define $\widetilde{X} \in \tilde{\Sigma}^*$ so that $|\widetilde{X}| = \lfloor |X| / h \rfloor$ and $\widetilde{X}[i] = X(h(i-1)\,..\,hi]$ for $i \in [1\,..\,|\widetilde{X}|]$. Moreover, we set $\tilde{W}[1\,..\,|W|]$ so that $\tilde{W}[j] = \widetilde{W[j]}$ for $j \in [1\,..\,|W|]$. Note that the recursive application of Proposition 4.3 to $\tilde{W}$ is possible because $2 \leq \tilde{\sigma}^{\tilde{\ell}} \leq \sigma^\ell \leq m$ and $\tilde{\ell} \geq 1$ hold when $h \leq \ell$.

Data structure size It is easy to see that, for a fixed length $d \in [0\,..\,\ell)$, the strings B_X for $X \in \Sigma^d$ are of total length m. Across all $X \in \Sigma^{\leq \ell - 1}$, this sums up to $m\ell$, so the raw strings B_X occupy $\mathcal{O}(m\ell \log \sigma) = \mathcal{O}(m \log m)$ bits. The augmentation of B_X using Lemma 4.1 adds $\mathcal{O}((\sigma + |B_X|) \log \sigma)$ extra bits (we set $n := \max(\sigma, |B_X|) = \Theta(\sigma + |B_X|)$ to ensure $\sigma \leq n$), which sums up to $\mathcal{O}((\sigma^\ell + m\ell) \log \sigma) = \mathcal{O}(m \log m)$ bits, i.e., $\mathcal{O}(m)$ machine words. The recursion depth is $\mathcal{O}(\log_h(h\ell))$, so the overall size is $\mathcal{O}(m \log_h(h\ell))$.

Answering queries To handle any query concerning $X \in \Sigma^{\leq \ell}$, we compute auxiliary strings $\widetilde{X}$ (as defined above) and $X' := X[1\,..\,|X| - (|X| \bmod h)]$ (obtained by expanding the letters in $\widetilde{X}$ into length-h strings).

Answering a prefix rank query $\mathsf{rank}_{W,X}(j)$, we traverse the path from $v_{X'}$ to v_X, maintaining a value r such that $r = \mathsf{rank}_{W,Y}(j)$ holds while the algorithm visits v_Y. We initialize $r := j = \mathsf{rank}_{W,\varepsilon}(j)$ if $X' = \varepsilon$ and $r := \mathsf{rank}_{\tilde{W},\widetilde{X}}(j)$ (computed recursively) otherwise; this is valid due to $\mathsf{rank}_{\tilde{W},\widetilde{X}}(j) = \mathsf{rank}_{W,X'}(j)$. Upon entering a node v_{Ya} from its parent v_Y, we set $r := \mathsf{rank}_{B_Y,a}(r)$ since $\mathsf{rank}_{W,Ya}(j) = \mathsf{rank}_{B_Y,a}(\mathsf{rank}_{W,Y}(j))$; see [41]. When reaching v_X, we return $r = \mathsf{rank}_{W,X}(j)$. The running time is $\mathcal{O}(h \log \log m)$ per recursive level, for a total of $\mathcal{O}(h \log \log m \cdot \log_h(h\ell))$.

Answering a prefix selection query $\mathsf{select}_{W,X}(r)$, we traverse the path from v_X to $v_{X'}$, maintaining a value q such that $\mathsf{select}_{W,Y}(q) = \mathsf{select}_{W,X}(r)$ holds while the algorithm visits v_Y. We initialize $q := r$ and, upon entering a node v_Y from its child v_{Ya}, we set $q := \mathsf{select}_{B_Y,a}(q)$ since $\mathsf{select}_{W,Ya}(r) = \mathsf{select}_{W,Y}(\mathsf{select}_{B_Y,a}(r))$;

[7] The statement of [9, Lemma C.2] does not bound the space consumption of the construction algorithm. Nevertheless, it is straightforward to implement the underlying construction procedure in $\mathcal{O}(n \log \sigma)$ bits of working space. The original algorithm scans the input sequence S from left to right and, for each $a \in \Sigma$, builds an array $P_a[1\,..\,n_a]$ such that $n_a = \mathsf{rank}_{S,a}(|S|)$ and $P_a[r] = \mathsf{select}_{S,a}(r)$ for $r \in [1\,..\,n_a]$. The array $P_a[1\,..\,n_a]$ is then converted to the Elias–Fano representation: an array $A_a[1\,..\,n_a]$ with $A_a[r] = P_a[r] \bmod \sigma$ for $r \in [1\,..\,n_a]$ and a bit vector $V_a = \mathrm{unary}((\lfloor P_a[r]/\sigma \rfloor - \lfloor P_a[r-1]/\sigma \rfloor)_{r \in [1\,..\,n_a]})$, where we assume $P_a[0] = 0$ to streamline the formula. To achieve $\mathcal{O}(n \log \sigma)$ bits of working space, instead of storing P_a explicitly, we convert P_a to the Elias–Fano representation on the fly as subsequent positions are appended to P_a.

see [41]. When reaching $v_{X'}$, we return $q = \mathsf{select}_{W,\varepsilon}(q)$ if $X' = \varepsilon$ and $\mathsf{select}_{\tilde{W},\tilde{X}}(q)$ otherwise; this is valid due to $\mathsf{select}_{\tilde{W},\tilde{X}}(q) = \mathsf{select}_{W,X'}(q)$. The running time is $\mathcal{O}(h)$ per recursive level, for a total of $\mathcal{O}(h \cdot \log_h(h\ell))$.

Construction algorithm If $\ell \le \sqrt{\log m}$, we use the original wavelet tree construction algorithm [41], which takes $\mathcal{O}(m\ell)$ time and $\mathcal{O}(m)$ space. Building the data structure of Lemma 4.1 for B_X takes $\mathcal{O}(\sigma + |B_X|)$ time and $\mathcal{O}((\sigma + |B_X|)\log\sigma/\log m)$ space, which sums up to $\mathcal{O}(\sigma^\ell + m\ell) = \mathcal{O}(m\ell)$ time and $\mathcal{O}(m\ell\log\sigma/\log m) = \mathcal{O}(m)$ space across $X \in \Sigma^{\le \ell-1}$ (due to $\ell\log\sigma \le \log m$). Precomputing the table shared by all instances of Lemma 4.1 takes $\mathcal{O}(m)$ time and space. Considering all levels of recursion, we get $\mathcal{O}(m\ell)$ time (due to $\tilde{\ell} \le \frac{1}{2}\ell$) and $\mathcal{O}(m\log_h(h\ell))$ space.

If $\ell > \sqrt{\log m}$, on the other hand, we apply the bit-parallel wavelet tree construction algorithm of [64, 3], which has been adapted to large alphabets in [50, Lemma 6.4]. Due to $\ell\log\sigma \le \log m$, this procedure takes $\mathcal{O}(m\ell\log\sigma/\sqrt{\log m} + m\ell\log^2\sigma/\log m) = \mathcal{O}(m\sqrt{\log m})$ time and $\mathcal{O}(m)$ space. Building the data structure of Lemma 4.1 for B_X takes $\mathcal{O}(\sigma + (\sigma + |B_X|)\log\sigma/\sqrt{\log m}) = \mathcal{O}(\sigma + |B_X|\log\sigma/\sqrt{\log m})$ time and $\mathcal{O}((\sigma + |B_X|)\log\sigma/\log m)$ space, which sums up to $\mathcal{O}(m\sqrt{\log m})$ time and $\mathcal{O}(m)$ space across $X \in \Sigma^{\le \ell-1}$. Precomputing the table shared by all instances of Lemma 4.1 takes $\mathcal{O}(m)$ time and space. Considering all levels of recursion, we get a multiplicative overhead of $\mathcal{O}(\log_h(h\ell))$, for a total of $\mathcal{O}(m\sqrt{\log m}\log_h(h\ell))$ time and $\mathcal{O}(m\log_h(h\ell))$ space. □

4.4 Range Counting and Selection

PROPOSITION 2.1. *An array $A[1..m']$ of $m' \in [2..m]$ nonnegative integers satisfying $\sum_{i=1}^{m'} A[i] = \mathcal{O}(m\log m)$ can be preprocessed in $\mathcal{O}(m)$ time so that range counting and selection queries can be answered in $\mathcal{O}(\log\log m)$ time and $\mathcal{O}(1)$ time, respectively.*

Proof. We use the following definitions. Denote $h = \lfloor \log m \rfloor$. For any $k \ge 0$, by $P_k[1..m_k]$, where $m_k = \mathsf{rcount}_A(kh, m)$, we denote the array defined by $P_k[i] = \mathsf{rselect}_A(kh, i)$. Let $v \ge 0$. We define a bitvector $M_v[1..m_k]$, where $k = \lfloor \frac{v}{h} \rfloor$ as follows. For any $i \in [1..m_k]$, $M_v[i] = 1$ holds if and only if $A[P_k[i]] \ge v$. For any $k \ge 0$, we define the concatenation $M'_k = M_{kh}M_{kh+1}\cdots M_{(k+1)h-1}$. Let $k_{\max} = \max\{k \ge 0 : m_k > 0\}$. Since all elements of A are nonnegative, and $\sum_{i=1}^{m'} A[i] \in \mathcal{O}(m\log m)$, we obtain $\max_{i\in[1..m']} A[i] \in \mathcal{O}(m\log m)$, and consequently, $k_{\max} = \lfloor \frac{1}{h}\max_{i\in[1..m']} A[i] \rfloor \in \mathcal{O}(m)$.

Components The data structure consists of two components:

1. First, for $k \in [0..k_{\max}]$, we store a plain representation of the sequence $P_k[1..m_k]$ using $\mathcal{O}(m_k)$ space. Each array is augmented with a static predecessor data structure. We use [30, Proposition 2], and hence achieve linear space and $\mathcal{O}(\log\log m)$ query time. Each $i \in [1..m']$ occurs in $\lceil \frac{A[i]+1}{h} \rceil$ arrays. Thus, $\sum_{k\ge0} m_k = \sum_{i=1}^{m'} \lceil \frac{A[i]+1}{h} \rceil \le 2m' + \sum_{i=1}^{m'} \lfloor \frac{A[i]}{h} \rfloor \le 2m' + \frac{1}{h}\sum_{i=1}^{m'} A[i] \in \mathcal{O}(m)$ and hence we can store the arrays P_k (including the associated predecessor data structures) using $\mathcal{O}((k_{\max}+1) + \sum_{k\ge0} m_k) \subseteq \mathcal{O}(m)$ space, so that we can access each array in $\mathcal{O}(1)$ time.
2. Second, for every $k \in [0..k_{\max}]$, we store the plain representation of bitvector M'_k, augmented using Theorem 2.1. By $|M'_k| = h \cdot m_k$, the total length of bitvectors M'_k is $\sum_{k\ge0} |M'_k| = h\sum_{k\ge0} m_k \in \mathcal{O}(m\log m)$. All bitvectors M'_k can thus be stored in $\mathcal{O}((k_{\max}+1) + \frac{1}{\log m}\sum_{k\ge0} |M'_k|) \subseteq \mathcal{O}(m)$ words of space, so that we can access each in $\mathcal{O}(1)$ time. For a bitvector of length t, the augmentation of Theorem 2.1 adds only $\mathcal{O}(\log m + t)$ bits of space, and hence does not increase the space usage.

In total, the data structure takes $\mathcal{O}(m)$ space.

Implementation of queries Using the above two components, we answer range counting/selection queries on A as follows. To compute $\mathsf{rcount}_A(v, j)$, we first let $k = \lfloor \frac{v}{h} \rfloor$. If $k > k_{\max}$, then we return $\mathsf{rcount}_A(v, j) = 0$. Otherwise, we observe that if $j' = |\{i \in [1..m_k] : P_k[i] \le j\}|$, then $\mathsf{rcount}_A(v, j) = \mathsf{rank}_{M_v,1}(j')$. Computing j' using the predecessor data structure takes $\mathcal{O}(\log\log m)$ time, and then $\mathsf{rank}_{M_v,1}(j')$ is computed using the rank support data structure of the bitvector M'_k as $\mathsf{rank}_{M'_k,1}(j' + (v - kh)m_k) - \mathsf{rank}_{M'_k,1}((v - kh)m_k)$ in $\mathcal{O}(1)$ time. To compute $\mathsf{rselect}_A(v, r)$, we observe that letting again $k = \lfloor \frac{v}{h} \rfloor$, it holds $\mathsf{rselect}_A(v, r) = P_k[\mathsf{select}_{M_v,1}(r)]$. The value $\mathsf{select}_{M_v,1}(r)$ is computed using the select support data structure of the bitvector M'_k as $\mathsf{select}_{M'_k,1}(\mathsf{rank}_{M'_k,1}((v - kh)m_k) + r) - (v - kh)m_k$ in $\mathcal{O}(1)$ time.

Construction algorithm We start by initializing $P_0[i] = i$ for $i \in [1\,..\,m']$. For $k \in [1\,..\,k_{\max}]$, the array P_k is computed by iterating over P_{k-1} and including only elements $P_{k-1}[i]$ satisfying $A[P_{k-1}[i]] \geq kh$. By $\sum_{k\geq 0} m_k \in \mathcal{O}(m)$, this takes $\mathcal{O}(m)$ time in total. We then augment all arrays P_k with the predecessor data structures. Since the arrays are sorted, using [30, Proposition 2], the construction altogether again takes $\mathcal{O}(m)$ time. We then construct bitvectors M'_k in the order of increasing $k \in [0\,..\,k_{\max}]$. To build M'_k we first scan P_k and check if there exists $i \in [1\,..\,m_k]$ such that $A[P_k[i]] < (k+1)h$.

1. If there is no such i, we set $M'_k := 1^{hm_k}$ in $\mathcal{O}(1 + \frac{1}{\log m} hm_k) = \mathcal{O}(m_k)$ time.
2. Otherwise, we scan again $P_k[1\,..\,m_k]$ and prepare h lists $L_0, L_1, \ldots, L_{h-1}$ such that L_y contains all $i \in [1\,..\,m_k]$ satisfying $A[P_k[i]] = kh+y$. Construction of all lists takes $\mathcal{O}(m_k+h)$ time. The bitvector M'_k is then obtained as the concatenation of bitvectors $M_{kh}, M_{kh+1}, \ldots, M_{(k+1)h-1}$ computed in this order. We first initialize $M_{kh} := 1^{m_k}$ in $\mathcal{O}(1 + \frac{m_k}{\log m})$ time. The bitvector M_{kh+y} for $y > 0$ is obtained by first copying the bitvector M_{kh+y-1} in $\mathcal{O}(1 + \frac{m_k}{\log m})$ time, and then setting $M_{kh+y}[i] = 0$ for every position i stored in L_{y-1}. The total length of all lists L_y is bounded by m_k. Thus, the construction of M'_k takes $\mathcal{O}(h+m_k+\frac{1}{\log m} hm_k) \subseteq \mathcal{O}(h+m_k)$ time.

To bound the total time spent constructing bitvectors M'_k, we consider two cases:

- $k \leq \frac{m}{h}$: The total time spent in the construction of bitvectors M'_k for such k is bounded by the sum $\sum_{k=0}^{\lfloor m/h \rfloor} \mathcal{O}(h + m_k) \subseteq \mathcal{O}(m + \sum_{k\geq 0} m_k) \subseteq \mathcal{O}(m)$.
- $k > \frac{m}{h}$: Let $k' = \lfloor \frac{m}{h} \rfloor + 1$. Note that for any t, it holds $m_{t+1} \leq m_t$. Moreover, whenever Case 2 above happens for some t, it holds $m_{t+1} < m_t$. Thus, Case 2 above can happen for $k > \frac{m}{h}$ only $m_{k'}$ times. Since for every $i \in [1\,..\,m_{k'}]$ we have $A[P_{k'}[i]] \geq m$, by $\sum_{i\in[1\,..\,m']} A[i] \in \mathcal{O}(m \log m)$ it holds $m_{k'} \in \mathcal{O}(\log m)$. The total time spend computing M'_k for $k > \frac{m}{h}$ is thus bounded by $\mathcal{O}(m_{k'}(h + m_{k'}) + \sum_{k\geq k'} m_k) \subseteq \mathcal{O}(\log^2 m + \sum_{k\geq 0} m_k) \subseteq \mathcal{O}(m)$.

The total length of bitvectors M'_k for $k \in [0\,..\,k_{\max}]$, is $\sum_{k\in[0\,..\,k_{\max}]} hm_k \in \mathcal{O}(hm)$. Thus, augmenting them all using Theorem 2.1 takes $\mathcal{O}((k_{\max}+1) + \frac{1}{\log m} hm) \subseteq \mathcal{O}(m)$ time. □

5 SA and ISA Queries

Let $\epsilon \in (0,1)$ be any fixed constant and let $T \in [0\,..\,\sigma)^n$, where $2 \leq \sigma < n^{1/7}$. In this section, we show how, given the packed representation of T, in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space, to construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that answers SA and ISA queries in $\mathcal{O}(\log^\epsilon n)$ time. We also derive a general reduction depending on prefix rank and selection queries.

Let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is any positive constant smaller than $\frac{1}{6}$ such that $\tau \geq 1$ (such μ exists by $\sigma < n^{1/7}$), be fixed for the duration of this section. Throughout, we also use R as a shorthand for $\mathsf{R}(\tau, T)$.

Definition 5.1. *Let $j \in [1\,..\,n]$. We call position j* periodic *if $j \in \mathsf{R}$. Otherwise, j is* nonperiodic.

Organization The structure and the query algorithm to compute $\mathrm{SA}[i]$ (resp. $\mathrm{ISA}[j]$), given any $i \in [1\,..\,n]$ (resp. $j \in [1\,..\,n]$), are different depending on whether $\mathrm{SA}[i]$ (resp. j) is periodic (Definition 5.1). Our description is thus split as follows. First (Section 5.1), we describe the set of data structures called collectively the index "core" that enables efficiently checking if $\mathrm{SA}[i] \in \mathsf{R}$ (resp. $j \in \mathsf{R}$); the core also contain some common components utilized by the remaining parts. In the following two parts (Sections 5.2 and 5.3), we describe structures handling each of the two cases. All ingredients are then put together in Section 5.4. Finally, we present our result in the general form (Section 5.5).

5.1 The Index Core

In this section, we present a data structure that, given any $j \in [1\,..\,n]$ (resp. $i \in [1\,..\,n]$), lets us in $\mathcal{O}(1)$ time determine if $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$).

The section is organized as follows. First, we introduce the components of the data structure (Section 5.1.1). Next, we describe the query algorithms (Section 5.1.2). Finally, we show the construction algorithm (Section 5.1.3).

5.1.1 The Data Structure

Definitions Let L_{range} be a mapping from $X \in [0\,..\,\sigma)^{\leq 3\tau-1} := \{\varepsilon\} \cup [0\,..\,\sigma) \cup \ldots \cup [0\,..\,\sigma)^{3\tau-1}$ to the pair of integers $(b, e) := (\text{RangeBeg}(X, T), \text{RangeEnd}(X, T))$. Let also L_{per} denote the mapping from $[0\,..\,\sigma)^{3\tau-1}$ to $\mathbb{Z}_+$ such that every X is mapped to $\text{per}(X)$.

Let $B_{3\tau-1}[1\,..\,n]$ be a bitvector defined such that $B_{3\tau-1}[i] = 1$ holds if and only if $i = n$, or $i < n$ and $X_{\text{SA}[i]} \neq X_{\text{SA}[i+1]}$, where $X_j = T[j\,..\,\min(n+1, j+3\tau-1))$ for every $j \in [1\,..\,n]$.

Let $A_{\text{short}}[1\,..\,t]$ $(t = \mathsf{rank}_{B_{3\tau-1},1}(n))$ be defined by $A_{\text{short}}[i] = X_{\text{SA}[j]}$, where $j = \mathsf{select}_{B_{3\tau-1},1}(i)$.

Components The index core, denoted $\text{C}_{\text{SA}}(T)$, consists of five components:

1. The packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{range}. When accessing L_{range}, strings $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ are converted to small integers using the mapping $\text{int}(X)$ defined in the proof of Proposition 4.2. By $\text{int}(X) \in [0\,..\,\sigma^{6\tau})$, L_{range} needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n^{6\mu}) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The lookup table L_{per}. Similarly as above, we utilize the mapping $\text{int}(X)$. L_{per} thus also needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.
4. The bitvector $B_{3\tau-1}$ augmented using Theorem 2.1 for rank and selection queries. The augmented bitvector takes $\mathcal{O}(n/\log n)$ space.
5. The array A_{short}. Every string $X \in \{A_{\text{short}}[i]\}_{i\in[1\,..\,t]}$ is encoded as $\text{int}(X)$ using $6\tau\log\sigma = \mathcal{O}(\log n)$ bits. This implicitly encodes the length of the string and ensures that all strings are encoded using the same number of bits. By $\{A_{\text{short}}[i]\}_{i\in[1\,..\,t]} \subseteq [0\,..\,\sigma)^{\leq 3\tau-1}$, we have $t = \mathcal{O}(n^{1/2})$, and hence the array A_{short} needs $\mathcal{O}(n^{1/2}) = \mathcal{O}(n/\log n)$ space.

In total, $\text{C}_{\text{SA}}(T)$ takes $\mathcal{O}(n/\log_\sigma n)$ space.

5.1.2 Navigation Primitives

PROPOSITION 5.1. *Given* $\text{C}_{\text{SA}}(T)$*, for any* $j \in [1\,..\,n]$ *we can in* $\mathcal{O}(1)$ *time determine if* $j \in \mathsf{R}$.

Proof. If $j > n - 3\tau + 2$, we return that $j \notin \mathsf{R}$ (Definition 2.1). Otherwise, we use the packed encoding of T to extract $X = T[j\,..\,j+3\tau-1)$ in $\mathcal{O}(1)$ time and convert it to $x = \text{int}(X)$. We then use the lookup table L_{per}, to determine $p = \text{per}(X)$, and return that $j \in \mathsf{R}$ if $p \leq \frac{1}{3}\tau$. □

PROPOSITION 5.2. *Given* $\text{C}_{\text{SA}}(T)$*, for any* $i \in [1\,..\,n]$ *we can in* $\mathcal{O}(1)$ *time determine if* $\text{SA}[i] \in \mathsf{R}$.

Proof. Given the position $i \in [1\,..\,n]$, we first compute $y = \mathsf{rank}_{B_{3\tau-1},1}(i-1)$. The string $X = A_{\text{short}}[y+1]$ is then a prefix of $T[\text{SA}[i]\,..\,n]$. If $|X| < 3\tau-1$, we must have $\text{SA}[i] > n-3\tau+2$, and thus we return that $\text{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$ (see Definition 2.1). Otherwise (i.e., $|X| = 3\tau-1$), using L_{per} we determine $p = \text{per}(X)$ and return that $\text{SA}[i] \in \mathsf{R}$ if $p \leq \frac{1}{3}\tau$. □

5.1.3 Construction Algorithm

PROPOSITION 5.3. *Given the packed representation of* $T \in [0\,..\,\sigma)^n$*, we can construct* $\text{C}_{\text{SA}}(T)$ *in* $\mathcal{O}(n/\log_\sigma n)$ *time.*

Proof. To compute L_{range}, we first compute for every $X \in [0\,..\,\sigma)^\ell$ (where $\ell = 3\tau-1$), its frequency $f_X := |\text{Occ}(X, T)|$. Using the simple generalization of the algorithm described in [50, Section 6.1.2], this takes $\mathcal{O}(n/\log_\sigma n)$ time (note that the algorithm requires $\ell\sigma^{2\ell-1} = O(n/\log_\sigma n)$, which is satisfied here, since $2\ell - 1 < 6\mu\log_\sigma n$ and $\mu < \frac{1}{6}$). From the frequencies of $X \in [0\,..\,\sigma)^{3\tau-1}$ we then compute the values of f_X for all $X \in [0\,..\,\sigma)^{<3\tau-1}$ by observing that unless X is a nonempty suffix of T, it holds $f_X = \sum_{c\in[0\,..\,\sigma)} f_{Xc}$, i.e., the frequency of each string shorter than $3\tau-1$ is obtained in $\mathcal{O}(\sigma)$ time. If X is a nonempty suffix of T (which we can check in $\mathcal{O}(1)$ time), we additionally add one to the count. Since each string contributes exactly once to the frequency of another string, over all $X \in [0\,..\,\sigma)^{<3\tau-1}$, this takes $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ time. Once f_X is computed for all $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, we compute L_{range} as follows. Denote $\Sigma = [0\,..\,\sigma)$. Assume that $L_{\text{range}}[\text{int}(X)] = (b, e)$ holds for some $X \in [0\,..\,\sigma)^{<3\tau-1}$. Then, for any $c \in \Sigma$, it holds $L_{\text{range}}[\text{int}(Xc)] = (e - x - f_{Xc}, e - x)$, where $x = \sum_{c'\in\Sigma, c'>c} f_{Xc'}$, e.g., for $\sigma = 2$, $L_{\text{range}}[\text{int}(X0)] = (e - f_{X1} - f_{X0}, e - f_{X1})$. We thus compute $L_{\text{range}}[\text{int}(X)]$ by

initializing $L_{\mathrm{range}}[\mathrm{int}(\varepsilon)] = (0, n)$, and then enumerating all $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ in the order of non-decreasing length (and, in case of ties, in the reverse lexicographical order). During the enumeration of strings of the form Xc, where $c \in \Sigma$, we maintain the sum $x = \sum_{c' \in \Sigma, c' > c} f_{Xc'}$. Then, using the above formula, the value of $L_{\mathrm{range}}[\mathrm{int}(Xc)]$ can be obtained in $\mathcal{O}(1)$ time. Over all X, the computation of $L_{\mathrm{range}}[\mathrm{int}(X)]$ thus takes $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ time.

To construct L_{per}, we enumerate all $X \in [0\,..\,\sigma)^{3\tau-1}$, and for each X in $\mathcal{O}(\tau^2)$ time we compute $\mathrm{per}(X)$ by trying all $\ell \in [1\,..\,3\tau-1)$. Initializing L_{per} takes $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$. Over all $X \in [0\,..\,\sigma)^{3\tau-1}$, we spend $\mathcal{O}(\sigma^{3\tau-1}\tau^2) = \mathcal{O}(n^{1/2}\log^2 n) = \mathcal{O}(n/\log_\sigma n)$ time.

We finish with the construction of $B_{3\tau-1}$ and A_{short}. First, in $\mathcal{O}(n/\log n)$ time we initialize $B_{3\tau-1}$ to zeros. Next, we initialize temporary counters k and f to zero, and simulate a preorder traversal of the trie of $[0\,..\,\sigma)^{3\tau-1}$. For each visited node with label X, we consider two cases:

- If $|X| < 3\tau - 1$, we check if X is a suffix of T. If so, increment k and f by one, and report X.
- Otherwise (i.e., if $|X| = 3\tau - 1$), if $f_X > 0$, we increment k by one, f by f_X, and report X.

Each time some string X is reported, we set $A_{\mathrm{short}}[k] = X$ and $B_{3\tau-1}[f] = 1$. The correctness of this procedure follows by noting that labels of nodes visited during the preorder traversal are lexicographically sorted. The traversal takes $\mathcal{O}(\sigma^{3\tau}) = \mathcal{O}(n^{3\mu}) = \mathcal{O}(n/\log n)$ time. Finally, using Theorem 2.1, in $\mathcal{O}(n/\log n)$ time we augment $B_{3\tau-1}$ with $\mathcal{O}(1)$-time rank and select queries. □

5.2 The Nonperiodic Positions

In this section, we describe a data structure that, given any $j \in [1\,..\,n]$ (resp. $i \in [1\,..\,n]$) satisfying $j \in [1\,..\,n] \setminus \mathsf{R}$ (resp. $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$) computes $\mathrm{ISA}[j]$ (resp. $\mathrm{SA}[i]$) in $\mathcal{O}(\log^\epsilon n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 5.2.1). Next, we describe the query algorithms (Sections 5.2.2 and 5.2.3). Finally, we show the construction algorithm (Section 5.2.4).

5.2.1 The Data Structure

Definitions We fix some τ-synchronizing set S of T obtained using Theorem 2.4 (recall, that $\tau = \lfloor \mu \log_\sigma n \rfloor$ is fixed for Section 5). We denote $n' = |\mathsf{S}| = \mathcal{O}(n/\tau)$. Let $(s_t^{\mathrm{text}})_{t \in [1\,..\,n']}$ be the sequence containing the elements of S in sorted order, i.e., if $i < j$ then $s_i^{\mathrm{text}} < s_j^{\mathrm{text}}$. Let also $(s_t^{\mathrm{lex}})_{t \in [1\,..\,n']}$ be the sequence containing elements of S sorted according to the lexicographical order of the corresponding suffixes, i.e., if $i < j$ then $T[s_i^{\mathrm{lex}}\,..\,n] \prec T[s_j^{\mathrm{lex}}\,..\,n]$. Let $W[1\,..\,n']$ be a sequence of length-3τ strings such that $W[i] = \overline{X_i}$, where $X_i = T^\infty[s_i^{\mathrm{lex}} - \tau\,..\,s_i^{\mathrm{lex}} + 2\tau)$.

For any $i \in [1\,..\,n{-}2\tau{+}1]$, we define $\mathrm{succ}_{\mathsf{S}}(i) = \min\{j \in \mathsf{S} \cup \{n{-}2\tau{+}2\} : j \geq i\}$ and denote $\mathcal{D} := \{T[i\,..\,\mathrm{succ}_{\mathsf{S}}(i) + 2\tau) : i \in [1\,..\,n{-}3\tau{+}2] \setminus \mathsf{R}\}$. Let $L_{\mathcal{D}}$ be a mapping from $[0\,..\,\sigma)^{3\tau-1}$ to $[0\,..\,\sigma)^{\leq 3\tau-1}$ such that for any $Y \in [0\,..\,\sigma)^{3\tau-1}$, if there exists $X \in \mathcal{D}$ that is a prefix of Y (by the consistency of S, there can be at most one such X), then $L_{\mathcal{D}}$ maps Y to X. Otherwise (i.e., there is no such X), $L_{\mathcal{D}}$ maps Y to ε. Let L_{rev} be a mapping that for every string $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$, returns the packed representation of $\overline{X}$.

Let $B_{\mathsf{S}}[1\,..\,n]$ be a bitvector defined so that $B_{\mathsf{S}}[i] = 1$ holds if and only if $i \in \mathsf{S}$.

Let $A_{\mathrm{smap}}[1\,..\,n']$ be an array storing a permutation of $[1\,..\,n']$ such that $A_{\mathrm{smap}}[i] = j$ holds if and only if $s_j^{\mathrm{text}} = s_i^{\mathrm{lex}}$. Let $A_{\mathrm{smap}}^{-1}[1\,..\,n']$ be an array storing a permutation of $[1\,..\,n']$ such that $A_{\mathrm{smap}}^{-1}[j] = i$ holds if and only if $s_j^{\mathrm{text}} = s_i^{\mathrm{lex}}$.

Components The data structure to handle nonperiodic positions consists of seven components:

1. The index core $\mathrm{C}_{\mathrm{SA}}(T)$ (Section 5.1.1). It takes $\mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{rev}. When accessing L_{rev}, strings $X \in [0\,..\,\sigma)^{\leq 3\tau-1}$ are converted to $\mathrm{int}(X)$. Thus, the mapping L_{rev} needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n^{6\mu}) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The lookup table $L_{\mathcal{D}}$. As above, $L_{\mathcal{D}}$ needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.
4. The bitvector B_{S} augmented using Theorem 2.1. It needs $\mathcal{O}(n/\log n)$ space.
5. The array $A_{\mathrm{smap}}[1\,..\,n']$ in plain form, using $n' = \mathcal{O}(n/\log_\sigma n)$ words of space.
6. The array $A_{\mathrm{smap}}^{-1}[1\,..\,n']$ in plain form, using $n' = \mathcal{O}(n/\log_\sigma n)$ words of space.
7. The data structure of Theorem 2.2 for the sequence $W[1\,..\,n']$. By $n' = \mathcal{O}(n/\log_\sigma n)$ and $\sigma^{3\tau} = \mathcal{O}(\sqrt{n}) = o(n/\log n)$, it needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

5.2.2 Implementation of ISA Queries

For any $j \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}$, letting $X \in \mathcal{D}$ be a prefix of $T[j\,..\,n]$ (by [50, Lemma 6.1], $\mathcal{D}$ is prefix-free, and hence there is exactly one such X), we define

$$\mathrm{Pos}(j) = \{j' \in [1\,..\,n] : \mathrm{LCE}(j,j') \geq |X| \text{ and } T[j'\,..\,n] \preceq T[j\,..\,n]\},$$

and denote $\delta(j) := |\mathrm{Pos}(j)|$.

LEMMA 5.1. *Let $j \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}$ and $X \in \mathcal{D}$ be a prefix of $T[j\,..\,n]$. Denote $\delta_{\mathrm{text}} = |X| - 2\tau$ and $b_X = \mathrm{RangeBeg}(X,T)$. Then:*

1. *It holds $\mathrm{ISA}[j] = b_X + \delta(j)$.*
2. *If $y \in [1\,..\,n']$ is such that $s^{\mathrm{lex}}_y = j + \delta_{\mathrm{text}}$, then $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$.*

Proof. 1. Observe that $j' \in \mathrm{Occ}(X,T)$ holds if and only if $\mathrm{LCE}(j,j') \geq |X|$. Thus, by definition of ISA, we have $\mathrm{ISA}[j] = \mathrm{RangeBeg}(X,T) + |\{j' \in \mathrm{Occ}(X,T) : T[j'\,..\,n] \preceq T[j\,..\,n]\}| = b_X + |\{j' \in [1\,..\,n] : \mathrm{LCE}(j,j') \geq |X| \text{ and } T[j'\,..\,n] \preceq T[j\,..\,n]\}| = b_X + \delta(j)$.

2. Denote $s = j + \delta_{\mathrm{text}}$. By definition of $\mathcal{D}$, we have $s \in \mathsf{S}$. By the consistency of S, there exists a bijection (given by the mapping $j' \mapsto j' + \delta_{\mathrm{text}}$) between positions $j' \in [1\,..\,n] \setminus \mathsf{R}$ satisfying $T[j'\,..\,\mathrm{succ}_{\mathsf{S}}(j') + 2\tau) = X$ and $T[j'\,..\,n] \preceq T[j\,..\,n]$, and positions $s' \in \mathsf{S}$ such that $T^\infty[s' - \delta_{\mathrm{text}}\,..\,s' + 2\tau) = X$ and $T[s'\,..\,n] \preceq T[s\,..\,n]$. Thus, letting $y \in [1\,..\,n']$ be such that $s^{\mathrm{lex}}_y = s$, we obtain that $\delta(j) = |\{i \in [1\,..\,y] : T^\infty[s^{\mathrm{lex}}_i - \delta_{\mathrm{text}}\,..\,s^{\mathrm{lex}}_i + 2\tau) = X\}|$. Since we defined $W[i] = \overline{X_i}$, where $X_i = T^\infty[s^{\mathrm{lex}}_i - \tau\,..\,s^{\mathrm{lex}}_i + 2\tau)$, we conclude that $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$. □

PROPOSITION 5.4. *Let $j \in [1\,..\,n]$ be such that $j \in [1\,..\,n] \setminus \mathsf{R}$. Given the data structure from Section 5.2.1 and the position j, we can compute $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. Given $j \in [1\,..\,n] \setminus \mathsf{R}$, we compute $\mathrm{ISA}[j]$ as follows. If $j > n - 3\tau + 2$, then letting $X = T[j\,..\,n]$, in $\mathcal{O}(1)$ time we compute $(b_X, e_X) = (\mathrm{RangeBeg}(X,T), \mathrm{RangeEnd}(X,T))$ using the lookup table L_{range}. By definition of the lexicographical order, we then have $\mathrm{SA}[b+1] = j$, and hence we return $\mathrm{ISA}[j] = b+1$. Let us thus assume $j \leq n - 3\tau - 2$. By $j \notin \mathsf{R}$ and the density condition of S (see Definition 2.1), this implies that $\mathsf{S} \cap [j\,..\,j+\tau) \neq \emptyset$. In $\mathcal{O}(1)$ time we compute $x = \mathsf{rank}_{B_{\mathsf{S}},1}(j-1)$. Then, in $\mathcal{O}(1)$ we compute $s = \mathsf{select}_{B_{\mathsf{S}},1}(x+1) = \mathrm{succ}_{\mathsf{S}}(j) \in \mathsf{S}$. We then have $X = T[j\,..\,s+2\tau) \in \mathcal{D}$, and in particular, $|X| = s + 2\tau - j$. In $\mathcal{O}(1)$ time we lookup $(b_X, e_X) = L_{\mathrm{range}}[\mathrm{int}(X)]$, i.e., $b_X = \mathrm{RangeBeg}(X,T)$. Letting $y = A^{-1}_{\mathrm{smap}}[x+1]$, we then have $s^{\mathrm{lex}}_y = s = j + |X| - 2\tau$. By Lemma 5.1, it thus remains to determine $\mathsf{rank}_{W,\overline{X}}(y)$. In $\mathcal{O}(1)$ time we compute $\overline{X}$ using the lookup table L_{rev}. In $\mathcal{O}(\log^\epsilon n)$ time, we then compute $\delta(j) = \mathsf{rank}_{W,\overline{X}}(y)$ using Theorem 2.2, and finally return $\mathrm{ISA}[j] = b_X + \delta(j)$. □

5.2.3 Implementation of SA Queries

LEMMA 5.2. *Let $i \in [1\,..\,n]$ be such that $\mathrm{SA}[i] \in [1\,..\,n-3\tau+2] \setminus \mathsf{R}$ and $X \in \mathcal{D}$ be a prefix of $T[\mathrm{SA}[i]\,..\,n]$. Denote $\delta_{\mathrm{text}} = |X| - 2\tau$ and $b_X = \mathrm{RangeBeg}(X,T)$. Then:*

1. *It holds $i = b_X + \delta(\mathrm{SA}[i])$.*
2. *If $y = \mathsf{select}_{W,\overline{X}}(i - b_X)$, then $s^{\mathrm{lex}}_y = \mathrm{SA}[i] + \delta_{\mathrm{text}}$.*

Proof. 1. Denote $j = \mathrm{SA}[i]$. By Lemma 5.1(1), $i = \mathrm{ISA}[j] = b_X + \delta(j) = b_X + \delta(\mathrm{SA}[i])$.

2. By the consistency of S, we have $\mathrm{SA}[i] + \delta_{\mathrm{text}} \in \mathsf{S}$. Thus, there exists $y \in [1\,..\,n']$ such that $s^{\mathrm{lex}}_y = \mathrm{SA}[i] + \delta_{\mathrm{text}}$. By Lemma 5.1(2) applied for $j = \mathrm{SA}[i]$, for any such y it holds $\delta(\mathrm{SA}[i]) = \mathsf{rank}_{W,\overline{X}}(y)$. By (1), we thus have $i - b_X = \mathsf{rank}_{W,\overline{X}}(y)$. Since X is a prefix of $T[\mathrm{SA}[i]\,..\,n]$, such y must also satisfy $T[s^{\mathrm{lex}}_y - \delta_{\mathrm{text}}\,..\,s^{\mathrm{lex}}_y + 2\tau) = X$, or equivalently, $\overline{X}$ must be a prefix of $W[y]$. The only $y \in [1\,..\,n']$ for which $\overline{X}$ is a prefix of $W[y]$ and that satisfies $\mathsf{rank}_{W,\overline{X}}(y) = i - b_X$, is $y = \mathsf{select}_{W,\overline{X}}(i - b_X)$. □

PROPOSITION 5.5. *Let $i \in [1\,..\,n]$ be such that $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$. Given the data structure from Section 5.2.1 and the index i, we can compute $\mathrm{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

Proof. Given $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}$, we compute $\mathrm{SA}[i]$ as follows. First, we compute $y = \mathsf{rank}_{B_{3\tau-1},1}(i-1)$. The string $Y = A_{\mathrm{short}}[y+1]$ is then a prefix of $T[\mathrm{SA}[i]\,..\,n]$ of length $\min(3\tau-1, n+1-i)$. If $|Y| < 3\tau-1$, we therefore have $\mathrm{SA}[i] > n-3\tau+2$ and moreover, $\mathrm{SA}[i]+|Y| = n+1$. Thus, we return $\mathrm{SA}[i] = n+1-|Y|$. Otherwise (i.e., $|Y| = 3\tau-1$), using $L_{\mathcal{D}}$ on Y we determine $x = \mathrm{int}(X)$, where $X \in \mathcal{D}$ is a prefix of $T[\mathrm{SA}[i]\,..\,n]$. In $\mathcal{O}(1)$ time we lookup $(b_X, e_X) = L_{\mathrm{range}}[x]$, i.e., $b_X = \mathrm{RangeBeg}(X, T)$. In $\mathcal{O}(\log^{\epsilon} n)$ time, we then compute $y = \mathsf{select}_{W,\overline{X}}(i-b_X)$ using Theorem 2.2 (the packed representation of $\overline{X}$ is obtained using the lookup table L_{rev} in $\mathcal{O}(1)$ time). By Lemma 5.2(2), we then have $\mathrm{SA}[i] = s^{\mathrm{lex}}_y - \delta_{\mathrm{text}}$, where $\delta_{\mathrm{text}} = |X| - 2\tau$. Using B_{S}, in $\mathcal{O}(1)$ time we compute $j' = \mathsf{select}_{B_{\mathsf{S}},1}(A_{\mathrm{smap}}[y])$. We then have $j' = s^{\mathrm{lex}}_y$, and hence we return $\mathrm{SA}[i] = j' - \delta_{\mathrm{text}}$. Altogether, the query takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

5.2.4 Construction Algorithm

PROPOSITION 5.6. *Given* $\mathrm{C}_{\mathrm{SA}}(T)$*, we can augment it into a data structure from Section 5.2.1 in* $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ *time and using* $\mathcal{O}(n/\log_\sigma n)$ *working space.*

Proof. First, using Theorem 2.4, we construct a τ-synchronizing set S of size $\mathcal{O}(n/\tau)$ in $\mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$ time from a packed representation of T. The set S is returned as an array taking $\mathcal{O}(n/\log_\sigma n)$ space. Using this array, we initialize the bitvector B_{S} in $\mathcal{O}(n/\log_\sigma n)$ time. Augmenting B_{S} with Theorem 2.1 takes $\mathcal{O}(n/\log n)$ time.

Next, we construct the arrays A^{-1}_{smap} and A_{smap}. We start by creating the sequence $(s^{\mathrm{text}}_t)_{t\in[1\,..\,n']}$ using select queries on B_{S}. This takes $\mathcal{O}(n/\log_\sigma n)$ time. Then, given $(s^{\mathrm{text}}_t)_{t\in[1\,..\,n']}$, and the packed representation of T, by [50, Theorem 4.3], we compute the sequence $(s^{\mathrm{lex}}_t)_{t\in[1\,..\,n']}$ in $\mathcal{O}(n/\log_\sigma n)$ time. Given $(s^{\mathrm{lex}}_t)_{t\in[1\,..\,n']}$, we then easily obtain the arrays A^{-1}_{smap} and A_{smap}: simply scan the sequence $(s^{\mathrm{lex}}_t)_{t\in[1\,..\,n']}$ and for each $i \in [1\,..\,n']$, let $j = \mathsf{rank}_{B_{\mathsf{S}},1}(s^{\mathrm{lex}}_i)$ and note that then $s^{\mathrm{text}}_j = s^{\mathrm{lex}}_i$ and hence we can set $A^{-1}_{\mathrm{smap}}[j] = i$ and $A_{\mathrm{smap}}[i] = j$.

Next, we initialize L_{rev}. In the RAM model, such array is easily initialized in $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ time. The sequence $W[1\,..\,n']$ is then obtained from $(s^{\mathrm{lex}}_t)_{t\in[1\,..\,n']}$ using L_{rev} in $\mathcal{O}(n/\log_\sigma n)$ time. We then process W using Theorem 2.2, which takes $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space.

Finally, to construct $L_{\mathcal{D}}$, we first compute a lookup table that for every $Z \in [0\,..\,\sigma)^{2\tau}$ tells whether $T[j\,..\,j+2\tau) = Z$ implies $j \in \mathsf{S}$ (by consistency of S this does not depend on j). Given the array containing the positions in S and the packed representation of T, this takes $\mathcal{O}(\sigma^{2\tau} + |\mathsf{S}|) = \mathcal{O}(n/\log_\sigma n)$ time. Given such lookup table, we iterate through every $Y \in [0\,..\,\sigma)^{3\tau-1}$ and in $\mathcal{O}(\tau)$ time we compute the shortest prefix X of Y whose length-2τ suffix is marked true in the above lookup table. If such X exists, we have $X \in \mathcal{D}$. Accounting for the initialization of $L_{\mathcal{D}}$, over all $Y \in [0\,..\,\sigma)^{3\tau-1}$, this takes $\mathcal{O}(\sigma^{6\tau} + \sigma^{3\tau-1}\log_\sigma n) = \mathcal{O}(n/\log_\sigma n)$ time. □

5.3 The Periodic Positions

In this section, we describe a data structure that, given any $j \in [1\,..\,n]$ (resp. $i \in [1\,..\,n]$) satisfying $j \in \mathsf{R}$ (resp. $\mathrm{SA}[i] \in \mathsf{R}$) computes $\mathrm{ISA}[j]$ (resp. $\mathrm{SA}[i]$) in $\mathcal{O}(\log\log n)$ time.

The section is organized as follows. First, we present the toolbox of combinatorial properties for periodic positions (Section 5.3.1). Next, we introduce the components of the data structure (Section 5.3.2). We then show how using this structure to implement some basic navigational routines (Section 5.3.3). Next, we describe the query algorithms (Sections 5.3.4 and 5.3.5). Finally, we show the construction algorithm (Section 5.3.6).

5.3.1 Preliminaries

We start by introducing the definitions to express the properties utilized in our data structures. For any $j \in \mathsf{R}$, we define $\text{L-root}(j) = \min\{T[j+t\,..\,j+t+p) : t \in [0\,..\,p)\}$, where $p = \mathrm{per}(T[j\,..\,j+3\tau-1))$. We denote $\mathrm{Roots} = \{\text{L-root}(j) : j \in \mathsf{R}\}$. For any $j \in \mathsf{R}$, let $e(j) = \min\{j' \geq j : j' \notin \mathsf{R}\} + 3\tau - 2$.

LEMMA 5.3. *Let* $j \in \mathsf{R}$ *and* $p = \mathrm{per}(T[j\,..\,j+3\tau-1))$*. Then:*

1. *If* $j+1 \in \mathsf{R}$ *then* $\mathrm{per}(T[j+1\,..\,j+3\tau)) = p$*,*
2. *It holds* $e(j) = j + p + \mathrm{LCE}(j, j+p)$*.*

Proof. 1. Denote $P = T[j\,..\,j+3\tau-1)$, $P' = T[j+1\,..\,j+3\tau)$, and $p' = \mathrm{per}(P')$. Our goal is to show that $p' = p$. For a proof by contradiction, assume $p' \neq p$. By the assumption, $\mathrm{per}(P) = p$. Denote $Y = P'[1\,..\,\tau]$, and note that

since P and P' overlap by $3\tau-2 \geq \tau$ symbols, Y is a substring of P, and hence has periods p and p'. Observe that we cannot have $p \mid p'$ since this would imply that $Y[1\,..\,p']$ is not primitive which would contradict $p' = \mathrm{per}(P')$. Observe now that we have $p, p' \leq \frac{1}{3}\tau$. By the Weak Periodicity Lemma [29], we thus have that Y has period $p'' = \gcd(p, p')$. By our assumptions, this implies $p'' < p'$ and $p'' \mid p'$. Thus, again we obtain that $Y[1\,..\,p']$ is not primitive. Therefore, we must have $p' = p$.

2. Denote $j' = e(j) - 3\tau + 2$. By definition, we then have $[j\,..\,j') \subseteq \mathsf{R}$ and $j' \notin \mathsf{R}$. By the above, for every $t \in [0\,..\,j'-j)$, it holds $\mathrm{per}(T[j+t\,..\,j+t+3\tau-1)) = p$. Thus, for every $j'' \in [j\,..\,j'+3\tau-2-p)$, we have $T[j''] = T[j''+p]$, i.e., the substring $T[j\,..\,j'+3\tau-2)$ has period p, and thus $\mathrm{LCE}(j, j+p) \geq (j'+3\tau-2)-j-p$, or equivalently, $j+p+\mathrm{LCE}(j, j+p) \geq j'+3\tau-2 = e(j)$. To show that this lower bound on $j+p+\mathrm{LCE}(j, j+p)$ is tight, let us assume that $e(j) \leq n$ (otherwise, the claim follows immediately). Equivalently, we then have $j'+3\tau-2 = e(j) \leq n$ and to finish the proof, it remains to show $T[e(j)] \neq T[e(j)-p]$. Recall that $\mathrm{per}(T[j'-1\,..\,j'+3\tau-2)) = p$. Thus, $T[j'+3\tau-2] = T[e(j)] = T[e(j)-p] = T[j'+3\tau-2-p]$ would imply that $\mathrm{per}(T[j'\,..\,j'+3\tau-1)) = p$, or equivalently, that $j' \in \mathsf{R}$, a contradiction. □

Observe that by definition of L-root, letting $p = |\text{L-root}(j)|$, there exists $s \in [0\,..\,p)$ such that $T[j+s\,..\,j+s+p) = \text{L-root}(j)$. Combining this with Lemma 5.3 implies that for every $j \in \mathsf{R}$, we can write $T[j\,..\,e(j)) = H'H^kH''$, where $H = \text{L-root}(j)$, and H' (resp. H'') is a proper suffix (resp. prefix) of H. We call such factorization the *L-decomposition* of $T[j\,..\,e(j))$. Note that the L-decomposition is unique, since otherwise would contradict the synchronization property of primitive strings [24, Lemma 1.11]. We denote $\text{L-head}(j) = |H'|$, $\text{L-exp}(j) = k$, and $\text{L-tail}(j) = |H''|$. For $j \in \mathsf{R}$, we let $\mathrm{type}(j) = +1$ if $e(j) \leq n$ and $T[e(j)] \succ T[e(j)-p]$ (where $p = |\text{L-root}(j)|$), and $\mathrm{type}(j) = -1$ otherwise. For any $j \in \mathsf{R}$, we denote $e^{\mathrm{full}}(j) = e(j) - \text{L-tail}(j)$. Observe that $e^{\mathrm{full}}(j) = j + \text{L-head}(j) + \text{L-exp}(j) \cdot |\text{L-root}(j)|$.

We repeatedly refer to the following subsets of R. First, denote $\mathsf{R}^- = \{j \in \mathsf{R} : \mathrm{type}(j) = -1\}$ and $\mathsf{R}^+ = \mathsf{R} \setminus \mathsf{R}^-$. For any $H \in \Sigma^+$ and any $s \in \mathbb{Z}_{\geq 0}$ we then let $\mathsf{R}_H = \{j \in \mathsf{R} : \text{L-root}(j) = H\}$, $\mathsf{R}^-_H = \mathsf{R}^- \cap \mathsf{R}_H$, $\mathsf{R}^+_H = \mathsf{R}^+ \cap \mathsf{R}_H$, $\mathsf{R}_{s,H} = \{j \in \mathsf{R}_H : \text{L-head}(j) = s\}$, $\mathsf{R}^-_{s,H} = \mathsf{R}^- \cap \mathsf{R}_{s,H}$, and $\mathsf{R}^+_{s,H} = \mathsf{R}^+ \cap \mathsf{R}_{s,H}$.

The following lemmas establish the key properties of periodic positions. First, we prove that the set of positions $\mathsf{R}_{s,H}$ occupies a contiguous block in SA and describe the structure of such block.

LEMMA 5.4. *Let $j \in \mathsf{R}_{s,H}$. For any $j' \in [1\,..\,n]$, $\mathrm{LCE}(j, j') \geq 3\tau-1$ holds if and only if $j' \in \mathsf{R}_{s,H}$. Moreover, if $j' \in \mathsf{R}_{s,H}$ then, letting $t = e(j) - j$ and $t' = e(j') - j'$, it holds $\mathrm{LCE}(j, j') \geq \min(t, t')$ and:*

1. *If $\mathrm{type}(j) \neq \mathrm{type}(j')$, then $T[j\,..\,n] \prec T[j'\,..\,n]$ if and only if $\mathrm{type}(j) < \mathrm{type}(j')$,*
2. *If $\mathrm{type}(j) = \mathrm{type}(j') = -1$ and $t \neq t'$, then $T[j\,..\,n] \prec T[j'\,..\,n]$ if and only if $t < t'$,*
3. *If $\mathrm{type}(j) = \mathrm{type}(j') = +1$ and $t \neq t'$, then $T[j\,..\,n] \prec T[j'\,..\,n]$ if and only if $t > t'$,*
4. *If $\mathrm{type}(j) \neq \mathrm{type}(j')$ or $t \neq t'$, then $\mathrm{LCE}(j, j') = \min(t, t')$.*

Proof. Let $j' \in [1\,..\,n]$ be such that $\mathrm{LCE}(j, j') \geq 3\tau-1$. Denoting $p = \mathrm{per}(T[j\,..\,j+3\tau-1))$ and $p' = \mathrm{per}(T[j'\,..\,j'+3\tau-1))$ we then have $p' = p \leq \frac{1}{3}$. Thus, $j' \in \mathsf{R}$ and $\text{L-root}(j') = \min\{T[j'+t\,..\,j'+t+p') : t \in [0\,..\,p')\} = \min\{T[j'+t\,..\,j'+t+p) : t \in [0\,..\,p)\} = \min\{T[j+t\,..\,j+t+p) : t \in [0\,..\,p)\} = H$. To show that $\text{L-head}(j') = s$, note that by $|H| \leq \tau$, the string $H'H^2$ (where H' is a length-s suffix of H) is a prefix of $T[j\,..\,j+3\tau-1) = T[j'\,..\,j'+3\tau-1)$. On the other hand, $\text{L-head}(j') = s'$ implies that $\widehat{H}'H^2$ (where $\widehat{H}'$ is a length-s' suffix of H) is a prefix of $T[j'\,..\,j'+3\tau-1)$. Thus, by the synchronization property of primitive strings [24, Lemma 1.11] applied to the two copies of H, we have $s' = s$, and consequently, $j' \in \mathsf{R}_{s,H}$. For the converse implication, assume $j' \in \mathsf{R}_{s,H}$. This implies that both $T[j\,..\,e(j))$ and $T[j'\,..\,e(j'))$ are prefixes of $H' \cdot H^\infty[1\,..)$ (where H' is as above). Thus, by $e(j) - j$, $e(j') - j' \geq 3\tau-1$, we obtain $\mathrm{LCE}(j, j') \geq 3\tau-1$.

Let us now assume $j' \in \mathsf{R}_{s,H}$. Since, as noted above, both $T[j\,..\,e(j)) = T[j\,..\,j+t)$ and $T[j'\,..\,e(j')) = T[j'\,..\,j'+t')$ are prefixes of $H' \cdot H^\infty[1\,..)$, we have $\mathrm{LCE}(j, j') \geq \min(t, t')$.

1. Assume $\mathrm{type}(j) < \mathrm{type}(j')$. Let $Q = H' \cdot H^\infty[1\,..)$, where H' is a length-s suffix of H. We will prove $T[j\,..\,n] \prec Q \prec T[j'\,..\,n]$, which implies the claim. First, we note that $\mathrm{type}(j) = -1$ implies that either $e(j) = n+1$, or $e(j) \leq n$ and $T[e(j)] \prec T[e(j)-|H|]$. In the first case, $T[j\,..\,e(j)) = T[j\,..\,n]$ is a proper prefix of Q and hence $T[j\,..\,n] \prec Q$. In the second case, we have $T[j\,..\,e(j)) = T[j\,..\,j+t) = Q[1\,..\,t]$ and $T[j+t] \prec T[j+t-|H|] = Q[1+t-|H|] = Q[1+t]$. Consequently, $T[j\,..\,n] \prec Q$. To show $Q \prec T[j'\,..\,n]$ we observe that $\mathrm{type}(j') = +1$ implies $e(j') \leq n$. Thus, we have $Q[1\,..\,t'] = T[j'\,..\,e(j')) = T[j'\,..\,j'+t')$ and $Q[1+t'] = Q[1+t'-|H|] = T[j'+t'-|H|] \prec T[j'+t']$. Hence, we obtain $Q \prec T[j'\,..\,n]$. We have thus obtained $T[j\,..\,n] \prec Q \prec T[j'\,..\,n]$ which implies $T[j\,..\,n] \prec T[j'\,..\,n]$. The opposite implication follows easily by symmetry.

More precisely, in a proof by contraposition, assuming $\text{type}(j) \geq \text{type}(j')$ we immediately obtain $\text{type}(j) > \text{type}(j')$ from the assumption. By the analogous argument as above we then have $T[j\,..\,n] \succ T[j'\,..\,n]$.

2. Assume $t < t'$. Similarly as above, we consider two cases for $e(j)$. If $e(j) = n+1$, then by $t < t'$, the string $T[j\,..\,e(j)) = T[j\,..\,n]$ is a proper prefix of $T[j'\,..\,e(j')) = T[j'\,..\,j'+t')$ and hence $T[j\,..\,n] \prec T[j'\,..\,j'+t') \preceq T[j'\,..\,n]$. On the other hand, if $e(j) \leq n$, then we have $T[j\,..\,j+t) = T[j'\,..\,j'+t)$ and by $t < t'$, $T[j+t] \prec T[j+t-|H|] = T[j'+t-|H|] = T[j'+t]$. Hence, $T[j\,..\,n] \prec T[j'\,..\,n]$. The opposite implication follows by symmetry similarly as in Item 1.

3. Assume $t > t'$. By $\text{type}(j') = +1$ we have $e(j') \leq n$. Thus, by $t > t'$, we have $T[j\,..\,j+t') = T[j'\,..\,j'+t')$ and $T[j+t'] = T[j+t'-|H|] = T[j'+t'-|H|] \prec T[j'+t']$. Hence, $T[j\,..\,n] \prec T[j'\,..\,n]$. The opposite implication follows by symmetry similarly as in Item 1.

4. By the earlier implication, $\text{LCE}(j,j') \geq \min(t,t')$. Thus, it remains to show $\text{LCE}(j,j') \leq \min(t,t')$. First, let $\text{type}(j) \neq \text{type}(j')$ and without the loss of generality let us assume $\text{type}(j) < \text{type}(j')$ (i.e., $\text{type}(j) = -1$ and $\text{type}(j') = +1$). Consider two cases:

- First, assume $t \leq t'$. Our goal is to prove $\text{LCE}(j,j') \leq t$. If $j+t = n+1$, then we immediately obtain the claim. Let us thus assume $j+t \leq n$. In the proof of Item 1 we showed that in this case $\text{type}(j) = -1$ implies $T[j+t] \prec Q[1+t]$. On the other hand, there we also proved that $\text{type}(j') = +1$ implies $Q[1\,..\,t'] = T[j'\,..\,j'+t')$ and $Q[1+t'] \prec T[j'+t']$. By $t \leq t'$, we thus obtain $Q[1+t] \preceq T[j'+t]$. Consequently, $T[j+t] \neq T[j'+t]$ and hence $\text{LCE}(j,j') \leq t$.
- Let us now assume $t > t'$. Our goal is to prove $\text{LCE}(j,j') \leq t'$. In the proof of Item 1 we showed that $\text{type}(j) = -1$ implies that $T[j\,..\,j+t) = Q[1\,..\,t]$. Thus, by $t > t'$ we have $T[j+t'] = Q[1+t']$. On the other hand, in the proof of Item 1 we also proved that $\text{type}(j') = +1$ implies $Q[1+t'] \prec T[j'+t']$. Thus, we obtain $T[j+t'] \neq T[j'+t']$ and hence $\text{LCE}(j,j') \leq t'$.

This concludes the proof of the claim in the case $\text{type}(j) \neq \text{type}(j')$. Let us thus assume $\text{type}(j) = \text{type}(j')$ and $t \neq t'$. First, consider the case $\text{type}(j) = \text{type}(j') = -1$ and assume without the loss of generality that $t < t'$ (to match the assumption in Item 2). Our goal is thus to show $\text{LCE}(j,j') \leq t$. In the proof of Item 2, we showed that we then either have $T[j\,..\,j+t) = T[j\,..\,n]$ (in which case $\text{LCE}(j,j') \leq n-j+1 = t$), or $T[j\,..\,j+t) = T[j'\,..\,j'+t)$ and $T[j+t] \prec T[j'+t]$ (which also immediately implies $\text{LCE}(j,j') \leq t$). Let us now consider the case $\text{type}(j) = \text{type}(j') = +1$ and assume without the loss of generality that $t > t'$ (to match the assumption in Item 3). Our goal is thus to show $\text{LCE}(j,j') \leq t'$. In the proof of Item 3, we showed that we then have $T[j\,..\,j+t') = T[j'\,..\,j'+t')$ and $T[j+t'] \prec T[j'+t']$. This implies $\text{LCE}(j,j') \leq t'$. □

The key to the efficient computation of SA and ISA values for periodic positions is processing of the elements of R in blocks (note that unlike in Lemma 5.4, which describes the structure of blocks in SA, here we mean blocks of positions in the text). The starting positions of these blocks are defined as $\mathsf{R}' := \{j \in \mathsf{R} : j-1 \notin \mathsf{R}\}$. We also let $\mathsf{R}'^{-} = \mathsf{R}' \cap \mathsf{R}^{-}$, $\mathsf{R}'^{+} = \mathsf{R}' \cap \mathsf{R}^{+}$, $\mathsf{R}'^{-}_H = \mathsf{R}' \cap \mathsf{R}^{-}_H$, and $\mathsf{R}'^{+}_H = \mathsf{R}' \cap \mathsf{R}^{+}_H$ for any $H \in \Sigma^+$. The following lemma justifies this strategy.

LEMMA 5.5. *For every $j \in \mathsf{R} \setminus \mathsf{R}'$ it holds:*

- $\text{L-root}(j-1) = \text{L-root}(j)$,
- $e(j-1) = e(j)$,
- $\text{L-tail}(j-1) = \text{L-tail}(j)$,
- $e^{\text{full}}(j-1) = e^{\text{full}}(j)$,
- $\text{type}(j-1) = \text{type}(j)$.

Proof. Denote $p = \text{per}(T[j-1\,..\,j-1+3\tau-1))$. By Lemma 5.3(1), it holds $\text{per}(T[j\,..\,j+3\tau-1)) = p$. By $p \leq \frac{\tau}{3}$, we thus have $T[j-1\,..\,j-1+p) = T[j-1+p\,..\,j-1+2p)$. Consequently, $\{T[j-1+t\,..\,j-1+t+p) : t \in [0\,..\,p)\} = \{T[j+t\,..\,j+t+p) : t \in [0\,..\,p)\}$, and hence $\text{L-root}(j-1) = \text{L-root}(j)$.

Denote $p' = \text{per}(T[j\,..\,j+3\tau-1))$. By Lemma 5.3(2), $e(j-1) = j-1+p+\text{LCE}(j-1,j-1+p)$ and $e(j) = j+p'+\text{LCE}(j,j+p')$. Thus, by $p = p'$ (following by the above) and $T[j-1] = T[j-1+p]$, we have $e(j-1) = j-1+p+\text{LCE}(j-1,j-1+p) = j+p+\text{LCE}(j,j+p) = j+p'+\text{LCE}(j,j+p') = e(j)$.

Assume $T[j-1\,..\,e(j-1)) = H'H^kH''$, where $H = \text{L-root}(j-1)$, $|H'| = \text{L-head}(j-1)$, and $|H''| = \text{L-tail}(j-1)$. By $e(j-1) = e(j)$ and the uniqueness of L-decomposition, this implies that either $T[j\,..\,e(j)) = H'[2\,..\,|H'|]H^kH''$ (if $|H'| > 0$) or $T[j\,..\,e(j)) = H[2\,..\,|H|]H^{k-1}H''$ (otherwise) is the L-decomposition of $T[j\,..\,e(j))$. In both cases, $\text{L-tail}(j-1) = \text{L-tail}(j) = |H''|$.

By the above two properties, $e^{\text{full}}(j-1) = e(j-1) - \text{L-tail}(j-1) = e(j) - \text{L-tail}(j) = e^{\text{full}}(j)$.
The last claim follows from the definition of type and equalities $e(j-1) = e(j)$ and $p = p'$. □

The above is complemented by the following results establishing the lower bound on the gap between blocks of positions in R, and that a mapping from j to $e^{\text{full}}(j)$ establishes an injective mapping of blocks of positions in R to positions in T.

LEMMA 5.6. *Let $j, j', j'' \in [1\,..\,n]$ be such that $j, j'' \in \mathsf{R}$, $j' \notin \mathsf{R}$, and $j < j' < j''$. Then, it holds $e(j) \le j'' + \tau - 1$ and $j'' - j \ge 2\tau$.*

Proof. Let $r = \min\{i \in (j'\,..\,j''] : i \in \mathsf{R}\}$. Then, $r \in \mathsf{R}'$. Observe that by Lemma 5.3 (resp. by $r-1 \notin \mathsf{R}$), it holds $\text{per}(T[j\,..\,e(j))) \le \lfloor \frac{1}{3}\tau \rfloor$ (resp. $\text{per}(T[r\,..\,e(r))) \le \lfloor \frac{1}{3}\tau \rfloor$), $e(j) - j \ge 3\tau - 1$ (resp. $e(r) - r \ge 3\tau - 1$), and the substring $T[j\,..\,e(j))$ (resp. $T[r\,..\,e(r))$ cannot be extended in T to the right (resp. left) without changing its shortest period. By [54, Fact 2.2.4], the fragments $T[j\,..\,e(j))$ and $T[r\,..\,e(r))$ must therefore overlap by less than $2\lfloor \frac{1}{3}\tau \rfloor$ symbols. In other words, $e(j) - r < 2\lfloor \frac{1}{3}\tau \rfloor$. By $r \le j''$ we thus obtain $e(j) \le r + 2\lfloor \frac{1}{3}\tau \rfloor \le j'' + \tau - 1$, i.e., the first claim. Equivalently, we can state that $j'' \ge e(j) - \tau + 1$. By combining this with $e(j) - j \ge 3\tau - 1$, we then obtain $j'' \ge e(j) - \tau + 1 \ge j + 3\tau - 1 - \tau + 1 = j + 2\tau$, i.e., the second claim. □

LEMMA 5.7. *For any $j, j' \in \mathsf{R}'$, $j \ne j'$ implies $e^{\text{full}}(j) \ne e^{\text{full}}(j')$.*

Proof. Assume without the loss of generality that $j < j'$. Then, $j' - 1 \notin \mathsf{R}$. By Lemma 5.6 applied for j, $j' - 1$, and j' we obtain $e(j) \le j' + \tau - 1$. Consequently, $e^{\text{full}}(j) \le e(j) \le j' + \tau - 1$. Let now $r' = \min\{t \in (j'\,..\,n] : t \notin \mathsf{R}\}$. We then have $e(j') = r' + 3\tau - 2$. Since for every $t \in \mathsf{R}$, it holds $e(t) - e^{\text{full}}(t) = \text{L-tail}(t) = |\text{L-root}(t)| \le \lfloor \frac{1}{3}\tau \rfloor$, we thus have $e^{\text{full}}(j') \ge e(j') - \lfloor \frac{1}{3}\tau \rfloor = r' + 3\tau - 2 - \lfloor \frac{1}{3}\tau \rfloor \ge j' + 2\tau - 1$. Combining this with the earlier upper bound on $e^{\text{full}}(j)$, we thus obtain $e^{\text{full}}(j) \le j' + \tau - 1 < j' + 2\tau - 1 \le e^{\text{full}}(j')$. In particular, $e^{\text{full}}(j) \ne e^{\text{full}}(j')$. □

5.3.2 The Data Structure

Definitions Let $q = |\mathsf{R}'^{-}|$ and let $(r_i^{\text{text}})_{i \in [1\,..\,q]}$ be a sequence containing all elements of R'^{-} in sorted order, i.e, for any $i, i' \in [1\,..\,q]$, $i < i'$ implies $r_i^{\text{text}} < r_{i'}^{\text{text}}$. Let $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$ also be a sequence containing all elements $k \in \mathsf{R}'^{-}$, but sorted first according to L-root(k) and in case of ties, by $T[e^{\text{full}}(k)\,..\,n]$. Formally, for any $i, i' \in [1\,..\,q]$, $i < i'$ implies that L-root$(r_i^{\text{lex}}) \prec$ L-root$(r_{i'}^{\text{lex}})$, or L-root$(r_i^{\text{lex}}) =$ L-root$(r_{i'}^{\text{lex}})$ and $T[e^{\text{full}}(r_i^{\text{lex}})\,..\,n] \prec T[e^{\text{full}}(r_{i'}^{\text{lex}})\,..\,n]$. Note that by Lemma 5.7, the sequence $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$ is well-defined. Based on $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$ we define the sequence of integers $(\ell_i)_{i \in [1\,..\,q]}$ as $\ell_i = e^{\text{full}}(r_i^{\text{lex}}) - r_i^{\text{lex}}$.

Let L_{root} denote the mapping from $[0\,..\,\sigma)^{3\tau-1}$ to $\mathbb{N}^2$ such that for any $X \in [0\,..\,\sigma)^{3\tau-1}$ satisfying $\text{per}(X) \le \frac{1}{3}\tau$, L_{root} maps X to a pair (s, p), where $p = \text{per}(X)$ and $s \in [0\,..\,p)$ is such that $X[1{+}s\,..\,1{+}s{+}p) = \min\{X[1{+}t\,..\,1{+}t{+}p) : t \in [0\,..\,p)\}$. We also define $L_{\text{minexp}} : [0\,..\,\sigma)^{3\tau-1} \to [1\,..\,n]$ as the mapping such that for every $X \in [0\,..\,\sigma)^{3\tau-1}$ satisfying $\text{per}(X) \le \frac{1}{3}\tau$, if we let $p = \text{per}(X)$, $H = \min\{X[1+t\,..\,1+t+p) : t \in [0\,..\,p)\}$ and $s \in [0\,..\,p)$ be such that $X[1{+}s\,..\,1{+}s{+}p) = H$, then assuming $\mathsf{R}^{-}_{s,H} \ne \emptyset$, L_{minexp} maps X to $\min\{\text{L-exp}(j) : j \in \mathsf{R}^{-}_{s,H}\}$. Let L_{runs} be a mapping, such that for every $H \in [0\,..\,\sigma)^{\le\tau}$ and every $H' \in [0\,..\,\sigma)^{\le\tau}$, L_{runs} maps the pair (H, H') to (b, e) defined by $b = |\{k \in \mathsf{R}'^{-} : \text{L-root}(k) \prec H, \text{ or L-root}(k) = H \text{ and } T[e^{\text{full}}(k)\,..\,n] \prec H'\}|$ and $e = b + |\{k \in \mathsf{R}'^{-}_{H} : H' \text{ is a prefix of } T[e^{\text{full}}(k)\,..\,n]\}|$. Note that then the set $\{r_i^{\text{lex}} : i \in (b\,..\,e]\}$ consists of all positions $k \in \mathsf{R}'^{-}_{H}$ such that H' is a prefix of $T[e^{\text{full}}(k)\,..\,n]$. In particular, every (H, ε) maps to a pair (b, e) such that $e = \sum_{H' \preceq H} |\mathsf{R}'^{-}_{H'}|$. For any $\ell > 0$, $H \in [0\,..\,\sigma)^{+}$, and $s \in [0\,..\,|H|)$, we define $\text{Pref}_{\ell}(s, H)$ as the length-ℓ prefix of $H' \cdot H^{\infty}[1\,..)$, where H' is a length-s suffix of H. Let L_{pref} denote the mapping that, given the pair (H, s), where $H \in [0\,..\,\sigma)^{\le\tau}$, and $s \in [0\,..\,|H|)$, returns the packed encoding of $\text{Pref}_{3\tau-1}(s, H)$.

Let $B_{\text{exp}}[1\,..\,n]$ be a bitvector such that for every $i \in [1\,..\,n]$, it holds $B_{\text{exp}}[i] = 0$ if and only if $\text{SA}[i] \in [1\,..\,n] \setminus \mathsf{R}^{-}$, or $i < n$ and the positions $j = \text{SA}[i]$ and $j' = \text{SA}[i+1]$ satisfy $j, j' \in \mathsf{R}^{-}_{s,H}$ and L-exp$(j) =$ L-exp(j') for some $H \in \text{Roots}$ and $s \in [0\,..\,|H|)$. Let $B_{\mathsf{R}'}[1\,..\,n]$ be a bitvector defined such that $B_{\mathsf{R}'}[i] = 1$ holds if and only if $i \in \mathsf{R}'$.

Let $A_{\text{len}}[1\,..\,q]$ by an array defined by $A_{\text{len}}[i] = \ell_i$. Let $A_{\text{rmap}}[1\,..\,q]$ be an array containing a permutation of $[1\,..\,q]$ such that $A_{\text{rmap}}[i] = i'$ holds if and only if $r_i^{\text{text}} = r_{i'}^{\text{lex}}$. By $A^{-1}_{\text{rmap}}[1\,..\,q]$ we denote an array containing a permutation of $[1\,..\,q]$ such that $A^{-1}_{\text{rmap}}[i'] = i$ holds if and only if $r_i^{\text{text}} = r_{i'}^{\text{lex}}$.

Components The data structure consists of two parts. The first part, designed to compute $\text{SA}[i]$ (resp. $\text{ISA}[j]$) for $i \in [1\,..\,n]$ (resp. $j \in [1\,..\,n]$) satisfying $\text{SA}[i] \in \mathsf{R}^{-}$ (resp. $j \in \mathsf{R}^{-}$), consists of the following eleven components:

1. $\mathrm{C}_{\mathrm{SA}}(T)$ (Section 5.1.1). It takes $\mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{root}. When accessing L_{root}, strings $X \in [0\,..\,\sigma)^{3\tau-1}$ are converted to $\mathrm{int}(X)$. Thus, L_{root} needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n^{6\mu}) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The lookup table L_{minexp}. As above, L_{minexp} also needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.
4. The lookup table L_{runs}. When storing the mapping from the key (H, H'), we first concatenate H and H' and convert it to an integer $x = \mathrm{int}(HH')$ in the range $[0\,..\,\sigma^{6\tau})$. We then create a triple $(x, |H|)$ (this contains enough information to decode H and H') and injectively map it to a positive integer not exceeding $\sigma^{6\tau}\tau < \sigma^{6\tau}\log_\sigma n$. Thus, L_{runs} can be stored using $\mathcal{O}(n^{6\mu}\log n) = \mathcal{O}(n/\log_\sigma n)$ space.
5. The lookup table L_{pref}. When storing the mapping, we convert the string H to $\mathrm{int}(H)$. By $\mathrm{int}(H) \in [0\,..\,\sigma^{6\tau})$ and $|H| \le \tau$, each pair $(\mathrm{int}(H), s)$ can then be injectively encoded as an integer in the range of size $\sigma^{6\tau}\tau < \sigma^{6\tau}\log_\sigma n$ and hence L_{pref} needs $\mathcal{O}(n^{6\mu}\log n) = \mathcal{O}(n/\log_\sigma n)$ space.
6. The bitvector B_{exp} augmented using Theorem 2.1. It needs $\mathcal{O}(n/\log n)$ space.
7. The bitvector $B_{\mathsf{R}'}$ augmented using Theorem 2.1. It needs $\mathcal{O}(n/\log n)$ space.
8. The array $A_{\mathrm{len}}[1\,..\,q]$ augmented with a structure from Proposition 2.1. To analyze its space usage, consider any $j_1, j_2, j_3 \in \mathsf{R}'$ such that $j_1 < j_2 < j_3$. Then, $j_2 - 1 \notin \mathsf{R}$ and $j_3 - 1 \notin \mathsf{R}$. By Lemma 5.6 applied first for j_1, $j_2 - 1$, and j_2 we have $e(j_1) \le j_2 + \tau - 1$. Applying it again for j_2, $j_3 - 1$, and j_3, we obtain $j_3 - j_2 \ge 2\tau$, or equivalently, $j_2 \le j_3 - 2\tau$. Combining the two inequalities, we thus obtain that $e(j_1) \le j_3 - \tau - 1 < j_3$. This implies that each position of T belongs to at most two intervals in the collection $\{[j\,..\,e(j)) : j \in \mathsf{R}'\}$, and consequently, $\sum_{i=1}^{q} \ell_i \le 2n$. On the other hand, by Lemma 5.6, for every $j, j' \in \mathsf{R}'$, $j \ne j'$ implies $|j' - j| \ge 2\tau$. Thus, $q = \mathcal{O}(n/\tau) = \mathcal{O}(n/\log_\sigma n)$. The array A augmented using Proposition 2.1 thus needs $\mathcal{O}(n/\log_\sigma n)$ space.
9. The array A_{rmap} in plain form using $\mathcal{O}(1+q) = \mathcal{O}(n/\log_\sigma n)$ space.
10. The array A^{-1}_{rmap} in plain form using $\mathcal{O}(1+q) = \mathcal{O}(n/\log_\sigma n)$ space
11. The $\mathcal{O}(n/\log_\sigma n)$-space data structure from [50, Theorem 5.4] that, given any $i, i' \in [1\,..\,n]$, returns $\mathrm{LCE}(i, i')$ in $\mathcal{O}(1)$ time.

The second part of the structure, designed to compute $\mathrm{SA}[i]$ (resp. $\mathrm{ISA}[j]$) for $i \in [1\,..\,n]$ (resp. $j \in [1\,..\,n]$) satisfying $\mathrm{SA}[i] \in \mathsf{R}^+$ (resp. $j \in \mathsf{R}^+$), consists of the symmetric counterparts adapted according to Lemma 5.4.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

5.3.3 Navigation Primitives

Proposition 5.7. *Given the data structure from Section 5.3.2 and any position $j \in \mathsf{R}$, we can in $\mathcal{O}(1)$ time compute the values* $\text{L-root}(j)$, $\text{L-head}(j)$, $\text{L-exp}(j)$, $\text{L-tail}(j)$, *and* $\mathrm{type}(j)$.

Proof. We first compute $x \in [0\,..\,\sigma^{6\tau})$ such that $x = \mathrm{int}(T[j\,..\,j+3\tau-1))$. Given the packed encoding of text T, such x is obtained in $\mathcal{O}(1)$ time. We then look up $(s, p) = L_{\mathrm{root}}[x]$, and in $\mathcal{O}(1)$ time obtain $\text{L-root}(j) = T[j+s\,..\,j+s+p)$ and $\text{L-head}(j) = s$. Next, we compute $\text{L-exp}(j)$ and $\text{L-tail}(j)$. For this we recall that by Lemma 5.3(2), it holds $e(j) = j + p + \mathrm{LCE}(j, j+p)$. Thus, given j and p, we can compute $e(j)$ in $\mathcal{O}(1)$ time. We then obtain $\text{L-exp}(j) = \lfloor \frac{e(j)-j-s}{p} \rfloor$ and $\text{L-tail}(j) = (e(j) - j - s) \bmod p$. Finally, to test if $\mathrm{type}(j) = +1$, we check whether $e(j) \le n$, and if so, whether $T[e(j)] \succ T[e(j) - p]$. □

Proposition 5.8. *Let $i \in [1\,..\,n]$ be such that $\mathrm{SA}[i] \in \mathsf{R}$. Given the data structure from Section 5.3.2 and the index i, in $\mathcal{O}(1)$ time we can compute* $\text{L-root}(\mathrm{SA}[i])$ *and* $\text{L-head}(\mathrm{SA}[i])$.

Proof. We first compute $y = \mathsf{rank}_{B_{3\tau-1},1}(i-1)$. The string $X = A_{\mathrm{short}}[y+1]$ is then a prefix of $T[\mathrm{SA}[i]\,..\,n]$ of length $3\tau-1$. Let $x = \mathrm{int}(X)$. We then look up $(s, p) = L_{\mathrm{root}}[x]$, and in $\mathcal{O}(1)$ time obtain $\text{L-root}(\mathrm{SA}[i]) = X[1+s\,..\,1+s+p)$ and $\text{L-head}(\mathrm{SA}[i]) = s$. □

5.3.4 Implementation of ISA Queries

For any $j \in \mathsf{R}$, we define

$$\mathrm{Pos}(j) = \{j' \in [1\,..\,n] : \mathrm{LCE}(j, j') \ge 3\tau - 1 \text{ and } T[j'\,..\,n] \preceq T[j\,..\,n]\},$$

and denote $\delta(j) = |\mathrm{Pos}(j)|$.

LEMMA 5.8. *Let* $j \in \mathsf{R}$ *and* $X = T[j\,..\,j+3\tau-1)$. *Then,* $\mathrm{ISA}[j] = \mathrm{RangeBeg}(X,T) + \delta(j)$.

Proof. It suffices to observe that $j' \in \mathrm{Occ}(X,T)$ holds if and only if $\mathrm{LCE}(j,j') \geq 3\tau-1$. Thus, it holds by definition of $\mathrm{ISA}[j]$ that $\mathrm{ISA}[j] = \mathrm{RangeBeg}(X,T) + |\{j' \in \mathrm{Occ}(X,T) : T[j'\,..\,n] \preceq T[j\,..\,n]\}| = \mathrm{RangeBeg}(X,T) + |\{j' \in [1\,..\,n] : \mathrm{LCE}(j,j') \geq 3\tau-1 \text{ and } T[j'\,..\,n] \preceq T[j\,..\,n]\}| = \mathrm{RangeBeg}(X,T) + \delta(j)$. □

We focus on computing $\delta(j)$ for $j \in \mathsf{R}^-$. The elements of R^+ are processed symmetrically (see the proof of Proposition 5.11). For any $H \in \mathrm{Roots}$, $s \in [0\,..\,|H|)$, and $j \in \mathsf{R}^-_{s,H}$, we define $\mathrm{Pos}^{\mathsf{a}}(j) = \{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') \leq \text{L-exp}(j)\}$ and $\mathrm{Pos}^{\mathsf{s}}(j) = \{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') = \text{L-exp}(j) \text{ and } T[j'\,..\,n] \succ T[j\,..\,n]\}$. For any $j \in \mathsf{R}^-$, we denote $\delta^{\mathsf{a}}(j) = |\mathrm{Pos}^{\mathsf{a}}(j)|$ and $\delta^{\mathsf{s}}(j) = |\mathrm{Pos}^{\mathsf{s}}(j)|$.

LEMMA 5.9. *For any* $j \in \mathsf{R}^-$, *it holds* $\delta(j) = \delta^{\mathsf{a}}(j) - \delta^{\mathsf{s}}(j)$.

Proof. We will prove that $\mathrm{Pos}^{\mathsf{a}}(j)$ is a disjoint union of $\mathrm{Pos}(j)$ and $\mathrm{Pos}^{\mathsf{s}}(j)$. This implies $\delta(j) + \delta^{\mathsf{s}}(j) = \delta^{\mathsf{a}}(j)$, and consequently, the equality in the claim.

By Lemma 5.4, letting $j \in \mathsf{R}^-_{s,H}$, we have $\mathrm{Pos}(j) = \{j' \in \mathsf{R}^-_{s,H} : T[j'\,..\,n] \preceq T[j\,..\,n]\}$, and moreover, if $j' \in \mathrm{Pos}(j)$, then $e(j') - j' \leq e(j) - j$. In particular, $\text{L-exp}(j') = \lfloor \frac{e(j')-j'-s}{|H|} \rfloor \leq \lfloor \frac{e(j)-j-s}{|H|} \rfloor = \text{L-exp}(j)$. Hence, $\mathrm{Pos}(j) \subseteq \mathrm{Pos}^{\mathsf{a}}(j)$. On the other hand, clearly $\mathrm{Pos}^{\mathsf{s}}(j) \subseteq \mathrm{Pos}^{\mathsf{a}}(j)$ and $\mathrm{Pos}^{\mathsf{s}}(j) \cap \mathrm{Pos}(j) = \emptyset$. Thus, to obtain the claim, it suffices to show that $\mathrm{Pos}^{\mathsf{a}}(j) \setminus \mathrm{Pos}^{\mathsf{s}}(j) \subseteq \mathrm{Pos}(j)$.

Let $j' \in \mathrm{Pos}^{\mathsf{a}}(j) \setminus \mathrm{Pos}^{\mathsf{s}}(j)$. Consider two cases. If $\text{L-exp}(j') = \text{L-exp}(j)$, then by definition of $\mathrm{Pos}^{\mathsf{s}}(j)$, it must hold $T[j'\,..\,n] \preceq T[j\,..\,n]$. Thus, we have $j' \in \mathrm{Pos}(j)$. Let us therefore assume $\text{L-exp}(j') < \text{L-exp}(j)$. Then, $e(j') - j' = s + \text{L-exp}(j') \cdot |H| + \text{L-tail}(j') < s + \text{L-exp}(j') \cdot |H| + |H| \leq s + \text{L-exp}(j) \cdot |H| \leq s + \text{L-exp}(j) \cdot |H| + \text{L-tail}(j) = e(j) - j$. By Lemma 5.4(2), this implies $T[j'\,..\,n] \prec T[j\,..\,n]$, and consequently, $j' \in \mathrm{Pos}(j)$. □

Computing $\delta^{\mathsf{a}}(j)$ We now describe the algorithm to compute $\delta^{\mathsf{a}}(j)$ for $j \in \mathsf{R}^-$.

PROPOSITION 5.9. *Given the data structure from Section 5.3.2 and any* $j \in \mathsf{R}^-$, *in* $\mathcal{O}(1)$ *time we can compute* $\delta^{\mathsf{a}}(j)$.

Proof. Let $X = T[j\,..\,j+3\tau-1)$. First, using the lookup table L_{range}, we compute $(b_X, e_X) = (\mathrm{RangeBeg}(X,T), \mathrm{RangeEnd}(X,T))$. Then, by Lemma 5.4, $\mathrm{SA}(b_X\,..\,e_X]$ contains all positions from $\mathsf{R}_{s,H}$, where $H = \text{L-root}(j)$ and $s = \text{L-head}(j)$. Next, using Proposition 5.7, we compute in $\mathcal{O}(1)$ time the value $k = \text{L-exp}(j)$. Finally, we retrieve $k_{\min} = L_{\mathrm{minexp}}[\mathrm{int}(X)]$. Observe now that by Lemma 5.4, all positions in $\mathsf{R}^-_{s,H}$ occur in $\mathrm{SA}(b_X\,..\,e_X]$ before $\mathsf{R}^+_{s,H}$. Furthermore, by Lemma 5.4(2), $[k_{\min}\,..\,k] \subseteq \{\text{L-exp}(j') : j' \in \mathsf{R}^-_{s,H}\}$ (for $k' \in (k_{\min}\,..\,k]$, we can take $j' = j + (k-k')|H|$). Thus, by the definition of B_{exp}, we can finally return $\delta^{\mathsf{a}}(j) = \mathsf{select}_{B_{\mathrm{exp}},1}(\mathsf{rank}_{B_{\mathrm{exp}},1}(b_X) + (k - k_{\min}) + 1) - b_X$ in $\mathcal{O}(1)$ time. □

Computing $\delta^{\mathsf{s}}(j)$ Next, we describe the algorithm to compute $\delta^{\mathsf{s}}(j)$ for any position $j \in \mathsf{R}^-$.

LEMMA 5.10. *Assume* $i, j \in \mathsf{R}^-_H$ *and let* $\ell = e(i) - i - 3\tau + 2$. *Then* $|\mathrm{Pos}^{\mathsf{s}}(j) \cap [i\,..\,i+\ell)| \leq 1$. *Moreover,* $|\mathrm{Pos}^{\mathsf{s}}(j) \cap [i\,..\,i+\ell)| = 1$ *if and only if* $T[e^{\mathrm{full}}(i)\,..\,n] \succ T[e^{\mathrm{full}}(j)\,..\,n]$ *and* $e^{\mathrm{full}}(i) - i \geq e^{\mathrm{full}}(j) - j$.

Proof. By Lemma 5.5, we have $[i\,..\,i+\ell) \subseteq \mathsf{R}^-_H$ with $e(i+\delta) = e(i)$ for every $\delta \in [0\,..\,\ell)$. Moreover, by the uniqueness of L-decomposition, $\text{L-tail}(i+\delta) = \text{L-tail}(i)$. Together, these imply that $e^{\mathrm{full}}(i+\delta) = e^{\mathrm{full}}(i)$, and consequently $e^{\mathrm{full}}(i+\delta) - (i+\delta) = e^{\mathrm{full}}(i) - i - \delta$. It remains to observe that, letting $j \in \mathsf{R}^-_{s,H}$, for $j' \in \mathrm{Pos}^{\mathsf{s}}(j)$ it holds $e^{\mathrm{full}}(j') - j' = s + \text{L-exp}(j') \cdot |H| = s + \text{L-exp}(j) \cdot |H| = e^{\mathrm{full}}(j) - j$. Thus, $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(j)$ implies $e^{\mathrm{full}}(i+\delta) - (i+\delta) = e^{\mathrm{full}}(i) - (i+\delta) = e^{\mathrm{full}}(j) - j$, or equivalently, $\delta = (e^{\mathrm{full}}(i) - i) - (e^{\mathrm{full}}(j) - j)$, and therefore, $|\mathrm{Pos}^{\mathsf{s}}(j) \cap [i\,..\,i+\ell)| \leq 1$.

For the second part, assume first that $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(j)$ holds for some $\delta \in [0\,..\,\ell)$. Then, as noted above, we have $e^{\mathrm{full}}(j) - j = e^{\mathrm{full}}(i) - (i+\delta) \leq e^{\mathrm{full}}(i) - i$. Moreover, letting $j \in \mathsf{R}^-_{s,H}$, by definition of $\mathrm{Pos}^{\mathsf{s}}(j)$, we have $i + \delta \in \mathsf{R}^-_{s,H}$, $\text{L-exp}(j) = \text{L-exp}(i+\delta)$, and $T[i+\delta\,..\,n] \succ T[j\,..\,n]$. Therefore, we obtain that $T[i+\delta\,..\,e^{\mathrm{full}}(i+\delta)) = T[i+\delta\,..\,e^{\mathrm{full}}(i)) = T[j\,..\,e^{\mathrm{full}}(j)) = H'H^k$ (where $k = \text{L-exp}(j)$ and H' is the length-s suffix of H), and consequently, $T[e^{\mathrm{full}}(i)\,..\,n] \succ T[e^{\mathrm{full}}(j)\,..\,n]$. To show the converse implication, assume $T[e^{\mathrm{full}}(i)\,..\,n] \succ T[e^{\mathrm{full}}(j)\,..\,n]$ and $e^{\mathrm{full}}(i) - i \geq e^{\mathrm{full}}(j) - j$. Let $\delta = (e^{\mathrm{full}}(i) - i) - (e^{\mathrm{full}}(j) - j)$. We will prove that $\delta \in [0\,..\,\ell)$ and $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(j)$. Clearly $\delta \geq 0$. To show $\delta < \ell$, we first prove $e(i) - e^{\mathrm{full}}(i) \geq e(j) - e^{\mathrm{full}}(j)$.

Suppose that $q = e(i) - e^{\text{full}}(i) < e(j) - e^{\text{full}}(j)$. By $i \in \mathsf{R}^-_H$, we then either have $e^{\text{full}}(i) + q = n + 1$, or $e^{\text{full}}(i) + q \le n$ and $T[e^{\text{full}}(i) + q] \prec T[e^{\text{full}}(i) + q - |H|] = T[e^{\text{full}}(j) + q - |H|] = T[e^{\text{full}}(j) + q]$, both of which contradict $T[e^{\text{full}}(i)\,..\,n] \succ T[e^{\text{full}}(j)\,..\,n]$. Thus, $e(i) - e^{\text{full}}(i) \ge e(j) - e^{\text{full}}(j)$. This implies, $e(i) - (i+\delta) = (e^{\text{full}}(i) - (i+\delta)) + (e(i) - e^{\text{full}}(i)) = (e^{\text{full}}(j) - j) + (e(i) - e^{\text{full}}(i)) \ge (e^{\text{full}}(j) - j) + (e(j) - e^{\text{full}}(j)) = e(j) - j \ge 3\tau - 1$, or equivalently $\delta \le e(i) - i - 3\tau + 1 < \ell$. To show $i + \delta \in \text{Pos}^{\mathsf{s}}(j)$, it remains to observe that $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - (i+\delta) = e^{\text{full}}(j) - j$ and $i + \delta, j \in \mathsf{R}^-_H$ imply $T[i+\delta\,..\,e^{\text{full}}(i))] = T[j\,..\,e^{\text{full}}(j))]$. This in particular gives, letting $j \in \mathsf{R}_{s,H}$, that $i + \delta \in \mathsf{R}_{s,H}$ and $\text{L-exp}(i+\delta) = \text{L-exp}(j)$. Moreover, combining it with $T[e^{\text{full}}(i)\,..\,n] \succ T[e^{\text{full}}(j)\,..\,n]$ yields $T[i+\delta\,..\,n] \succ T[j\,..\,n]$. Finally, by Lemma 5.5, $\text{type}(i+\delta) = \text{type}(i) = -1$. Therefore, $i + \delta \in \text{Pos}^{\mathsf{s}}(j)$. □

Proposition 5.10. *Given the data structure from Section 5.3.2 and any $j \in \mathsf{R}^-$, in $\mathcal{O}(\log\log n)$ time we can compute $\delta^{\mathsf{s}}(j)$.*

Proof. Given $j \in \mathsf{R}^-$, we first compute $H = \text{L-root}(j)$, $s = \text{L-head}(j)$, and $k = \text{L-exp}(j)$. By Proposition 5.7, this takes $\mathcal{O}(1)$ time. This lets us deduce $e^{\text{full}}(j) = j + s + k|H|$. Then, we compute $i \in [1\,..\,q]$ satisfying $j \in [r_i^{\text{text}}\,..\,e(r_i^{\text{text}}) - 3\tau + 2)$, i.e., j is in the maximal block of positions from R^- starting at position r_i^{text}. Using $B_{\mathsf{R}'}$ we obtain $i = \mathsf{rank}_{B_{\mathsf{R}'},1}(j)$ in $\mathcal{O}(1)$ time. Observe now that, letting $j' = r_i^{\text{text}}$, by $e^{\text{full}}(j') = e^{\text{full}}(j)$, we have $T[e^{\text{full}}(j')\,..\,n] = T[e^{\text{full}}(j)\,..\,n]$. Therefore, letting $x = A_{\text{rmap}}[i]$ and $x' = \sum_{H' \preceq H} |\mathsf{R}'^-_{H'}|$ (obtained in $\mathcal{O}(1)$ time using L_{runs}), by Lemma 5.10 we have $\delta^{\mathsf{s}}(j) = |\{i' \in (x\,..\,x'] : \ell_{i'} \ge e^{\text{full}}(j) - j\}| = \mathsf{rcount}_{A_{\text{len}}}(e^{\text{full}}(j) - j, x') - \mathsf{rcount}_{A_{\text{len}}}(e^{\text{full}}(j) - j, x)$, which we compute in $\mathcal{O}(\log\log n)$ time using the data structure from Proposition 2.1. □

Remark 5.1. In Lemma 5.9, we presented an equation relating the sizes of $\text{Pos}^{\mathsf{a}}(j)$ and $\text{Pos}^{\mathsf{s}}(j)$, and the size of $\text{Pos}(j)$, where $j \in \mathsf{R}^-$. In this formula, some positions are first counted as part of $\text{Pos}^{\mathsf{a}}(j)$, and then canceled when subtracting the size of $\text{Pos}^{\mathsf{s}}(j)$. To see the reason for this counterintuitive formula, let $J := \{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') = \text{L-exp}(j)\}$, where $s = \text{L-head}(j)$ and $H = \text{L-root}(j)$, and consider the problem of computing the size of $J' = \{j' \in J : T[j'\,..\,n] \succeq T[j\,..\,n]\}$. As shown in Lemma 5.10, to count such positions, it suffices to first align all $j'' \in \mathsf{R}'^-$ by the position $e^{\text{full}}(j'')$, and then count those j'' that satisfy (1) $T[e^{\text{full}}(j'')\,..\,n] \succeq T[e^{\text{full}}(j)\,..\,n]$, and (2) $e^{\text{full}}(j'') - j'' \ge e^{\text{full}}(j) - j$. For every $j'' \in \mathsf{R}'^-$ satisfying these conditions, there exists exactly one $j' \in \mathsf{R}^-_{s,H}$ such that $[j''\,..\,j'] \subseteq \mathsf{R}$, $\text{L-exp}(j') = \text{L-exp}(j)$ and $T[j'\,..\,n] \succeq T[j\,..\,n]$, because for $j, j'' \in \mathsf{R}^-$, $T[e^{\text{full}}(j'')\,..\,n] \succeq T[e^{\text{full}}(j)\,..\,n]$ implies $e(j'') - e^{\text{full}}(j'') \ge e(j) - e^{\text{full}}(j)$ (Lemma 5.4). Thus, letting $\ell = e^{\text{full}}(j) - j$, such j' is given by $j' = e^{\text{full}}(j'') - \ell$. In particular, such j' satisfies $j' \in \mathsf{R}$ because $(e(j'') - e^{\text{full}}(j'')) + \ell \ge e(j) - e^{\text{full}}(j) + \ell = e(j) - j \ge 3\tau - 1$.

Consider now the problem of computing the size of $J'' = \{j' \in J : T[j'\,..\,n] \prec T[j\,..\,n]\}$ (defining $\text{Pos}^{\mathsf{s}}(j)$ as J'' may seem like a simpler alternative to the current definition). Observe that the above method does not work for this problem. The reason for this is that position $j'' \in \mathsf{R}'^-$ satisfying $T[e^{\text{full}}(j')\,..\,n] \prec T[e^{\text{full}}(j)\,..\,n]$ does not necessarily imply that $e^{\text{full}}(j'') - \ell \in \mathsf{R}$. This is because we may have $e(j'') - e^{\text{full}}(j'') < e(j) - e^{\text{full}}(j)$, which implies that it is possible that $(e(j'') - e^{\text{full}}(j'')) + \ell < 3\tau - 1$. This motivates the current definition of $\text{Pos}^{\mathsf{s}}(j)$.

Summary By combining all above results, we obtain the following algorithm to compute $\text{ISA}[j]$ for periodic positions.

Proposition 5.11. *Given the data structure from Section 5.3.2 and any $j \in \mathsf{R}$, in $\mathcal{O}(\log\log n)$ time we can compute* $\text{ISA}[j]$.

Proof. First, in $\mathcal{O}(1)$ time we compute $x = \text{int}(X)$, where $X = T[j\,..\,j + 3\tau - 1)$. In $\mathcal{O}(1)$ we then look up $(b_X, e_X) = L_{\text{range}}[x]$. In particular, we have $b_X = \text{RangeBeg}(X, T)$. Then, using Proposition 5.7 we determine $\text{type}(j)$. Depending on whether $j \in \mathsf{R}^-$ or $j \in \mathsf{R}^+$ we use either a combination of Propositions 5.9 and 5.10, or their symmetric counterparts (more precisely, if $j \in \mathsf{R}^+$, letting $s = \text{L-head}(j)$ and $H = \text{L-root}(j)$, we have $\delta^{\mathsf{a}}(j) = |\text{Pos}^{\mathsf{a}}(j)|$ and $\delta^{\mathsf{s}}(j) = |\text{Pos}^{\mathsf{s}}(j)|$, where $\text{Pos}^{\mathsf{a}}(j) = \{j' \in \mathsf{R}^+_{s,H} : \text{L-exp}(j') \le \text{L-exp}(j)\}$ and $\text{Pos}^{\mathsf{s}}(j) = \{j' \in \mathsf{R}^+_{s,H} : \text{L-exp}(j) = \text{L-exp}(j) \text{ and } T[j'\,..\,n] \prec T[j\,..\,n]\}$), to compute $\delta^{\mathsf{a}}(j)$ and $\delta^{\mathsf{s}}(j)$ in $\mathcal{O}(1)$ and $\mathcal{O}(\log\log n)$ time, respectively. If $j \in \mathsf{R}^-$, then by Lemma 5.9 we have $\delta(j) = \delta^{\mathsf{a}}(j) - \delta^{\mathsf{s}}(j)$. Otherwise, by the counterpart of Lemma 5.9, $\delta(j) = (e_X - b_X) - (\delta^{\mathsf{a}}(j) - \delta^{\mathsf{s}}(j))$. Finally, we return $\text{ISA}[j] = b_X + \delta(j)$ as the answer. In total, the query takes $\mathcal{O}(\log\log n)$ time. □

5.3.5 Implementation of SA Queries

We focus on positions $i \in [1\,..\,n]$ satisfying $\mathrm{SA}[i] \in \mathsf{R}^-$. Positions satisfying $\mathrm{SA}[i] \in \mathsf{R}^+$ are processed symmetrically (see the proof of Proposition 5.14). The algorithm to query $\mathrm{SA}[i]$ for $i \in [1\,..\,n]$ satisfying $\mathrm{SA}[i] \in \mathsf{R}^-$ proceeds in two steps. First, we compute $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^{\mathsf{s}}(\mathrm{SA}[i])$. In the second steps, these values are used to compute $\mathrm{SA}[i]$.

Computing $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^{\mathsf{s}}(\mathrm{SA}[i])$ We now describe the first step during the computation of $\mathrm{SA}[i]$ for $i \in [1\,..\,n]$ satisfying $\mathrm{SA}[i] \in \mathsf{R}$.

PROPOSITION 5.12. *Let $i \in [1\,..\,n]$ be such that $\mathrm{SA}[i] \in \mathsf{R}$. Given the data structure from Section 5.3.2 and the index i, in $\mathcal{O}(1)$ time we can check if $\text{type}(\mathrm{SA}[i]) = -1$, and if so, return $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^{\mathsf{s}}(\mathrm{SA}[i])$.*

Proof. To check if $\text{type}(\mathrm{SA}[i]) = -1$, we first compute $y = \mathsf{rank}_{B_{3\tau-1},1}(i-1)$. The string $X = A_{\text{short}}[y+1]$ is then a prefix of $T[\mathrm{SA}[i]\,..\,n]$ of length $3\tau-1$. Let $x = \text{int}(X)$. In $\mathcal{O}(1)$ time we then look up $(b_X, e_X) = L_{\text{range}}[x]$. By Lemma 5.4 we then have $\text{type}(\mathrm{SA}[i]) = -1$ if and only if $B_{\text{exp}}[i\,..\,e_X]$ contains a bit with value 1. This can be checked in $\mathcal{O}(1)$ time by checking if $\mathsf{rank}_{B_{\text{exp}},1}(e_X) > \mathsf{rank}_{B_{\text{exp}},1}(i-1)$. Let us assume $\text{type}(\mathrm{SA}[i]) = -1$. To compute $\text{L-exp}(\mathrm{SA}[i])$, we first in $\mathcal{O}(1)$ retrieve $k_{\min} = L_{\text{minexp}}[x]$, and then compute $\text{L-exp}(\mathrm{SA}[i]) = k_{\min} + (\mathsf{rank}_{B_{\text{exp}},1}(i-1) - \mathsf{rank}_{B_{\text{exp}},1}(b_X))$. Then, $\delta^{\mathsf{a}}(\mathrm{SA}[i])$ can be computed in $\mathcal{O}(1)$ time as $\delta^{\mathsf{a}}(\mathrm{SA}[i]) = \mathsf{select}_{B_{\text{exp}},1}(\mathsf{rank}_{B_{\text{exp}},1}(i-1)+1) - b_X$. Finally, by applying Lemma 5.8 and Lemma 5.9 for $j = \mathrm{SA}[i]$, it holds $i - b_X = \delta^{\mathsf{a}}(\mathrm{SA}[i]) - \delta^{\mathsf{s}}(\mathrm{SA}[i])$. Thus, we obtain $\delta^{\mathsf{s}}(\mathrm{SA}[i]) = b_X + \delta^{\mathsf{a}}(\mathrm{SA}[i]) - i$. □

Computing $\mathrm{SA}[i]$ We now describe the algorithm to complete the computation of $\mathrm{SA}[i]$ for any $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in \mathsf{R}^-$.

PROPOSITION 5.13. *In $\mathcal{O}(n/\log_\sigma n)$ time, we can augment the structure of Proposition 5.8 so that, given any $i \in [1\,..\,n]$ such that $\mathrm{SA}[i] \in \mathsf{R}^-$, along with $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^{\mathsf{s}}(\mathrm{SA}[i])$, we can compute $\mathrm{SA}[i]$ in $\mathcal{O}(\log\log n)$ time.*

Proof. First, we compute $H = \text{L-root}(\mathrm{SA}[i])$ and $\text{L-head}(\mathrm{SA}[i])$ in $\mathcal{O}(1)$ time using Proposition 5.8. This lets us deduce that $e^{\text{full}}(\mathrm{SA}[i]) - \mathrm{SA}[i] = \ell$, where $\ell = \text{L-head}(\mathrm{SA}[i]) + \text{L-exp}(\mathrm{SA}[i])|H|$. Let $x = \sum_{H' \preceq H} |\mathsf{R}'^-_{H'}|$ (obtained using L_{runs} in $\mathcal{O}(1)$ time). Next, we compute $\delta = \mathsf{rcount}_{A_{\text{len}}}(\ell, x)$. Using the structure from Proposition 2.1, this takes $\mathcal{O}(\log\log n)$ time. Let $k = \delta - \delta^{\mathsf{s}}(\mathrm{SA}[i])$. We then compute the position $p \in [1\,..\,q]$ of the kth leftmost element in A_{len} that is greater or equal than ℓ. Using Proposition 2.1, we compute $p = \mathsf{rselect}_{A_{\text{len}}}(\ell, k)$ in $\mathcal{O}(1)$ time. By Lemma 5.7 and Lemma 5.10, we then have $e^{\text{full}}(r_p^{\text{lex}}) = e^{\text{full}}(\mathrm{SA}[i])$. By combining Lemma 5.5 and Lemma 5.7, for any $j', j'' \in \mathsf{R}$ such that $j' < j''$ and $e^{\text{full}}(j') = e^{\text{full}}(j'')$, it holds $[j'\,..\,j''] \subseteq \mathsf{R}$, i.e., j' and j'' must belong to the same contiguous block of positions from R. Since $r_p^{\text{lex}} \in \mathsf{R}'$, we thus have $\mathrm{SA}[i] \in [r_p^{\text{lex}}\,..\,e(r_p^{\text{lex}}) - 3\tau + 2) \subseteq \mathsf{R}^-_H$. In $\mathcal{O}(1)$ time we obtain $p' = A^{-1}_{\text{rmap}}[p]$ and $j := \mathsf{select}_{B_{R'},1}(p') = r_p^{\text{lex}}$. Observe now that in the block $[j\,..\,e(j) - 3\tau + 2)$ there is at most one element with given values of L-exp and L-head, and we already have values $\text{L-exp}(\mathrm{SA}[i])$ and $\text{L-head}(\mathrm{SA}[i])$. We thus proceed as follows. First, we compute $e(j)$. For this, we recall that by Lemma 5.3(2), it holds $e(j) = j + |H| + \text{LCE}(j, j+|H|)$. Thus, given j and $|H|$, we can compute $e(j)$ in $\mathcal{O}(1)$ time. We then in $\mathcal{O}(1)$ time compute $s = \text{L-head}(j)$ using the lookup table L_{root}. This lets us determine $e^{\text{full}}(j) = e(j) - ((e(j) - j - s) \bmod |H|)$. In $\mathcal{O}(1)$ time we then obtain $\mathrm{SA}[i] = e^{\text{full}}(j) - \text{L-head}(\mathrm{SA}[i]) - \text{L-exp}(\mathrm{SA}[i])|H|$. In total, the query takes $\mathcal{O}(\log\log n)$ time. □

Summary By combining all above results, we obtain the following algorithm to compute $\mathrm{SA}[i]$ for periodic positions.

PROPOSITION 5.14. *Let $i \in [1\,..\,n]$ be such that $\mathrm{SA}[i] \in \mathsf{R}$. Given the data structure from Section 5.3.2 and the index i, in $\mathcal{O}(\log\log n)$ time we can compute $\mathrm{SA}[i]$.*

Proof. First, using Proposition 5.12, in $\mathcal{O}(1)$ time we compute $\text{type}(\mathrm{SA}[i])$. Depending on whether $\mathrm{SA}[i] \in \mathsf{R}^-$ or $\mathrm{SA}[i] \in \mathsf{R}^+$, we use either a combination of Propositions 5.12 and 5.13 or their symmetric counterparts (see the proof of Proposition 5.11), to first compute $\text{L-exp}(\mathrm{SA}[i])$ and $\delta^{\mathsf{s}}(\mathrm{SA}[i])$ in $\mathcal{O}(1)$ time, and then $\mathrm{SA}[i]$ in $\mathcal{O}(\log\log n)$ time. □

5.3.6 Construction Algorithm

PROPOSITION 5.15. *Given* $\mathrm{C}_{\mathrm{SA}}(T)$*, we can in* $\mathcal{O}(n/\log_\sigma n)$ *time augment it into a data structure from Section 5.3.2.*

Proof. Due to a large number of components, as well as dependency of some components on others, we present the description in separate paragraphs, in the order in which it occurs.

Construction of L_{root} To compute L_{root}, we observe that, given $X \in [0\,..\,\sigma)^{3\tau-1}$, we can check in $\mathcal{O}(\tau^2)$ time if $\mathrm{per}(X) \le \frac{1}{3}\tau$, and if so, determine the value $L_{\mathrm{root}}[\mathrm{int}(X)] = (s,p)$. To compute $\mathrm{per}(X)$, we try all $\ell \in [1\,..\,\lfloor\frac{\tau}{3}\rfloor]$ until we find that ℓ is a period of X, or that there is no such ℓ. Assuming $p := \mathrm{per}(X) \le \frac{1}{3}\tau$, finding $s \in [0\,..\,p)$ satisfying $X[1+s\,..\,1+s+p) = \min\{X[t\,..\,t+p) : t \in [1\,..\,p]\}$ also takes $\mathcal{O}(\tau^2)$ time. Initializing L_{root} takes $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$. Over all $X \in [0\,..\,\sigma)^{3\tau-1}$, we spend $\mathcal{O}(\sigma^{3\tau-1}\tau^2) = \mathcal{O}(n^{1/2}\log^2 n) = \mathcal{O}(n/\log_\sigma n)$ time.

Construction of the structure for LCE *queries* By [50, Theorem 5.4], the data structure for LCE queries on T can be constructed from the packed representation of T in $\mathcal{O}(n/\log_\sigma n)$ time.

Construction of B_{exp} To simplify the notation, for the duration of this proof, we denote $E := B_{\mathrm{exp}}$. We use the following definitions. For any $H \in \mathrm{Roots}$ and $s \in [0\,..\,|H|)$, let $E^-_{s,H}$ denote the block of E corresponding to suffixes starting in $\mathsf{R}^-_{s,H}$, i.e., $E^-_{s,H} = E(b\,..\,e]$, where $(b\,..\,e] \subseteq [1\,..\,n]$ is such that $\mathsf{R}^-_{s,H} = \{\mathrm{SA}[i] : i \in (b\,..\,e]\}$ (such $(b\,..\,e]$ exists by Lemma 5.4(1)). Finally, let $\mathrm{unary}(x) := \mathsf{0}^x\mathsf{1}$ denote the unary encoding of an integer $x \ge 0$, and let $\mathrm{unary}^+(x)$ be $\mathrm{unary}(x)$ with the first symbol removed (in particular, $\mathrm{unary}^+(0)$ is the empty string). If $(a_i)_{i\in[1\,..\,k]}$ is a sequence of non-negative integers, we define $\mathrm{unary}((a_i)_{i\in[1\,..\,k]}) := \bigodot_{i=1}^{k}\mathrm{unary}(a_i)$, where $\odot$ denotes concatenation. Analogously, $\mathrm{unary}^+((a_i)_{i\in[1\,..\,k]}) := \bigodot_{i=1}^{k}\mathrm{unary}^+(a_i)$. The definitions of unary and unary^+ are naturally extended to infinite sequences $(a_i)_{i\in[1\,..\,\infty)}$.

Let $\alpha < 1$ be a positive constant. We first show an algorithm that, given the set of positions R'^-_H (where $H \in \mathrm{Roots}$) as input, computes all bitvectors $E^-_{0,H},\ldots,E^-_{|H|-1,H}$ in $\mathcal{O}(|\mathsf{R}'^-_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. For any $s \in [0\,..\,|H|)$ and $k \ge 0$, denote $e_{s,k,H} = |\{j' \in \mathsf{R}^-_{s,H} : \text{L-exp}(j') = k\}|$. We start by observing that by Lemma 5.4(2), $E^-_{s,H} = \mathrm{unary}^+((e_{s,k,H})_{k\in[0\,..\,\infty)})$. The values $e_{s,k,H}$ can be efficiently determined based on the following observation. First, note that if $j \in \mathsf{R}'^-_H$, then $[j\,..\,e(j)-3\tau+2) \subseteq \mathsf{R}^-_H$, and $j-1, e(j)-3\tau+2 \notin \mathsf{R}$, i.e., the block of positions in R^-_H is maximal. By Lemma 5.5, for any $j' \in [j\,..\,e(j)-3\tau+2)$, it holds $e(j') = e(j)$. Thus, for any $j' \in [j\,..\,e(j)-3\tau+2)$, we have $\text{L-exp}(j') = \lfloor\frac{e-j'}{|H|}\rfloor$ and $\text{L-head}(j') = (e-j') \bmod |H|$, where $e = e(j) - \text{L-tail}(j)$. With this in mind, for any $j \in \mathsf{R}'^-_H$, we let $\mathcal{I}_j = (3\tau-2-t\,..\,s+k|H|]$, where $s = \text{L-head}(j)$, $k = \text{L-exp}(j)$, and $t = \text{L-tail}(j)$. By the above discussion, for any $s \in [0\,..\,|H|)$ and $k \ge 0$, we have $e_{s,k,H} = |\{j \in \mathsf{R}'^-_H : s+k|H| \in \mathcal{I}_j\}|$. The algorithm consists of three steps:

1. First, we compute the string $\mathrm{unary}((e_{0,k,H})_{k=0}^{k_{\max}})$, where $k_{\max} = \max\{\text{L-exp}(j') : j' \in \mathsf{R}^-_H\}$. We start by computing $k_{\max}$. For this we observe that $k_{\max} = \max\{\text{L-exp}(j') : j' \in \mathsf{R}'^-_H\}$. Thus, using Proposition 5.7, we can compute $k_{\max}$ in $\mathcal{O}(|\mathsf{R}'^-_H|)$ time. To compute $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$, we generate the sequence of "events" from R'^-_H, sort them, and then output $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$ left-to-right. More precisely, let $m = |\mathsf{R}'^-_H|$, and let $(p_i,v_i)_{i\in[0\,..\,2m]}$ be a sequence containing the multiset $\{(0,0),(k_{\max}+1,0)\} \cup \{(\lceil\min\mathcal{I}_j/|H|\rceil,+1) : j \in \mathsf{R}'^-_H\} \cup \{(\lfloor\max\mathcal{I}_j/|H|\rfloor+1,-1) : j \in \mathsf{R}'^-_H\}$ such that for any $i \in [1\,..\,2m]$, it holds $p_{i-1} \le p_i$. To compute the sequence $(p_i,v_i)_{i\in[0\,..\,2m]}$, we observe that, given $j \in \mathsf{R}'^-_H$, we can compute $\mathcal{I}_j$ in $\mathcal{O}(1)$ time using Proposition 5.7. Thus, in $\mathcal{O}(m)$ time we can generate all pairs in the above multiset. We then sort the pairs by the first element. Using $\lceil 1/\alpha\rceil$-round radix sort, this takes $\mathcal{O}(m+n^\alpha)$ time. Consequently, we can compute $(p_i,v_i)_{i\in[0\,..\,2m]}$ in $\mathcal{O}(|\mathsf{R}'^-_H| + n^\alpha)$ time. Given the sequence $(p_i,v_i)_{i\in[0\,..\,2m]}$, we compute $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$ as follows. First, we initialize the output bitvector to the empty string and set $v = 0$. We then iterate through $i = 1,\ldots,2m$. For every i, we first append $p_i - p_{i-1}$ copies of the string $\mathrm{unary}(v)$ to the output string. We then add v_i to v. To efficiently append multiple copies of $\mathrm{unary}(v)$ to the output, we first precompute (in $\mathcal{O}(\log^2 n) = \mathcal{O}(n^\alpha)$ time) the prefix of length $\log n$ of the string $\mathrm{unary}(x)^\infty[1\,..)$ for every $x \in [0\,..\,\log n)$. This way, we can append $\mathrm{unary}(v)^\ell$ to the output in $\mathcal{O}(1+(v+1)\ell/\log n)$ time. Consequently, the construction of $\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})$ takes $\mathcal{O}(|\mathsf{R}'^-_H| + |\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})|/\log n + n^\alpha) = \mathcal{O}(|\mathsf{R}'^-_H| + |E^-_{0,H}|/\log n + n^\alpha) = \mathcal{O}(|\mathsf{R}'^-_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. To show the first upper bound, observe that $k_{\max} \le |E^-_{0,H}| + \mathcal{O}(\tau/|H|)$. Thus, $|\mathrm{unary}((e_{0,k,H})_{k\in[0\,..\,k_{\max}]})| = |\mathrm{unary}^+((e_{0,k,H})_{k\in[0\,..\,\infty)})| + k_{\max} + 1 = |E^-_{0,H}| + k_{\max} + 1 \le 2|E^-_{0,H}| + \mathcal{O}(\log n)$. The second upper bound follows by observing that $|E^-_{0,H}| + \cdots + |E^-_{|H|-1,H}| = |\mathsf{R}^-_H|$.

2. The second step of the algorithm is to compute the strings $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$ for $s \in [1\,..\,|H|)$. For any $s \in [1\,..\,|H|)$, let $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i\in[0\,..\,m_s]}$ denote the sequence containing all the elements (q,p,v) of the multiset $\{(q,0,0) : q \in [1\,..\,|H|)\} \cup \{(q, k_{\max}+1, 0) : q \in [1\,..\,|H|)\} \cup \{(\min \mathcal{I}_j \bmod |H|, \lfloor \min \mathcal{I}_j/|H| \rfloor, +1) : j \in \mathsf{R}'^{-}_H\} \cup \{((\max \mathcal{I}_j + 1) \bmod |H|, \lfloor (\max \mathcal{I}_j + 1)/|H| \rfloor, -1) : j \in \mathsf{R}'^{-}_H\}$ that satisfy $q = s$, and for any $i \in [1\,..\,m_s]$, it holds $p_{i-1}^{(s)} \le p_i^{(s)}$ (note that the elements of this multiset satisfying $q = 0$ are not included in any sequence). To compute the sequences $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i\in[0\,..\,m_s]}$ for all $s \in [1\,..\,|H|)$, we first enumerate all triples in the above multiset. Using Proposition 5.7, this takes $\mathcal{O}(m)$ time. We then sort the triples lexicographically. Using $\lceil 1/\alpha \rceil$-round radix sort, this takes $\mathcal{O}(m + n^\alpha)$ time. This yields all sequences concatenated together. It is easy to discard unused elements, and to detect boundaries between lists with a single scan. Consequently, we can construct all sequences in $\mathcal{O}(|\mathsf{R}'^{-}_H| + n^\alpha)$ time. Given the above sequences, we can compute the strings $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$ for $s \in [1\,..\,|H|)$ as follows. The algorithm computes the strings in the order of increasing s. More precisely, given the string $U := \mathrm{unary}((e_{s-1,k,H})_{k\in[0\,..\,k_{\max}]})$ and the sequence $(q_i^{(s)}, p_i^{(s)}, v_i^{(s)})_{i\in[0\,..\,m_s]}$ (where $s \in [1\,..\,|H|)$), we compute the string $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$ in $\mathcal{O}(m_s + |U|/\log n)$ time as follows. First, we initialize the output bitvector to the empty string, and set $v = 0$ and $y = 0$. We then iterate through $i = 1, \ldots, m_s$. For every i, we first check if $p_i^{(s)} > p_{i-1}^{(s)}$. If yes, we perform the following three steps. First, find the position y' of the $p_i^{(s)}$th 1-bit in U. Second, append the substring $U(y\,..\,y']$ to the output, except we first prepend it with v zeros (if $v \ge 0$) or discard its first $-v$ bits (if $v < 0$). Finally, we set $y = y'$ and $v = 0$. Then (regardless of whether $p_i^{(s)} > p_{i-1}^{(s)}$), we add $v_i^{(s)}$ to v. To efficiently compute y' we observe that the arguments of the consecutive select queries are increasing. We can thus precompute in $\mathcal{O}(n^\alpha)$ time a lookup table such that the computation of y' takes $\mathcal{O}(1 + (y' - y)/\log n)$ time (these lookup tables can be shared among algorithms for different s). Note that for any $s \in [0\,..\,|H|)$, we have $k_{\max} \le |E^-_{s,H}| + \mathcal{O}(\tau/|H|)$. Thus, $|U| \le 2|E^-_{s-1,H}| + \mathcal{O}(\log n)$, and hence the algorithm runs in $\mathcal{O}(m_s + |E^-_{s-1,H}|/\log n)$ time. Consequently, by $m_0 + \cdots + m_{|H|-1} \le 2|\mathsf{R}'^{-}_H| + 2|H|$ and $|E^-_{0,H}| + \cdots + |E^-_{|H|-1,H}| = |\mathsf{R}^-_H|$, over all $s \in [1\,..\,|H|)$, we spend $\mathcal{O}(|\mathsf{R}'^{-}_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time.
3. The third and final step of the algorithm is to convert the string $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$ into $\mathrm{unary}^+((e_{s,k,H})_{k\in[0\,..\,k_{\max}]}) = E^-_{s,H}$ for every $s \in [0\,..\,|H|)$. Let us fix some $s \in [0\,..\,|H|)$. Observe that to implement the conversion, it suffices to remove the first bit, as well as every bit following a 1-bit in $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$. In the RAM model, such local operation is easy implemented in $\mathcal{O}(1 + |\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})|/\log n)$ time after a $\mathcal{O}(n^\alpha)$-time preprocessing (we do the preprocessing once for all $s \in [0\,..\,|H|)$). As observed above, $|\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})| \le 2|E^-_{s,H}| + \mathcal{O}(\log n)$. Thus, the total time to perform the conversion for all s is $\mathcal{O}(|R^-_H|/\log n + n^\alpha)$.

Using the above algorithm, we construct E as follows. We start by computing the set $\{(\mathrm{int}(\text{L-root}(j)), j)\}_{j\in\mathsf{R}'^{-}}$. For this, observe that for every τ-synchronizing set P of T, by the density condition (see also [50, Section 6.1.2]), $i \in \mathsf{R}'$ implies that either $i = 1$ or $i > 1$ and $i - 1 \in \mathsf{P}$. In particular, $|\mathsf{R}'^{-}| \le |\mathsf{R}'| \le 1 + |\mathsf{P}|$. We thus proceed as follows. First, using [50, Theorem 8.11] in $\mathcal{O}(n/\log_\sigma n)$ time we construct any τ-synchronizing set P of T of size $\mathcal{O}(n/\tau)$. Then, using the above observation together with Proposition 5.7, we enumerate the set $\{(\mathrm{int}(\text{L-root}(j)), j)\}_{j\in\mathsf{R}'^{-}}$ in $\mathcal{O}(1 + |\mathsf{P}|) = \mathcal{O}(n/\log_\sigma n)$ time. We then discard P. Using $\lceil 1/\alpha \rceil$-round radix sort we then sort in $\mathcal{O}(|\mathsf{R}'^{-}| + n^\alpha) = \mathcal{O}(n/\log_\sigma n + n^\alpha)$ time the set of pairs by the first coordinate. This yields the representation of sets R'^{-}_H for all $H \in \mathrm{Roots}$. For each $H \in \mathrm{Roots}$, we then use the above algorithm to compute bitvectors $E^-_{0,H}, \ldots, E^-_{|H|-1,H}$ in $\mathcal{O}(|\mathsf{R}'^{-}_H| + |\mathsf{R}^-_H|/\log n + n^\alpha)$ time. By $\mathrm{Roots} \subseteq [0\,..\,\sigma)^{\le\tau}$, over all H, this takes $\mathcal{O}(|\mathsf{R}'^{-}| + |R^-|/\log n + n^{\alpha+\mu})$ time (recall that $\tau = \lfloor \mu \log_\sigma n \rfloor$ and $\mu < \frac{1}{6}$). Choosing $\alpha < 1 - \mu$ results in $\mathcal{O}(n/\log_\sigma n)$ total time. When bitvectors $E^-_{s,H}$ are computed for all $H \in \mathrm{Roots}$ and $s \in [0\,..\,|H|)$, we initialize E to the string $\mathsf{0}^n$ in $\mathcal{O}(n/\log n)$ time, and then "paste" all the non-empty bitvectors $E^-_{s,H}$ into their correct positions. Given $H \in \mathrm{Roots}$ and $s \in [0\,..\,|H|)$, we first compute in $\mathcal{O}(\log n)$ time the corresponding string $X \in [0\,..\,\sigma)^{3\tau-1}$, and then compute the position to paste $E^-_{s,H}$ using the lookup table L_{range}. Over all $H \in \mathrm{Roots}$ and $s \in [0\,..\,|H|)$, this takes $\mathcal{O}(n^\mu \log^2 n + n/\log n) = \mathcal{O}(n/\log_\sigma n)$ time. Thus, altogether, constructing E and augmenting it using Theorem 2.1 takes $\mathcal{O}(n/\log_\sigma n)$ time.

Construction of L_{minexp} Observe that in the above algorithm, if i is the position of the leftmost 0-bit in $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$, then $\min\{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\} = i - 1$. Given the packed representation of $\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})$, the position i can be easily found in $\mathcal{O}(1 + |\mathrm{unary}((e_{s,k,H})_{k\in[0\,..\,k_{\max}]})|/\log n)$ time. Thus, accounting for the computation of $X \in [0\,..\,\sigma)^{3\tau-1}$ corresponding to the choice of $H \in \mathrm{Roots}$ and $s \in [0\,..\,|H|)$, we can initialize L_{minexp} in $\mathcal{O}(n/\log n + n^\mu \log^2 n) = \mathcal{O}(n/\log_\sigma n)$ time.

Construction of $B_{\mathsf{R}'}$ As seen above, we can enumerate R', and thereby compute $B_{\mathsf{R}'}$, in $\mathcal{O}(n/\log_\sigma n)$ time. Augmenting $B_{\mathsf{R}'}$ with Theorem 2.1 takes $\mathcal{O}(n/\log n)$ time.

Construction of A_{rmap} Since for any $j \in \mathsf{R}$, we can in $\mathcal{O}(1)$ compute L-root(j), $e(j)$, $s =$ L-head(j), and $k =$ L-exp(j), in $\mathcal{O}(n/\log_\sigma n)$ time we can also enumerate all $j \in \mathsf{R}'^{-}$. The key challenge is computing the sequence $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$. By the density condition, for every τ-synchronizing set P of T, it holds that if $j \in \mathsf{R}$, then $e(j) - 2\tau + 1 \in \mathsf{P}$ (for a proof, simply compare the claims of Lemma 5.3 and [50, Fact 3.2]). This lets us compute $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$ as follows. First, using [50, Theorem 8.11], in $\mathcal{O}(n/\log_\sigma n)$ time we construct any τ-synchronizing set P of T of size $\mathcal{O}(n/\tau)$. The set P is returned as an array of size $|\mathsf{P}|$. In $\mathcal{O}(n/\log_\sigma n)$ time we then create a bitvector $B_{\mathsf{P}}[1\,..\,n]$ such that $B_{\mathsf{P}}[i] = 1$ holds if and only if $i \in \mathsf{P}$. In $\mathcal{O}(n/\log n)$ time we augment B_{P} using Theorem 2.1. Let $(p_t^{\mathrm{text}})_{t\in[1\,..\,|\mathsf{P}|]}$ denote a sequence containing elements of P in increasing order and let $(p_t^{\mathrm{lex}})_{t\in[1\,..\,|\mathsf{P}|]}$ denote a sequence containing P sorted according to the lexicographical order of the corresponding suffixes in T, i.e., such that for any $i, i' \in [1\,..\,|\mathsf{P}|]$, $i < i'$ implies $T[p_i^{\mathrm{lex}}\,..\,n] \prec T[p_{i'}^{\mathrm{lex}}\,..\,n]$. Given the array containing P, we compute the sequence $(p_t^{\mathrm{lex}})_{t\in[1\,..\,|\mathsf{P}|]}$ in $\mathcal{O}(n/\log_\sigma n)$ time using [50, Theorem 4.3]. Let $\mathrm{ISA}_{\mathsf{P}}[1\,..\,|\mathsf{P}|]$ be an array storing a permutation of $[1\,..\,|\mathsf{P}|]$ such that $\mathrm{ISA}_{\mathsf{P}}[j] = i$ holds if and only if $p_j^{\mathrm{text}} = p_i^{\mathrm{lex}}$. Using the sequence $(p_t^{\mathrm{lex}})_{t\in[1\,..\,|\mathsf{P}|]}$ and the bitvector B_{P}, we compute $\mathrm{ISA}_{\mathsf{P}}$ in $\mathcal{O}(|\mathsf{P}|) = \mathcal{O}(n/\log_\sigma n)$ time: For every $i \in [1\,..\,|\mathsf{P}|]$, we first compute $j = \mathsf{rank}_{B_{\mathsf{P}},1}(p_i^{\mathrm{lex}})$ and then set $\mathrm{ISA}_{\mathsf{P}}[j] = i$. Next, for each $j \in \mathsf{R}'^{-}$, letting $H =$ L-root(j) and $j_{\mathsf{P}} = \mathsf{rank}_{B_{\mathsf{P}},1}(e(j) - 2\tau + 1)$, we form a tuple $(\mathrm{int}(H), e(j) - e^{\mathrm{full}}(j), \mathrm{ISA}_{\mathsf{P}}[j_{\mathsf{P}}], j)$. Observe, that $X \prec X'$ holds if and only if $\mathrm{int}(X) < \mathrm{int}(X')$. Let $j, j' \in \mathsf{R}'^{-}_H$. Note that since both $T[e^{\mathrm{full}}(j)\,..\,e(j))$ and $T[e^{\mathrm{full}}(j')\,..\,e(j'))$ are prefixes of H, by definition of R^{-}, $e(j) - e^{\mathrm{full}}(j) < e(j') - e^{\mathrm{full}}(j')$ implies $T[e^{\mathrm{full}}(j)\,..\,n] \prec T[e^{\mathrm{full}}(j')\,..\,n]$. If $e(j) - e^{\mathrm{full}}(j) = e(j') - e^{\mathrm{full}}(j')$, then $T[e(j) - 2\tau + 1\,..\,e^{\mathrm{full}}(j)) = T[e(j') - 2\tau + 1\,..\,e^{\mathrm{full}}(j'))$, and consequently, $T[e^{\mathrm{full}}(j)\,..\,n] \prec T[e^{\mathrm{full}}(j')\,..\,n]$ holds if and only if $\mathrm{ISA}_{\mathsf{P}}[j_{\mathsf{P}}] < \mathrm{ISA}_{\mathsf{P}}[j'_{\mathsf{P}}]$. We have thus shown that sorting the tuples lexicographically yields a sequence $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$ on the fourth coordinate. Given $j \in \mathsf{R}'^{-}$, we can compute the corresponding tuple in $\mathcal{O}(1)$ time. Thus, since all its elements are integers in the range $[1\,..\,n]$, using LSD radix-sort, we can compute $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$ in $\mathcal{O}(n/\log_\sigma n)$ time. With a single scan of $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$ and the help of rank queries on $B_{\mathsf{R}'}$ we can then compute table A_{rmap} in $\mathcal{O}(n/\log_\sigma n)$ time.

Construction of A_{rmap}^{-1} Given A_{rmap}, we can compute A_{rmap}^{-1} in $\mathcal{O}(q) = \mathcal{O}(n/\log_\sigma n)$ time, since these two arrays are inverses of each other.

Construction of L_{runs} In $\mathcal{O}(\sigma^\tau + |\mathsf{R}'^{-}|)$ time we perform a synchronized enumeration of all $H \in [0\,..\,\sigma)^{\le\tau}$ in lexicographical order and the L-root values (obtained using Proposition 5.7) for positions in the sequence $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$. This lets us obtain the pair (b_H, e_H) satisfying $\{r_i^{\mathrm{lex}} : i \in (b_H\,..\,e_H]\} = \mathsf{R}'^{-}_H$ for every $H \in [0\,..\,\sigma)^{\le\tau}$ satisfying $\mathsf{R}'^{-}_H \neq \emptyset$. For each such H, we then enumerate all $H' \in [0\,..\,\sigma)^{\le\tau}$ and for each we find corresponding subrange of $(b_H\,..\,e_H]$ in $\mathcal{O}(\tau\log n)$ time using binary search. Overall, the initialization of L_{runs} takes $\mathcal{O}(\sigma^{6\tau}\tau + |\mathsf{R}'^{-}| + \sigma^{2\tau}\tau\log n) = \mathcal{O}(n/\log_\sigma n)$ time.

Construction of L_{pref} To construct L_{pref}, we enumerate all possible $H \in [0\,..\,\sigma)^{\le\tau}$. For each H, we try all $s \in [0\,..\,|H|)$, and for each we construct the string $\mathrm{Pref}_{3\tau-1}(s, H)$ in $\mathcal{O}(\tau)$ time. Over all H, and including the initialization of L_{pref}, this takes $\mathcal{O}(\sigma^{6\tau}\tau + \sigma^\tau\tau^2) = \mathcal{O}(n^{6\mu}\log n) = \mathcal{O}(n/\log_\sigma n)$ time.

Construction of range counting/selection for A From $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$ we construct in $\mathcal{O}(n/\log_\sigma n)$ time the sequence $(\ell_i)_{i\in[1\,..\,q]}$, and then build the array $A_{\mathrm{len}}[1\,..\,q]$ and augment it with a range counting/selection data structure. Using Proposition 2.1, by $q = \mathcal{O}(n/\log_\sigma n)$ and $\sum_{i=1}^{q} A_{\mathrm{len}}[i] = \mathcal{O}(n)$, this takes $\mathcal{O}(n/\log_\sigma n)$ time.

Construction of the remaining components After the above components are constructed, we then analogously construct their symmetric counterparts (adapted according to Lemma 5.4). □

5.4 The Final Data Structure

In this section, we put together Sections 5.1 to 5.3 to obtain a data structure that, given any $j \in [1\,..\,n]$ (resp. $i \in [1\,..\,n]$) computes $\mathrm{ISA}[j]$ (resp. $\mathrm{SA}[i]$) in $\mathcal{O}(\log^\epsilon n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 5.4.1). Next, we describe the query algorithms (Sections 5.4.2 and 5.4.3). Finally, we show the construction algorithm (Section 5.4.4).

5.4.1 The Data Structure

The data structure consists of two components:

1. The structure from Section 5.2.1 (used to handle nonperiodic positions).
2. The structure from Section 5.3.2 (used to handle periodic positions).

In total, the data structure needs $\mathcal{O}(n/\log_\sigma n)$ space.

5.4.2 Implementation of ISA Queries

PROPOSITION 5.16. *Given the data structure from Section 5.4.1 and any $j \in [1\,..\,n]$, in $\mathcal{O}(\log^\epsilon n)$ time we can compute* $\mathrm{ISA}[j]$.

Proof. First, we use Proposition 5.1 to check in $\mathcal{O}(1)$ time if $j \in \mathsf{R}$. Depending on whether $j \in \mathsf{R}$ or not, we use Proposition 5.4 or Proposition 5.11 to compute $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ or $\mathcal{O}(\log\log n)$ time (respectively). □

5.4.3 Implementation of SA Queries

PROPOSITION 5.17. *Given the data structure from Section 5.4.1 and any $i \in [1\,..\,n]$, in $\mathcal{O}(\log^\epsilon n)$ time we can compute* $\mathrm{SA}[i]$.

Proof. First, we use Proposition 5.2 to check in $\mathcal{O}(1)$ time if $\mathrm{SA}[i] \in \mathsf{R}$. Depending on whether $\mathrm{SA}[i] \in \mathsf{R}$ or not, we use Proposition 5.5 or Proposition 5.14 to compute $\mathrm{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ or $\mathcal{O}(\log\log n)$ time (respectively). □

5.4.4 Construction Algorithm

PROPOSITION 5.18. *Given the packed representation of $T \in [0\,..\,\sigma)^n$, we can construct the data structure from Section 5.4.1 in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space.*

Proof. First, from a packed representation of T, we construct $\mathrm{C}_{\mathrm{SA}}(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 5.3. Then, using Propositions 5.6 and 5.15, we augment $\mathrm{C}_{\mathrm{SA}}(T)$ into the two components of the structure from Section 5.4.1 in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ and $\mathcal{O}(n/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space. □

5.5 Summary

By combining Propositions 5.16 to 5.18 we obtain the following final result of this section.

THEOREM 5.1. *Given any constant $\epsilon \in (0, 1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/7}$, in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space we can construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that:*

- *Given any $i \in [1\,..\,n]$ returns $\mathrm{SA}[i]$ in $\mathcal{O}(\log^\epsilon n)$ time,*
- *Given any $j \in [1\,..\,n]$ returns $\mathrm{ISA}[j]$ in $\mathcal{O}(\log^\epsilon n)$ time.*

We also immediately obtain the following more general result.

THEOREM 5.2. *Consider a data structure answering prefix rank and selection queries that, for any string of length m over alphabet $[0\,..\,\sigma)^\ell$, achieves the following complexities:*

1. *Space usage $S(m, \ell, \sigma)$,*
2. *Preprocessing time $P_t(m, \ell, \sigma)$,*
3. *Preprocessing space $P_s(m, \ell, \sigma)$,*
4. *Query time $Q(m, \ell, \sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/7}$, there exist $m = \mathcal{O}(n/\log_\sigma n)$ and $\ell = \mathcal{O}(\log_\sigma n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(m, \ell, \sigma))$ time and $\mathcal{O}(n/\log_\sigma n + P_s(m, \ell, \sigma))$ working space build a structure of size $\mathcal{O}(n/\log_\sigma n + S(m, \ell, \sigma))$ that:

- *Given any $i \in [1\,..\,n]$ returns $\mathrm{SA}[i]$ in $\mathcal{O}(\log\log n + Q(m, \ell, \sigma))$ time,*
- *Given any $j \in [1\,..\,n]$ returns $\mathrm{ISA}[j]$ in $\mathcal{O}(\log\log n + Q(m, \ell, \sigma))$ time.*

6 Pattern Matching Queries

Let $\epsilon \in (0,1)$ be any fixed constant and let $T \in [0\,..\,\sigma)^n$, where $2 \leq \sigma < n^{1/7}$. In this section we show how, given the packed representation of T, to construct in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space a structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of a pattern $P \in [0\,..\,\sigma)^m$, returns $\mathrm{RangeBeg}(P,T)$ and $\mathrm{RangeEnd}(P,T)$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time. We also derive a general reduction depending on prefix rank and selection queries.

As in Section 5, we let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is some positive constant smaller than $\frac{1}{6}$ such that $\tau \geq 1$, be fixed for the duration of this section. Throughout, we also use R as a shorthand for $\mathsf{R}(\tau, T)$.

Definition 6.1. *Let $P \in [0\,..\,\sigma)^m$. We call pattern P* periodic *if it holds that $m \geq 3\tau - 1$ and $\mathrm{per}(P[1\,..\,3\tau-1]) \leq \frac{1}{3}\tau$. Otherwise, P is* nonperiodic.

Organization The structure and the query algorithm for a pattern P are different depending on whether P is periodic (Definition 6.1). Our description is thus split as follows. First (Section 6.1), we describe the set of data structures called collectively the index "core" that enables efficiently checking if P is periodic (it is also used to handle very short patterns and contains some common components utilized by the remaining parts). In the following two parts (Sections 6.2 and 6.3), we describe structures handling each of the two cases. All ingredients are then put together in Section 6.4. Finally, we present our result in the general form (Section 6.5).

6.1 The Index Core

In this section, we present a data structure that, given a packed representation of any pattern $P \in [0\,..\,\sigma)^m$, lets us in $\mathcal{O}(1)$ time check if P is periodic. It also let us compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ if $m < 3\tau - 1$.

The section is organized as follows. First, we introduce the components of the data structure (Section 6.1.1). We then show how using this structure to implement the periodicity check (Section 6.1.2). Next, we describe the query algorithm for short patterns (Section 6.1.3). Finally, we show the construction algorithm (Section 6.1.4).

6.1.1 The Data Structure

The index core, denoted $\mathrm{C}_{\mathrm{PM}}(T)$ consists of the following subset of components of $\mathrm{C}_{\mathrm{SA}}(T)$:

1. The packed representation of T using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The lookup table L_{range} using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.
3. The lookup table L_{per} using $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space.

In total, $\mathrm{C}_{\mathrm{PM}}(T)$ needs $\mathcal{O}(n/\log_\sigma n)$ space.

6.1.2 Navigation Primitives

Proposition 6.1. *Given $\mathrm{C}_{\mathrm{PM}}(T)$ and a packed representation of $P \in [0\,..\,\sigma)^m$, we can in $\mathcal{O}(1)$ time determine whether P is periodic (Definition 6.1).*

Proof. If $m < 3\tau - 1$, we return false. Otherwise, in $\mathcal{O}(1)$ time we compute $x = \mathrm{int}(X)$, where $X = P[1\,..\,3\tau-1]$. We then look up $p = L_{\mathrm{per}}[x]$ and return true if and only if $p \leq \frac{1}{3}\tau$. □

6.1.3 Implementation of Queries

Proposition 6.2. *Let $P \in [0\,..\,\sigma)^m$ be a pattern satisfying $m < 3\tau - 1$. Given $\mathrm{C}_{\mathrm{PM}}(T)$ and the packed representation of P, in $\mathcal{O}(1)$ time we can compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.*

Proof. Using L_{range} on P, we immediately obtain and return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(1)$ time. □

6.1.4 Construction Algorithm

Proposition 6.3. *Given the packed representation of $T \in [0\,..\,\sigma)^n$, we can construct $\mathrm{C}_{\mathrm{PM}}(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time.*

Proof. Since $\mathrm{C}_{\mathrm{PM}}(T)$ contains a subset of components of $\mathrm{C}_{\mathrm{SA}}(T)$, this follows by Proposition 5.3. □

6.2 The Nonperiodic Patterns

In this section, we describe a data structure that, given a packed representation of any nonperiodic pattern $P \in [0\,..\,\sigma)^m$ (see Definition 6.1), computes $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 6.2.1). Next, we describe the query algorithm (Section 6.2.2). Finally, we show the construction algorithm (Section 6.2.3).

6.2.1 The Data Structure

Definitions Let S be a τ-synchronizing set, as defined in Section 5.2.1. Let $A_\mathsf{S}[1\,..\,n']$ be an array defined by $A_\mathsf{S}[i] = s_i^{\mathrm{lex}}$ (where $(s_t^{\mathrm{lex}})_{t\in[1\,..\,n']}$ is a sequence as defined in Section 5.2.1).

Components The data structure to handle nonperiodic patterns consists of three components:

1. The index core $\mathrm{C}_{\mathrm{PM}}(T)$ (Section 6.1.1) using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The data structure from Section 5.2.1 using $\mathcal{O}(n/\log_\sigma n)$ space.
3. The data structure from Proposition 4.2 for the array $A_\mathsf{S}[1\,..\,n']$. By $n' = \mathcal{O}(n/\log_\sigma n)$ and Proposition 4.2, it needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

6.2.2 Implementation of Queries

LEMMA 6.1. *Let $P \in [0\,..\,\sigma)^m$ be a nonperiodic pattern satisfying $m \geq 3\tau - 1$, and let $X \in \mathcal{D}$ be a prefix of P. Denote $\delta_{\mathrm{text}} = |X| - 2\tau$ and $P' = P(\delta_{\mathrm{text}}\,..\,m]$. Let $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ be such that $b_{\mathrm{pre}} = |\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}}\,..\,n] \prec P'\}|$ and $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,n'] : P'$ is a prefix of $T[s_i^{\mathrm{lex}}\,..\,n]\}$. Then, it holds*

$$(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (b_X + \delta_1, b_X + \delta_2),$$

where $b_X = \mathrm{RangeBeg}(X,T)$, $\delta_1 = \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}})$, and $\delta_2 = \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}})$.

Proof. Observe that by the consistency of S and $X \in \mathcal{D}$, $j \in \mathrm{Occ}(X,T)$ implies $j + \delta_{\mathrm{text}} \in \mathsf{S}$. Thus, $\mathrm{Occ}(X,T) = \{s - \delta_{\mathrm{text}} : s \in \mathsf{S} \text{ and } s - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T)\}$. Note also that if S_1 is a prefix of S_2 then $\mathrm{RangeBeg}(S_2,T) = \mathrm{RangeBeg}(S_1,T) + |\{j \in \mathrm{Occ}(S_1,T) : T[j\,..\,n] \prec S_2\}|$. Together with the definition of b_{pre}, this implies

$$\begin{aligned}
\mathrm{RangeBeg}(P,T) &= \mathrm{RangeBeg}(X,T) + |\{j \in \mathrm{Occ}(X,T) : T[j\,..\,n] \prec P\}| \\
&= b_X + |\{s - \delta_{\mathrm{text}} : s \in \mathsf{S}, s - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T), \text{ and } T[s - \delta_{\mathrm{text}}\,..\,n] \prec P\}| \\
&= b_X + |\{s \in \mathsf{S} : s - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T) \text{ and } T[s - \delta_{\mathrm{text}}\,..\,n] \prec P\}| \\
&= b_X + |\{s \in \mathsf{S} : s - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T) \text{ and } T[s\,..\,n] \prec P'\}| \\
&= b_X + |\{i \in [1\,..\,n'] : s_i^{\mathrm{lex}} - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T) \text{ and } T[s_i^{\mathrm{lex}}\,..\,n] \prec P'\}| \\
&= b_X + |\{i \in [1\,..\,n'] : s_i^{\mathrm{lex}} - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T) \text{ and } i \leq b_{\mathrm{pre}}\}| \\
&= b_X + |\{i \in [1\,..\,b_{\mathrm{pre}}] : s_i^{\mathrm{lex}} - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T)\}| \\
&= b_X + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}) \\
&= b_X + \delta_1,
\end{aligned}$$

where the second-to-last equality follows by $W[i] = \overline{X_i}$, where $X_i = T^\infty[s_i^{\mathrm{lex}} - \tau\,..\,s_i^{\mathrm{lex}} + 2\tau)$, since $s_i^{\mathrm{lex}} - \delta_{\mathrm{text}} \in \mathrm{Occ}(X,T)$ holds if and only if X if a suffix of X_i (i.e., if $\overline{X}$ is a prefix of $\overline{X_i}$).

Next, we show that $|\mathrm{Occ}(P,T)| = \delta_2 - \delta_1$. We start by observing that (similarly as above, except applied to P) by the consistency of S and $X \in \mathcal{D}$ being a prefix of P, $j \in \mathrm{Occ}(P,T)$ implies $j + \delta_{\mathrm{text}} \in \mathsf{S}$. Thus, $\mathrm{Occ}(P,T) = \{s - \delta_{\mathrm{text}} : s \in \mathsf{S} \text{ and } s - \delta_{\mathrm{text}} \in \mathrm{Occ}(P,T)\}$ and hence,

$$|\mathrm{Occ}(P,T)| = |\{s - \delta_{\mathrm{text}} : s \in \mathsf{S},\ s - \delta_{\mathrm{text}} \in \mathrm{Occ}(P,T)\}|$$

$$
\begin{aligned}
&= |\{s \in \mathsf{S} : s - \delta_{\text{text}} \in \mathrm{Occ}(P,T)\}| \\
&= |\{i \in [1\,..\,n'] : s^{\text{lex}}_i - \delta_{\text{text}} \in \mathrm{Occ}(P,T)\}| \\
&= |\{i \in [1\,..\,n'] : s^{\text{lex}}_i - \delta_{\text{text}} \in \mathrm{Occ}(X,T) \text{ and } s^{\text{lex}}_i \in \mathrm{Occ}(P',T)\}| \\
&= |\{i \in [1\,..\,n'] : s^{\text{lex}}_i - \delta_{\text{text}} \in \mathrm{Occ}(X,T) \text{ and } b_{\text{pre}} < i \leq e_{\text{pre}}\}| \\
&= |\{i \in (b_{\text{pre}}\,..\,e_{\text{pre}}] : s^{\text{lex}}_i - \delta_{\text{text}} \in \mathrm{Occ}(X,T)\}| \\
&= \mathsf{rank}_{W,\overline{X}}(b_{\text{pre}}) - \mathsf{rank}_{W,\overline{X}}(e_{\text{pre}}) \\
&= \delta_1 - \delta_2.
\end{aligned}
$$

Combining the above with the earlier equality, we obtain $\mathrm{RangeEnd}(P,T) = \mathrm{RangeBeg}(P,T) + |\mathrm{Occ}(P,T)| = b_X + \delta_2$, i.e., the second part of the claim. □

REMARK 6.1. Note that since the range $(b_{\text{pre}}\,..\,e_{\text{pre}}]$ is well-defined even if $e_{\text{pre}} - b_{\text{pre}} = 0$, the above lemma holds even if $|\mathrm{Occ}(P,T)| = 0$.

PROPOSITION 6.4. *Let $P \in [0\,..\,\sigma)^m$ be a nonperiodic pattern satisfying $m \geq 3\tau - 1$. Given the data structure from Section 6.2.1 and the packed representation of P, in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time we can compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.*

Proof. Let $Y = P[1\,..\,3\tau-1]$. First, using the lookup table L_{range} on Y, in $\mathcal{O}(1)$ time we compute $(b_Y, e_Y) = (\mathrm{RangeBeg}(Y,T), \mathrm{RangeEnd}(Y,T))$. If $e_Y - b_Y = 0$, then $\mathrm{Occ}(Y,T) = \emptyset$, and it is easy to see that then we have $\mathrm{RangeBeg}(P,T) = \mathrm{RangeBeg}(Y,T)$ and $\mathrm{RangeEnd}(P,T) = \mathrm{RangeEnd}(Y,T)$. We thus return (b_Y, e_Y). Let us thus assume $b_Y \neq e_Y$, i.e., $\mathrm{Occ}(Y,T) \neq \emptyset$. Together with $\mathrm{per}(Y) > \frac{1}{3}\tau$, this implies (see Section 5.2.1) that there exists a unique prefix $X \in \mathcal{D}$ of P. Using $L_{\mathcal{D}}$ on Y in $\mathcal{O}(1)$ time we compute the prefix $X \in \mathcal{D}$ of P. Let $\delta = |X| - 2\tau$. Using again the lookup table L_{range}, in $\mathcal{O}(1)$ time we compute $(b_X, e_X) = (\mathrm{RangeBeg}(X,T), \mathrm{RangeEnd}(X,T))$. Using Proposition 4.2, we then compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time the pair $(b_{\text{pre}}, e_{\text{pre}})$ for the pattern $P' := P(\delta\,..\,m]$. By Lemma 6.1, we then return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (b_X + \mathsf{rank}_{W,\overline{X}}(b_{\text{pre}}), b_X + \mathsf{rank}_{W,\overline{X}}(e_{\text{pre}}))$, with the two prefix rank queries implemented using Theorem 2.2, in $\mathcal{O}(\log^\epsilon n)$ time each (the string $\overline{X}$ is obtained using the lookup table L_{rev}). Altogether, the query time is $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$. □

6.2.3 Construction Algorithm

PROPOSITION 6.5. *Given $\mathrm{C}_{\mathrm{PM}}(T)$, we can in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and in $\mathcal{O}(n/\log_\sigma n)$ working space augment it into a data structure from Section 6.2.1.*

Proof. First, we combine Propositions 5.3 and 5.6 (recall that the packed representation of T is a component of $\mathrm{C}_{\mathrm{PM}}(T)$) to construct the structure from Section 5.2.1 in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and using $\mathcal{O}(n/\log_\sigma n)$ working space. In particular, this constructs $(s^{\text{lex}}_i)_{i \in [1\,..\,n']}$. We thus initialize $A_{\mathsf{S}}[i] = s^{\text{lex}}_i$ for $i \in [1\,..\,n']$ and in $\mathcal{O}(n/\log_\sigma n)$ time and $\mathcal{O}(n/\log_\sigma n)$ working space construct the data structure from Proposition 4.2. The overall runtime is $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$. The working space never exceed $\mathcal{O}(n/\log_\sigma n)$ words. □

6.3 The Periodic Patterns

In this section, we describe a data structure that, given a packed representation of any periodic pattern $P \in [0\,..\,\sigma)^m$ (see Definition 6.1), computes $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.

The section is organized as follows. First, we present the toolbox of combinatorial properties for periodic patterns (Section 6.3.1). Next, we introduce the components of the data structure (Section 6.3.2). We then show how using this structure to implement some basic navigational routines (Section 6.3.3). Next, we describe the query algorithm (Section 6.3.4). Finally, we show the construction algorithm (Section 6.3.5).

6.3.1 Preliminaries

Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern (see Definition 6.1). We define $\text{L-root}(P) = \min\{P[1+t\,..\,1+t+p) : t \in [0\,..\,p)\}$, where $p = \mathrm{per}(P[1\,..\,3\tau-1])$. Let $H = \text{L-root}(P)$. We define $e(P) = 1 + p + \mathrm{lcp}(P[1\,..\,m], P[1+p\,..\,m])$, where $p = |H|$. By definition, there exists $s \in [0\,..\,p)$ such that $P[1+s\,..\,1+s+p) = H$. Thus, we can write

$P[1\,..\,e(P)) = H'H^kH''$, where H' (resp. H'') is a proper suffix (resp. prefix) of H. By $e(P) \geq 3\tau$ and $|H| \leq \tau$, such decomposition is unique (see also Section 5.3.1). We denote L-head$(P) = |H'|$, L-exp$(P) = k$, and L-tail$(P) = |H''|$. We also let $e^{\text{full}}(P) = e(P) - \text{L-tail}(P)$. We define type$(P) = +1$ if $e(P) \leq m$ and $P[e(P)] \succ P[e(P) - p]$ (where $p = |\text{L-root}(P)|$), and type$(P) = -1$ otherwise.

LEMMA 6.2. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern and let $s = \text{L-head}(P)$ and $H = \text{L-root}(P)$. For any $j \in [1\,..\,n]$, $\text{lcp}(P, T[j\,..\,n]) \geq 3\tau - 1$ holds if and only if $j \in \mathsf{R}_{s,H}$. Moreover, if $j \in \mathsf{R}_{s,H}$ then, letting $t = e(P) - 1$ and $t' = e(j) - j$, it holds $\text{lcp}(P, T[j\,..\,n]) \geq \min(t, t')$ and:*

1. *If $\text{type}(P) \neq \text{type}(j)$, then $P \prec T[j\,..\,n]$ if and only if $\text{type}(P) < \text{type}(j)$,*
2. *If $\text{type}(P) = \text{type}(j) = -1$ and $t \neq t'$, then $P \prec T[j\,..\,n]$ if and only if $t < t'$,*
3. *If $\text{type}(P) = \text{type}(j) = +1$ and $t \neq t'$, then $P \prec T[j\,..\,n]$ if and only if $t > t'$,*
4. *If $\text{type}(P) \neq \text{type}(j)$ or $t \neq t'$, then $P \neq T[j\,..\,n]$ and $\text{lcp}(P, T[j\,..\,n]) = \min(t, t')$.*

Proof. Let $j \in [1\,..\,n]$ be such that $\text{lcp}(P, T[j\,..\,n]) \geq 3\tau - 1$. Denoting $p = \text{per}(P[1\,..\,3\tau - 1])$ and $p' = \text{per}(T[j\,..\,j + 3\tau - 1))$ we then have $p' = p \leq \frac{1}{3}\tau$. Thus, $j \in \mathsf{R}$. Moreover, this implies L-root$(j) = \min\{T[j + \delta\,..\,j + \delta + p') : \delta \in [0\,..\,p')\} = \min\{P[1 + \delta\,..\,1 + \delta + p) : \delta \in [0\,..\,p)\} = H$. To show that L-head$(j) = s$, note that by $|H| \leq \tau$, the string $H'H^2$ (where H' is a length-s suffix of H) is a prefix of $P[1\,..\,3\tau - 1] = T[j\,..\,j + 3\tau - 1)$. On the other hand, L-head$(j) = s'$ implies that $\widehat{H}'H^2$ (where $\widehat{H}'$ is a length-s' suffix of H) is a prefix of $T[j\,..\,j + 3\tau - 1)$. Thus, by the synchronization property of primitive strings [24, Lemma 1.11] applied to the two copies of H, we have $s' = s$, and hence, $j \in \mathsf{R}_{s,H}$. For the converse implication, assume $j \in \mathsf{R}_{s,H}$. This implies that both $P[1\,..\,e(P))$ and $T[j\,..\,e(j))$ are prefixes of $H' \cdot H^\infty[1\,..)$ (where H' is as above). Thus, by $e(P) - 1$, $e(j) - j \geq 3\tau - 1$, we obtain $\text{lcp}(P, T[j\,..\,n]) \geq 3\tau - 1$.

Let us now assume $j \in \mathsf{R}_{s,H}$. Since, as noted above, both $P[1\,..\,e(P)) = P[1\,..\,t]$ and $T[j\,..\,e(j)) = T[j\,..\,j + t')$ are prefixes of $H' \cdot H^\infty[1\,..)$, we have $\text{lcp}(P, T[j\,..\,n]) \geq \min(t, t')$.

1. Let $Q = H' \cdot H^\infty[1\,..)$, where H' is a length-s suffix of H. In the proof of Lemma 5.4, it is shown that type$(j) = -1$ implies $T[j\,..\,n] \prec Q$, and type$(j) = +1$ implies $Q \prec T[j\,..\,n]$. We now prove an analogous fact for P. We first note that type$(P) = -1$ implies that either $e(P) = m+1$, or $e(P) \leq m$ and $P[e(P)] \prec P[e(P) - |H|]$. In the first case, $P[1\,..\,e(P)) = P$ is a proper prefix of Q and hence $P \prec Q$. In the second case, we have $P[1\,..\,t] = Q[1\,..\,t]$ and $P[1 + t] \prec P[1 + t - |H|] = Q[1 + t - |H|] = Q[1 + t]$. Consequently, $P \prec Q$. If type$(P) = +1$ holds, then $e(P) \leq m$. Thus, we have $Q[1\,..\,t] = P[1\,..\,t]$ and $Q[1 + t] = Q[1 + t - |H|] = P[1 + t - |H|] \prec P[1 + t]$. Hence, we obtain $Q \prec P$. We are now ready to prove the claim. Assume first that type$(P) <$ type(j). By the above we then have $P \prec Q \prec T[j\,..\,n]$. The opposite implication is proved by contraposition. Assume type$(P) >$ type(j). By the above we then have $T[j\,..\,n] \prec Q \prec P$.

2. Assume $t < t'$. If $e(P) = m+1$, then $P[1\,..\,t] = P[1\,..\,e(P)) = P$ is proper prefix of $T[j\,..\,j+t') = T[j\,..\,e(j))$, and hence $P \prec T[j\,..\,e(j)) \preceq T[j\,..\,n]$. If $e(P) \leq m$, then we have $P[1\,..\,t] = T[j\,..\,j + t)$ and by $t < t'$, $P[1 + t] \prec P[1 + t - |H|] = T[j + t - |H|] = T[j + t]$. Thus, we also obtain $P \prec T[j\,..\,n]$. The opposite implication is proved by contraposition. Assume $t > t'$. If $e(j) = n + 1$, then by $t > t'$, the string $T[j\,..\,j + t') = T[j\,..\,e(j)) = T[j\,..\,n]$ is a proper prefix of $P[1\,..\,t] = P[1\,..\,e(P))$, and hence $T[j\,..\,n] \prec P[1\,..\,e(P)) \preceq P$. If $e(j) \leq n$, then we have $T[j\,..\,j + t') = P[1\,..\,t']$ and by $t > t'$, $T[j + t'] \prec T[j + t' - |H|] = P[1 + t' - |H|] = P[1 + t']$. Consequently, we also obtain $T[j\,..\,n] \prec P$.

3. Assume $t > t'$. By type$(j) = +1$, we have $e(j) \leq n$. Thus, by $t > t'$ we have $P[1\,..\,t'] = T[j\,..\,j + t')$ and $P[1 + t'] = P[1 + t' - |H|] = T[j + t' - |H|] \prec T[j + t']$. Consequently, $P \prec T[j\,..\,n]$. The opposite implication is proved by contraposition. Assume $t < t'$. By type$(P) = +1$, we have $e(P) \leq m$. Thus, by $t < t'$ we have $T[j\,..\,j + t) = P[1\,..\,t]$ and $T[j + t] = T[j + t - |H|] = P[1 + t - |H|] \prec P[1 + t]$. Consequently, we obtain $T[j\,..\,n] \prec P$.

4. By the earlier implication, $\text{lcp}(P, T[j\,..\,n]) \geq \min(t, t')$. Thus, it remains to show that $P \neq T[j\,..\,n]$ and $\text{lcp}(P, T[j\,..\,n]) \leq \min(t, t')$. First, let us assume type$(P) <$ type(j) (i.e., type$(P) = -1$ and type$(j) = +1$). Consider two cases:

- First, assume $t \leq t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\text{lcp}(P, T[j\,..\,n]) \leq t$. First, recall from the proof of Lemma 5.4(1) that type$(j) = +1$ implies $j + t' \leq n$, $Q[1\,..\,t'] = T[j\,..\,j + t')$, and $Q[1 + t'] \prec T[j + t']$. Consider now two subcases. If $e(P) = m + 1$, then $t = m$, and hence $\text{lcp}(P, T[j\,..\,n]) \leq m = t$. By $t' \leq n - j$ we then also have $|P| = t < t' + 1 \leq n - j + 1 = |T[j\,..\,n]|$. Thus, $P \neq T[j\,..\,n]$. Let us thus assume $e(P) \leq m$. In the proof of Item 1 we showed that in this case type$(P) = -1$ implies $P[1 + t] \prec Q[1 + t]$. On the other

hand, as noted above, $\mathrm{type}(j) = +1$ implies $Q[1\,..\,t'] = T[j\,..\,j+t')$, and $Q[1+t'] \prec T[j+t']$. By $t \le t'$ we thus have $Q[1+t] \preceq T[j+t]$. Consequently, $P[1+t] \neq T[j+t]$. This immediately implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$.

- Let us now assume $t > t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$. In the proof of Item 1 we showed that $\mathrm{type}(P) = -1$ implies $P[1\,..\,t] = Q[1\,..\,t]$. Thus, by $t > t'$ we have $P[1+t'] = Q[1+t']$. On the other hand, in the proof of Lemma 5.4(1) we showed that $\mathrm{type}(j) = +1$ implies $Q[1+t'] \prec T[j+t']$. Thus, we obtain $P[1+t'] \neq T[j+t']$. This immediately implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$.

Assume now $\mathrm{type}(P) > \mathrm{type}(j)$ (i.e., $\mathrm{type}(P) = +1$ and $\mathrm{type}(j) = -1$). Consider two cases:

- First, assume $t < t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$. In the proof of Lemma 5.4(1) we showed that $\mathrm{type}(j) = -1$ implies $T[j\,..\,j+t') = Q[1\,..\,t']$. Thus, by $t < t'$ we have $T[j+t] = Q[1+t]$. On the other hand, in the proof of Item 1 we showed that $\mathrm{type}(P) = +1$ implies $Q[1+t] \prec P[1+t]$. Thus, we obtain $T[j+t] \neq P[1+t]$. This immediately implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$.
- Let us now assume $t \ge t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$. First, recall from the proof of Item 1 that $\mathrm{type}(P) = +1$ implies $t+1 = e(P) \le m$, $Q[1\,..\,t] = P[1\,..\,t]$, and $Q[1+t] \prec P[1+t]$. Consider now two subcases. If $e(j) = n+1$, then $j + t' = n+1$ (or equivalently, $t' = n-j+1$) and hence $\mathrm{lcp}(P, T[j\,..\,n]) \le |T[j\,..\,n]| = t'$. By $t+1 \le m$ we then also have $|T[j\,..\,n]| = t' < t+1 \le m = |P|$. Thus, $P \neq T[j\,..\,n]$. Let us thus assume $e(j) \le n$. In the proof of Lemma 5.4(1) we showed that in this case $\mathrm{type}(j) = -1$ implies $T[j+t'] \prec Q[1+t']$. On the other hand, as noted above, $\mathrm{type}(P) = +1$ implies $Q[1\,..\,t] = P[1\,..\,t]$ and $Q[1+t] \prec P[1+t]$. By $t \ge t'$ we thus have $Q[1+t'] \preceq P[1+t']$. Consequently, $T[j+t'] \neq P[1+t']$. This immediately implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$.

This concludes the proof of the claim if $\mathrm{type}(P) \neq \mathrm{type}(j)$. Let us now assume $\mathrm{type}(P) = \mathrm{type}(j) = -1$ and $t \neq t'$. Consider two cases:

- First, assume $t < t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$. In the proof of Item 2 we showed that either it holds $P[1\,..\,t] = P$ (in which case $\mathrm{lcp}(P, T[j\,..\,n]) \le |P| = t$ and $|P| = t < t' = e(j) - j \le n+1-j = |T[j\,..\,n]|$ which in turn implies $P \neq T[j\,..\,n]$), or $P[1+t] \prec T[j+t]$ (which also implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$).
- Let us now assume $t > t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$. In the proof of Item 2, we showed that either it holds $T[j\,..\,j+t') = T[j\,..\,n]$ (in which case $\mathrm{lcp}(P, T[j\,..\,n]) \le n-j+1 = t'$ and $|T[j\,..\,n]| = n-j+1 = t' < t = e(P) - 1 \le |P|$ which in turn implies $P \neq T[j\,..\,n]$), or $T[j+t'] \prec P[1+t']$ (which also implies $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$).

Let us now assume $\mathrm{type}(P) = \mathrm{type}(j) = +1$ and $t \neq t'$. Consider two cases:

- First, assume $t < t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t$. In the proof of Item 3 we showed that $T[j+t] \prec P[1+t]$. This immediately implies the claims.
- Let us now assume $t > t'$. Our goal is to prove $P \neq T[j\,..\,n]$ and $\mathrm{lcp}(P, T[j\,..\,n]) \le t'$. In the proof of Item 3 we showed that $P[1+t'] \prec T[j+t']$. This immediately implies the claims. □

LEMMA 6.3. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. For every $S \in [0\,..\,\sigma)^+$, $\mathrm{lcp}(P, S) \ge 3\tau - 1$ implies that S is periodic, and that it holds* $\text{L-root}(S) = \text{L-root}(P)$ *and* $\text{L-head}(S) = \text{L-head}(P)$.

Proof. Denote $X = P[1\,..\,3\tau-1]$. Letting $p := \mathrm{per}(X)$ we then have $p \le \frac{1}{3}\tau$. By $\mathrm{lcp}(P, S) \ge 3\tau - 1$, X is thus a prefix of S and hence $\mathrm{per}(S[1\,..\,3\tau-1]) = p \le \frac{1}{3}\tau$, i.e., S is periodic. Moreover, we then have $\text{L-root}(S) = \min\{S[1+t\,..\,1+t+p) : t \in [0\,..\,p)\} = \min\{X[1+t\,..\,1+t+p) : t \in [0\,..\,p)\} = \min\{P[1+t\,..\,1+t+p) : t \in [0\,..\,p)\} = \text{L-root}(P)$. To show the last claim, denote $s = \text{L-head}(P)$ and $s' = \text{L-head}(S)$. Then, letting $H = \text{L-root}(P) = \text{L-root}(S)$, the string $H'H^2$ (resp. $\widehat{H}'H^2$) is a prefix of P (resp. S), where H' (resp. $\widehat{H}'$) is a length-s (resp. length-s') suffix of H. Note, however, that $s, s' < |H| = p \le \frac{1}{3}\tau$ and $|X| \ge \tau \ge 3|H|$. This implies that $H'H^2$ and $\widehat{H}'H^2$ are both prefixes of X. By the synchronization property of primitive strings [24, Lemma 1.11], this implies $|H'| = |\widehat{H}'|$. Thus, we obtain $\text{L-head}(P) = s = |H'| = |\widehat{H}'| = s' = \text{L-head}(S)$. □

LEMMA 6.4. *Let $P \in [0\,..\,\sigma)^+$ be a periodic pattern satisfying $e(P) \le |P|$. Then:*

1. *For every $S \in [0\,..\,\sigma)^+$, $\mathrm{lcp}(P, S) \ge e(P)$ (in particular, P being a prefix of S) implies that S is periodic and it holds:*

- $e(S) = e(P)$,
- L-tail(S) = L-tail(P),
- $e^{\text{full}}(S) = e^{\text{full}}(P)$,
- L-exp(S) = L-exp(P),
- type(S) = type(P).

2. *If* $j \in \text{Occ}(P,T)$, *then* $j \in \mathsf{R}$ *and it holds:*

- $e(j) - j = e(P) - 1$,
- L-tail(j) = L-tail(P),
- $e^{\text{full}}(j) - j = e^{\text{full}}(P) - 1$,
- L-exp(j) = L-exp(P),
- type(j) = type(P).

Proof. Denote $X = P[1\,..\,3\tau-1]$, $H = \text{L-root}(P)$, $s = \text{L-head}(P)$, and $p = \text{per}(X) = |H| \le \frac{1}{3}\tau$.

1. First, observe that by definition, $e(P) = 1+p+\text{lcp}(P, P[1+p\,..\,|P|]) > |X|$. Thus, $\text{lcp}(P,S) \ge e(P)$ implies that X is a prefix of S, and hence S is periodic. By Lemma 6.3, we then also have L-root$(S) = H$ and L-head$(S) = s$. To show $e(S) = e(P)$, observe that by $e(P) \le |P|$ and the definition of $e(P)$, we have $P[e(P)] \ne P[e(P)-p]$. Consequently, $\text{lcp}(P,S) \ge e(P)$ yields $\text{lcp}(P, P[1+p\,..\,|P|]) = \text{lcp}(S, S[1+p\,..\,|S|])$. Combining this with $|\text{L-root}(S)| = p$ we thus obtain $e(S) = 1+p+\text{lcp}(S, S[1+p\,..\,|S|]) = 1+p+\text{lcp}(P, P[1+p\,..\,|P|]) = e(P)$. By L-head$(S) = s$ we then also obtain L-tail$(S) = (e(S)-1-\text{L-head}(S)) \bmod |\text{L-root}(S)| = (e(P)-1-s) \bmod |H| = \text{L-tail}(P)$, and consequently $e^{\text{full}}(S) = e(S) - \text{L-tail}(S) = e(P) - \text{L-tail}(P) = e^{\text{full}}(P)$. We then also have $\text{L-exp}(S) = \lfloor \frac{e(S)-1-\text{L-head}(S)}{|\text{L-root}(S)|} \rfloor = \lfloor \frac{e(P)-1-s}{|H|} \rfloor = \text{L-exp}(P)$. Finally, by $e(P) \le |P|$ and $\text{lcp}(P,S) \ge e(P)$, we then also have $S[e(S)] = P[e(P)]$. Consequently, $S[e(S)] \prec S[e(S)-p]$ holds if and only of $P[e(P)] \prec P[e(P)-p]$. Therefore, type(S) = type(P).

2. We start by noting that $j \in \text{Occ}(P,T)$ implies $j \in \text{Occ}(X,T)$. Thus, $\text{per}(T[j\,..\,j+3\tau-1)) = \text{per}(X) = p \le \frac{1}{3}\tau$, and hence $j \in \mathsf{R}$. By Lemma 6.2, we then also have L-root$(j) = H$ and L-head$(j) = s$. To show $e(j) - j = e(P) - 1$, denote $S = T[j\,..\,n]$. Since P is a prefix of S, by Item 1, it follows that $e(S) = e(P)$. By L-root(S) = L-root(P) (Lemma 6.3) and the definition of $e(S)$, we thus have $1+p+\text{lcp}(S, S[1+p\,..\,|S|]) = e(S) = e(P)$, or equivalently, $\text{lcp}(S, S[1+p\,..\,|S|]) = e(P) - p - 1$. Since $\text{lcp}(S, S[1+p\,..\,|S|]) = \text{LCE}(j, j+p)$, we thus obtain $p + \text{LCE}(j, j+p) = e(P) - 1$. It remains to note that for $j \in \mathsf{R}$, by Lemma 5.3(2), $e(j) - j = p + \text{LCE}(j, j+p)$. Therefore, we have $e(j) - j = e(P) - 1$. Combining this with L-root$(j) = H$ and L-head$(j) = s$ yields L-tail$(j) = (e(j) - j - \text{L-head}(j)) \bmod |\text{L-root}(j)| = (e(P) - 1 - s) \bmod |H| = \text{L-tail}(P)$, $e^{\text{full}}(j) - j = e(j) - j - \text{L-tail}(j) = e(P) - 1 - \text{L-tail}(P) = e^{\text{full}}(P) - 1$, and $\text{L-exp}(j) = \lfloor \frac{e(j)-j-s}{|\text{L-root}(j)|} \rfloor = \lfloor \frac{e(P)-1-s}{|H|} \rfloor = \text{L-exp}(P)$. Finally, by $e(P) \le |P|$ we have $e(j) \le n$ and $T[e(j)] = P[e(P)]$. Consequently, $T[e(j)] \prec T[e(j)-p]$ holds if and only if $P[e(P)] \prec P[e(P)-p]$. Therefore, type(j) = type(P). □

6.3.2 The Data Structure

Definitions Let $q = |\mathsf{R}'^{-}|$. Recall (Section 5.3.2), that $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$ denotes the sequence containing all positions $j \in \mathsf{R}'^{-}$ sorted first by L-root(j), and in case of ties, by $T[e^{\text{full}}(j)\,..\,n]$. Recall also that Roots = $\{\text{L-root}(j) : j \in \mathsf{R}\}$. For any string $H \in$ Roots, let $\text{pow}(H) = H^{\infty}[1\,..\,|H|\lceil \frac{\tau}{|H|} \rceil]$. This function satisfies the following properties:

- The set $\{\text{pow}(H) : H \in \text{Roots}\}$ is prefix-free.
- For any $X, Y \in$ Roots, $X \prec Y$ implies $\text{pow}(X) \prec \text{pow}(Y)$.

For a proof, consider $X, Y \in$ Roots such that $X \prec Y$. By [54, Fact 9.1.6], it holds $X \preceq \text{pow}(X) \prec X^{\infty}[1\,..) \prec Y \preceq \text{pow}(Y)$. Since $|Y| < \tau \le |\text{pow}(X)|$, the set $\{\text{pow}(X), \text{pow}(Y)\}$ is prefix-free.

We define $\mathsf{Z} = \{e^{\text{full}}(j) - |\text{pow}(\text{L-root}(j))| : j \in \mathsf{R}'^{-}\}$. We also define an array $A_{\mathsf{Z}}[1\,..\,q]$ so that, for any $i \in [1\,..\,q]$, $A_{\mathsf{Z}}[i] = e^{\text{full}}(j) - |\text{pow}(H_i)|$, where $j = r_i^{\text{lex}}$ and $H_i = \text{L-root}(r_i^{\text{lex}})$. Note that $\{A_{\mathsf{Z}}[i] : i \in [1\,..\,q]\} = \mathsf{Z}$. Observe also that $T[A_{\mathsf{Z}}[i]\,..\,n] = \text{pow}(H_i) \cdot T[e^{\text{full}}(j)\,..\,n]$. Together with the properties of the pow function and with the definition of $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$, this implies that the positions in A_{Z} are sorted according to the lexicographic order of the corresponding suffixes of T, i.e., $i < i'$ implies $T[A_{\mathsf{Z}}[i]\,..\,n] \prec T[A_{\mathsf{Z}}[i']\,..\,n]$.

Components The data structure to handle periodic patterns consists of two parts. The first part (designed to handle periodic patterns P satisfying type$(P) = -1$) consists of three components:

1. The index core $\mathrm{C}_{\mathrm{PM}}(T)$ (Section 6.1.1) using $\mathcal{O}(n/\log_\sigma n)$ space.
2. The first part of the structure from Section 5.3.2 using $\mathcal{O}(n/\log_\sigma n)$ space.
3. The data structure from Proposition 4.2 for the array $A_{\mathsf{Z}}[1\,..\,q]$. By $q = \mathcal{O}(n/\log_\sigma n)$ and Proposition 4.2, it needs $\mathcal{O}(n/\log_\sigma n)$ space.

The second part of the structure (to handle P satisfying $\mathrm{type}(P) = +1$) consists of the symmetric counterparts of the above components adapted according to Lemma 6.2.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

6.3.3 Navigation Primitives

PROPOSITION 6.6. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Given the data structure from Section 6.3.2 and the packed representation of P, we can in $\mathcal{O}(1 + m/\log_\sigma n)$ time compute* L-root(P), L-head(P), L-exp(P), L-tail(P), *and* type(P).

Proof. We first compute $x \in [0\,..\,\sigma^{6\tau})$ such that $x = \mathrm{int}(P[1\,..\,3\tau{-}1])$. Given the packed encoding of P, such x is obtained in $\mathcal{O}(1)$ time. We then look up $(s,p) = L_{\mathrm{root}}[x]$, and in $\mathcal{O}(1)$ time obtain L-root$(P) = P[1{+}s\,..\,1{+}s{+}p)$ and L-head$(P) = s$. Next, we compute L-exp(P) and L-tail(P). For this, we first determine the length ℓ of the longest common prefix of P and $P(p\,..\,m]$. Using the packed representation of P, we can do this in $\mathcal{O}(1{+}m/\log_\sigma n)$ time (see, e.g., [50, Proposition 2.3]). Consequently, we obtain $e(P) = 1 + p + \ell$, L-exp$(P) = \lfloor \frac{e(P)-1-s}{p} \rfloor$, and L-tail$(P) = (e(P) - 1 - s) \bmod p$. Finally, to test if $\mathrm{type}(P) = +1$, we check whether $e(P) \le m$, and if so, whether $P[e(P)] \succ P[e(P) - p]$. □

6.3.4 Implementation of Queries

Overview The query algorithm is derived in two steps. First, we establish how, given the structure from Section 6.3.2 and a packed representation of any periodic pattern $P \in [0\,..\,\sigma)^m$ to compute $|\mathrm{Occ}(P,T)|$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time. This culminates in Proposition 6.9. We then show how to extend this algorithm to instead return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in the same time complexity, culminating in Proposition 6.12. The reason for this two-step approach is explained in Remark 6.2.

Computing $|\mathrm{Occ}(P,T)|$ Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Denote $s =$ L-head(P) and $H =$ L-root(P). We define $\mathrm{Occ}^{\mathsf{a}}(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) > \text{L-exp}(P)\}$ and $\mathrm{Occ}^{\mathsf{s}}(P,T) = \{j \in \mathsf{R}_{s,H} \cap \mathrm{Occ}(P,T) : \text{L-exp}(j) = \text{L-exp}(P)\}$.

LEMMA 6.5. *For any periodic pattern $P \in [0\,..\,\sigma)^m$, the set* $\mathrm{Occ}(P,T)$ *is a disjoint union of* $\mathrm{Occ}^{\mathsf{a}}(P,T)$ *and* $\mathrm{Occ}^{\mathsf{s}}(P,T)$.

Proof. By definition, $\mathrm{Occ}^{\mathsf{a}}(P,T) \cap \mathrm{Occ}^{\mathsf{s}}(P,T) = \emptyset$ and $\mathrm{Occ}^{\mathsf{a}}(P,T) \cup \mathrm{Occ}^{\mathsf{s}}(P,T) \subseteq \mathrm{Occ}(P,T)$. Thus, it suffices to show $\mathrm{Occ}(P,T) \subseteq \mathrm{Occ}^{\mathsf{a}}(P,T) \cup \mathrm{Occ}^{\mathsf{s}}(P,T)$. Assume $j \in \mathrm{Occ}(P,T)$. By $m \ge 3\tau - 1$, this implies $\mathrm{lcp}(T[j\,..\,n], P) \ge 3\tau - 1$. Thus, by Lemma 6.2, it holds $j \in \mathsf{R}_{s,H}$, where $s =$ L-head(P) and $H =$ L-root(P). To obtain $j \in \mathrm{Occ}^{\mathsf{a}}(P,T) \cup \mathrm{Occ}^{\mathsf{s}}(P,T)$ it remains to show L-exp$(j) \ge$ L-exp(P). First, note that for any $t \in [1\,..\,m)$, $j \in \mathrm{Occ}(P,T)$ implies $\mathrm{LCE}(j, j+t) \ge \mathrm{lcp}(P[1\,..\,m], P[1+t\,..\,m])$. In particular, letting $p = |H|$, by definition of $e(P)$ and Lemma 5.3(2), we have $e(j) - j = p + \mathrm{LCE}(j, j+p) \ge p + \mathrm{lcp}(P[1\,..\,m], P[1+p\,..\,m]) = e(P) - 1$. Consequently, L-exp$(j) = \lfloor \frac{e(j)-j-s}{p} \rfloor \ge \lfloor \frac{e(P)-1-s}{p} \rfloor =$ L-exp(P). □

By the above lemma, if $P \in [0\,..\,\sigma)^m$ is periodic, then $\mathrm{Occ}(P,T) \subseteq \mathsf{R}$. We focus on computing sizes of sets $\mathrm{Occ}^{\mathsf{a}-}(P,T) := \mathrm{Occ}^{\mathsf{a}}(P,T) \cap \mathsf{R}^-$ and $\mathrm{Occ}^{\mathsf{s}-}(P,T) := \mathrm{Occ}^{\mathsf{s}}(P,T) \cap \mathsf{R}^-$. The sizes of the sets $\mathrm{Occ}^{\mathsf{a}+}(P,T) := \mathrm{Occ}^{\mathsf{a}}(P,T) \cap \mathsf{R}^+$ and $\mathrm{Occ}^{\mathsf{s}+}(P,T) := \mathrm{Occ}^{\mathsf{s}}(P,T) \cap \mathsf{R}^+$ are computed analogously (see Proposition 6.9).

We now describe the algorithm to compute $|\mathrm{Occ}^{\mathsf{a}-}(P,T)|$ for any periodic pattern $P \in [0\,..\,\sigma)^m$.

LEMMA 6.6. *Assume that $P \in [0\,..\,\sigma)^m$ is periodic. If $e(P) \le m$, then it holds* $\mathrm{Occ}^{\mathsf{a}-}(P,T) = \emptyset$. *Otherwise, it holds* $\mathrm{Occ}^{\mathsf{a}-}(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$, *where $s =$* L-head(P) *and $H =$* L-root(P).

Proof. Let $e(P) \le m$. Denote $k =$ L-exp(P). Suppose $\mathrm{Occ}^{\mathsf{a}-}(P,T) \neq \emptyset$, and let $j \in \mathrm{Occ}^{\mathsf{a}-}(P,T)$. By definition, $s + k|H| \le e(P) - 1 < s + (k+1)|H|$ and $P[e(P)] \neq P[e(P) - |H|]$. On the other hand, by $j \in \mathsf{R}_{s,H}$ and

L-exp$(j) > k$, the string $H'H^{k+1}$ (where H' is a length-s suffix of H) is a prefix of $T[j\,..\,n]$. Thus, we have $T[j+e(P)-1] = T[j+e(P)-1-|H|] = P[e(P)-|H|] \neq P[e(P)]$. This implies $j \notin \mathrm{Occ}(P,T)$, contradicting $j \in \mathrm{Occ}^{\mathsf{a}-}(P,T)$. Thus, $\mathrm{Occ}^{\mathsf{a}-}(P,T) = \emptyset$.

Let $e(P) > m$. The inclusion $\mathrm{Occ}^{\mathsf{a}-}(P,T) \subseteq \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$ follows by definition. To show the opposite inclusion, let $j \in \mathsf{R}^-_{s,H}$ be such that L-exp$(j) >$ L-exp(P). Denote $k =$ L-exp(P). Then, $P = H'H^kH''$, where $|H'| = s$, and H' (resp. H'') is a suffix (resp. prefix) of H. Thus, P is a prefix of $H'H^{k+1}$. The latter string, on the other hand, is by L-exp$(j) \geq k+1$ and $j \in \mathsf{R}_{s,H}$, a prefix of $T[j\,..\,n]$. Thus, $j \in \mathrm{Occ}(P,T)$. By $j \in \mathsf{R}^-_{s,H}$ and L-exp$(j) >$ L-exp(P), we therefore also have $j \in \mathrm{Occ}^{\mathsf{a}-}(P,T)$. □

Proposition 6.7. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Given the data structure from Section 6.3.2 and the packed representation of P, we can compute $|\mathrm{Occ}^{\mathsf{a}-}(P,T)|$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time.*

Proof. First, using Proposition 6.6, we compute $s =$ L-head(P), $H =$ L-root(P), $k =$ L-exp(P), and $t =$ L-tail(P) in $\mathcal{O}(1 + m/\log_\sigma n)$ time. This lets us determine $e(P) = 1+s+k|H|+t$. If $e(P) \leq m$, then by Lemma 6.6, we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = 0$. Otherwise, using the array L_{range}, we compute in $\mathcal{O}(1)$ time a pair of integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T prefixed with $X = P[1\,..\,3\tau-1]$. Equivalently, by Lemma 5.4 (see also the implementation of queries in Proposition 5.9), $\mathrm{SA}(b\,..\,e]$ contains all positions from $\mathsf{R}_{s,H}$. If $b = e$, then it holds $\mathsf{R}_{s,H} = \emptyset$, and thus we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = 0$. Let us thus assume $b < e$. Our goal now is to determine the subrange of $\mathrm{SA}(b\,..\,e]$ containing all positions in $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) > \text{L-exp}(P)\}$ (these positions form a subrange by Lemma 5.4). For that, we first compute $d = \mathsf{rank}_{B_{\text{exp}},1}(e) - \mathsf{rank}_{B_{\text{exp}},1}(b)$ in $\mathcal{O}(1)$ time. If $d = 0$, then $\mathsf{R}^-_{s,H} = \emptyset$, and hence we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = 0$. Otherwise, we retrieve $k_{\min} = L_{\text{minexp}}[\text{int}(X)]$ in $\mathcal{O}(1)$ time. Then, letting $k_{\max} = k_{\min} + d - 1$, we have $k_{\min} \leq k_{\max}$ and $[k_{\min}\,..\,k_{\max}] = \{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\}$ (see the proof of Proposition 5.9). If $k \geq k_{\max}$, by Lemma 6.6, we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = 0$. Otherwise, we have two cases. Let $p = \mathsf{rank}_{B_{\text{exp}},1}(b)$. If $k < k_{\min}$, then we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = |\mathsf{R}^-_{s,H}| = \mathsf{select}_{B_{\text{exp}},1}(p+d) - b$. Otherwise (i.e., $k \geq k_{\min}$), we return $|\mathrm{Occ}^{\mathsf{a}-}(P,T)| = \mathsf{select}_{B_{\text{exp}},1}(p+d) - \mathsf{select}_{B_{\text{exp}},1}(p+k-k_{\min}+1)$. In total, the query takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. □

Next, we now describe the algorithm compute $|\mathrm{Occ}^{\mathsf{s}-}(P,T)|$ for any periodic pattern $P \in [0\,..\,\sigma)^m$.

Lemma 6.7. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Denote $H =$ L-root(P). Assume $i \in \mathsf{R}^-_H$ and let $\ell = e(i) - i - 3\tau + 2$. Then, $|\mathrm{Occ}^{\mathsf{s}-}(P,T) \cap [i\,..\,i+\ell)| \leq 1$. Moreover, $|\mathrm{Occ}^{\mathsf{s}-}(P,T) \cap [i\,..\,i+\ell)| = 1$ holds if and only if $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}(i)\,..\,n]$ and $e^{\text{full}}(i) - i \geq e^{\text{full}}(P) - 1$.*

Proof. As observed in the proof of Lemma 5.10, $[i\,..\,i+\ell) \subseteq \mathsf{R}^-_H$, and for any $\delta \in [0\,..\,\ell)$, it holds $e(i+\delta) = e(i)$, L-tail$(i+\delta) =$ L-tail(i), and consequently, $e^{\text{full}}(i+\delta) = e^{\text{full}}(i)$ and $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - i - \delta$. Moreover, by definition of $\mathrm{Occ}^{\mathsf{s}-}(P,T)$, letting L-head$(P) = s$, for any $j \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$ it holds $e^{\text{full}}(j) - j = s + \text{L-exp}(j)\cdot|H| = s + \text{L-exp}(P)\cdot|H| = e^{\text{full}}(P) - 1$. Thus, $i+\delta \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$ implies $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - (i+\delta) = e^{\text{full}}(P) - 1$, or equivalently, $\delta = (e^{\text{full}}(i) - i) - (e^{\text{full}}(P) - 1)$, and therefore, $|\mathrm{Occ}^{\mathsf{s}-}(P,T) \cap [i\,..\,i+\ell)| \leq 1$.

For the second part, assume first that $i + \delta \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$ holds for some $\delta \in [0\,..\,\ell)$. Then, as noted above, we have $e^{\text{full}}(P) - 1 = e^{\text{full}}(i) - (i+\delta) \leq e^{\text{full}}(i) - i$. Moreover, letting L-head$(P) = s$, by definition of $\mathrm{Occ}^{\mathsf{s}-}(P,T)$, we have $i + \delta \in \mathsf{R}^-_{s,H}$, L-exp$(P) =$ L-exp$(i+\delta)$, and $T[i+\delta\,..\,i+\delta+m) = P$. Therefore, we obtain that $T[i+\delta\,..\,e^{\text{full}}(i+\delta)) = T[i+\delta\,..\,e^{\text{full}}(i)) = P[1\,..\,e^{\text{full}}(P)) = H'H^k$ (where $k =$ L-exp(P) and H' is the length-s suffix of H), and consequently, $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}(i)\,..\,n]$. To show the converse implication, assume that $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}(i)\,..\,n]$ and $e^{\text{full}}(i) - i \geq e^{\text{full}}(P) - 1$. Let $\delta = (e^{\text{full}}(i) - i) - (e^{\text{full}}(P) - 1)$. We will prove that $\delta \in [0\,..\,\ell)$ and $i + \delta \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$. Clearly $\delta \geq 0$. To show $\delta < \ell$, we first prove $e(i) - e^{\text{full}}(i) \geq e(P) - e^{\text{full}}(P)$. Suppose that $q = e(i) - e^{\text{full}}(i) < e(P) - e^{\text{full}}(P)$. By $i \in \mathsf{R}^-_H$, we then either have $e^{\text{full}}(i) + q = n+1$, or $e^{\text{full}}(i) + q \leq n$ and $T[e^{\text{full}}(i)+q] \neq T[e^{\text{full}}(i)+q-|H|] = P[e^{\text{full}}(P)+q-|H|] = P[e^{\text{full}}(P)+q]$, both of which contradict that $P[e^{\text{full}}(P)\,..\,m]$ is a prefix of $T[e^{\text{full}}(i)\,..\,n]$. Thus, $e(i) - e^{\text{full}}(i) \geq e(P) - e^{\text{full}}(P)$. This implies, $e(i) - (i+\delta) = (e^{\text{full}}(i) - (i+\delta)) + (e(i) - e^{\text{full}}(i)) = (e^{\text{full}}(P) - 1) + (e(i) - e^{\text{full}}(i)) \geq (e^{\text{full}}(P) - 1) + (e(P) - e^{\text{full}}(P)) = e(P) - 1 \geq 3\tau - 1$, or equivalently $\delta \leq e(i) - i - 3\tau + 1 < \ell$. To show $i + \delta \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$, it remains to observe that $e^{\text{full}}(i+\delta) - (i+\delta) = e^{\text{full}}(i) - (i+\delta) = e^{\text{full}}(P) - 1$ and L-root$(i+\delta) =$ L-root$(i) = H =$ L-root(P) (following from Lemma 5.5) imply $T[i+\delta\,..\,e^{\text{full}}(i)) = P[1\,..\,e^{\text{full}}(P))$. This in particular gives, letting L-head$(P) = s$, that $i + \delta \in \mathsf{R}_{s,H}$ and L-exp$(i+\delta) =$ L-exp(P). Moreover, combining it with $P[e^{\text{full}}(P)\,..\,m]$ being a prefix of $T[e^{\text{full}}(i)\,..\,n]$ yields $T[i+\delta\,..\,i+\delta+m) = P$. Finally, by Lemma 5.5, $\text{type}(i+\delta) = \text{type}(i) = -1$. Therefore, $i + \delta \in \mathrm{Occ}^{\mathsf{s}-}(P,T)$. □

PROPOSITION 6.8. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Given the data structure from Section 6.3.2 and the packed representation of P, we can compute $|\mathrm{Occ}^{\mathsf{s}-}(P,T)|$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.*

Proof. First, using Proposition 6.6, we compute $s = \text{L-head}(P)$, $H = \text{L-root}(P)$, and $k = \text{L-exp}(P)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time. This lets us determine $e^{\mathrm{full}}(P) = 1+s+k|H|$ and $P' := P[e^{\mathrm{full}}(P) - |\mathrm{pow}(H)|\,..\,m]$. Then, using Proposition 4.2, we compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time a range $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,q] : P' \text{ is a prefix of } T[A_{\mathsf{Z}}[i]\,..\,n]\}$. Observe that the set $\{r_i^{\mathrm{lex}} : i \in (b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]\}$ consists of all positions $j \in \mathsf{R}'^{-}_H$ such that $P[e^{\mathrm{full}}(P)\,..\,m]$ is a prefix of $T[e^{\mathrm{full}}(j)\,..\,n]$. Thus, by Lemma 6.7, we have $|\mathrm{Occ}^{\mathsf{s}-}(P,T)| = |\{i \in (b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] : e^{\mathrm{full}}(r_i^{\mathrm{lex}}) - r_i^{\mathrm{lex}} \geq e^{\mathrm{full}}(P) - 1\}|$, which we compute in $\mathcal{O}(\log\log n)$ time using the range counting structure as $\mathsf{rcount}_{A_{\mathrm{len}}}(e^{\mathrm{full}}(P)-1, e_{\mathrm{pre}}) - \mathsf{rcount}_{A_{\mathrm{len}}}(e^{\mathrm{full}}(P)-1, b_{\mathrm{pre}})$ (recall, that $A_{\mathrm{len}}[i] = e^{\mathrm{full}}(r_i^{\mathrm{lex}}) - r_i^{\mathrm{lex}}$; see Section 5.3.2). □

By combining all above results, we obtain the following algorithm to compute $|\mathrm{Occ}(P,T)|$ for any periodic pattern P.

PROPOSITION 6.9. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Given the data structure from Section 6.3.2 and the packed representation of P, we can compute $|\mathrm{Occ}(P,T)|$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time.*

Proof. Given a packed representation of a periodic pattern P, we compute $|\mathrm{Occ}(P,T)| = |\mathrm{Occ}^{\mathsf{a}-}(P,T)| + |\mathrm{Occ}^{\mathsf{s}-}(P,T)| + |\mathrm{Occ}^{\mathsf{a}+}(P,T)| + |\mathrm{Occ}^{\mathsf{s}+}(P,T)|$ using Propositions 6.7 and 6.8 and their symmetric counterparts (adapted according to Lemma 6.2). The total time is $\mathcal{O}(m/\log_\sigma n + \log\log n)$. □

Generalizing the Query Algorithm We now show how to generalize the above algorithms to compute $|\mathrm{Occ}(P,T)|$, to instead return $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.

For any periodic pattern $P \in [0\,..\,\sigma)^m$ we define

$$\mathrm{Pos}(P,T) = \{j \in [1\,..\,n] : \mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1 \text{ and } T[j\,..\,n] \prec P\},$$

and denote $\delta(P,T) = |\mathrm{Pos}(P,T)|$.

LEMMA 6.8. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern and let $X = P[1\,..\,3\tau-1]$. Then, it holds* $\mathrm{RangeBeg}(P,T) = \mathrm{RangeBeg}(X,T) + \delta(P,T)$.

Proof. It suffices to observe that $j \in \mathrm{Occ}(X,T)$ holds if and only if $\mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1$. Thus, it holds by definition of $\mathrm{RangeBeg}(P,T)$ that $\mathrm{RangeBeg}(P,T) = \mathrm{RangeBeg}(X,T) + |\{j \in \mathrm{Occ}(X,T) : T[j\,..\,n] \prec P\}| = \mathrm{RangeBeg}(X,T) + |\{j \in [1\,..\,n] : \mathrm{lcp}(T[j\,..\,n], P) \geq 3\tau - 1 \text{ and } T[j\,..\,n] \prec P\}| = \mathrm{RangeBeg}(X,T) + \delta(P,T)$. □

We focus on computing $\delta(P,T)$ for P satisfying $\mathrm{type}(P) = -1$ (the structure for P satisfying $\mathrm{type}(P) = +1$ is symmetric; see the proof of Proposition 6.12). We define $\mathrm{Pos}^{\mathsf{a}}(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) \leq \text{L-exp}(P)\}$ and $\mathrm{Pos}^{\mathsf{s}}(P,T) = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = \text{L-exp}(P) \text{ and } T[j\,..\,n] \succeq P\}$, where $s = \text{L-head}(P)$ and $H = \text{L-root}(P)$. We denote $\delta^{\mathsf{a}}(P,T) = |\mathrm{Pos}^{\mathsf{a}}(P,T)|$ and $\delta^{\mathsf{s}}(P,T) = |\mathrm{Pos}^{\mathsf{s}}(P,T)|$.

LEMMA 6.9. *For any periodic pattern $P \in [0\,..\,\sigma)^m$ that satisfies* $\mathrm{type}(P) = -1$*, it holds $\delta(P,T) = \delta^{\mathsf{a}}(P,T) - \delta^{\mathsf{s}}(P,T)$.*

Proof. We will prove that $\mathrm{Pos}^{\mathsf{a}}(P,T)$ is a disjoint union of $\mathrm{Pos}(P,T)$ and $\mathrm{Pos}^{\mathsf{s}}(P,T)$. This implies $\delta(P,T) + \delta^{\mathsf{s}}(P,T) = \delta^{\mathsf{a}}(P,T)$, and consequently, the equality in the claim.

Denote $s = \text{L-head}(P)$ and $H = \text{L-root}(P)$. By Lemma 6.2, letting $j \in \mathsf{R}^-_{s,H}$, we have $\mathrm{Pos}(P,T) = \{j \in \mathsf{R}^-_{s,H} : T[j\,..\,n] \prec P\}$, and moreover, if $j \in \mathrm{Pos}(P,T)$, then $e(j) - j \leq e(P) - 1$. In particular, $\text{L-exp}(j) = \lfloor \frac{e(j)-j-s}{|H|} \rfloor \leq \lfloor \frac{e(P)-1-s}{|H|} \rfloor = \text{L-exp}(P)$. Hence, $\mathrm{Pos}(P,T) \subseteq \mathrm{Pos}^{\mathsf{a}}(P,T)$. On the other hand, clearly $\mathrm{Pos}^{\mathsf{s}}(P,T) \subseteq \mathrm{Pos}^{\mathsf{a}}(P,T)$ and $\mathrm{Pos}^{\mathsf{s}}(P,T) \cap \mathrm{Pos}(P,T) = \emptyset$. Thus, to obtain the claim, it suffices to show that $\mathrm{Pos}^{\mathsf{a}}(P,T) \setminus \mathrm{Pos}^{\mathsf{s}}(P,T) \subseteq \mathrm{Pos}(P,T)$.

Let $j \in \mathrm{Pos}^{\mathsf{a}}(P,T) \setminus \mathrm{Pos}^{\mathsf{s}}(P,T)$. Consider two cases. If $\text{L-exp}(j) = \text{L-exp}(P)$, then by definition of $\mathrm{Pos}^{\mathsf{s}}(P,T)$, it must hold $T[j\,..\,n] \prec P$. Thus, we have $j \in \mathrm{Pos}(P,T)$. Let us therefore assume $\text{L-exp}(j) < \text{L-exp}(P)$. Then, $e(j) - j = s + \text{L-exp}(j)\cdot|H| + \text{L-tail}(j) < s + \text{L-exp}(j)\cdot|H| + |H| \leq s + \text{L-exp}(P)\cdot|H| \leq s + \text{L-exp}(P)\cdot|H| + \text{L-tail}(P) = e(P) - 1$. By Lemma 6.2(2) and Lemma 6.2(4), this implies $T[j\,..\,n] \prec P$, and consequently, $j \in \mathrm{Pos}(P,T)$. □

We now describe how, given any periodic pattern $P \in [0\,..\,\sigma)^m$ that satisfies $\mathrm{type}(P) = -1$, to compute $\delta^{\mathsf{a}}(P,T)$.

PROPOSITION 6.10. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern satisfying $\mathrm{type}(P) = -1$. Given the data structure from Section 6.3.2 and the packed representation of P, we can in $\mathcal{O}(1 + m/\log_\sigma n)$ time compute $\delta^{\mathsf{a}}(P,T)$.*

Proof. First, using Proposition 6.6, we compute $H = \text{L-root}(P)$ and $k = \text{L-exp}(P)$ in $\mathcal{O}(1+m/\log_\sigma n)$ time. Then, using L_{range}, we compute in $\mathcal{O}(1)$ time a pair of integers b, e such that $\mathrm{SA}(b\,..\,e]$ contains the starting positions of all suffixes of T prefixed with $X = P[1\,..\,3\tau{-}1]$. Equivalently, by Lemma 6.2, $\mathrm{SA}(b\,..\,e]$ contains all positions from $\mathsf{R}_{s,H}$, where $s = \text{L-head}(P)$. If $b = e$, then it holds $\mathsf{R}_{s,H} = \emptyset$, and thus we return $\delta^{\mathsf{a}}(P,T) = 0$. Let us thus assume $b < e$. Our goal now is to determine the subrange of $\mathrm{SA}(b\,..\,e]$ containing all positions in $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) \le \text{L-exp}(P)\}$ (these positions form a subrange by Lemma 5.4). For that, we first compute $d = \mathsf{rank}_{B_{\exp},1}(e) - \mathsf{rank}_{B_{\exp},1}(b)$ in $\mathcal{O}(1)$ time. If $d = 0$, then $\mathsf{R}^-_{s,H} = \emptyset$, and hence we return $\delta^{\mathsf{a}}(P,T) = 0$. Otherwise, we retrieve $k_{\min} = L_{\mathrm{minexp}}[\mathrm{int}(X)]$ in $\mathcal{O}(1)$ time. Then, letting $k_{\max} = k_{\min} + d - 1$, we have $k_{\min} \le k_{\max}$ and $[k_{\min}\,..\,k_{\max}] = \{\text{L-exp}(j) : j \in \mathsf{R}^-_{s,H}\}$ (see the proof of Proposition 5.9). If $k < k_{\min}$, we return $\delta^{\mathsf{a}}(P,T) = 0$. Otherwise, we have two cases. Let $p = \mathsf{rank}_{B_{\exp},1}(b)$. If $k \ge k_{\max}$, then we return $\delta^{\mathsf{a}}(P,T) = |\mathsf{R}^-_{s,H}| = \mathsf{select}_{B_{\exp},1}(p+d) - b$. Otherwise (i.e., $k < k_{\max}$), we return $\delta^{\mathsf{a}}(P,T) = \mathsf{select}_{B_{\exp},1}(p + k - k_{\min} + 1) - b$. In total, the query takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. □

We now describe how, given any periodic pattern $P \in [0\,..\,\sigma)^m$ that satisfies $\mathrm{type}(P) = -1$, to compute $\delta^{\mathsf{s}}(P,T)$.

LEMMA 6.10. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern that satisfies $\mathrm{type}(P) = -1$. Denote $H = \text{L-root}(P)$. Assume $i \in \mathsf{R}^-_H$ and let $\ell = e(i) - i - 3\tau + 2$. Then, we have $|\mathrm{Pos}^{\mathsf{s}}(P,T) \cap [i\,..\,i+\ell)| \le 1$. Moreover, $|\mathrm{Pos}^{\mathsf{s}}(P,T) \cap [i\,..\,i+\ell)| = 1$ holds if and only if $T[e^{\mathrm{full}}(i)\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ and $e^{\mathrm{full}}(i) - i \ge e^{\mathrm{full}}(P) - 1$.*

Proof. In the proof of Lemma 6.7, it is shown that $[i\,..\,i+\ell) \subseteq \mathsf{R}^-_H$, and for any $\delta \in [0\,..\,\ell)$, it holds $e^{\mathrm{full}}(i+\delta) - (i+\delta) = e^{\mathrm{full}}(i) - i - \delta$. By definition of $\mathrm{Pos}^{\mathsf{s}}(P,T)$, letting $s = \text{L-head}(P)$, for any $j \in \mathrm{Pos}^{\mathsf{s}}(P,T)$ it holds $e^{\mathrm{full}}(j) - j = s + \text{L-exp}(j) \cdot |H| = s + \text{L-exp}(P) \cdot |H| = e^{\mathrm{full}}(P) - 1$. Thus, $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(P,T)$ implies $e^{\mathrm{full}}(i+\delta) - (i+\delta) = e^{\mathrm{full}}(i) - (i+\delta) = e^{\mathrm{full}}(P) - 1$, or equivalently, $\delta = (e^{\mathrm{full}}(i) - i) - (e^{\mathrm{full}}(P) - 1)$, and therefore, $|\mathrm{Pos}^{\mathsf{s}}(P,T) \cap [i\,..\,i+\ell)| \le 1$.

For the second part, assume first that $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(P,T)$ holds for some $\delta \in [0\,..\,\ell)$. Then, as noted above, we have $e^{\mathrm{full}}(P) - 1 = e^{\mathrm{full}}(i) - (i+\delta) \le e^{\mathrm{full}}(i) - i$. Moreover, letting $\text{L-head}(P) = s$, by definition of $\mathrm{Pos}^{\mathsf{s}}(P,T)$, we have $i + \delta \in \mathsf{R}^-_{s,H}$, $\text{L-exp}(P) = \text{L-exp}(i+\delta)$, and $T[i{+}\delta\,..\,n] \succeq P$. Therefore, we obtain that $T[i{+}\delta\,..\,e^{\mathrm{full}}(i+\delta)) = T[i{+}\delta\,..\,e^{\mathrm{full}}(i)) = P[1\,..\,e^{\mathrm{full}}(P)) = H'H^k$ (where $k = \text{L-exp}(P)$ and H' is the length-s suffix of H), and consequently, $T[e^{\mathrm{full}}(i)\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$. To show the converse implication, assume that $T[e^{\mathrm{full}}(i)\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ and $e^{\mathrm{full}}(i) - i \ge e^{\mathrm{full}}(P) - 1$. Let $\delta = (e^{\mathrm{full}}(i) - i) - (e^{\mathrm{full}}(P) - 1)$. We will prove that $\delta \in [0\,..\,\ell)$ and $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(P,T)$. Clearly $\delta \ge 0$. To show $\delta < \ell$, we first prove $e(i) - e^{\mathrm{full}}(i) \ge e(P) - e^{\mathrm{full}}(P)$. Suppose that $q = e(i) - e^{\mathrm{full}}(i) < e(P) - e^{\mathrm{full}}(P)$. By $i \in \mathsf{R}^-_H$, we then either have $e^{\mathrm{full}}(i) + q = n + 1$, or $e^{\mathrm{full}}(i) + q \le n$ and $T[e^{\mathrm{full}}(i) + q] \prec T[e^{\mathrm{full}}(i) + q - |H|] = P[e^{\mathrm{full}}(P) + q - |H|] = P[e^{\mathrm{full}}(P) + q]$, both of which contradict $T[e^{\mathrm{full}}(i)\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$. Thus, $e(i) - e^{\mathrm{full}}(i) \ge e(P) - e^{\mathrm{full}}(P)$. This implies, $e(i) - (i+\delta) = (e^{\mathrm{full}}(i) - (i+\delta)) + (e(i) - e^{\mathrm{full}}(i)) = (e^{\mathrm{full}}(P) - 1) + (e(i) - e^{\mathrm{full}}(i)) \ge (e^{\mathrm{full}}(P) - 1) + (e(P) - e^{\mathrm{full}}(P)) = e(P) - 1 \ge 3\tau - 1$, or equivalently $\delta \le e(i) - i - 3\tau + 1 < \ell$. To show $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(P,T)$, it remains to observe that $e^{\mathrm{full}}(i+\delta) - (i+\delta) = e^{\mathrm{full}}(i) - (i+\delta) = e^{\mathrm{full}}(P) - 1$ and $\text{L-root}(i+\delta) = \text{L-root}(i) = H = \text{L-root}(P)$ (following from Lemma 5.5) imply $T[i + \delta\,..\,e^{\mathrm{full}}(i)) = P[1\,..\,e^{\mathrm{full}}(P))$. This in particular gives, letting $\text{L-head}(P) = s$, that $i + \delta \in \mathsf{R}_{s,H}$ and $\text{L-exp}(i+\delta) = \text{L-exp}(P)$. Moreover, combining it with $T[e^{\mathrm{full}}(i)\,..\,n] \succeq P[e^{\mathrm{full}}(P)\,..\,m]$ yields $T[i + \delta\,..\,n] \succeq P$. Finally, by Lemma 5.5, $\mathrm{type}(i+\delta) = \mathrm{type}(i) = -1$. Therefore, $i + \delta \in \mathrm{Pos}^{\mathsf{s}}(P,T)$. □

PROPOSITION 6.11. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern satisfying $\mathrm{type}(P) = -1$. Given the data structure from Section 6.3.2 and the packed representation of P, we can in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time compute $\delta^{\mathsf{s}}(P,T)$.*

Proof. First, using Proposition 6.6, we compute $s = \text{L-head}(P)$, $H = \text{L-root}(P)$, and $k = \text{L-exp}(P)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ time. This lets us determine $e^{\mathrm{full}}(P) = 1{+}s{+}k|H|$ and $P' := P[e^{\mathrm{full}}(P) - |\mathrm{pow}(H)|\,..\,m]$. Then, using Proposition 4.2, we compute in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time a value $x = |\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec P'\}|$. Then, letting $x' = \sum_{H' \preceq H} |\mathsf{R}'^-_{H'}|$ (obtained from L_{runs} in $\mathcal{O}(1)$ time as explained in the proof of Proposition 5.10), by definition of A_{Z} and properties of function pow (see the proof of Proposition 6.8), the set $\{r^{\mathrm{lex}}_i : i \in (x\,..\,x']\}$

(where r_i^{lex} is defined as in the proof of Proposition 6.8) consists of all positions $j \in \mathsf{R}'^{-}_H$ satisfying $T[e^{\text{full}}(j)\,..\,n] \succeq P[e^{\text{full}}(P)\,..\,m]$. Thus, by Lemma 6.10, it holds $\delta^{\mathsf{s}}(P,T) = |\text{Pos}^{\mathsf{s}}(P,T)| = |\{i \in (x\,..\,x'] : \ell_i \geq e^{\text{full}}(P) - 1\}|$ (where ℓ_i is defined as in Proposition 6.8), which we compute in $\mathcal{O}(\log\log n)$ time using the range counting structure as $\mathsf{rcount}_{A_{\text{len}}}(e^{\text{full}}(P) - 1, x') - \mathsf{rcount}_{A_{\text{len}}}(e^{\text{full}}(P) - 1, x)$. □

By combining the above results, we obtain the algorithm to efficiently compute the pair $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T))$ for periodic patterns.

PROPOSITION 6.12. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern. Given the data structure from Section 6.3.2 and the packed representation of P, we can in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time compute the pair* $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T))$.

Proof. First, using Proposition 6.9 in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time we compute $|\text{Occ}(P,T)|$. Next, using the lookup table L_{range}, in $\mathcal{O}(1)$ time we compute $(b_X, e_X) = (\text{RangeBeg}(X,T), \text{RangeEnd}(X,T))$, where $X = P[1\,..\,3\tau{-}1]$. Then, in $\mathcal{O}(1 + m/\log_\sigma n)$ time using Proposition 6.6 we determine $\text{type}(P)$. Depending on whether $\text{type}(P) = -1$ or $\text{type}(P) = +1$, we use either a combination of Propositions 6.10 and 6.11, or their symmetric counterparts (more precisely, if $\text{type}(P) = +1$, we have $\delta^{\mathsf{a}}(P,T) = |\text{Pos}^{\mathsf{a}}(P,T)|$ and $\delta^{\mathsf{s}}(P,T) = |\text{Pos}^{\mathsf{s}}(P,T)|$, where $\text{Pos}^{\mathsf{a}}(P,T) = \{j \in \mathsf{R}^{+}_{s,H} : \text{L-exp}(j) \leq \text{L-exp}(P)\}$ and $\text{Pos}^{\mathsf{s}}(P,T) = \{j \in \mathsf{R}^{+}_{s,H} : \text{L-exp}(j) = \text{L-exp}(j) \text{ and } T[j\,..\,n] \prec P\}$), to compute $\delta^{\mathsf{a}}(P,T)$ and $\delta^{\mathsf{s}}(P,T)$ in $\mathcal{O}(1 + m/\log_\sigma n)$ and $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time, respectively. If $\text{type}(P) = -1$, then by Lemma 6.9 we have $\delta(P,T) = \delta^{\mathsf{a}}(P,T) - \delta^{\mathsf{s}}(P,T)$. Otherwise, by the counterpart of Lemma 6.9, $\delta(P,T) = (e_X - b_X) - (\delta^{\mathsf{a}}(P,T) - \delta^{\mathsf{s}}(P,T))$. Finally, we return $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T)) = (b_X + \delta(P,T), b_X + \delta(P,T) + |\text{Occ}(P,T)|)$ (see Lemma 6.8) as the answer. In total, the query takes $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time. □

REMARK 6.2. Note the subtle difference in the type of symmetry used during the computation of $|\text{Pos}(P,T)|$ and $|\text{Occ}(P,T)|$. When computing $\delta(P,T) = |\text{Pos}(P,T)|$, by Lemma 6.2 we have $\text{Pos}(P,T) \subseteq \mathsf{R}^{-}$ for any P satisfying $\text{type}(P) = -1$ (and $\text{Pos}(P,T) \subseteq \mathsf{R}^{+}$ for P satisfying $\text{type}(P) = +1$). However, when computing $|\text{Occ}(P,T)|$ for P satisfying $\text{type}(P) = -1$, it is possible that $\text{Occ}(P,T) \cap \mathsf{R}^{-} \neq \emptyset$ and $\text{Occ}(P,T) \cap \mathsf{R}^{+} \neq \emptyset$. Consequently, during the computation of $|\text{Occ}(P,T)|$, we partition the output set $\text{Occ}^{\mathsf{a}}(P,T)$ (resp. $\text{Occ}^{\mathsf{s}}(P,T)$) into two subsets $\text{Occ}^{\mathsf{a}-}(P,T)$ and $\text{Occ}^{\mathsf{a}+}(P,T)$ (resp. $\text{Occ}^{\mathsf{s}-}(P,T)$ and $\text{Occ}^{\mathsf{s}+}(P,T)$), but the computation is always performed regardless of $\text{type}(P)$, leading to two queries for each periodic pattern P. During the computation of $\delta(P,T)$, on the other hand, the computation is performed separately for P satisfying $\text{type}(P) = -1$ and P satisfying $\text{type}(P) = +1$, without the need to partition $\text{Pos}(P,T)$ within each case, leading to a single query but only on the appropriate structure depending on $\text{type}(P)$. This is the reason for why the seemingly related computation of $|\text{Pos}(P,T)|$ and $|\text{Occ}(P,T)|$ is (unlike for nonperiodic patterns; see Section 6.2.2) described separately.

6.3.5 Construction Algorithm

PROPOSITION 6.13. *Given $\text{C}_{\text{PM}}(T)$, we can in $\mathcal{O}(n/\log_\sigma n)$ time augment it into a data structure from Section 6.3.2.*

Proof. First, we combine Propositions 5.3 and 5.15 (recall that the packed representation of T is a component of $\text{C}_{\text{PM}}(T)$) to construct the data structure from Section 5.3.2 in $\mathcal{O}(n/\log_\sigma n)$ time. In particular, this constructs $(r_i^{\text{lex}})_{i \in [1\,..\,q]}$. Using Proposition 5.7, we can now compute $A_{\mathsf{Z}}[i]$ for any $i \in [1\,..\,q]$ in $\mathcal{O}(1)$ time. Then, in $\mathcal{O}(n/\log_\sigma n)$ time, we construct the data structure from Proposition 4.2.

After the above components are constructed, we then analogously construct their symmetric counterparts (adapted according to Lemma 6.2). □

6.4 The Final Data Structure

In this section, we put together Sections 6.1 to 6.3 to obtain a data structure that, given a packed representation of any pattern $P \in [0\,..\,\sigma)^m$, computes $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 6.4.1). Next, we describe the query algorithms (Section 6.4.2). Finally, we show the construction algorithm (Section 6.4.3).

6.4.1 The Data Structure

The data structure consists of two components:

1. The structure from Section 6.2.1 (used to handle nonperiodic patterns).
2. The structure from Section 6.3.2 (used to handle periodic patterns).

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

6.4.2 Implementation of Queries

PROPOSITION 6.14. *Given the data structure from Section 6.4.1 and the packed representation of any $P \in [0\,..\,\sigma)^m$, we can in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time compute the pair* $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.

Proof. First, using Proposition 6.1, in $\mathcal{O}(1)$ time we check if P is periodic. If so, we obtain $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time using Proposition 6.12. Otherwise (i.e., if P is not periodic), we consider two cases, depending on whether it holds $m < 3\tau - 1$. If so, then we obtain $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(1)$ time using Proposition 6.2. Otherwise, we obtain $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time using Proposition 6.4. □

6.4.3 Construction Algorithm

PROPOSITION 6.15. *Given the packed representation of $T \in [0\,..\,\sigma)^n$, we can construct the data structure from Section 6.4.1 in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space.*

Proof. First, from a packed representation of T, we construct $\mathrm{C_{PM}}(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 6.3. Then, using Propositions 6.5 and 6.13, we augment $\mathrm{C_{PM}}(T)$ into the two components of the structure from Section 6.4.1 in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ and $\mathcal{O}(n/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space. The overall runtime is thus $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$. □

6.5 Summary

By combining Proposition 6.14 and Proposition 6.15 we obtain the following final result of this section.

THEOREM 6.1. *Given any constant $\epsilon \in (0,1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/7}$, we can in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of any $P \in [0\,..\,\sigma)^m$, returns the pair* $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ *(and hence, in particular, the value $|\mathrm{Occ}(P,T)|$) in $\mathcal{O}(m/\log_\sigma n + \log^\epsilon n)$ time.*

By combining the above result with Theorem 5.1, we moreover obtain the following result.

THEOREM 6.2. *Given any constant $\epsilon \in (0,1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \le \sigma < n^{1/7}$, we can in $\mathcal{O}(n\min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space construct a data structure of size $\mathcal{O}(n/\log_\sigma n)$ that, given the packed representation of any $P \in [0\,..\,\sigma)^m$, returns the set $\mathrm{Occ}(P,T)$ in $\mathcal{O}(m/\log_\sigma n + (|\mathrm{Occ}(P,T)| + 1)\log^\epsilon n)$ time.*

By observing that the dominating operations in the above index are prefix rank and selection queries, we obtain the following more general result.

THEOREM 6.3. *Consider a data structure answering prefix rank and selection queries that, for any string of length m over alphabet $[0\,..\,\sigma)^\ell$, achieves the following complexities:*

1. *Space usage $S(m,\ell,\sigma)$,*
2. *Preprocessing time $P_t(m,\ell,\sigma)$,*
3. *Preprocessing space $P_s(m,\ell,\sigma)$,*
4. *Query time $Q(m,\ell,\sigma)$.*

Operation	Description
isleaf(v)	Return true if and only if v is a leaf
index(v)	Any position $j \in \mathrm{Occ}(\mathrm{str}(v), T)$
findleaf(j)	The leaf v satisfying $\mathrm{str}(v) = T[j\,..\,n]$
count(v)	The number of leaves in the subtree rooted in v
sdepth(v)	The string-depth of node v, i.e., $\lvert\mathrm{str}(v)\rvert$
parent(v)	The parent of $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$
firstchild(v)	The leftmost child of v, or $\perp$ if v is a leaf
lastchild(v)	The rightmost child of v, or $\perp$ if v is a leaf
rightsibling(v)	The right sibling of v, or $\perp$ if there is no such node
leftsibling(v)	The left sibling of v, or $\perp$ if there is no such node
slink(v)	A node v' satisfying $\mathrm{str}(v') = \mathrm{str}(v)[2\,..\,\lvert\mathrm{str}(v)\rvert]$
slink(v, i)	A node v' satisfying $\mathrm{str}(v') = \mathrm{str}(v)[i{+}1\,..\,\lvert\mathrm{str}(v)\rvert]$, i.e., iterated slink
wlink(v, c)	A node v' satisfying $\mathrm{str}(v') = c \cdot \mathrm{str}(v)$, or $\perp$ if there is no such node [8]
child(v, c)	A child v' of v satisfying $\mathrm{str}(v')[\lvert\mathrm{str}(v)\rvert{+}1] = c$, or $\perp$ if there is no such node
pred(v, c)	A node child(v, c'), where $c' = \max\{c'' \in [0\,..\,c) : \mathrm{child}(v, c'') \neq \perp\}$ (or $\perp$)
letter(v, i)	The ith leftmost character of $\mathrm{str}(v)$
WA(v, d)	The most shallow ancestor of v satisfying $\mathrm{sdepth}(v) \geq d$
LCA(u, v)	The lowest common ancestor of nodes u and v
isancestor(u, v)	Return true if and only if u is an ancestor of v

Table 1: Operations on suffix tree $\mathcal{T}_{\mathrm{st}}$ supported by our data structure.

For every $T \in [0\,..\,\sigma)^n$ with $2 \leq \sigma < n^{1/7}$, there exist $m = \mathcal{O}(n/\log_\sigma n)$ and $\ell = \mathcal{O}(\log_\sigma n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(m, \ell, \sigma))$ time and $\mathcal{O}(n/\log_\sigma n + P_s(m, \ell, \sigma))$ working space build a structure of size $\mathcal{O}(n/\log_\sigma n + S(m, \ell, \sigma))$ that, given the packed representation of any $P \in [0\,..\,\sigma)^m$, performs the following queries:

- *Return* $(\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T))$ *in* $\mathcal{O}(m/\log_\sigma n + \log\log n + Q(m, \ell, \sigma))$ *time,*
- *Return* $\mathrm{Occ}(P, T)$ *in* $\mathcal{O}(m/\log_\sigma n + (\lvert\mathrm{Occ}(P, T)\rvert + 1)(\log\log n + Q(m, \ell, \sigma)))$ *time.*

7 Suffix Tree Queries

Let $\epsilon \in (0, 1)$ be any fixed constant and let $T \in [0\,..\,\sigma)^n$, where $2 \leq \sigma < n^{1/7}$. Let $\mathcal{T}_{\mathrm{st}}$ denote the suffix tree of T, i.e., a compact trie of the set $\{T[1\,..\,n], T[2\,..\,n], \ldots, T[n]\}$. In this section, we show how given the packed representation of T, to construct in $\mathcal{O}(n \min(1, \log\sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space a representation of $\mathcal{T}_{\mathrm{st}}$ occupying $\mathcal{O}(n/\log_\sigma n)$ space, and supporting each of the operations listed in Table 1 in $\mathcal{O}(\log^\epsilon n)$ time. [9] We also derive a general reduction depending on prefix rank and selection queries.

As in Sections 5 and 6, we let $\tau = \lfloor \mu \log_\sigma n \rfloor$, where μ is any positive constant smaller than $\frac{1}{6}$ such that $\tau \geq 1$, be fixed for the duration of this section. Throughout, we also use R as a shorthand for $\mathsf{R}(\tau, T)$.

Definition 7.1. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. The node v is said to be* periodic *if $\mathrm{str}(v)$ is periodic (Definition 6.1). Otherwise, v is* nonperiodic.

Representation of a Node For any explicit node v of $\mathcal{T}_{\mathrm{st}}$ we denote $\mathrm{Occ}(v) := \mathrm{Occ}(\mathrm{str}(v), T)$. In our data structure we represent each explicit node v of $\mathcal{T}_{\mathrm{st}}$ in one of two ways:

[8] Our data structure supports also a slightly stronger operation $\mathrm{wlink}'(v, c)$ (see Proposition 7.31), that returns a node v' satisfying $\mathrm{repr}(v') = (\mathrm{RangeBeg}(c \cdot \mathrm{str}(v), T), \mathrm{RangeEnd}(c \cdot \mathrm{str}(v), T))$, if such node exists. This generalizes $\mathrm{wlink}(v, c)$, since $\mathrm{wlink}(v, c) \neq \perp$ holds if and only if $\mathrm{wlink}'(v, c) \neq \perp$ and $\mathrm{sdepth}(\mathrm{wlink}'(v, c)) = \mathrm{sdepth}(v) + 1$. Therefore, we can use $\mathrm{wlink}'(v, c)$ to compute $\mathrm{wlink}(v, c)$. Note, however, that it is possible that $\mathrm{wlink}(v, c) = \perp$ and yet $\mathrm{wlink}'(v, c) \neq \perp$. In that case, there exists a node corresponding to $\mathrm{wlink}(v, c)$ in the suffix *trie* of T, but in $\mathcal{T}_{\mathrm{st}}$ this node is not explicit.

[9] Similarly as in prior CST implementations [79, 31, 77, 34, 14, 16], the time complexity of some operations is actually $\mathcal{O}(1)$. We also note that some prior CST implementations (e.g., [79, 77, 34, 16]) support two additional operations called *tree depth* and *tree level ancestor* which are analogous to $\mathrm{sdepth}(v)$ and $\mathrm{WA}(v, d)$ but with distance to the root defined by the number of ancestor nodes rather than the total length of edge labels.

- A pair (j, ℓ), where $j \in \mathrm{Occ}(v)$ (i.e., j is the starting position of some occurrence of $\mathrm{str}(v)$ in T) and $\ell = \mathrm{sdepth}(v)$).
- A pair $(\mathrm{lrank}(v), \mathrm{rrank}(v))$. Note that since v is a node of suffix tree, in this special case we have $(\mathrm{lrank}(v), \mathrm{rrank}(v)) = (\mathrm{RangeBeg}(\mathrm{str}(v), T), \mathrm{RangeEnd}(\mathrm{str}(v), T))$. Thus, letting $(b, e) = (\mathrm{lrank}(v), \mathrm{rrank}(v))$, we then have $\{\mathrm{SA}[i]\}_{i \in (b\,..\,e]} = \mathrm{Occ}(v)$. Note also that $b < e$.

In most cases, the latter representation leads to a more convenient implementation. Thus, we adopt it as a default and denote $\mathrm{repr}(v) := (\mathrm{lrank}(v), \mathrm{rrank}(v))$ (while using the first one mostly as a temporary internal representation). We also define $\mathrm{repr}(\bot) = (0, 0)$.

Organization The structure and query algorithms for a node v are different depending on whether v is periodic (Definition 7.1). Our description is thus split as follows. First (Section 7.1), we describe the set of data structures called collectively the index "core" that enables efficiently checking if v is periodic (it is also used to perform operations on nodes with very small depth and contains some common components utilized by the remaining parts). In the following two parts (Sections 7.2 and 7.3), we describe structures handling each of the two cases. All ingredients are then put together in Section 7.4. Finally, we present our result in the general form (Section 7.5).

7.1 The Index Core

In this section, we describe a data structure used to check in $\mathcal{O}(1)$ time if a given node is periodic. It also lets us perform operations concerning nodes at depth smaller than $3\tau - 1$ in $\mathcal{O}(1)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 7.1.1). We then show how using this structure to implement some basic navigational routines (Section 7.1.2). Next, we describe the query algorithms for the fundamental operations (Sections 7.1.3 to 7.1.6). Finally, we show the construction algorithm (Section 7.1.7).

7.1.1 The Data Structure

Definitions For any $k \geq 1$, let $\mathcal{S}_k := \{S \in [0\,..\,\sigma)^k : S \text{ occurs in } T\}$ denote the set of length-k substrings of T. Let $\mathcal{T}_{3\tau-1}$ denote the compact trie of $\mathcal{S}_{3\tau-1}$.

Components The index core, denoted $\mathrm{C}_{\mathrm{ST}}(T)$, consists of two components:

1. The index core $\mathrm{C}_{\mathrm{SA}}(T)$ (Section 5.1.1). It takes $\mathcal{O}(n/\log_\sigma n)$ space.
2. The compact trie $\mathcal{T}_{3\tau-1}$. All nodes of $\mathcal{T}_{3\tau-1}$ are stored in an array and pointers to nodes are implemented as indexes to this array. Each node v of $\mathcal{T}_{3\tau-1}$ stores the string $\mathrm{str}(v)$ encoded as an integer $\mathrm{int}(\mathrm{str}(v))$, the pointer $\mathrm{parent}(v)$, the value $\mathrm{sdepth}(v)$, and the doubly linked list containing pointers to all children of v, in ascending order of the first letter on the connecting edge. Since each node v of $\mathcal{T}_{3\tau-1}$ corresponds to a unique string $S \in [0\,..\,\sigma)^{\leq 3\tau-1}$, in total $\mathcal{T}_{3\tau-1}$ needs $\mathcal{O}(\sigma^{3\tau}) = \mathcal{O}(\sqrt{n})$ space. The trie $\mathcal{T}_{3\tau-1}$ is augmented with the following structures:
   (a) A linear-space data structure answering the LCA queries in $\mathcal{T}_{3\tau-1}$ in $\mathcal{O}(1)$ time [10]. By the above bound, the data structure uses $\mathcal{O}(\sqrt{n})$ space.
   (b) A lookup table L_{child} that for each edge of $\mathcal{T}_{3\tau-1}$ connecting a node v to its parent p and labeled with a string starting with the character c, maps the pair (i_p, c) to i_v, where i_p and i_v are pointers to p and v. $\mathcal{T}_{3\tau-1}$ has less than $2\sigma^{3\tau-1}$ nodes and thus $i_v < 2\sigma^{3\tau-1}$. On the other hand, $c \in [0\,..\,\sigma)$. Thus, each pair (i_v, c) can be (in $\mathcal{O}(1)$ time) injectively mapped to an integer not exceeding $2\sigma^{3\tau} = \mathcal{O}(\sqrt{n})$ and hence L_{child} needs $\mathcal{O}(n/\log_\sigma n)$ space.
   (c) A lookup table L_{WA} that for every node v of $\mathcal{T}_{3\tau-1}$ and every $d \in [0\,..\,3\tau-1)$, maps the pair (i_v, d) to i_u, where $u = \mathrm{WA}(v, d)$ and i_v (resp. i_u) is the pointer to v (resp. u). Since $i_v < 2\sigma^{3\tau-1}$ and $\tau = \mathcal{O}(\log n)$, each pair (i_v, d) can be injectively mapped to an integer not exceeding $\mathcal{O}(\sqrt{n}\log n)$ and hence L_{WA} needs $\mathcal{O}(n/\log_\sigma n)$ space.
   (d) An array storing the pointers to leaves of $\mathcal{T}_{3\tau-1}$ in the left-to-right order. Since the number of leaves is $\mathcal{O}(\sigma^{3\tau-1})$, the array needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, $\mathrm{C}_{\mathrm{ST}}(T)$ takes $\mathcal{O}(n/\log_\sigma n)$ space.

REMARK 7.1. Note that $\mathcal{T}_{3\tau-1}$ corresponds to $\mathcal{T}_{\text{st}}$ truncated at depth $3\tau-1$. The key reason motivating this definition is that the pair $(b,e) = (\text{RangeBeg}(\text{str}(v),T), \text{RangeEnd}(\text{str}(v),T))$ for every node v of $\mathcal{T}_{3\tau-1}$ at depth $3\tau-1$ that corresponds to an implicit node of $\mathcal{T}_{\text{st}}$ (in the middle of an edge connecting some node v' of $\mathcal{T}_{\text{st}}$ to one of its children v'') satisfies $(b,e) = (\text{RangeBeg}(\text{str}(v''),T), \text{RangeEnd}(\text{str}(v''),T))$. In all our uses, this is sufficient, and the value $\text{sdepth}(v'')$ is never needed.

7.1.2 Navigation Primitives

Mapping from $\mathcal{T}_{\text{st}}$ to $\mathcal{T}_{3\tau-1}$ For any explicit node v of $\mathcal{T}_{\text{st}}$, we define $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v)$ as the deepest explicit node u of $\mathcal{T}_{3\tau-1}$ such that $\text{str}(u)$ is a prefix of $\text{str}(v)$.

LEMMA 7.1. *Let v be an explicit node of $\mathcal{T}_{\text{st}}$ and $u = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v)$. Then, $\text{sdepth}(v) \geq 3\tau-1$ holds if and only if $\text{sdepth}(u) = 3\tau-1$. Moreover,*

1. *If $\text{sdepth}(v) \geq 3\tau-1$ then $\text{str}(u) = \text{str}(v)[1\,..\,3\tau-1]$.*
2. *Otherwise (i.e., if $\text{sdepth}(v) < 3\tau-1$), $\text{str}(u) = \text{str}(v)$.*

Proof. 1. If $\text{sdepth}(v) \geq 3\tau-1$ then $\text{str}(v)[1\,..\,3\tau-1] \in \mathcal{S}_{3\tau-1}$. Therefore, by definition of $\mathcal{T}_{3\tau-1}$, there exists a node u' in $\mathcal{T}_{3\tau-1}$ satisfying $\text{str}(u') = \text{str}(v)[1\,..\,3\tau-1]$. Since $\text{str}(u')$ is a prefix of $\text{str}(v)$ and u' is a leaf of $\mathcal{T}_{3\tau-1}$, we thus have $u' = u$, and hence $\text{str}(u) = \text{str}(v)[1\,..\,3\tau-1]$.

2. Let $\text{sdepth}(v) < 3\tau-1$ and $X = \text{str}(v)$. Since v is explicit, there exists distinct $c, c' \in \Sigma$ such that Xc and Xc' occur in T. By $|X| < 3\tau-1$, $\mathcal{T}_{3\tau-1}$ therefore has an explicit node u' satisfying $\text{str}(u') = X$. By $\text{sdepth}(u') = \text{sdepth}(v)$, we thus have $u' = u$ and hence $\text{str}(u) = \text{str}(v)$.

The equivalence follows immediately from the two items. □

LEMMA 7.2. *Let v be an explicit node of $\mathcal{T}_{\text{st}}$. Let $i_1 = \text{lrank}(v)+1$, $i_2 = \text{rrank}(v)$, $y_1 = \mathsf{rank}_{B_{3\tau-1},1}(i_1-1)+1$, $y_2 = \mathsf{rank}_{B_{3\tau-1},1}(i_2-1)+1$, u_1 (resp. u_2) be the y_1th (resp. y_2th) leftmost leaf of $\mathcal{T}_{3\tau-1}$, and $u = \text{LCA}(u_1,u_2)$. Then, $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v) = u$.*

Proof. By definition of $\mathcal{T}_{3\tau-1}$ and A_{short} (Section 5.1), if $\widehat{u}$ is the kth leftmost leaf of $\mathcal{T}_{3\tau-1}$, then $\text{str}(\widehat{u}) = A_{\text{short}}[k]$. Thus, $\text{str}(u_1) = A_{\text{short}}[y_1]$ and $\text{str}(u_2) = A_{\text{short}}[y_2]$. Denote $Q = \text{str}(v)$ and consider two cases:

- Let $\text{sdepth}(v) \geq 3\tau-1$. Denote $X = Q[1\,..\,3\tau-1]$. By $i_1 = \text{lrank}(v)+1$ and $i_2 = \text{rrank}(v)$, we then have $\text{SA}[i_1], \text{SA}[i_2] \in \text{Occ}(Q,T) \subseteq \text{Occ}(X,T)$. By definition of $B_{3\tau-1}$ and A_{short}, positions $y_1 = \mathsf{rank}_{B_{3\tau-1},1}(i_1-1)+1$ and $y_2 = \mathsf{rank}_{B_{3\tau-1},1}(i_2-1)+1$ then satisfy $A_{\text{short}}[y_1] = A_{\text{short}}[y_2] = X$. Thus, by the above observation, $\text{str}(u_1) = \text{str}(u_2) = X$ and hence, by Observation 4.1, $\text{str}(u) = X = \text{str}(v)[1\,..\,3\tau-1]$. Consequently, by Lemma 7.1(1) and since all nodes of $\mathcal{T}_{3\tau-1}$ have different value of str, this yields $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v) = u$.
- Let us now assume $\text{sdepth}(v) < 3\tau-1$. Let v_1 (resp. v_2) be the i_1th (resp. i_2th) leftmost leaf of $\mathcal{T}_{\text{st}}$. Then, $\text{str}(v_1) = T[\text{SA}[i_1]\,..\,n]$ and $\text{str}(v_2) = T[\text{SA}[i_2]\,..\,n]$. By $i_1 = \text{lrank}(v)+1$ and $i_2 = \text{rrank}(v)$ we have $v = \text{LCA}(v_1,v_2)$. Thus, by Observation 4.1, $\text{lcp}(T[\text{SA}[i_1]\,..\,n], T[\text{SA}[i_2]\,..\,n]) = \text{sdepth}(v) = |Q|$. Observe now that:
  - By definition of A_{short} and $B_{3\tau-1}$, the string $A_{\text{short}}[y_1]$ (resp. $A_{\text{short}}[y_2]$) is a prefix of $T[\text{SA}[i_1]\,..\,n]$ (resp. $T[\text{SA}[i_2]\,..\,n]$). Thus, it holds $\text{lcp}(A_{\text{short}}[y_1], A_{\text{short}}[y_2]) \leq \text{lcp}(T[\text{SA}[i_1]\,..\,n], T[\text{SA}[i_2]\,..\,n]) = |Q|$.
  - On the other hand, since Q is a prefix of $\text{str}(v_1) = T[\text{SA}[i_1]\,..\,n]$ and $\text{str}(v_2) = T[\text{SA}[i_2]\,..\,n]$, and it holds $|A_{\text{short}}[y_1]| = \min(3\tau-1, n-\text{SA}[i_1]+1)$, $|A_{\text{short}}[y_2]| = \min(3\tau-1, n-\text{SA}[i_2]+1)$, and $|Q| < 3\tau-1$, we obtain that Q is a prefix of $A_{\text{short}}[y_1]$ and $A_{\text{short}}[y_2]$. Thus, $\text{lcp}(A_{\text{short}}[y_1], A_{\text{short}}[y_2]) \geq |Q|$.

  We thus proved that Q is a prefix of $A_{\text{short}}[y_1]$ and $A_{\text{short}}[y_2]$, and $\text{lcp}(A_{\text{short}}[y_1], A_{\text{short}}[y_2]) = |Q|$. Thus, since $\text{str}(u_1) = A_{\text{short}}[y_1]$ and $\text{str}(u_2) = A_{\text{short}}[y_2]$, we obtain from Observation 4.1, that $u = \text{LCA}(u_1,u_2)$ satisfies $\text{str}(u) = Q$. By Lemma 7.1(2) and since all nodes of $\mathcal{T}_{3\tau-1}$ have different value of str, this yields $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v) = u$. □

PROPOSITION 7.1. *Let v be an explicit node of $\mathcal{T}_{\text{st}}$. Given $\text{C}_{\text{ST}}(T)$ and $\text{repr}(v)$, in $\mathcal{O}(1)$ time we can compute the pointer to the node $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v)$.*

Proof. Denote $(b,e) = \mathrm{repr}(v)$, $i_1 = b+1$, and $i_2 = e$. First, in $\mathcal{O}(1)$ time we compute $y_1 = \mathsf{rank}_{B_{3\tau-1},1}(i_1-1)+1$ and $y_2 = \mathsf{rank}_{B_{3\tau-1},1}(i_2-1)+1$. In $\mathcal{O}(1)$ time we then retrieve the y_1th and y_2th leftmost leaves u_1 and u_2 of $\mathcal{T}_{3\tau-1}$ (respectively). Finally, using the LCA structure for $\mathcal{T}_{3\tau-1}$, we compute in $\mathcal{O}(1)$ time the pointer to node $u = \mathrm{LCA}(u_1,u_2)$ of $\mathcal{T}_{3\tau-1}$. By Lemma 7.2, we then have $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v) = u$. □

PROPOSITION 7.2. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given $\mathrm{C}_{\mathrm{ST}}(T)$ and $\mathrm{repr}(v)$, we can in $\mathcal{O}(1)$ time check if v is periodic. If v is not periodic, then in $\mathcal{O}(1)$ we can additionally determine if it holds $\mathrm{sdepth}(v) < 3\tau-1$.*

Proof. First, using Proposition 7.1, in $\mathcal{O}(1)$ time we compute the pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. If $\mathrm{sdepth}(u) = 3\tau-1$ then by Lemma 7.1 it holds $\mathrm{sdepth}(v) \geq 3\tau-1$, and we can in $\mathcal{O}(1)$ time determine if v is periodic by checking if $\mathrm{per}(X) \leq \frac{1}{3}\tau$ for $X = \mathrm{str}(v)[1\,..\,3\tau-1] = \mathrm{str}(u)$ (stored with u) using the lookup table L_{per}. If $\mathrm{sdepth}(u) < 3\tau-1$, then by Lemma 7.1, we have $\mathrm{sdepth}(v) < 3\tau-1$, and hence v is nonperiodic. Note that in the above algorithm, whenever v is nonperiodic, we always know if $\mathrm{sdepth}(v) < 3\tau-1$. Thus, we can additionally return this information at no extra cost. Each of the steps takes $\mathcal{O}(1)$ time. □

7.1.3 Implementation of $\mathrm{LCA}(u,v)$

LEMMA 7.3. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\mathrm{st}}$. Then,*

$$\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(\mathrm{LCA}(v_1,v_2)) = \mathrm{LCA}(\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_1), \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_2)).$$

Proof. Let $u_1 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_1)$, $u_2 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_2)$, $v = \mathrm{LCA}(v_1,v_2)$, and $u = \mathrm{LCA}(u_1,u_2)$. Then, the claim is that $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v) = u$. Denote $\ell = \mathrm{lcp}(\mathrm{str}(v_1),\mathrm{str}(v_2))$ and recall that by Observation 4.1, we have $\mathrm{sdepth}(v) = \ell$. We consider two cases:

- First, assume $\mathrm{sdepth}(v) \geq 3\tau-1$. By $\mathrm{sdepth}(v_1) \geq \mathrm{sdepth}(v) \geq 3\tau-1$, we obtain from Lemma 7.1(1) and Observation 4.1 that $\mathrm{str}(u_1) = \mathrm{str}(v_1)[1\,..\,3\tau-1] = \mathrm{str}(v)[1\,..\,3\tau-1]$. Analogously, $\mathrm{str}(u_2) = \mathrm{str}(v)[1\,..\,3\tau-1]$, and consequently, $\mathrm{str}(u) = \mathrm{str}(v)[1\,..\,3\tau-1]$. Since all nodes in $\mathcal{T}_{3\tau-1}$ have different values of str, by Lemma 7.1(1), this implies $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$.
- Let us now assume $\mathrm{sdepth}(v) < 3\tau-1$. We will show that then $\mathrm{str}(u) = \mathrm{str}(v)$. Since all nodes in $\mathcal{T}_{3\tau-1}$ have different values of str, by Lemma 7.1(2), this immediately implies $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. We first show that $\mathrm{sdepth}(u_1) \geq \ell$. Consider two cases. If $\mathrm{sdepth}(v_1) \geq 3\tau-1$, then by Lemma 7.1(1), $\mathrm{str}(u_1) = \mathrm{str}(v_1)[1\,..\,3\tau-1]$, i.e., $\mathrm{sdepth}(u_1) = 3\tau-1 > \mathrm{sdepth}(v) = \ell$. Otherwise, by Lemma 7.1(2), it holds $\mathrm{str}(u_1) = \mathrm{str}(v_1)$, and thus also $\mathrm{sdepth}(u_1) = \mathrm{sdepth}(v_1) \geq \mathrm{sdepth}(v) = \ell$. By the analogous argument, $\mathrm{sdepth}(u_2) \geq \ell$. Recall now that, by definition, $\mathrm{str}(u_1)$ (resp. $\mathrm{str}(u_2)$) is a prefix of $\mathrm{str}(v_1)$ (resp. $\mathrm{str}(v_2)$). Thus, $\mathrm{str}(u_1)[1\,..\,\ell] = \mathrm{str}(v_1)[1\,..\,\ell] = \mathrm{str}(v_2)[1\,..\,\ell] = \mathrm{str}(u_2)[1\,..\,\ell]$. Denoting $\ell' = \mathrm{lcp}(\mathrm{str}(u_1),\mathrm{str}(u_2))$, we therefore have $\ell' \geq \ell$. On the other hand, $\mathrm{str}(u_1)$ (resp. $\mathrm{str}(u_2)$) being a prefix of $\mathrm{str}(v_1)$ (resp. $\mathrm{str}(v_2)$), implies $\ell' \leq \ell$. Consequently, $\ell' = \ell$. By Observation 4.1, we therefore obtain $\mathrm{str}(u) = \mathrm{str}(\mathrm{LCA}(u_1,u_2)) = \mathrm{str}(u_1)[1\,..\,\ell'] = \mathrm{str}(v_1)[1\,..\,\ell] = \mathrm{str}(\mathrm{LCA}(v_1,v_2)) = \mathrm{str}(v)$. □

PROPOSITION 7.3. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\mathrm{st}}$. Given $\mathrm{C}_{\mathrm{ST}}(T)$ and the pairs $\mathrm{repr}(v_1)$ and $\mathrm{repr}(v_2)$, we can in $\mathcal{O}(1)$ time check if $\mathrm{sdepth}(\mathrm{LCA}(v_1,v_2)) \geq 3\tau-1$. If so, in $\mathcal{O}(1)$ time we can additionally determine if $\mathrm{LCA}(v_1,v_2)$ is periodic. Otherwise (i.e., if $\mathrm{sdepth}(\mathrm{LCA}(v_1,v_2)) < 3\tau-1$) in $\mathcal{O}(1)$ time we can compute $\mathrm{repr}(\mathrm{LCA}(v_1,v_2))$.*

Proof. Denote $v = \mathrm{LCA}(v_1,v_2)$. First, using Proposition 7.1, in $\mathcal{O}(1)$ time we compute pointers to $u_1 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_1)$ and $u_2 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v_2)$ of $\mathcal{T}_{3\tau-1}$. Then, using the LCA structure for $\mathcal{T}_{3\tau-1}$, we compute in $\mathcal{O}(1)$ time the pointer to node $u = \mathrm{LCA}(u_1,u_2)$ of $\mathcal{T}_{3\tau-1}$. By Lemma 7.3, we now have $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v) = u$. If $\mathrm{sdepth}(u) = 3\tau-1$, by Lemma 7.1 it holds $\mathrm{sdepth}(v) \geq 3\tau-1$ and $\mathrm{str}(u) = \mathrm{str}(v)[1\,..\,3\tau-1]$, and thus we can in $\mathcal{O}(1)$ determine if v is periodic by checking if $\mathrm{per}(X) \leq \frac{1}{3}\tau$ for $X = \mathrm{str}(u)$ (stored with u) using the lookup table L_{per}. Otherwise (i.e., if $\mathrm{sdepth}(u) < 3\tau-1$), by Lemma 7.1 we have $\mathrm{sdepth}(v) < 3\tau-1$ and $\mathrm{str}(u) = \mathrm{str}(v)$. Thus, we return that v is nonperiodic and in $\mathcal{O}(1)$ time we obtain the pair $\mathrm{repr}(v) = (\mathrm{RangeBeg}(\mathrm{str}(v),T),\mathrm{RangeEnd}(\mathrm{str}(v),T)) = (\mathrm{RangeBeg}(\mathrm{str}(u),T),\mathrm{RangeEnd}(\mathrm{str}(u),T))$ using the lookup table L_{range} on $\mathrm{str}(u)$. Each of the steps takes $\mathcal{O}(1)$ time. □

7.1.4 Implementation of $\mathrm{child}(v,c)$

LEMMA 7.4. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) < 3\tau-1$. Let $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. For any $c \in [0\,..\,\sigma)$, $\mathrm{child}(v,c) = \perp$ holds if and only if $\mathrm{child}(u,c) = \perp$. Moreover, if $\mathrm{child}(v,c) \neq \perp$ then, letting $u' = \mathrm{child}(u,c)$, it holds*

$$\mathrm{repr}(\mathrm{child}(v,c)) = (\mathrm{RangeBeg}(\mathrm{str}(u'),T), \mathrm{RangeEnd}(\mathrm{str}(u'),T)).$$

Proof. By Lemma 7.1(2), u satisfies $\mathrm{str}(u) = \mathrm{str}(v)$. Thus, since $\mathcal{T}_{3\tau-1}$ is a compact trie of substrings of T of length $3\tau-1$, we immediately obtain that for any $c \in [0\,..\,\sigma)$, $\mathrm{child}(v,c) \neq \perp$ if and only if $\mathrm{child}(u,c)$. Let us assume for some $c \in [0\,..\,\sigma)$, it holds $\mathrm{child}(v,c) = v' \neq \perp$. If $\mathrm{sdepth}(v') \leq 3\tau-1$, then by definition of $\mathcal{T}_{3\tau-1}$, the node $u' = \mathrm{child}(u,c)$ must satisfy $\mathrm{str}(v') = \mathrm{str}(u')$. This implies the claim immediately. Otherwise ($\mathrm{sdepth}(v') > 3\tau-1$), u' satisfies $\mathrm{sdepth}(u') = 3\tau-1$, and corresponds to the implicit node of $\mathcal{T}_{\mathrm{st}}$ on the edge connecting v to v'. By definition of suffix tree, however, letting S be such that $\mathrm{str}(v)S = \mathrm{str}(v')[1\,..\,3\tau-1]$, we have $(\mathrm{RangeBeg}(\mathrm{str}(v'),T), \mathrm{RangeEnd}(\mathrm{str}(v'),T)) = (\mathrm{RangeBeg}(\mathrm{str}(v)S,T), \mathrm{RangeEnd}(\mathrm{str}(v)S,T)) = (\mathrm{RangeBeg}(\mathrm{str}(u'),T), \mathrm{RangeEnd}(\mathrm{str}(u'),T))$, which by definition of repr implies the claim. □

PROPOSITION 7.4. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) < 3\tau-1$. Given $\mathrm{C}_{\mathrm{ST}}(T)$, $\mathrm{repr}(v)$, and $c \in [0\,..\,\sigma)$, we can in $\mathcal{O}(1)$ time compute $\mathrm{repr}(\mathrm{child}(v,c))$.*

Proof. First, using Proposition 7.1, in $\mathcal{O}(1)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. Using the lookup table L_{child}, in $\mathcal{O}(1)$ time we check if $\mathrm{child}(u,c) = \perp$. If so, then by Lemma 7.4, it holds $\mathrm{child}(v,c) = \perp$ and we return $\mathrm{repr}(\mathrm{child}(v,c)) = (0,0)$. Otherwise (i.e., $\mathrm{child}(u,c) \neq \perp$), we obtain a pointer to $u' = \mathrm{child}(u,c)$. By Lemma 7.4, we then have $\mathrm{repr}(\mathrm{child}(v,c)) = (\mathrm{RangeBeg}(\mathrm{str}(u'),T), \mathrm{RangeEnd}(\mathrm{str}(u'),T))$, which we obtain using the lookup table L_{range} on $\mathrm{str}(u')$. Each of the steps takes $\mathcal{O}(1)$ time. □

7.1.5 Implementation of $\mathrm{pred}(v,c)$

PROPOSITION 7.5. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) < 3\tau-1$. Given $\mathrm{C}_{\mathrm{ST}}(T)$, $\mathrm{repr}(v)$, and $c \in [0\,..\,\sigma)$, we can in $\mathcal{O}(1)$ time compute $\mathrm{RangeBeg}(\mathrm{str}(v)c,T)$.*

Proof. First, using Proposition 7.1 we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. By $\mathrm{sdepth}(v) < 3\tau-1$ and Lemma 7.1(2), node u satisfies $\mathrm{str}(u) = \mathrm{str}(v)$. Thus, we have $\mathrm{RangeBeg}(\mathrm{str}(v)c,T) = \mathrm{RangeBeg}(\mathrm{str}(u)c,T)$. Next, we compute $Y = \mathrm{str}(u)c$ (recall, that $\mathrm{str}(u)$ is stored with u). Using the lookup table L_{range}, we then compute and return $\mathrm{RangeBeg}(Y,T)$. Each of the steps takes $\mathcal{O}(1)$ time. □

7.1.6 Implementation of $\mathrm{WA}(v,d)$

LEMMA 7.5. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$ and d be such that $0 \leq d \leq |\mathrm{str}(v)|$ and $d < 3\tau-1$. Then, letting $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$ and $u' = \mathrm{WA}(u,d)$, it holds*

$$\mathrm{repr}(\mathrm{WA}(v,d)) = (\mathrm{RangeBeg}(\mathrm{str}(u'),T), \mathrm{RangeEnd}(\mathrm{str}(u'),T)).$$

Proof. Denote $v' = \mathrm{WA}(v,d)$. We consider two cases:

- First, assume $\mathrm{sdepth}(v) \geq 3\tau-1$. By Lemma 7.1(1), we then have $\mathrm{str}(u) = \mathrm{str}(v)[1\,..\,3\tau-1]$. Therefore, utilizing one of the assumptions about d, we have $d < 3\tau-1 = \mathrm{sdepth}(u)$, i.e., u' is well-defined (see Section 4.1). Moreover, this implies that for any ancestor $\bar{v}$ of v at depth at most $3\tau-1$, there exist a corresponding ancestor $\bar{u}$ of u and there exists a one-to-one mapping between ancestors of $\bar{v}$ in $\mathcal{T}_{\mathrm{st}}$ and ancestors of $\bar{u}$ in $\mathcal{T}_{3\tau-1}$ (with corresponding nodes having equal root-to-node labels). Therefore, if $\mathrm{sdepth}(v') \leq 3\tau-1$ then $\mathrm{str}(u') = \mathrm{str}(v')$ and the claim follows. Otherwise ($\mathrm{sdepth}(v') > 3\tau-1$), by $d < 3\tau-1$, we must have $u' = u$ and u' then corresponds to the implicit node on the edge connecting v' to $\mathrm{parent}(v')$. This implies $\mathrm{repr}(v') = (\mathrm{RangeBeg}(\mathrm{str}(u'),T), \mathrm{RangeEnd}(\mathrm{str}(u'),T))$.
- Let us now assume $\mathrm{sdepth}(v) < 3\tau-1$. By Lemma 7.1(2), we then have $\mathrm{str}(u) = \mathrm{str}(v)$. In particular, utilizing one of the assumptions on d, we have $d \leq \mathrm{sdepth}(v) = \mathrm{sdepth}(u)$, i.e., u' is well-defined (see Section 4.1). Moreover, this implies that there is a one-to-one correspondence between ancestors of v in $\mathcal{T}_{\mathrm{st}}$ and ancestors or u in $\mathcal{T}_{3\tau-1}$. In particular, $\mathrm{str}(v') = \mathrm{str}(u')$, which implies the claim. □

PROPOSITION 7.6. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given $\mathrm{C}_{\mathrm{ST}}(T)$, $\mathrm{repr}(v)$, and an integer d satisfying $0 \le d \le |\mathrm{str}(v)|$ and $d < 3\tau - 1$, in $\mathcal{O}(1)$ time we can compute $\mathrm{repr}(\mathrm{WA}(v, d))$.*

Proof. First, using Proposition 7.1, we compute a pointer to node $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{3\tau-1}}(v)$. Then, using the lookup table L_{WA}, in $\mathcal{O}(1)$ time we obtain the pointer to $u' = \mathrm{WA}(u, d)$. By Lemma 7.5, we then have $\mathrm{repr}(\mathrm{WA}(v, d)) = (\mathrm{RangeBeg}(\mathrm{str}(u'), T), \mathrm{RangeEnd}(\mathrm{str}(u'), T))$, which is obtained using the lookup table L_{range} on $\mathrm{str}(u')$. Each of the steps takes $\mathcal{O}(1)$ time. □

7.1.7 Construction Algorithm

PROPOSITION 7.7. *Given the packed representation of $T \in [0\,..\,\sigma)^n$, we can construct $\mathrm{C}_{\mathrm{ST}}(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time.*

Proof. First, in $\mathcal{O}(n/\log_\sigma n)$ time we construct $\mathrm{C}_{\mathrm{SA}}(T)$ using Proposition 5.3. Note that during the construction, we compute the frequency $f_X = |\mathrm{Occ}(X, T)|$ for every $X \in [0\,..\,\sigma)^{\le 3\tau-1}$ (note that by definition of $\mathrm{Occ}(X, T)$ (Section 2), we have $f_X = n$ for the empty string $X = \varepsilon$).

Next, we construct the trie $\mathcal{T}_{3\tau-1}$ and the associated data structures. Observe that for every $X \in [0\,..\,\sigma)^{\le 3\tau-1}$, the trie $\mathcal{T}_{3\tau-1}$ contains an explicit node v satisfying $\mathrm{str}(v) = X$ if and only if $f_X > 0$, and either $|X| = 3\tau - 1$ or $|X| < 3\tau - 1$ and there exist distinct $c, c' \in [0\,..\,\sigma)$ such that $f_{Xc} > 0$ and $f_{Xc'} > 0$. [10] Thus, given any X, we can in $\mathcal{O}(\sigma)$ time check if there exists a node of $\mathcal{T}_{3\tau-1}$ corresponding to X. Moreover, if such v exists and $|X| > 0$ then to find X' satisfying $\mathrm{str}(\mathrm{parent}(v)) = X'$, it suffices to compute the longest prefix X' of X such that L_{range} for X' is different from L_{range} for X. Thus, such X' can be computed in $\mathcal{O}(\tau) = \mathcal{O}(\log n)$ time. Using the above observations, we construct $\mathcal{T}_{3\tau-1}$ as follows. We maintain a lookup table L_{node} that for any $X \in [0\,..\,\sigma)^{\le 3\tau-1}$ maps the integer $\mathrm{int}(X)$ to a pointer to the node v of $\mathcal{T}_{3\tau-1}$ satisfying $\mathrm{str}(v) = X$ if such v exists. By $\mathrm{int}(X) \in [0\,..\,\sigma^{6\tau})$, the table needs $\mathcal{O}(\sigma^{6\tau}) = \mathcal{O}(n/\log_\sigma n)$ space and its initialization takes $\mathcal{O}(n/\log_\sigma n)$ time. During the construction, nodes are stored in a dynamic array with amortized $\mathcal{O}(1)$-time insertion at the end, and pointers are implemented as indexes of this array. We enumerate all $X \in [0\,..\,\sigma)^{\le 3\tau-1}$ in the order of non-decreasing length, and in case of ties, in lexicographical order. For each X, using the above method in $\mathcal{O}(\sigma)$ time we check whether there should be a node in $\mathcal{T}_{3\tau-1}$ satisfying $\mathrm{str}(v) = X$. If so, we create a new node v, add it to the array of nodes, and update the lookup table L_{node}. Associated with v we store the string X encoded as $\mathrm{int}(X)$ and the length $|X| = \mathrm{sdepth}(v)$. If $|X| > 0$, in $\mathcal{O}(\log n)$ time we then compute the longest prefix X' of X for which $(\mathrm{RangeBeg}(X', T), \mathrm{RangeEnd}(X', T)) \neq (\mathrm{RangeBeg}(X, T), \mathrm{RangeEnd}(X, T))$ (utilizing the lookup table L_{range}), and then using L_{node} obtain v' satisfying $\mathrm{str}(v') = X'$. We then set $\mathrm{parent}(v) = v'$ and add v to the list of children of v', updating also the links between children of v'. Over all $X \in [0\,..\,\sigma)^{\le 3\tau-1}$, the construction takes $\mathcal{O}(\sigma^{3\tau-1}(\sigma + \log n)) = \mathcal{O}(n/\log_\sigma n)$ time. After constructing $\mathcal{T}_{3\tau-1}$, we augment it with auxiliary structures as follows:

(a) In $\mathcal{O}(\sigma^{3\tau-1}) = \mathcal{O}(n/\log_\sigma n)$ time we preprocess $\mathcal{T}_{3\tau-1}$ for $\mathcal{O}(1)$-time LCA queries using the structure from [10].

(b) Next, we perform a traversal of $\mathcal{T}_{3\tau-1}$. For each node v different from the root, we obtain the pointer to $p = \mathrm{parent}(v)$ and $c = \mathrm{str}(v)[|\mathrm{str}(p)| + 1]$. We then injectively map (i_p, c) (where i_p is the pointer to p) to an integer x not exceeding $2\sigma^{3\tau} = \mathcal{O}(\sqrt{n})$ and set $L_{\mathrm{child}}[x] := i_v$, where i_v is the pointer to v. The construction takes $\mathcal{O}(\sqrt{n}) = \mathcal{O}(n/\log_\sigma n)$ time.

(c) Next, starting from each node v of $\mathcal{T}_{3\tau-1}$ we compute the pointer to $\mathrm{WA}(v, d)$ for every $d \in [0\,..\,3\tau - 1)$. It suffices to perform one traversal towards the roots and thus this takes $\mathcal{O}(\log n)$ time per node (in total for all d). For each computed node v', we map the pair (i_v, d) (where i_v is the pointer to v) to an integer x not exceeding $\mathcal{O}(\sqrt{n}\log n)$ and set $L_{\mathrm{WA}}[x] := i_{v'}$, where $i_{v'}$ is the pointer to v'. Including the initialization of L_{WA}, the construction takes $\mathcal{O}(\sqrt{n}\log n) = \mathcal{O}(n/\log_\sigma n)$ time.

(d) Finally, we perform the in-order traversal of the tree, collecting the leaves of $\mathcal{T}_{3\tau-1}$ in an array. By the bound on the number of nodes, this takes $\mathcal{O}(n/\log_\sigma n)$ time. □

[10] Note, that this holds also for $X = \varepsilon$ because we defined $f_\varepsilon = n$ and assumed in Section 2 that T contains at least two distinct symbols.

7.2 The Nonperiodic Nodes

In this section, we describe a data structure used to perform operations on nonperiodic nodes (see Definition 7.1) in $\mathcal{O}(\log^{\epsilon} n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 7.2.1). We then show how using this structure to implement some basic navigational routines (Section 7.2.2). Next, we describe the query algorithms for the fundamental operations (Sections 7.2.3 to 7.2.6). Finally, we show the construction algorithm (Section 7.2.7).

7.2.1 The Data Structure

Definitions Let S be a τ-synchronizing set, as defined in Section 5.2.1. Recall (Section 6.2.1) that $A_{\mathsf{S}}[1\,..\,n']$ is an array defined by $A_{\mathsf{S}}[i] = s_i^{\text{lex}}$. Let $\mathcal{T}_{\mathsf{S}}$ denote the compact trie of the set $\{T[i\,..\,n] : i \in \mathsf{S}\}$.

Components The data structure to handle nonperiodic nodes consists of three components:

1. The index core $\mathrm{C}_{\mathrm{ST}}(T)$ (Section 7.1.1). It takes $\mathcal{O}(n/\log_\sigma n)$ space.
2. The data structure from Section 5.2.1 using $\mathcal{O}(n/\log_\sigma n)$ space.
3. The compact trie $\mathcal{T}_{\mathsf{S}}$ represented as in Proposition 4.1 (i.e., for the array $A_{\mathsf{S}}[1\,..\,n']$ defined above). By $n' = \mathcal{O}(n/\log_\sigma n)$ and Proposition 4.1, it needs $\mathcal{O}(n/\log_\sigma n)$ space.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

7.2.2 Navigation Primitives

Mapping from $\mathcal{T}_{\mathrm{st}}$ to $\mathcal{T}_{\mathsf{S}}$ For any explicit nonperiodic node v of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) \geq 3\tau - 1$, we define $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$ as a node of $\mathcal{T}_{\mathsf{S}}$ satisfying $\mathrm{str}(v) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u)$, where $X \in \mathcal{D}$ is a prefix of $\mathrm{str}(v)$ and $\delta_{\mathrm{text}} = |X| - 2\tau$ (such X exists and is unique, since for $Y = \mathrm{str}(v)[1\,..\,3\tau{-}1]$ it holds $\mathrm{Occ}(Y,T) \neq \emptyset$ and $\mathrm{per}(Y) > \frac{1}{3}\tau$; see Section 5.2).

Lemma 7.6. *Let v be an explicit nonperiodic node of $\mathcal{T}_{\mathrm{st}}$ satisfying* $\mathrm{sdepth}(v) \geq 3\tau - 1$.

1. *The node* $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{S}}}(v)$ *is well-defined.*
2. *Let $X \in \mathcal{D}$ be a prefix of* $\mathrm{str}(v)$, $b_X = \mathrm{RangeBeg}(X,T)$, $i_1 = \mathrm{lrank}(v) + 1$, $i_2 = \mathrm{rrank}(v)$, $y_1 = \mathsf{select}_{W,\overline{X}}(i_1 - b_X)$, $y_2 = \mathsf{select}_{W,\overline{X}}(i_2 - b_X)$, u_1 *(resp. u_2) be the y_1th (resp. y_2th) leftmost leaf of $\mathcal{T}_{\mathsf{S}}$, and* $u = \mathrm{LCA}(u_1,u_2)$. *Then,* $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$.

Proof. 1. Let $X \in \mathcal{D}$ be a prefix of $\mathrm{str}(v)$ and let $\delta_{\mathrm{text}} = |X| - 2\tau$. If v is a leaf of $\mathcal{T}_{\mathrm{st}}$, then for $i \in \mathrm{Occ}(\mathrm{str}(v),T)$, it holds that X is a prefix of $T[i\,..\,n]$. Thus, by the consistency of S, $i + \delta_{\mathrm{text}} \in \mathsf{S}$, and consequently, there exist u in $\mathcal{T}_{\mathsf{S}}$ such that $\mathrm{str}(v) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u)$. Otherwise (i.e., v is an internal node), consider any two different leaves v_1 and v_2 in the subtree rooted in v such that $v = \mathrm{LCA}(v_1,v_2)$. Let $i_1 \in \mathrm{Occ}(\mathrm{str}(v_1),T)$ and $i_2 \in \mathrm{Occ}(\mathrm{str}(v_2),T)$. Then, $\mathrm{str}(v)$ is a prefix of both $T[i_1\,..\,n]$ and $T[i_2\,..\,n]$. Since X is a prefix of $\mathrm{str}(v)$, X is therefore also a prefix of $T[i_1\,..\,n]$ and $T[i_2\,..\,n]$. Thus, again by the consistency of S, we have $i_1 + \delta_{\mathrm{text}}, i_2 + \delta_{\mathrm{text}} \in \mathsf{S}$. Consequently, there exist nodes u_1 and u_2 in $\mathcal{T}_{\mathsf{S}}$ satisfying $\mathrm{str}(v_1) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_1)$ and $\mathrm{str}(v_2) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_2)$. By Observation 4.1 applied to v_1 and v_2, for $\ell = \mathrm{lcp}(\mathrm{str}(v_1),\mathrm{str}(v_2))$ it holds $\mathrm{str}(v) = \mathrm{str}(v_1)[1\,..\,\ell]$. On the other hand, applying Observation 4.1 to u_1 and u_2 implies that for $\ell' = \mathrm{lcp}(\mathrm{str}(u_1),\mathrm{str}(u_2))$ and $u = \mathrm{LCA}(u_1,u_2)$, it holds $\mathrm{str}(u) = \mathrm{str}(u_1)[1\,..\,\ell']$. Finally, by $\delta_{\mathrm{text}} < |X|$, we have $\ell = \mathrm{lcp}(\mathrm{str}(v_1),\mathrm{str}(v_2)) = \mathrm{lcp}(X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_1), X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_2)) = \delta_{\mathrm{text}} + \mathrm{lcp}(\mathrm{str}(u_1),\mathrm{str}(u_2)) = \delta_{\mathrm{text}} + \ell'$. Thus,

$$\begin{aligned}\mathrm{str}(v) &= \mathrm{str}(v_1)[1\,..\,\ell] \\ &= X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_1)[1\,..\,\ell - \delta_{\mathrm{text}}] \\ &= X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u_1)[1\,..\,\ell'] \\ &= X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u),\end{aligned}$$

i.e., there exists u in $\mathcal{T}_{\mathsf{S}}$ satisfying $\mathrm{str}(v) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u)$, i.e., $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{S}}}(v)$ is well-defined.

2. Let $\delta_{\text{text}} = |X| - 2\tau$. To see that y_1 and y_2 in the definition are well-defined (i.e., that $i_1 - b_X, i_2 - b_X \in [1\,..\,\mathsf{rank}_{W,\overline{X}}(n')]$), recall first that (similarly as in the proof of Lemmas 5.1 and 6.1) by consistency of S, there exists a bijection (given by $j \mapsto j + \delta_{\text{text}}$) between $\mathrm{Occ}(X,T)$ and positions $s \in \mathsf{S}$ such that $T^{\infty}[s - \delta_{\text{text}}\,..\,s + 2\tau) = X$. In particular, by definition of $W[1\,..\,n']$, this implies $\mathsf{rank}_{W,\overline{X}}(n') = |\mathrm{Occ}(X,T)| = e_X - b_X$, where $e_X = \mathrm{RangeEnd}(X,T)$. Observe now that in $\mathcal{T}_{\text{st}}$, for any node v, it holds $\mathrm{lrank}(v) = \mathrm{RangeBeg}(\mathrm{str}(v),T)$ and $\mathrm{rrank}(v) = \mathrm{RangeEnd}(\mathrm{str}(v),T)$ (this property does not hold, e.g., in $\mathcal{T}_{\mathsf{S}}$). Thus, since X is a prefix of $\mathrm{str}(v)$, we have $b_X < i_1 \le i_2 \le e_X$. Combining with the above, we thus obtain $1 \le i_1 - b_X \le i_2 - b_X \le \mathsf{rank}_{W,\overline{X}}(n')$.

Let v_1 (resp. v_2) be the i_1th (resp. i_2th) leftmost leaf of $\mathcal{T}_{\text{st}}$. Denote $\mathrm{str}(v) = Q$. By $i_1, i_2 \in (b_X\,..\,e_X]$, the string X is a prefix of $\mathrm{str}(v_1)$ and $\mathrm{str}(v_2)$. Since $v = \mathrm{LCA}(v_1,v_2)$, X is therefore also a prefix of $\mathrm{str}(v)$. To show $\mathrm{str}(v) = X[1\,..\,\delta_{\text{text}}] \cdot \mathrm{str}(u)$, it thus suffices to show $\mathrm{str}(u) = Q(\delta_{\text{text}}\,..\,|Q|]$. To this end, we will prove that $Q(\delta_{\text{text}}\,..\,|Q|]$ is a prefix of $\mathrm{str}(u_1)$ and $\mathrm{str}(u_2)$, and that it holds $\mathrm{lcp}(\mathrm{str}(u_1),\mathrm{str}(u_1)) = |Q| - \delta_{\text{text}}$. By $u = \mathrm{LCA}(u_1,u_2)$, this immediately implies the claim. Let $i \in (b_X\,..\,e_X]$. By Lemma 5.1 for $j = \mathrm{SA}[i]$, if $s^{\text{lex}}_y = \mathrm{SA}[i] + \delta_{\text{text}}$ then $\delta(\mathrm{SA}[i]) = \mathsf{rank}_{W,\overline{X}}(y)$. Since by definition of b_X it holds $\delta(\mathrm{SA}[i]) = i - b_X$, we obtain $i - b_X = \mathsf{rank}_{W,\overline{X}}(y)$. Since y also satisfies $T[s^{\text{lex}}_y - \delta_{\text{text}}\,..\,s^{\text{lex}}_y + 2\tau) = X$, it must hold $y = \mathsf{select}_{W,\overline{X}}(i - b_X)$. For such y we have $s^{\text{lex}}_y = \mathrm{SA}[i] + \delta_{\text{text}}$. Applied for i_1 and i_2, we obtain $s^{\text{lex}}_{y_1} = \mathrm{SA}[i_1] + \delta_{\text{text}}$ and $s^{\text{lex}}_{y_2} = \mathrm{SA}[i_2] + \delta_{\text{text}}$. Recall now that the sequence $(s^{\text{lex}}_i)_{i \in [1\,..\,n']}$ contain the positions in S sorted according to the lexicographical order of the corresponding suffixes of T. This implies that the y_1th (resp. y_2th) leftmost leaf u_1 (resp. u_2) of $\mathcal{T}_{\mathsf{S}}$ satisfies $\mathrm{str}(u_1) = T[s^{\text{lex}}_{y_1}\,..\,n] = T[\mathrm{SA}[i_1] + \delta_{\text{text}}\,..\,n]$ (resp. $\mathrm{str}(u_2) = T[s^{\text{lex}}_{y_2}\,..\,n] = T[\mathrm{SA}[i_2] + \delta_{\text{text}}\,..\,n]$). Since all suffixes of T with starting positions in $\mathrm{SA}(\mathrm{lrank}(v)\,..\,\mathrm{rrank}(v)]$ have Q as a prefix, and we clearly have $i_1, i_2 \in (\mathrm{lrank}(v)\,..\,\mathrm{rrank}(v)]$, we immediately obtain that $Q(\delta_{\text{text}}\,..\,|Q|]$ is a prefix of both $T[\mathrm{SA}[i_1] + \delta_{\text{text}}\,..\,n] = \mathrm{str}(u_1)$ and $T[\mathrm{SA}[i_2] + \delta_{\text{text}}\,..\,n] = \mathrm{str}(u_2)$. To show the second claim, we first note that we have $\mathrm{lcp}(T[\mathrm{SA}[i_1]\,..\,n], T[\mathrm{SA}[i_2]\,..\,n]) = \mathrm{lcp}(\mathrm{str}(v_1),\mathrm{str}(v_2)) = |\mathrm{str}(\mathrm{LCA}(v_1,v_2))| = |\mathrm{str}(v)| = |Q|$. Together with $\delta_{\text{text}} \le |X| \le |Q|$, this implies $\mathrm{lcp}(\mathrm{str}(u_1),\mathrm{str}(u_2)) = \mathrm{lcp}(T[\mathrm{SA}[i_1]+\delta_{\text{text}}\,..\,n], T[\mathrm{SA}[i_2]+\delta_{\text{text}}\,..\,n]) = \mathrm{lcp}(T[\mathrm{SA}[i_1]\,..\,n], T[\mathrm{SA}[i_2]\,..\,n]) - \delta_{\text{text}} = |Q| - \delta_{\text{text}}$. As noticed earlier, these two facts yield $\mathrm{str}(u) = Q(\delta_{\text{text}}\,..\,|Q|]$, and consequently $\mathrm{str}(v) = X[1\,..\,\delta_{\text{text}}] \cdot \mathrm{str}(u)$. Thus, $\mathrm{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$. □

PROPOSITION 7.8. *Let v be an explicit nonperiodic node of $\mathcal{T}_{\text{st}}$ satisfying $\mathrm{sdepth}(v) \ge 3\tau - 1$. Given the data structure from Section 7.2.1 and the pair $\mathrm{repr}(v)$, we can in $\mathcal{O}(\log^{\epsilon} n)$ time compute the pointer to $\mathrm{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v)$.*

Proof. Denote $(b,e) = \mathrm{repr}(v)$. First, using Proposition 7.1, in $\mathcal{O}(1)$ time we compute pointer to $u = \mathrm{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v)$. By Lemma 7.1(1), we have $\mathrm{str}(u) = \mathrm{str}(v)[1\,..\,3\tau-1]$. Letting $Y = \mathrm{str}(u)$, we then have $\mathrm{per}(Y) > \frac{1}{3}\tau$ and $\mathrm{Occ}(Y,T) \neq \emptyset$. This implies (see Section 5.2.1) that there exists a unique prefix $X \in \mathcal{D}$ of $\mathrm{str}(v)$. Using $L_{\mathcal{D}}$ on Y, in $\mathcal{O}(1)$ time we obtain X. Using the lookup table L_{range} (stored as part of $\mathrm{C}_{\mathrm{ST}}(T)$; see Section 7.1), in $\mathcal{O}(1)$ time we compute $b_X = \mathrm{RangeBeg}(X,T)$. Using the lookup table L_{rev} stored in the structure from Section 7.2.1, we then obtain $\overline{X}$. Next, letting $i_1 = b + 1$ and $i_2 = e$ (recall that $\mathrm{repr}(v) = (\mathrm{lrank}(v), \mathrm{rrank}(v))$), using Theorem 2.2 in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $y_1 = \mathsf{select}_{W,\overline{X}}(i_1 - b_X)$ and $y_2 = \mathsf{select}_{W,\overline{X}}(i_2 - b_X)$. Then, using Proposition 4.1 in $\mathcal{O}(1)$ time we compute the pointers to the y_1th and y_2th leftmost leaves u_1 and u_2 (respectively) of $\mathcal{T}_{\mathsf{S}}$. Then, again using Proposition 4.1, in $\mathcal{O}(1)$ time we compute and return the pointer to $u = \mathrm{LCA}(u_1,u_2)$. By Lemma 7.6(2), it holds $\mathrm{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$. □

Mapping from $\mathcal{T}_{\mathsf{S}}$ to $\mathcal{T}_{\text{st}}$ For any string $X \in [0\,..\,\sigma)^{\le 3\tau-1}$ and any node u of the trie $\mathcal{T}_{\mathsf{S}}$, we define $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X,u) = (b_X + \delta_1, b_X + \delta_2)$, where $b_X = \mathrm{RangeBeg}(X,T)$, $z_1 = \mathrm{lrank}(u)$, $z_2 = \mathrm{rrank}(u)$, $\delta_1 = \mathsf{rank}_{W,\overline{X}}(z_1)$, and $\delta_2 = \mathsf{rank}_{W,\overline{X}}(z_2)$.

REMARK 7.2. Note that the mapping from $\mathcal{T}_{\text{st}}$ to $\mathcal{T}_{\mathsf{S}}$ is not necessarily injective, and hence it may not have an inverse. To perform the mapping from $\mathcal{T}_{\mathsf{S}}$ to $\mathcal{T}_{\text{st}}$, we will use the above function. Note, however, that although the pair $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X,u)$ is always defined, not for *every* X and u it yields $\mathrm{repr}(v)$ for some node v of $\mathcal{T}_{\text{st}}$. Below we show a simple but useful condition where it does. In the following sections we show more subtle uses of $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X,u)$ (see, e.g., Remark 7.3).

LEMMA 7.7. *Let v be an explicit nonperiodic node of $\mathcal{T}_{\text{st}}$ satisfying $\mathrm{sdepth}(v) \ge 3\tau - 1$ and let $u = \mathrm{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v)$. Then, letting $X \in \mathcal{D}$ be a prefix of $\mathrm{str}(v)$, it holds $\mathrm{repr}(v) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X,u)$.*

Proof. First, recall that $\mathcal{D} \subseteq [0\,..\,\sigma)^{\le 3\tau-1}$ (Section 5.2.1). Thus, $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X,u)$ is well-defined. Denote $b_X = \mathrm{RangeBeg}(X,T)$, $Q = \mathrm{str}(v)$, $\delta_{\text{text}} = |X| - 2\tau$, $Q_{\text{suf}} = Q(\delta_{\text{text}}\,..\,|Q|]$. Note that $\mathrm{str}(u) = Q_{\text{suf}}$. Since

$\{s_i^{\text{lex}}\}_{i\in[1..n']} = \mathsf{S}$ and $(T[s_i^{\text{lex}}..n])_{i\in[1..n']}$ is lexicographically sorted, it holds by definition of $\mathcal{T}_{\mathsf{S}}$ that $\text{lrank}(u) = |\{i \in [1..n'] : T[s_i^{\text{lex}}..n] \prec Q_{\text{suf}}\}|$ and $(\text{lrank}(u)..\text{rrank}(u)] = \{i \in [1..n'] : Q_{\text{suf}} \text{ is a prefix of } T[s_i^{\text{lex}}..n]\}$ (in particular, we have $\{s_i^{\text{lex}}\}_{i\in(\text{lrank}(u)..\text{rrank}(u)]} = \text{Occ}(Q_{\text{suf}}, T)$). Therefore, letting $\delta_1 = \mathsf{rank}_{W,\overline{X}}(\text{lrank}(u))$ and $\delta_2 = \mathsf{rank}_{W,\overline{X}}(\text{rrank}(u))$, by Lemma 6.1, it holds $\text{repr}(v) = (\text{RangeBeg}(Q,T), \text{RangeEnd}(Q,T)) = (b_X + \delta_1, b_X + \delta_2) = \text{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u)$. □

Proposition 7.9. *Let u be a node of $\mathcal{T}_{\mathsf{S}}$. Given the data structure from Section 7.2.1, a pointer to u, and the value $\text{int}(X)$ for some $X \in [0..\sigma)^{\leq 3\tau-1}$, we can in $\mathcal{O}(\log^{\epsilon} n)$ time compute the pair $\text{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u)$.*

Proof. First, using the L_{range} lookup table (stored as part of $\text{C}_{\text{ST}}(T)$; see Section 7.1), we compute $b_X = \text{RangeBeg}(X, T)$. Using the lookup table L_{rev} stored in the structure from Section 7.2.1, we compute $\overline{X}$. In $\mathcal{O}(1)$ we obtain $z_1 = \text{lrank}(u)$ and $z_2 = \text{rrank}(u)$ (Proposition 4.1). Finally, using Theorem 2.2, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\delta_1 = \mathsf{rank}_{W,\overline{X}}(z_1)$ and $\delta_2 = \mathsf{rank}_{W,\overline{X}}(z_2)$, and return $\text{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u) = (b_X + \delta_1, b_X + \delta_2)$. □

7.2.3 Implementation of $\text{LCA}(u, v)$

Lemma 7.8. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\text{st}}$ such that $\text{LCA}(v_1, v_2)$ is nonperiodic and it holds $\text{sdepth}(\text{LCA}(v_1, v_2)) \geq 3\tau - 1$. Then, v_1 and v_2 are nonperiodic and it holds $\text{sdepth}(v_1) \geq 3\tau - 1$ and $\text{sdepth}(v_2) \geq 3\tau - 1$. Moreover,*

$$\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(\text{LCA}(v_1, v_2)) = \text{LCA}(\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_1), \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_2)).$$

Proof. Denote $v = \text{LCA}(v_1, v_2)$ and $Y = \text{str}(v)[1..3\tau{-}1]$. By the assumption, we have $\text{per}(Y) > \frac{1}{3}\tau$. Since by definition of v, the string Y is a prefix of $\text{str}(v_1)$ and $\text{str}(v_2)$, we thus obtain that v_1 and v_2 are nonperiodic and it holds $\text{sdepth}(v_1) \geq 3\tau - 1$ and $\text{sdepth}(v_2) \geq 3\tau - 1$. Thus, $u_1 = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_1)$ and $u_2 = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_2)$ are well-defined (see Section 7.2.2).

Let $u = \text{LCA}(u_1, u_2)$, $\ell' = \text{sdepth}(u)$, and $\ell = \text{sdepth}(v)$. By Observation 4.1, we have $\ell = \text{lcp}(\text{str}(v_1), \text{str}(v_2)), \ell' = \text{lcp}(\text{str}(u_1), \text{str}(u_2))$, $\text{str}(v) = \text{str}(v_1)[1..\ell]$, and $\text{str}(u) = \text{str}(u_1)[1..\ell']$. Let now $X \in \mathcal{D}$ be a prefix of $\text{str}(v)$ (such X exists and is unique since $\text{per}(Y) > \frac{1}{3}\tau$ and since $\text{str}(v)$ being a substring of T implies $\text{Occ}(Y, T) \neq \emptyset$; see also Section 5.2.1). By definition of $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_1)$ and $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_2)$, we have $\text{str}(v_1) = X[1..\delta_{\text{text}}]\cdot\text{str}(u_1)$ and $\text{str}(v_2) = X[1..\delta_{\text{text}}]\cdot\text{str}(u_2)$, where $\delta_{\text{text}} = |X| - 2\tau$. This implies $\ell = \delta_{\text{text}} + \ell'$, and consequently, $\text{str}(v) = \text{str}(v_1)[1..\ell] = X[1..\delta_{\text{text}}] \cdot \text{str}(u_1)[1..\ell - \delta_{\text{text}}] = X[1..\delta_{\text{text}}] \cdot \text{str}(u_1)[1..\ell'] = X[1..\delta_{\text{text}}] \cdot \text{str}(u)$. Since v is an explicit node of $\mathcal{T}_{\text{st}}$, and no two nodes of $\mathcal{T}_{\mathsf{S}}$ have the same value of str, we therefore obtain $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$. □

Proposition 7.10. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\text{st}}$ such that $\text{LCA}(v_1, v_2)$ is nonperiodic and satisfies $\text{sdepth}(\text{LCA}(v_1, v_2)) \geq 3\tau - 1$. Given the data structure from Section 7.2.1 and the pairs $\text{repr}(v_1)$ and $\text{repr}(v_2)$, we can in $\mathcal{O}(\log^{\epsilon} n)$ time compute $\text{repr}(\text{LCA}(v_1, v_2))$.*

Proof. Denote $v = \text{LCA}(v_1, v_2)$. By Lemma 7.8, v_1 and v_2 are nonperiodic and it holds $\text{sdepth}(v_1) \geq 3\tau - 1$ and $\text{sdepth}(v_2) \geq 3\tau - 1$. Thus, $u_1 = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_1)$ and $u_2 = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v_2)$ are well-defined (see Section 7.2.2). Using Proposition 7.8, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute pointers to u_1 and u_2. Next, using the representation of $\mathcal{T}_{\mathsf{S}}$ stored as part of the structure in Section 7.2.1, and Proposition 4.1, in $\mathcal{O}(1)$ time we compute a pointer to $u = \text{LCA}(u_1, u_2)$. By Lemma 7.8, it holds $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$. We exploit this connection to compute $\text{repr}(v)$. Since v is nonperiodic, letting $Y = \text{str}(v)[1..3\tau{-}1]$, it holds $\text{per}(Y) > \frac{1}{3}\tau$. Since $\text{str}(v)$ is a substring of T, we have $\text{Occ}(Y, T) \neq \emptyset$. Together with $\text{per}(Y) > \frac{1}{3}\tau$, this implies (see Section 5.2.1) that there exists a unique prefix $X \in \mathcal{D}$ of $\text{str}(v)$. We compute X as follows. First, using Proposition 7.1, in $\mathcal{O}(1)$ time we compute pointers to $u_1' = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v_1)$ and $u_2' = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v_2)$ of $\mathcal{T}_{3\tau-1}$. Then, using the LCA structure for $\mathcal{T}_{3\tau-1}$, we compute in $\mathcal{O}(1)$ time the pointer to node $u' = \text{LCA}(u_1', u_2')$ of $\mathcal{T}_{3\tau-1}$. By Lemma 7.3, we now have $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{3\tau-1}}(v) = u'$. Moreover, by $\text{sdepth}(v) \geq 3\tau - 1$ and Lemma 7.1(1), $\text{str}(u') = Y$. Using $L_{\mathcal{D}}$ on Y, in $\mathcal{O}(1)$ time we thus obtain X. Using Proposition 7.9, in $\mathcal{O}(\log^{\epsilon} n)$ time, we then compute the pair $(b, e) = \text{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u)$. As noted above, $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v) = u$. By Lemma 7.7, we therefore have $\text{repr}(v) = (b, e)$. □

7.2.4 Implementation of $\text{child}(v, c)$

Lemma 7.9. *Let $c \in [0..\sigma)$ and v be an explicit nonperiodic internal node of $\mathcal{T}_{\text{st}}$ satisfying $\text{sdepth}(v) \geq 3\tau - 1$. Let $u = \text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{S}}}(v)$. If $\text{child}(u, c) = \perp$ then $\text{child}(v, c) = \perp$. Otherwise, letting $u' = \text{child}(u, c)$, it holds*

$$\mathrm{repr}(\mathrm{child}(v,c)) = \begin{cases} (b,e) & \text{if } b \neq e, \\ (0,0) & \text{otherwise,} \end{cases}$$

where $(b,e) = \mathrm{pseudoinv}_{\mathcal{T}_\mathsf{S}}(X,u')$ *and* $X \in \mathcal{D}$ *is a prefix of* $\mathrm{str}(v)$.

Proof. Denote $P = \mathrm{str}(v)c$, $\delta_{\mathrm{text}} = |X| - 2\tau$, and $P' = P(\delta_{\mathrm{text}} \,..\, |P|]$. Observe that since $\mathrm{str}(v)$ is nonperiodic and it holds $\mathrm{sdepth}(v) \geq 3\tau - 1$, P is also nonperiodic and satisfies $|P| \geq 3\tau - 1$. Let $(b_{\mathrm{pre}}, e_{\mathrm{pre}})$ be such that $b_{\mathrm{pre}} = |\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}}\,..\,n] \prec P'\}|$ and $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,n'] : P' \text{ is a prefix of } T[s_i^{\mathrm{lex}}\,..\,n]\}$. Recall now that u satisfies $\mathrm{str}(u) = \mathrm{str}(v)(\delta_{\mathrm{text}}\,..\,|\mathrm{str}(v)|]$, or equivalently, $\mathrm{str}(u)c = P'$. By definition of $\mathcal{T}_\mathsf{S}$ and $\mathrm{child}(u,c)$, we thus obtain that $\mathrm{child}(u,c) = \bot$ implies $e_{\mathrm{pre}} - b_{\mathrm{pre}} = 0$. Consequently, by Lemma 6.1, it holds $|\mathrm{Occ}(P,T)| = \mathrm{RangeEnd}(P,T) - \mathrm{RangeBeg}(P,T) = (b_X + \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}})) - (b_X + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}})) = \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}) - \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}) = 0$, and thus $\mathrm{child}(v,c) = \bot$.

Let us now assume $\mathrm{child}(u,c) = u' \neq \bot$. By definition of $\mathrm{pseudoinv}_{\mathcal{T}_\mathsf{S}}(X,u')$, we then have $(b,e) = (b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{lrank}(u')), b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{rrank}(u')))$, where $b_X = \mathrm{RangeBeg}(X,T)$. By definition of $\mathcal{T}_\mathsf{S}$ and $\mathrm{child}(u,c)$, however, we also have $b_{\mathrm{pre}} = \mathrm{lrank}(u')$ and $e_{\mathrm{pre}} = \mathrm{rrank}(u')$. Thus, by Lemma 6.1,

$$\begin{aligned} (\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) &= (b_X + \mathsf{rank}_{W,\overline{X}}(b_{\mathrm{pre}}), b_X + \mathsf{rank}_{W,\overline{X}}(e_{\mathrm{pre}})) \\ &= (b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{lrank}(u')), b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{rrank}(u'))) \\ &= (b,e). \end{aligned}$$

By the above, if $b \neq e$, then $\mathrm{Occ}(P,T) \neq \emptyset$. This implies $\mathrm{child}(v,c) \neq \bot$ and $\mathrm{repr}(\mathrm{child}(v,c)) = (\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$. We thus indeed have $\mathrm{repr}(\mathrm{child}(v,c)) = (b,e)$. Otherwise (i.e., if $b = e$), by the above we have $\mathrm{Occ}(P,T) = \emptyset$. This implies $\mathrm{child}(v,c) = \bot$ and hence indeed we also have $\mathrm{repr}(\mathrm{child}(v,c)) = (0,0)$. □

Remark 7.3. Note that even though in the above result we have $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{S}}(v) = u$ and $\mathrm{child}(u,c)$ contains information used to determine $\mathrm{child}(v,c)$, it does not necessarily hold that $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{S}}(\mathrm{child}(v,c)) = \mathrm{child}(u,c)$. The procedure is nevertheless correct, because such one-to-one correspondence is not required. The details of this mapping, however, become relevant for the $\mathrm{WA}(v,d)$ operation and are explained in detail in the proof of Lemma 7.11.

Proposition 7.11. *Let* v *be an explicit nonperiodic internal node of* $\mathcal{T}_{\mathrm{st}}$ *satisfying* $\mathrm{sdepth}(v) \geq 3\tau - 1$. *Given the data structure from Section 7.2.1,* $\mathrm{repr}(v)$, *and* $c \in [0\,..\,\sigma)$, *in* $\mathcal{O}(\log^\epsilon n)$ *time we can compute* $\mathrm{repr}(\mathrm{child}(v,c))$.

Proof. First, using Proposition 7.8, in $\mathcal{O}(\log^\epsilon n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{S}}(v)$. Then, using the representation of $\mathcal{T}_\mathsf{S}$ stored as part of the structure in Section 7.2.1, and Proposition 4.1, in $\mathcal{O}(\log\log n)$ time we check if $\mathrm{child}(u,c) = \bot$. If so, by Lemma 7.9 we have $\mathrm{child}(v,c) = \bot$, and thus we return $\mathrm{repr}(\mathrm{child}(v,c)) = (0,0)$. Otherwise ($\mathrm{child}(u,c) \neq \bot$), we obtain a pointer to $u' = \mathrm{child}(u,c)$. Next, using Proposition 7.1 in $\mathcal{O}(1)$ time we compute a pointer to $u'' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{3\tau-1}}(v)$. By Lemma 7.1(1), $\mathrm{sdepth}(v) \geq 3\tau - 1$ implies $\mathrm{sdepth}(u'') = 3\tau - 1$ and $\mathrm{str}(u'') = \mathrm{str}(v)[1\,..\,3\tau{-}1]$. We obtain $Y = \mathrm{str}(u'')$ (stored with u'') in $\mathcal{O}(1)$ time. Using $L_\mathcal{D}$ on Y, in $\mathcal{O}(1)$ time we then compute a prefix $X \in \mathcal{D}$ of Y (see Section 7.2.2). Finally, using Proposition 7.9, in $\mathcal{O}(\log^\epsilon n)$ time we compute the pair $(b,e) = \mathrm{pseudoinv}_{\mathcal{T}_\mathsf{S}}(X,u')$. If $b = e$ then by Lemma 7.9 we return $\mathrm{repr}(\mathrm{child}(v,c)) = (0,0)$. Otherwise, we return $\mathrm{repr}(\mathrm{child}(v,c)) = (b,e)$. □

7.2.5 Implementation of $\mathrm{pred}(v,c)$

Lemma 7.10. *Let* $c \in [0\,..\,\sigma)$ *and* v *be an explicit nonperiodic internal node of* $\mathcal{T}_{\mathrm{st}}$ *satisfying* $\mathrm{sdepth}(v) \geq 3\tau - 1$. *Let* $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{S}}(v)$. *If* $\mathrm{pred}(u,c) = \bot$ *then* $\mathrm{RangeBeg}(\mathrm{str}(v)c,T) = \mathrm{RangeBeg}(\mathrm{str}(v),T)$. *Otherwise, letting* $u' = \mathrm{pred}(u,c)$, *it holds*

$$\mathrm{RangeBeg}(\mathrm{str}(v)c,T) = e,$$

where $(b,e) = \mathrm{pseudoinv}_{\mathcal{T}_\mathsf{S}}(X,u')$ *and* $X \in \mathcal{D}$ *is a prefix of* $\mathrm{str}(v)$.

Proof. We start by characterizing $\mathrm{RangeBeg}(\mathrm{str}(v), T)$ using Lemma 6.1 for pattern $\mathrm{str}(v)$. First, note that v is nonperiodic and it holds $\mathrm{sdepth}(v) \geq 3\tau - 1$. On the other hand, by definition, we have $\mathrm{str}(u) = \mathrm{str}(v)(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v)|]$, where $\delta_{\mathrm{text}} = |X| - 2\tau$. Finally, by definition of $\mathcal{T}_{\mathsf{S}}$, we have $|\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}} \,..\, n] \prec \mathrm{str}(v)(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v)|]\}| = \mathrm{lrank}(u)$. Thus, by Lemma 6.1, we have $\mathrm{RangeBeg}(\mathrm{str}(v), T) = b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{lrank}(u))$, where $b_X = \mathrm{RangeBeg}(X, T)$.

Denote $P = \mathrm{str}(v)c$ and $P' = P(\delta_{\mathrm{text}} \,..\, |P|]$. Since $\mathrm{str}(v)$ is nonperiodic and satisfies $|\mathrm{str}(v)| \geq 3\tau - 1$, P is also nonperiodic and it holds $|P| \geq 3\tau - 1$. Note also that by $\mathrm{str}(u) = \mathrm{str}(v)(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v)|]$, we have $\mathrm{str}(u)c = P'$.

Let us first assume $\mathrm{pred}(u, c) = \bot$. By definition, this implies that $|\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}} \,..\, n] \prec \mathrm{str}(u)c\}| = \mathrm{lrank}(u)$. Equivalently, by $\mathrm{str}(u)c = P'$, $|\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}} \,..\, n] \prec P'\}| = \mathrm{lrank}(u)$. By Lemma 6.1 for pattern $P = \mathrm{str}(v)c$ we thus obtain $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{lrank}(u))$. Since above we also established that $\mathrm{RangeBeg}(\mathrm{str}(v), T) = b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{lrank}(u))$, we have thus proved that $\mathrm{pred}(u, c) = \bot$ implies $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = \mathrm{RangeBeg}(\mathrm{str}(v), T)$.

Let us now assume $\mathrm{pred}(u, c) = u' \neq \bot$. Observe that by definition of $\mathrm{pred}(u, c)$, this implies $|\{i \in [1\,..\,n'] : T[s_i^{\mathrm{lex}} \,..\, n] \prec \mathrm{str}(u)c\}| = \mathrm{rrank}(u')$. By Lemma 6.1 applied for pattern $P = \mathrm{str}(v)c$, we thus obtain $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{rrank}(u'))$. On the other hand, observe that by definition of $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u')$, we have $e = b_X + \mathsf{rank}_{W,\overline{X}}(\mathrm{rrank}(u'))$. We thus obtain $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e$. □

Proposition 7.12. *Let v be an explicit nonperiodic internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) \geq 3\tau - 1$. Given the data structure from Section 7.2.1, $\mathrm{repr}(v)$, and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{RangeBeg}(\mathrm{str}(v)c, T)$.*

Proof. First, using Proposition 7.8, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v)$. Then, using the representation of $\mathcal{T}_{\mathsf{S}}$ stored as part of the structure in Section 7.2.1, and Proposition 4.1, in $\mathcal{O}(\log \log n)$ time we check if $\mathrm{pred}(u, c) = \bot$. If so, by Lemma 7.10 we have $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = \mathrm{RangeBeg}(\mathrm{str}(v), T)$ and hence we return $\mathrm{RangeBeg}(\mathrm{str}(v), T)$ (given as input) as the result. Otherwise ($\mathrm{pred}(u, c) \neq \bot$), we obtain a pointer to $u' = \mathrm{pred}(u, c)$. Next, using Proposition 7.1 in $\mathcal{O}(1)$ time we compute a pointer to $u'' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{3\tau-1}}(v)$. By Lemma 7.1(1), $\mathrm{sdepth}(v) \geq 3\tau - 1$ implies $\mathrm{sdepth}(u'') = 3\tau - 1$ and $\mathrm{str}(u'') = \mathrm{str}(v)[1\,..\,3\tau{-}1]$. We obtain $Y = \mathrm{str}(u'')$ (stored with u'') in $\mathcal{O}(1)$ time. Using $L_{\mathcal{D}}$ on Y, in $\mathcal{O}(1)$ time we then compute a prefix $X \in \mathcal{D}$ of Y (see Section 7.2.2). Finally, using Proposition 7.9, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute the pair $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u')$. By Lemma 7.10, it holds $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e$. We thus return e as the answer. □

7.2.6 Implementation of $\mathrm{WA}(v, d)$

Lemma 7.11. *Let v be an explicit nonperiodic node of $\mathcal{T}_{\mathrm{st}}$ and d be an integer satisfying $3\tau - 1 \leq d \leq |\mathrm{str}(v)|$. Then, letting $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v)$, it holds*

$$\mathrm{repr}(\mathrm{WA}(v, d)) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, \widehat{u}),$$

where $X \in \mathcal{D}$ is a prefix of $\mathrm{str}(v)$, $\delta_{\mathrm{text}} = |X| - 2\tau$, and $\widehat{u} = \mathrm{WA}(u, d - \delta_{\mathrm{text}})$.

Proof. Denote $f^{(0)}(x) = x$ and $f^{(i)}(x) = f(f^{(i-1)}(x))$ for $i \in \mathbb{Z}_+$. Let

$$\mathcal{V} := \{\mathrm{parent}^{(i)}(v) : i \in \mathbb{Z}_{\geq 0} \text{ and } \mathrm{sdepth}(\mathrm{parent}^{(i)}(v)) \geq |X|\} \text{ and}$$
$$\mathcal{U} := \{\mathrm{parent}^{(i)}(u) : i \in \mathbb{Z}_{\geq 0} \text{ and } \mathrm{sdepth}(\mathrm{parent}^{(i)}(u)) \geq 2\tau\}$$

For any $v' \in \mathcal{V}$, the node $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v')$ satisfies $\mathrm{str}(v') = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u')$. In particular, $\mathrm{str}(u) = \mathrm{str}(v)(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v)|]$. Since for any $v' \in \mathcal{V}$, $\mathrm{str}(v') = \mathrm{str}(v)[1\,..\,|\mathrm{str}(v')|]$, we thus obtain that for $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v')$ it holds $\mathrm{str}(u') = \mathrm{str}(v')(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v')|] = \mathrm{str}(v)(\delta_{\mathrm{text}} \,..\, |\mathrm{str}(v')|] = \mathrm{str}(u)[1\,..\,|\mathrm{str}(u')|]$, i.e., u' is an ancestor of u. Moreover, $\mathrm{sdepth}(u') = |\mathrm{str}(v')| - \delta_{\mathrm{text}} \geq |X| - \delta_{\mathrm{text}} = 2\tau$. Consequently, $\mathcal{U}' := \{\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v') : v' \in \mathcal{V}\}$ satisfies $\mathcal{U}' \subseteq \mathcal{U}$. Note also, that $v' \neq v''$ implies $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v') \neq \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v'')$.

For any $u' \in \mathcal{U}$, denote $(s(u'), t(u')) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u')$. We prove the following property of $\mathcal{U}'$. Let $w, w' \in \mathcal{U}$ be such that $w' = \mathrm{parent}(w)$. We claim, that $(s(w), t(w)) \neq (s(w'), t(w'))$ implies $w' \in \mathcal{U}'$. Denote $Q' = \mathrm{str}(w')$ and $Q = X[1\,..\,\delta_{\mathrm{text}}] \cdot Q'$. The proof consists of three steps:

- First, we show that it holds $\{\mathrm{SA}[i]\}_{i \in (s'\,..\,t']} = \mathrm{Occ}(Q, T)$, where $s' = s(w')$ and $t' = t(w')$. By the above discussion, we have $\mathrm{str}(v) = X[1\,..\,\delta_{\mathrm{text}}] \cdot \mathrm{str}(u)$. Thus, $X(\delta_{\mathrm{text}} \,..\, |X|]$ is a prefix of $\mathrm{str}(u)$. On the other hand,

by $w' \in \mathcal{U}$, $\mathrm{str}(w')$ is a prefix of $\mathrm{str}(u)$ and we have $|\mathrm{str}(w')| \geq 2\tau$. Consequently, $X(\delta_{\mathrm{text}} \,..\, |X|]$ is a prefix of Q'. As noted in the proof of Lemma 7.7, by the consistency of S we then have $\{s_i^{\mathrm{lex}}\}_{i \in (\mathrm{lrank}(w') \,..\, \mathrm{rrank}(w')]} = \mathrm{Occ}(Q', T)$ and consequently $\{\mathrm{SA}[i]\}_{i \in (s' \,..\, t']} = \mathrm{Occ}(X[1 \,..\, \delta_{\mathrm{text}}] \cdot Q', T) = \mathrm{Occ}(Q, T)$.
- Second, we prove that there exists a node v' in $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{str}(v') = Q$. First, note that $\{\mathrm{SA}[i]\}_{i \in (s' \,..\, t']} = \mathrm{Occ}(Q, T)$ already implies that there exists some node v' of $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{repr}(v') = (s', t')$ and Q is a prefix of $\mathrm{str}(v')$. It thus remains to show that $\mathrm{str}(v') = Q$. For this, it suffices to show that there exists $c, c' \in [0 \,..\, \sigma)$ such that $c \neq c'$, $\mathrm{Occ}(Qc, T) \neq \emptyset$, and $\mathrm{Occ}(Qc', T) \neq \emptyset$. Observe that by $(s(w'), t(w')) \neq (s(w), t(w))$, there exists a child $w'' \neq w$ of w' such that $t(w'') > s(w'')$. This yields an occurrence of $\mathrm{str}(w')$ preceded in T with $X[1 \,..\, \delta_{\mathrm{text}}]$. The same holds for $\mathrm{str}(w)$, since $s(w') \leq s(u) < t(u) \leq t(w')$. In other words, for $c = \mathrm{str}(w)[|Q'| + 1]$ and $c' = \mathrm{str}(w'')[|Q'| + 1]$ we have $c \neq c'$, $\mathrm{Occ}(Qc, T) \neq \emptyset$, and $\mathrm{Occ}(Qc', T) \neq \emptyset$. This concludes the proof of $\mathrm{str}(v') = Q$.
- Finally, recall that by definition, the node $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v')$ satisfies $\mathrm{str}(v') = X[1 \,..\, \delta_{\mathrm{text}}] \cdot \mathrm{str}(\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v'))$. Thus, by $\mathrm{str}(v') = Q = X[1 \,..\, \delta_{\mathrm{text}}] \cdot Q'$ and $\mathrm{str}(w') = Q'$, we must have $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v') = w'$. This implies $w' \in \mathcal{U}'$.

We are now ready to prove the main claim. Let $v' = \mathrm{WA}(v, d)$ and $v'' = \mathrm{parent}(v')$. We then have $\mathrm{sdepth}(v'') < d \leq \mathrm{sdepth}(v')$. Moreover, by $|X| \leq 3\tau - 1 \leq d$, we have $v' \in \mathcal{V}$. Let $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v')$. Then, $u' \in \mathcal{U}'$. By the above discussion, we also have $d - \delta_{\mathrm{text}} \leq \mathrm{sdepth}(\widehat{u}) \leq \mathrm{sdepth}(u')$. By $3\tau - 1 \leq d$ this implies $2\tau = |X| - \delta_{\mathrm{text}} \leq 3\tau - 1 - \delta_{\mathrm{text}} \leq d - \delta_{\mathrm{text}} \leq \mathrm{sdepth}(\widehat{u})$, i.e., $\widehat{u} \in \mathcal{U}$. Let $k \in \mathbb{Z}_{\geq 0}$ be such that $\widehat{u} = \mathrm{parent}^{(k)}(u')$. This implies that $\mathrm{parent}^{(i)}(u') \notin \mathcal{U}'$ holds for $i \in [1 \,..\, k]$, since otherwise it would contradict $v' = \mathrm{WA}(v, d)$. If $k = 0$ then we trivially have $(s(u'), t(u')) = (s(\widehat{u}), t(\widehat{u}))$. Otherwise, by (the contraposition of) the above property of $\mathcal{U}'$ we have

$$\begin{aligned}(s(u'), t(u')) &= (s(\mathrm{parent}(u')), t(\mathrm{parent}(u'))) \\ &= \dots \\ &= (s(\mathrm{parent}^{(k)}(u')), t(\mathrm{parent}^{(k)}(u'))) \\ &= (s(\widehat{u}), t(\widehat{u})).\end{aligned}$$

By Lemma 7.7, we obtain $\mathrm{repr}(\mathrm{WA}(v, d)) = \mathrm{repr}(v') = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, u') = (s(u'), t(u')) = (s(\widehat{u}), t(\widehat{u})) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, \widehat{u})$. □

Proposition 7.13. *Let v be an explicit nonperiodic node of $\mathcal{T}_{\mathrm{st}}$ satisfying $3\tau - 1 \leq |\mathrm{str}(v)|$. Given the data structure from Section 7.2.1, $\mathrm{repr}(v)$, and an integer d satisfying $3\tau - 1 \leq d \leq |\mathrm{str}(v)|$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{WA}(v, d))$.*

Proof. First, using Proposition 7.8, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v)$. Next, using Proposition 7.1 in $\mathcal{O}(1)$ time we compute a pointer to $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{3\tau-1}}(v)$. By Lemma 7.1(1), $\mathrm{sdepth}(v) \geq 3\tau - 1$ implies $\mathrm{sdepth}(u') = 3\tau - 1$ and $\mathrm{str}(u') = \mathrm{str}(v)[1 \,..\, 3\tau - 1]$. We obtain $Y = \mathrm{str}(u')$ (stored with u') in $\mathcal{O}(1)$ time. Using $L_{\mathcal{D}}$ on Y, in $\mathcal{O}(1)$ time we then compute a prefix $X \in \mathcal{D}$ of Y (see Section 7.2.2). Let $\delta_{\mathrm{text}} = |X| - 2\tau$. Finally, using the representation of $\mathcal{T}_{\mathsf{S}}$ stored as part of the structure in Section 7.2.1, and Proposition 4.1, in $\mathcal{O}(\log \log n)$ time we compute a pointer to $\widehat{u} = \mathrm{WA}(u, d - \delta_{\mathrm{text}})$. Using Proposition 7.9, in $\mathcal{O}(\log^{\epsilon} n)$ time we then compute and return $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{S}}}(X, \widehat{u})$, which by Lemma 7.11 is equal to $\mathrm{repr}(\mathrm{WA}(v, d))$. □

7.2.7 Construction Algorithm

Proposition 7.14. *Given $\mathrm{C}_{\mathrm{ST}}(T)$, we can in $\mathcal{O}(n \min(1, \log \sigma / \sqrt{\log n}))$ time and $\mathcal{O}(n / \log_{\sigma} n)$ working space augment it into a data structure from Section 7.2.1.*

Proof. First, we combine Propositions 5.3 and 5.6 (recall that the packed representation of T is a component of $\mathrm{C}_{\mathrm{ST}}(T)$) to construct in $\mathcal{O}(n \min(1, \log \sigma / \sqrt{\log n}))$ time and using $\mathcal{O}(n / \log_{\sigma} n)$ working space the data structure from Section 5.2.1. In particular, this constructs $(s_i^{\mathrm{lex}})_{i \in [1 \,..\, n']}$. We then initialize $A_{\mathsf{S}}[i] = s_i^{\mathrm{lex}}$ for $i \in [1 \,..\, n']$ and in $\mathcal{O}(n / \log_{\sigma} n)$ time construct $\mathcal{T}_{\mathsf{S}}$ represented using Proposition 4.1. □

7.3 The Periodic Nodes

In this section, we describe a data structure used to perform operations on periodic nodes (see Definition 7.1) in $\mathcal{O}(\log \log n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 7.3.1). We then show how using this structure to implement some basic navigational routines (Section 7.3.2). Next, we describe the query algorithms for the fundamental operations (Sections 7.3.3 to 7.3.6). Finally, we show the construction algorithm (Section 7.3.7).

7.3.1 The Data Structure

Definitions Let v be a periodic node of $\mathcal{T}_{\text{st}}$. We define L-root(v) := L-root(str(v)), $e(v)$:= e(str(v)), L-head(v) := L-head(str(v)), L-exp(v) := L-exp(str(v)), L-tail(v) := L-tail(str(v)), $e^{\text{full}}(v) := e^{\text{full}}(\text{str}(v))$, and type$(v)$:= type(str(v)). Let $q = |\mathsf{R}'^{-}|$. Recall (Section 5.3.2) that $(r_i^{\text{lex}})_{i\in[1\,..\,q]}$ is a sequence containing all elements $k \in \mathsf{R}'^{-}$ sorted first according to L-root(k) and in case of ties, by $T[e^{\text{full}}(k)\,..\,n]$. Recall also (Section 6.3.2) that $\mathsf{Z} = \{e^{\text{full}}(j) - |\text{pow}(\text{L-root}(j))| : j \in \mathsf{R}'^{-}\}$ and $A_{\mathsf{Z}}[1\,..\,q]$ is an array defined by $A_{\mathsf{Z}}[i] = e^{\text{full}}(r_i^{\text{lex}}) - |\text{pow}(H_i)|$, where $H_i = \text{L-root}(r_i^{\text{lex}})$ and $\text{pow}(H_i) = H_i^{\infty}[1\,..\,|H_i|\lceil \frac{\tau}{|H_i|}\rceil]$. Let $\mathcal{T}_{\mathsf{Z}}$ denote the compact trie of the set $\{T[i\,..\,n] : i \in \mathsf{Z}\}$.

Components The data structure consists of two parts. The first part consists of the following three components:

1. The index core $\text{C}_{\text{ST}}(T)$ (Section 7.1). It takes $\mathcal{O}(n/\log_\sigma n)$ space.
2. The first part of the structure from Section 5.3.2 using $\mathcal{O}(n/\log_\sigma n)$ space.
3. The compact trie $\mathcal{T}_{\mathsf{Z}}$ represented as in Proposition 4.1 (i.e., for the array $A_{\mathsf{Z}}[1\,..\,q]$ defined as above). By $q = \mathcal{O}(n/\log_\sigma n)$ and Proposition 4.1, it needs $\mathcal{O}(n/\log_\sigma n)$ space.

The second part of the structure consists of the symmetric counterparts of the above components adapted according to Lemma 5.4 (see also Section 5.3.2)

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

7.3.2 Navigation Primitives

Mapping from $\mathcal{T}_{\text{st}}$ to $\mathcal{T}_{\mathsf{Z}}$ For any periodic explicit node v of $\mathcal{T}_{\text{st}}$ satisfying $e(v) \le |\text{str}(v)|$ and type$(v) = -1$, we define $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{Z}}}(v) = u$ as a node of $\mathcal{T}_{\mathsf{Z}}$ satisfying $\text{str}(u) = \text{pow}(H)\cdot\text{str}(v)[e^{\text{full}}(v)\,..\,|\text{str}(v)|]$, where $H = \text{L-root}(v)$.

LEMMA 7.12. *Let v be a periodic explicit node v of $\mathcal{T}_{\text{st}}$ such that $e(v) \le |\text{str}(v)|$ and type$(v) = -1$.*

1. *The node $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{Z}}}(v)$ is well-defined.*
2. *Let $i_1 = \text{lrank}(v) + 1$, $i_2 = \text{rrank}(v)$, y_1 and y_2 be such that $e^{\text{full}}(r_{y_1}^{\text{lex}}) = e^{\text{full}}(\text{SA}[i_1])$ and $e^{\text{full}}(r_{y_2}^{\text{lex}}) = e^{\text{full}}(\text{SA}[i_2])$ (respectively), u_1 and u_2 be the y_1th and y_2th leftmost leaf of $\mathcal{T}_{\mathsf{Z}}$ (respectively), and $u = \text{LCA}(u_1, u_2)$. Then, $\text{map}_{\mathcal{T}_{\text{st}},\mathcal{T}_{\mathsf{Z}}}(v) = u$.*

Proof. Denote $s = \text{L-head}(v)$, $H = \text{L-root}(v)$, $p = |H|$, $Q = \text{str}(v)$, and $Q_{\text{suf}} = Q[e^{\text{full}}(Q)\,..\,|Q|]$. Note that by $e^{\text{full}}(Q) \le e(Q) \le |Q|$, it holds $Q_{\text{suf}} \ne \varepsilon$.

1. We start by observing that by Lemma 6.2 and Lemma 6.4(2), for every $i \in \text{Occ}(Q,T)$, it holds $i \in \mathsf{R}^{-}_{s,H}$ and $e^{\text{full}}(i) - i = e^{\text{full}}(Q) - 1$. Note that this implies $e^{\text{full}}(i) \in \text{Occ}(Q_{\text{suf}}, T)$. To show that $\mathcal{T}_{\mathsf{Z}}$ has a node u satisfying $\text{str}(u) = \text{pow}(H)\cdot Q_{\text{suf}}$, consider two cases:

- Assume that v is a leaf. Let $i \in \text{Occ}(\text{str}(v), T)$. By the above, $i \in \mathsf{R}^{-}_{H}$. Let j be the smallest integer such that $[j\,..\,i] \subseteq \mathsf{R}$. It holds $j \in \mathsf{R}'$ and moreover, by Lemma 5.5, $j \in \mathsf{R}^{-}_{H}$ and $e^{\text{full}}(j) = e^{\text{full}}(i)$. Thus, by $e^{\text{full}}(i) \in \text{Occ}(Q_{\text{suf}}, T)$ (see above), we have $e^{\text{full}}(j) \in \text{Occ}(Q_{\text{suf}}, T)$. Finally, by $i \in \text{Occ}(Q,T)$ and $|Q| = n - i + 1$, we have $|Q_{\text{suf}}| = |Q| - e^{\text{full}}(Q) + 1 = |Q| - (e^{\text{full}}(Q) - 1) = (n - i + 1) - (e^{\text{full}}(i) - i) = n - e^{\text{full}}(i) + 1 = n - e^{\text{full}}(j) + 1$. We have thus shown that there exists $j \in \mathsf{R}'^{-}_{H}$ such that $e^{\text{full}}(j) \in \text{Occ}(Q_{\text{suf}}, T)$ and $n - e^{\text{full}}(j) + 1 = |Q_{\text{suf}}|$. By definition of $A_{\mathsf{Z}}[1\,..\,q]$ (see Section 6.3.2) this implies that there exists a leaf u of $\mathcal{T}_{\mathsf{Z}}$ such that $\text{str}(u) = \text{pow}(H)\cdot T[e^{\text{full}}(j)\,..\,n] = \text{pow}(H)\cdot Q_{\text{suf}}$.
- Assume that v is an internal node. Consider the leftmost and the rightmost leaves v_1 and v_2 (respectively) in the subtree rooted in v. Letting $i_1 \in \text{Occ}(\text{str}(v_1), T)$ and $i_2 \in \text{Occ}(\text{str}(v_2), T)$, we have $i_1 \ne i_2$ and $i_1, i_2 \in \text{Occ}(Q,T)$. Thus, $i_1, i_2 \in \mathsf{R}^{-}_{H}$ and $e^{\text{full}}(i_1) - i_1 = e^{\text{full}}(Q) - 1 = e^{\text{full}}(i_2) - i_2$. Therefore, $e^{\text{full}}(i_1) \ne e^{\text{full}}(i_2)$ and, by Lemma 5.5, i_1 and i_2 are in different maximal contiguous blocks of positions

from R, i.e., letting j_1 (resp. j_2) be the smallest integer such that $[j_1\,..\,i_1] \subseteq \mathsf{R}$ (resp. $[j_2\,..\,i_2] \subseteq \mathsf{R}$), we have $j_1, j_2 \in \mathsf{R}'$ and $j_1 \neq j_2$. By Lemma 5.5, it then holds $j_1, j_2 \in \mathsf{R}'^{-}_H$, $e^{\text{full}}(j_1) = e^{\text{full}}(i_1)$, and $e^{\text{full}}(j_2) = e^{\text{full}}(i_2)$. Thus, by $e^{\text{full}}(i_1), e^{\text{full}}(i_2) \in \text{Occ}(Q_{\text{suf}}, T)$ (following from $i_1, i_2 \in \text{Occ}(Q, T)$), we obtain $e^{\text{full}}(j_1), e^{\text{full}}(j_2) \in \text{Occ}(Q_{\text{suf}}, T)$. Next, we show $\text{LCE}(e^{\text{full}}(j_1), e^{\text{full}}(j_2)) = |Q_{\text{suf}}|$. As noted earlier, $e^{\text{full}}(i_1) - i_1 = e^{\text{full}}(i_2) - i_2 = e^{\text{full}}(Q) - 1$. Thus, by $\text{LCE}(i_1, i_2) = |\text{str}(\text{LCA}(v_1, v_2))| = |\text{str}(v)| = |Q|$ and $e^{\text{full}}(Q) - 1 \leq e(Q) - 1 < |Q|$, we have $|Q| = \text{LCE}(i_1, i_2) = e^{\text{full}}(Q) - 1 + \text{LCE}(e^{\text{full}}(i_1), e^{\text{full}}(i_2))$. Equivalently, $\text{LCE}(e^{\text{full}}(i_1), e^{\text{full}}(i_2)) = |Q| - e^{\text{full}}(Q) + 1 = |Q_{\text{suf}}|$, which by $e^{\text{full}}(j_1) = e^{\text{full}}(i_1)$ and $e^{\text{full}}(j_2) = e^{\text{full}}(i_2)$ yields $\text{LCE}(e^{\text{full}}(j_1), e^{\text{full}}(j_2)) = |Q_{\text{suf}}|$. We have thus shown that there exist distinct positions $j_1, j_2 \in \mathsf{R}'^{-}_H$ satisfying $e^{\text{full}}(j_1), e^{\text{full}}(j_2) \in \text{Occ}(Q_{\text{suf}}, T)$ and $\text{LCE}(e^{\text{full}}(j_1), e^{\text{full}}(j_1)) = |Q_{\text{suf}}|$. By definition of $A_{\mathsf{Z}}[1\,..\,q]$, this implies that there exists leaves u_1 and u_2 of $\mathcal{T}_{\mathsf{Z}}$ such that $\text{str}(u_1) = \text{pow}(H) \cdot T[e^{\text{full}}(j_1)\,..\,n]$ and $\text{str}(u_2) = \text{pow}(H) \cdot T[e^{\text{full}}(j_2)\,..\,n]$ (see the proof of Proposition 6.8) and consequently, by Observation 4.1, the node $u = \text{LCA}(u_1, u_2)$ satisfies $\text{str}(u) = \text{pow}(H) \cdot Q_{\text{suf}}$.

2. To show that y_1 and y_2 are well-defined note that for every $i \in \mathsf{R}^{-}$, letting j be the smallest integer satisfying $[j\,..\,i] \subseteq \mathsf{R}$, we have $j \in \mathsf{R}'^{-}$ and (by Lemma 5.5) $e^{\text{full}}(j) = e^{\text{full}}(i)$. Consequently, since $\{r^{\text{lex}}_i\}_{i \in [1\,..\,q]} = \mathsf{R}'^{-}$, taking $y \in [1\,..\,q]$ such that $r^{\text{lex}}_y = j$, it holds $e^{\text{full}}(r^{\text{lex}}_y) = e^{\text{full}}(i)$. Therefore, by $\text{SA}[i_1], \text{SA}[i_2] \in \text{Occ}(Q, T) \subseteq \mathsf{R}^{-}$, $y_1, y_2 \in [1\,..\,q]$ are (uniquely) defined.

We start by showing that $\text{str}(u_1) = \text{pow}(H) \cdot T[e^{\text{full}}(\text{SA}[i_1])\,..\,n]$ and $\text{str}(u_2) = \text{pow}(H) \cdot T[e^{\text{full}}(\text{SA}[i_2])\,..\,n]$. As noted in Section 6.3.2, the sequence $(T[A_{\mathsf{Z}}[i]\,..\,n])_{i \in [1\,..\,q]}$ is lexicographically sorted. Thus, by definition of u_1 and u_2, we have $\text{str}(u_1) = T[A_{\mathsf{Z}}[y_1]\,..\,n]$ and $\text{str}(u_2) = T[A_{\mathsf{Z}}[y_2]\,..\,n]$. As also noted in Section 6.3.2, for every $i \in [1\,..\,q]$, $T[A_{\mathsf{Z}}[i]\,..\,n] = \text{pow}(H_i) \cdot T[e^{\text{full}}(r^{\text{lex}}_i)\,..\,n]$, where $H_i = \text{L-root}(r^{\text{lex}}_i)$. Combining that with the assumptions $e^{\text{full}}(r^{\text{lex}}_{y_1}) = e^{\text{full}}(\text{SA}[i_1])$ and $e^{\text{full}}(r^{\text{lex}}_{y_2}) = e^{\text{full}}(\text{SA}[i_2])$, we therefore obtain

$$\text{str}(u_1) = \text{pow}(H_{y_1}) \cdot T[e^{\text{full}}(\text{SA}[i_1])\,..\,n],$$
$$\text{str}(u_2) = \text{pow}(H_{y_2}) \cdot T[e^{\text{full}}(\text{SA}[i_2])\,..\,n].$$

To obtain $\text{str}(u_1) = \text{pow}(H) \cdot T[e^{\text{full}}(\text{SA}[i_1])\,..\,n]$ and $\text{str}(u_2) = \text{pow}(H) \cdot T[e^{\text{full}}(\text{SA}[i_2])\,..\,n]$ it thus remains to show $H_{y_1} = H_{y_2} = H$. To this end, we first note that by $e^{\text{full}}(r^{\text{lex}}_{y_1}) = e^{\text{full}}(\text{SA}[i_1])$, (resp. $e^{\text{full}}(r^{\text{lex}}_{y_2}) = e^{\text{full}}(\text{SA}[i_2])$), and Lemmas 5.5 and 5.7, the positions $e^{\text{full}}(r^{\text{lex}}_{y_1})$ and $\text{SA}[i_1]$ (resp. $e^{\text{full}}(r^{\text{lex}}_{y_2})$ and $\text{SA}[i_2]$) belong to the same contiguous block of elements from R. Next, by $|Q| \geq 3\tau - 1$ and $\text{SA}[i_1] \in \text{Occ}(Q, T)$, we obtain $\text{lcp}(T[\text{SA}[i_1]\,..\,n], Q) \geq 3\tau - 1$. Thus, by combining Lemma 5.5 and Lemma 6.2, we have $H_{y_1} = \text{L-root}(r^{\text{lex}}_{y_1}) = \text{L-root}(\text{SA}[i_1]) = \text{L-root}(Q) = \text{L-root}(v) = H$. Analogously, $H_{y_2} = H$.

By the above and Observation 4.1, we thus have $\text{str}(u) = \text{str}(\text{LCA}(u_1, u_2)) = \text{pow}(H) \cdot T[e^{\text{full}}(\text{SA}[i_1])\,..\,e^{\text{full}}(\text{SA}[i_1]) + \ell)$, where $\ell = \text{LCE}(e^{\text{full}}(\text{SA}[i_1]), e^{\text{full}}(\text{SA}[i_2]))$. Moreover, by $\text{SA}[i_1] \in \text{Occ}(Q, T)$, we have $e^{\text{full}}(\text{SA}[i_1]) \in \text{Occ}(Q_{\text{suf}}, T)$. Thus, it remains to show that $\ell = |Q_{\text{suf}}|$. For this, recall that by $\text{SA}[i_1], \text{SA}[i_2] \in \text{Occ}(Q, T)$ we also have $e^{\text{full}}(\text{SA}[i_1]) - \text{SA}[i_1] = e^{\text{full}}(\text{SA}[i_2]) - \text{SA}[i_2] = e^{\text{full}}(Q) - 1$. Therefore, by $e^{\text{full}}(Q) - 1 \leq e(Q) - 1 < |Q|$, we have $|Q| = \text{LCE}(\text{SA}[i_1], \text{SA}[i_2]) = e^{\text{full}}(Q) - 1 + \text{LCE}(e^{\text{full}}(\text{SA}[i_1]), e^{\text{full}}(\text{SA}[i_2]))$. Equivalently, $\text{LCE}(e^{\text{full}}(\text{SA}[i_1]), e^{\text{full}}(\text{SA}[i_2])) = |Q| - e^{\text{full}}(Q) + 1 = |Q_{\text{suf}}|$. We therefore obtained $\text{str}(u) = \text{pow}(H) \cdot Q_{\text{suf}}$, i.e., $\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{Z}}}(v) = u$. □

Proposition 7.15. *Let v be a periodic explicit node v of $\mathcal{T}_{\text{st}}$ satisfying $e(v) \leq |\text{str}(v)|$ and $\text{type}(v) = -1$. Given the data structure from Section 7.3.1 and $\text{repr}(v)$, we can in $\mathcal{O}(\log \log n)$ time compute the pointer to the node $\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{Z}}}(v)$.*

Proof. Denote $(b, e) = \text{repr}(v)$, $i_1 = b + 1$, and $i_2 = e$. First, using Proposition 5.12 in $\mathcal{O}(1)$ time we compute the $\text{L-exp}(\text{SA}[i_1])$ and $\delta^{\mathsf{s}}(\text{SA}[i_1])$. Next, as explained in the proof of Proposition 5.13, given i_1, $\text{L-exp}(\text{SA}[i_1])$, and $\delta^{\mathsf{s}}(\text{SA}[i_1])$, in $\mathcal{O}(\log \log n)$ time we compute $y_1 \in [1\,..\,q]$ satisfying $e^{\text{full}}(r^{\text{lex}}_{y_1}) = e^{\text{full}}(\text{SA}[i_1])$. Note that Proposition 5.13 requires that $\text{SA}[i_1] \in \mathsf{R}^{-}$, which holds by $\text{SA}[i_1] \in \text{Occ}(Q, T)$, where $Q = \text{str}(v)$ (see the proof of Lemma 7.12). Analogously we compute $y_2 \in [1\,..\,q]$ satisfying $e^{\text{full}}(r^{\text{lex}}_{y_2}) = e^{\text{full}}(\text{SA}[i_2])$. Then, using Proposition 4.1 in $\mathcal{O}(1)$ time we compute the pointers to the y_1th and y_2th leftmost leaves u_1 and u_2 (respectively) of $\mathcal{T}_{\mathsf{Z}}$. Then, again using Proposition 4.1, in $\mathcal{O}(1)$ time we compute and return the pointer to $u = \text{LCA}(u_1, u_2)$. By Lemma 7.12, it holds $\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{Z}}}(v) = u$. □

Mapping from $\mathcal{T}_{\mathsf{Z}}$ to $\mathcal{T}_{\text{st}}$ Let u be a node of $\mathcal{T}_{\mathsf{Z}}$ such that there exists $H \in \text{Roots}$ for which $\text{pow}(H)$ is a prefix of $\text{str}(u)$. For any $\ell \geq 0$, we define $\text{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u)$ as follows. If, letting $s = \ell \bmod |H|$ and $k = \lfloor \frac{\ell}{|H|} \rfloor$,

it holds $\mathsf{P} := \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\} = \emptyset$, then $\text{pseudoinv}_{\mathcal{T}_\mathsf{Z}}(\ell, u) := (0, 0)$. Otherwise (i.e., $\mathsf{P} \neq \emptyset$), letting $b_\mathsf{P}, e_\mathsf{P} \in [0\,..\,n]$ be such that $\{\text{SA}[i]\}_{i\in(b_\mathsf{P}\,..\,e_\mathsf{P}]} = \mathsf{P}$, $b_H, e_H \in [0\,..\,q]$ be such that $\{r_i^{\text{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^-_H$, and letting $z_1 = \text{lrank}(u)$ and $z_2 = \text{rrank}(u)$, we define $\text{pseudoinv}_{\mathcal{T}_\mathsf{Z}}(\ell, u) := (e_\mathsf{P} - c_1, e_\mathsf{P} - c_2)$, where

$$c_1 := \mathsf{rcount}_{A_{\text{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\text{len}}}(\ell, z_1),$$
$$c_2 := \mathsf{rcount}_{A_{\text{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\text{len}}}(\ell, z_2).$$

REMARK 7.4. To see that H is well-defined, recall (see the proof of Proposition 6.8) that $\{\text{pow}(H)\}_{H\in\text{Roots}}$ is prefix-free. Thus, at most one element of $\{\text{pow}(H)\}_{H\in\text{Roots}}$ can be a prefix of $\text{str}(u)$. Furthermore, since $X \neq Y$ implies $\text{pow}(X) \neq \text{pow}(Y)$, $\text{pow}(H)$ uniquely identifies H.

To see that b_P and e_P are well-defined, recall that by Lemma 5.4, if $\mathsf{P} \neq \emptyset$, then all positions in P occupy a contiguous block in SA (see also the proof of Proposition 5.9).

Finally, to show that b_H and e_H are well-defined, note that $\text{pow}(H)$ being a prefix of $\text{str}(u)$ implies, by definition of $\mathcal{T}_\mathsf{Z}$, that there exists $i \in [1\,..\,q]$ such that $H = \text{L-root}(r_i^{\text{lex}})$. Recall (see the proof of Proposition 5.10), that for any $i, i' \in [1\,..\,q]$, $i < i'$ implies $\text{L-root}(r_i^{\text{lex}}) \preceq \text{L-root}(r_{i'}^{\text{lex}})$. Thus, there exists a unique (b_H, e_H) (with $0 \le b_H < e_H \le q$) such that $\{r_i^{\text{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^-_H$.

REMARK 7.5. Note that similarly as for $\text{map}_{\mathcal{T}_\text{st},\mathcal{T}_\mathsf{S}}$ (see Section 7.2.2), the mapping $\text{map}_{\mathcal{T}_\text{st},\mathcal{T}_\mathsf{Z}}$ is not necessarily injective, and hence it may not have an inverse (see also Remark 7.2). To perform the mapping from $\mathcal{T}_\mathsf{Z}$ to $\mathcal{T}_\text{st}$, we will use the above function. Although it is well-defined for every ℓ and u (specified as above), its value is not always meaningful. Below we show a simple but useful condition where it is, and in the following sections we show the more subtle uses.

LEMMA 7.13. *Let $P \in [0\,..\,\sigma)^m$ be a periodic pattern satisfying $e(P) \le m$ and $\text{type}(P) = -1$. Denote $H = \text{L-root}(P)$, $s = \text{L-head}(P)$, $k = \text{L-exp}(P)$, $\ell = e^{\text{full}}(P) - 1$, and $P' = \text{pow}(H) \cdot P(\ell\,..\,m]$. Assume that $\mathsf{P} := \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\} \neq \emptyset$ and let $b_\mathsf{P}, e_\mathsf{P} \in [0\,..\,n]$ be such that $\{\text{SA}[i]\}_{i\in(b_\mathsf{P}\,..\,e_\mathsf{P}]} = \mathsf{P}$ and $b_H, e_H \in [0\,..\,q]$ be such that $\{r_i^{\text{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^-_H$. Finally, let $(b_\text{pre}, e_\text{pre})$ be such that $b_\text{pre} = |\{i \in [1\,..\,q] : T[A_\mathsf{Z}[i]\,..\,n] \prec P'\}|$ and $(b_\text{pre}\,..\,e_\text{pre}] = \{i \in [1\,..\,q] : P' \text{ is a prefix of } T[A_\mathsf{Z}[i]\,..\,n]\}$. Then, it holds*

$$(\text{RangeBeg}(P, T), \text{RangeEnd}(P, T)) = (e_\mathsf{P} - c_1, e_\mathsf{P} - c_2),$$

where $c_1 = \mathsf{rcount}_{A_{\text{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\text{len}}}(\ell, b_\text{pre})$ and $c_2 = \mathsf{rcount}_{A_{\text{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\text{len}}}(\ell, e_\text{pre})$.

Proof. The proof consists of two steps:

1. First, we prove that $|\text{Occ}(P, T)| = c_1 - c_2 = \mathsf{rcount}_{A_{\text{len}}}(\ell, e_\text{pre}) - \mathsf{rcount}_{A_{\text{len}}}(\ell, b_\text{pre})$. By Lemma 6.5, $\text{Occ}(P, T)$ is a disjoint union of $\text{Occ}^\mathsf{a}(P, T)$ and $\text{Occ}^\mathsf{s}(P, T)$ (see the beginning of Section 6.3.4 for definitions). Moreover, since $e(P) \le m$, Lemma 6.6 and its symmetric version (adapted according to Lemma 6.2) imply that $\text{Occ}^\mathsf{a}(P, T) = \emptyset$. Thus, we need to prove $|\text{Occ}^\mathsf{s}(P, T)| = \mathsf{rcount}_{A_{\text{len}}}(\ell, e_\text{pre}) - \mathsf{rcount}_{A_{\text{len}}}(\ell, b_\text{pre})$. By Lemma 6.4(2), it holds $\text{Occ}(P, T) \subseteq \mathsf{R}^-$. Thus, $\text{Occ}^\mathsf{s}(P, T) = \text{Occ}^{\mathsf{s}-}(P, T)$. Recall now that $A_\mathsf{Z}[i] = e^{\text{full}}(r_i^{\text{lex}}) - |\text{pow}(\text{L-root}(r_i^{\text{lex}}))|$. Since the set $\{\text{pow}(H) : H \in \text{Roots}\}$ is prefix-free, it follows, letting $H_j = \text{L-root}(j)$ (where $j \in \mathsf{R}$), that

$$\begin{aligned}\{r_i^{\text{lex}}\}_{i\in(b_\text{pre}\,..\,e_\text{pre}]} &= \{j \in \mathsf{R}'^- : \text{pow}(H) \cdot P(\ell\,..\,m] \text{ is a prefix of } T[e^{\text{full}}(j) - |\text{pow}(H_j)|\,..\,n]\}\\ &= \{j \in \mathsf{R}'^- : \text{pow}(H) \cdot P(\ell\,..\,m] \text{ is a prefix of } \text{pow}(H_j) \cdot T[e^{\text{full}}(j)\,..\,n]\}\\ &= \{j \in \mathsf{R}'^-_H : P(\ell\,..\,m] \text{ is a prefix of } T[e^{\text{full}}(j)\,..\,n]\}\end{aligned}$$

   By Lemma 6.7, we thus have $|\text{Occ}^{\mathsf{s}-}(P, T)| = |\{i \in (b_\text{pre}\,..\,e_\text{pre}] : e^{\text{full}}(r_i^{\text{lex}}) - r_i^{\text{lex}} \ge e^{\text{full}}(P) - 1\}| = \mathsf{rcount}_{A_{\text{len}}}(\ell, e_\text{pre}) - \mathsf{rcount}_{A_{\text{len}}}(\ell, b_\text{pre})$ (recall, that $A_{\text{len}}[i] = e^{\text{full}}(r_i^{\text{lex}}) - r_i^{\text{lex}}$; see Section 5.3.2).
2. Second, we prove that $\text{RangeBeg}(P, T) = e_\mathsf{P} - c_1$. We start by observing that since P is periodic and satisfies $\text{type}(P) = -1$, it follows from Lemmas 6.8 and 6.9 that $\text{RangeBeg}(P, T) = \text{RangeBeg}(X, T) + \delta(P, T) = \text{RangeBeg}(X, T) + \delta^\mathsf{a}(P, T) - \delta^\mathsf{s}(P, T)$, where $X = P[1\,..\,3\tau - 1]$. On the other hand, combining the equalities $\text{L-head}(P) = s$, $\text{L-root}(P) = H$, $\text{L-exp}(P) = k$, and $\text{type}(P) = -1$ with the definition of P yields

$\mathrm{RangeBeg}(X,T) + \delta^{\mathsf{a}}(P,T) = e_{\mathsf{P}}$. Consequently, we obtain $\mathrm{RangeBeg}(P,T) = e_{\mathsf{P}} - \delta^{\mathsf{s}}(P,T)$. It thus remains to show $\delta^{\mathsf{s}}(P,T) = c_1$. By utilizing that by definition of the sequence $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$, for every $i, i' \in (b_H\,..\,e_H]$, $i < i'$ implies $T[e^{\mathrm{full}}(r_i^{\mathrm{lex}})\,..\,n] \preceq T[e^{\mathrm{full}}(r_{i'}^{\mathrm{lex}})\,..\,n]$, it follows by the above formula for $\{r_i^{\mathrm{lex}}\}_{i\in(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]}$ that

$$\{r_i^{\mathrm{lex}}\}_{i\in(b_{\mathrm{pre}}\,..\,e_H]} = \{j \in \mathsf{R}'^{-}_H : P(\ell\,..\,m] \preceq T[e^{\mathrm{full}}(j)\,..\,n]\}.$$

By Lemma 6.10, we thus have $\delta^{\mathsf{s}}(P,T) = |\{i \in (b_{\mathrm{pre}}\,..\,e_H] : e^{\mathrm{full}}(r_i^{\mathrm{lex}}) - r_i^{\mathrm{lex}} \geq e^{\mathrm{full}}(P) - 1\}| = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, b_{\mathrm{pre}}) = c_1$. □

REMARK 7.6. Note that since the range $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}]$ is well-defined even if $e_{\mathrm{pre}} - b_{\mathrm{pre}} = 0$, the above lemma holds even if $|\mathrm{Occ}(P,T)| = 0$.

LEMMA 7.14. *Let v be an explicit periodic node of $\mathcal{T}_{\mathrm{st}}$ such that $e(v) \leq |\mathrm{str}(v)|$ and $\mathrm{type}(v) = -1$. Let $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$ and $\ell = e^{\mathrm{full}}(v) - 1$. Then, it holds $\mathrm{repr}(v) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u)$.*

Proof. Denote $H = \text{L-root}(v)$, $P = \mathrm{str}(v)$, and $(b,e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u)$. Let $s = \ell \bmod |H|$, $k = \lfloor \frac{\ell}{|H|} \rfloor$, and $\mathsf{P} = \{j \in \mathsf{R}^{-}_{s,H} : \text{L-exp}(j) = k\}$. Note that we then have $\text{L-head}(P) = (e^{\mathrm{full}}(P) - 1) \bmod |H| = s$ and $\text{L-exp}(P) = \lfloor \frac{e^{\mathrm{full}}(P)-1}{|H|} \rfloor = k$. Observe that this implies $\mathsf{P} \neq \emptyset$. To see this, consider any $j \in \mathrm{Occ}(\mathrm{str}(v),T) = \mathrm{Occ}(P,T)$. By Lemma 6.2, it follows that $j \in \mathsf{R}$, $\text{L-root}(j) = \text{L-root}(P) = H$, and $\text{L-head}(j) = \text{L-head}(P) = s$, i.e., $j \in \mathsf{R}_{s,H}$. Furthermore, by $e(P) \leq |P|$ and $\mathrm{type}(P) = -1$ we obtain from Lemma 6.4(2) that $\text{L-exp}(j) = \text{L-exp}(P) = k$ and $\mathrm{type}(j) = \mathrm{type}(P) = -1$. Thus, $j \in \mathsf{P}$ and consequently $\mathsf{P} \neq \emptyset$. By definition of $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u)$, we thus obtain that $(b,e) = (e_{\mathsf{P}} - c_1, e_{\mathsf{P}} - c_2)$, where $b_{\mathsf{P}}, e_{\mathsf{P}} \in [0\,..\,n]$ are such that $\{\mathrm{SA}[i]\}_{i\in(b_{\mathsf{P}}\,..\,e_{\mathsf{P}}]} = \mathsf{P}$, $b_H, e_H \in [0\,..\,q]$ are such that $\{r_i^{\mathrm{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^{-}_H$, $z_1 = \mathrm{lrank}(u)$, $z_2 = \mathrm{rrank}(u)$, and

$$c_1 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, z_1),$$
$$c_2 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, z_2).$$

By definition of $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$, we have $\mathrm{str}(u) = \mathrm{pow}(H) \cdot P[e^{\mathrm{full}}(P)\,..\,|P|]$. Thus, denoting $P' = \mathrm{pow}(H) \cdot P[e^{\mathrm{full}}(P)\,..\,|P|]$, by definition of $\mathcal{T}_{\mathsf{Z}}$, we have $\mathrm{lrank}(u) = |\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec P'\}|$ and $(\mathrm{lrank}(u)\,..\,\mathrm{rrank}(u)] = \{i \in [1\,..\,q] : P' \text{ is a prefix of } T[A_{\mathsf{Z}}[i]\,..\,n]\}$. By Lemma 7.13, this implies that $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(u))), e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(u)))) = (e_{\mathsf{P}} - c_1, e_{\mathsf{P}} - c_2) = (b,e)$. This immediately implies $\mathrm{repr}(v) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u)$. □

PROPOSITION 7.16. *Let $H \in \mathrm{Roots}$ and let u be a node of $\mathcal{T}_{\mathsf{Z}}$ such that $\mathrm{pow}(H)$ is a prefix of $\mathrm{str}(u)$. Given the data structure from Section 7.3.1, a pointer to u, and integers $\mathrm{int}(H)$ and $\ell \geq 0$, we can in $\mathcal{O}(\log\log n)$ time compute the pair $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u)$.*

Proof. Let $p := |H|$. We first compute $s := \ell \bmod p$ and $k = \lfloor \frac{\ell}{p} \rfloor$. Next, using the lookup tables L_{pref} and L_{range}, we compute in $\mathcal{O}(1)$ time the pair $(b_{s,H}, e_{s,H}) = (\mathrm{RangeBeg}(X,T), \mathrm{RangeEnd}(X,T))$, where $X = \mathrm{Pref}_{3\tau-1}(s,H)$. By Lemma 6.2, we then have that $b_{s,H} = e_{s,H}$ holds if and only if $\mathsf{R}_{s,H} = \emptyset$, and if $b_{s,H} \neq e_{s,H}$ then $\{\mathrm{SA}[i] : i \in (b_{s,H}\,..\,e_{s,H}]\} = \mathsf{R}_{s,H}$. If $b_{s,H} = e_{s,H}$, we return $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u) = (0,0)$. Let us now assume $b_{s,H} \neq e_{s,H}$.

Next, using the data structure from Section 5.3.2, as explained in the proof of Proposition 5.9, in $\mathcal{O}(1)$ time we compute the pair $(b_{\mathsf{P}}, e_{\mathsf{P}})$ satisfying $\{\mathrm{SA}[i]\}_{i\in(b_{\mathsf{P}}\,..\,e_{\mathsf{P}}]} = \mathsf{P}$, where $\mathsf{P} = \{j \in \mathsf{R}^{-}_{s,H} : \text{L-exp}(j) = k\}$. More precisely, first, in $\mathcal{O}(1)$ time we compute $d = \mathsf{rank}_{B_{\mathrm{exp}},1}(e_{s,H}) - \mathsf{rank}_{B_{\mathrm{exp}},1}(b_{s,H})$. If $d = 0$, then $\mathsf{R}^{-}_{s,H} = \emptyset$, and hence we return $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u) = (0,0)$. Otherwise, in $\mathcal{O}(1)$ time we retrieve $k_{\min} = L_{\mathrm{minexp}}[\mathrm{int}(X)]$. Then, letting $k_{\max} = k_{\min} + d - 1$, we have $[k_{\min}\,..\,k_{\max}] = \{\text{L-exp}(j) : j \in \mathsf{R}^{-}_{s,H}\}$. If $k \notin [k_{\min}\,..\,k_{\max}]$, then $\mathsf{P} = \emptyset$, and thus we return $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u) = (0,0)$. Otherwise, we have two cases. Let $p = \mathsf{rank}_{B_{\mathrm{exp}},1}(b_{s,H})$. If $k = k_{\min}$, then in $\mathcal{O}(1)$ time we compute $(b_{\mathsf{P}}, e_{\mathsf{P}}) = (b_{s,H}, \mathsf{select}_{B_{\mathrm{exp}},1}(p+1))$. If $k > k_{\min}$, in $\mathcal{O}(1)$ time we compute $(b_{\mathsf{P}}, e_{\mathsf{P}}) = (\mathsf{select}_{B_{\mathrm{exp}},1}(p + k - k_{\min}), \mathsf{select}_{B_{\mathrm{exp}},1}(p + k + 1 - k_{\min}))$.

For the final step, we first in $\mathcal{O}(1)$ time compute $e_H = \sum_{H' \preceq H} |\mathsf{R}'^{-}_{H'}|$ using the lookup table L_{runs} stored as part of the structure from Section 5.3.2. Then, it holds that there exists $b_H < e_H$ such that $\{r_i^{\mathrm{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^{-}_H$. Then, in $\mathcal{O}(1)$ time we obtain $z_1 = \mathrm{lrank}(u)$ and $z_2 = \mathrm{rrank}(u)$ (Proposition 4.1). Finally, in $\mathcal{O}(\log\log n)$ time we compute $c_1 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, z_1)$ and $c_2 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, z_2)$ and return $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell,u) = (e_{\mathsf{P}} - c_1, e_{\mathsf{P}} - c_2)$. The range counting queries are implemented using the structure from Proposition 2.1 for the array A, which is stored as part of the structure from Section 5.3.2. □

Handling Nodes Satisfying $e(v) > |\mathrm{str}(v)|$ Next, we present a combinatorial result describing how to compute the value $e(v)$, and to check if it holds $e(v) > |\mathrm{str}(v)|$. We then show how to compute $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ and $(\mathrm{RangeBeg}(Pc,T), \mathrm{RangeEnd}(Pc,T))$ for any periodic pattern $P \in [0\,..\,\sigma)^+$ satisfying $e(P) > |P|$. We will use it to efficiently perform queries on periodic nodes v of $\mathcal{T}_{\mathrm{st}}$ satisfying $e(v) > |\mathrm{str}(v)|$.

LEMMA 7.15. *Let v be an explicit periodic node of $\mathcal{T}_{\mathrm{st}}$. Let $i_1 = \mathrm{lrank}(v)+1$ and $i_2 = \mathrm{rrank}(v)$.*

1. *It holds $\mathrm{SA}[i_1], \mathrm{SA}[i_2] \in \mathsf{R}$ and $e(v) = 1 + \min(e(\mathrm{SA}[i_1]) - \mathrm{SA}[i_1], e(\mathrm{SA}[i_2]) - \mathrm{SA}[i_2])$.*
2. *$e(v) \le |\mathrm{str}(v)|$ holds if and only if $T[\mathrm{SA}[i_1] + e(v) - 1] = T[\mathrm{SA}[i_2] + e(v) - 1]$.*

Proof. Denote $\ell = \mathrm{sdepth}(v)$, $b = \mathrm{lrank}(v)$, and $e = \mathrm{rrank}(v)$.

1. Let $s = \text{L-head}(v)$, $H = \text{L-root}(v)$, $p = |H|$, and $Q = \mathrm{str}(v)$. By definition, we have $b < e$ and $\{\mathrm{SA}[i]\}_{i\in(b\,..\,e]} = \mathrm{Occ}(Q,T)$. On the other hand, by $|Q| \ge 3\tau - 1$ and Lemma 6.2, for every $j \in \mathrm{Occ}(Q,T)$ it holds $j \in \mathsf{R}_{s,H}$. In particular, we thus obtain $\mathrm{SA}[i_1], \mathrm{SA}[i_2] \in \mathsf{R}$.

Next, we prove the following two facts.

- First, we show that there exists $t \in \{1,2\}$ satisfying $e(v) - 1 = e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t]$. By definition, it holds $\ell = \mathrm{LCE}(\mathrm{SA}[i_1], \mathrm{SA}[i_2])$ and $T[\mathrm{SA}[i_1]\,..\,\mathrm{SA}[i_1]+\ell) = T[\mathrm{SA}[i_2]\,..\,\mathrm{SA}[i_2]+\ell)$. If $e - b = 1$, then any $t \in \{1,2\}$ satisfies the claim, since then $\ell = n - \mathrm{SA}[i_t] + 1$, and thus it follows from $\mathrm{SA}[i_t] \in \mathrm{Occ}(Q,T)$ that

$$\begin{aligned} e(v) - 1 &= p + \mathrm{lcp}(Q(0\,..\,\ell-p], Q(p\,..\,\ell]) \\ &= p + \mathrm{lcp}(T[\mathrm{SA}[i_t]\,..\,\mathrm{SA}[i_t]+\ell-p), T[\mathrm{SA}[i_t]+p\,..\,\mathrm{SA}[i_t]+\ell)) \\ &= p + \mathrm{lcp}(T[\mathrm{SA}[i_t]\,..\,n-p], T[\mathrm{SA}[i_t]+p\,..\,n]) \\ &= p + \mathrm{LCE}(\mathrm{SA}[i_t], \mathrm{SA}[i_t]+p) \\ &= e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t]. \end{aligned}$$

  Assume now $e-b > 1$. Then, $T[\mathrm{SA}[i_1]+\ell] \ne T[\mathrm{SA}[i_2]+\ell]$. [11] Thus, by $T[\mathrm{SA}[i_1]+\ell-p] = T[\mathrm{SA}[i_2]+\ell-p]$ there exists $t \in \{1,2\}$ such that $T[\mathrm{SA}[i_t]+\ell] \ne T[\mathrm{SA}[i_t]+\ell-p]$. For such t, we have $\mathrm{LCE}(\mathrm{SA}[i_t], p+\mathrm{SA}[i_t]) \le \ell - p$ and hence $e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t] = p + \mathrm{LCE}(\mathrm{SA}[i_t], \mathrm{SA}[i_t]+p) \le \ell$. We therefore obtain $e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t] = p + \mathrm{lcp}(T[\mathrm{SA}[i_t]\,..\,\mathrm{SA}[i_t]+\ell-p), T[\mathrm{SA}[i_t]+p\,..\,\mathrm{SA}[i_t]+\ell))$. On the other hand, by $Q = T[\mathrm{SA}[i_t]\,..\,\mathrm{SA}[i_t]+\ell)$ we have $e(v)-1 = p+\mathrm{lcp}(Q(0\,..\,\ell-p], Q(p\,..\,\ell]) = p+\mathrm{lcp}(T[\mathrm{SA}[i_t]\,..\,\mathrm{SA}[i_t]+\ell-p), T[\mathrm{SA}[i_t]+p\,..\,\mathrm{SA}[i_t]+\ell))$. Therefore, $e(v) - 1 = e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t]$.
- Second, we show that for every $i \in (b\,..\,e]$, it holds $e(v) - 1 \le e(\mathrm{SA}[i]) - \mathrm{SA}[i]$. For this, recall that $e(\mathrm{SA}[i]) - \mathrm{SA}[i] = p + \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i]+p)$. Therefore, by $\mathrm{SA}[i] \in \mathrm{Occ}(Q,T)$, we obtain $e(v) - 1 = e(Q) - 1 = p + \mathrm{lcp}(Q(0\,..\,\ell-p], Q(p\,..\,\ell]) = p + \mathrm{lcp}(T[\mathrm{SA}[i]\,..\,\mathrm{SA}[i]+\ell-p), T[\mathrm{SA}[i]+p\,..\,\mathrm{SA}[i]+\ell)) \le p + \mathrm{LCE}(\mathrm{SA}[i], \mathrm{SA}[i]+p) = e(\mathrm{SA}[i]) - \mathrm{SA}[i]$.

By the above two facts, we obtain $\min(e(\mathrm{SA}[i_1]) - \mathrm{SA}[i_1], e(\mathrm{SA}[i_2]) - \mathrm{SA}[i_2]) = \min(e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t], e(\mathrm{SA}[i_{3-t}]) - \mathrm{SA}[i_{3-t}]) = e(v) - 1$.

2. We start by showing that $\mathrm{SA}[i_1] + e(v) - 1, \mathrm{SA}[i_2] + e(v) - 1 \le n$. Observe that for every $j \in \mathsf{R}$, by the uniqueness of $T[n]$, it holds $e(j) \le n$. Consider any $i \in (b\,..\,e]$. Above, we proved $e(v) - 1 \le e(\mathrm{SA}[i]) - \mathrm{SA}[i]$. Thus, we obtain $\mathrm{SA}[i] + e(v) - 1 \le e(\mathrm{SA}[i]) \le n$. In particular, $\mathrm{SA}[i_1] + e(v) - 1, \mathrm{SA}[i_2] + e(v) - 1 \le n$. We now prove the equivalence. Recall, that $|\mathrm{str}(v)| = \ell = \mathrm{LCE}(\mathrm{SA}[i_1], \mathrm{SA}[i_2])$ holds by definition. Let us first assume $e(v) \le \ell$. By the assumption $T[\mathrm{SA}[i_1]\,..\,\mathrm{SA}[i_1]+\ell) = T[\mathrm{SA}[i_2]\,..\,\mathrm{SA}[i_2]+\ell)$, this immediately implies $T[\mathrm{SA}[i_1] + e(v) - 1] = T[\mathrm{SA}[i_2] + e(v) - 1]$. To show the opposite implication, assume by contraposition that $e(v) > \ell$. Since by definition we have $e(v) \le |\mathrm{str}(v)| + 1$, we must have $e(v) = \ell + 1$. Then, by definition of LCE, we have $T[\mathrm{SA}[i_1] + e(v) - 1] = T[\mathrm{SA}[i_1] + \ell] \ne T[\mathrm{SA}[i_2] + \ell] = T[\mathrm{SA}[i_2] + e(v) - 1]$. □

PROPOSITION 7.17. *Let $P \in [0\,..\,\sigma)^+$ be a periodic pattern satisfying $e(P) > |P|$. Given the structure from Section 7.3.1, and the values* $\text{L-head}(P)$, $\text{L-root}(P)$, *and $|P|$, we can in $\mathcal{O}(\log\log n)$ time compute the pair* $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$.

[11] To see that symbols $T[\mathrm{SA}[i_1] + \ell]$ and $T[\mathrm{SA}[i_2] + \ell]$ are well-defined, observe that by $\ell > 0$ and $b + 1 < e$, it follows that $\mathrm{SA}[i_1] + \ell - 1 \ne \mathrm{SA}[i_2] + \ell - 1$. On the other hand, we have $T[\mathrm{SA}[i_1] + \ell - 1] = T[\mathrm{SA}[i_2] + \ell - 1]$. Thus, by the uniqueness of $T[n]$ we must have $\mathrm{SA}[i_1] + \ell - 1 < n$ and $\mathrm{SA}[i_2] + \ell - 1 < n$

Proof. Denote $s = \text{L-head}(P)$, $H = \text{L-root}(P)$, and $m = |P|$. First, in $\mathcal{O}(1)$ time we compute $k := \text{L-exp}(P) = \lfloor \frac{m-s}{|H|} \rfloor$ and $t := \text{L-tail}(P) = m - s - k|H|$. Next, using the lookup table L_{pref}, in $\mathcal{O}(1)$ time we compute $X := \text{Pref}_{3\tau-1}(s, H) = P[1\,..\,3\tau{-}1]$.

Next, we compute $|\text{Occ}(P,T)|$. Recall that by Lemma 6.5, $|\text{Occ}(P,T)| = |\text{Occ}^{\text{a}}(P,T)| + |\text{Occ}^{\text{s}}(P,T)| = |\text{Occ}^{\text{a}-}(P,T)| + |\text{Occ}^{\text{a}+}(P,T)| + |\text{Occ}^{\text{s}-}(P,T)| + |\text{Occ}^{\text{s}+}(P,T)|$ (see Section 6.3.4).

- To compute $|\text{Occ}^{\text{a}-}(P,T)|$, we proceed as in the proof of Proposition 6.7, except for one modification: Since we already have L-head(P), L-root(P), and L-exp(P) (note that we do not need L-tail(P) here since we assumed $e(P) = |P| + 1$), we can skip the first step which takes $\mathcal{O}(1 + m/\log_\sigma n)$ time. Note that after such modification, we no longer need the packed representation of the whole pattern P, but only $P[1\,..\,3\tau-1]$, which we computed above. The rest of the algorithm in Proposition 6.7 takes $\mathcal{O}(1)$ time. The structures from Proposition 6.7 that we used (augmented bitvector B_{exp} and lookup tables L_{minexp} and L_{range}) are components of the structure from Section 7.3.1.
- To compute $|\text{Occ}^{\text{s}-}(P,T)|$, we proceed as in Proposition 6.8, except for two modifications. First, we again already have L-head(P), L-root(P), and L-exp(P), which lets us skip the first step taking $\mathcal{O}(1 + m/\log_\sigma n)$ time. Second, rather than computing b_{pre} and e_{pre} in $\mathcal{O}(m/\log_\sigma n + \log\log n)$ time, we use the lookup table L_{runs} stored in the structure from Section 7.3.1. More precisely, b_{pre} and e_{pre} are obtained in $\mathcal{O}(1)$ time by looking up in L_{runs} the pair associated with the key (H, H'), where H' is a length-t prefix of H (note that $P[e^{\text{full}}(P)\,..\,m] = H'$). The rest of the algorithm in Proposition 6.8 takes $\mathcal{O}(\log\log n)$ time. Again, the components used in Proposition 6.8 are present in structure from Section 7.3.1.

The values $|\text{Occ}^{\text{a}+}(P,T)|$ and $|\text{Occ}^{\text{s}+}(P,T)|$ are computed analogously (see the proof of Proposition 6.9) using the symmetric components of the structure from Section 7.3.1. We can thus compute $|\text{Occ}(P,T)|$ in $\mathcal{O}(\log\log n)$ time.

The next step of the algorithm is to compute $\delta(P,T)$ (Section 6.3.4). Observe (see Section 6.3.1) that $e(P) > |P|$ implies $\text{type}(P) = -1$. Recall that for such P, by Lemma 6.9, $\delta(P,T) = \delta^{\text{a}}(P,T) - \delta^{\text{s}}(P,T)$.

- To compute $\delta^{\text{a}}(P,T)$, we proceed as in the proof of Proposition 6.10, employing the same modification as when computing $|\text{Occ}^{\text{a}-}(P,T)|$ above. Thus, the computation takes $\mathcal{O}(1)$ time. Proposition 6.10 uses the structure from Proposition 6.7 and, as above, the used components are already present in the structure from Section 7.3.1.
- To compute $\delta^{\text{s}}(P,T)$, we observe that for a periodic pattern P satisfying $e(P) > |P|$, it holds by Lemma 6.2(2) that $\text{Pos}^{\text{s}}(P,T) = \text{Occ}^{\text{s}-}(P,T)$. Consequently, we can compute $\delta^{\text{s}}(P,T) = |\text{Occ}^{\text{s}-}(P,T)|$ as above in $\mathcal{O}(\log\log n)$ time.

Combining the above two steps, the computation of $\delta(P,T)$ takes $\mathcal{O}(\log\log n)$ time.

We use the above values to obtain $(\text{RangeBeg}(P,T), \text{RangeEnd}(P,T))$ as follows. By Lemma 6.8, $\text{RangeBeg}(P,T) = \text{RangeBeg}(X,T) + \delta(P,T)$, where $X = P[1\,..\,3\tau{-}1]$. The value $\text{RangeBeg}(X,T)$ is obtained in $\mathcal{O}(1)$ time using the lookup table L_{range}. We thus obtain $\text{RangeBeg}(P,T)$. By definition, we then compute $\text{RangeEnd}(P,T) = \text{RangeEnd}(P,T) + |\text{Occ}(P,T)|$. In total, the query takes $\mathcal{O}(\log\log n)$ time. □

Remark 7.7. Note that the above result holds even if $\text{Occ}(P,T) = \emptyset$. Thus, it is more general than the result needed to support efficient processing of periodic nodes v of $\mathcal{T}_{\text{st}}$ satisfying $e(v) > |\text{str}(v)|$, since for such nodes we have $\text{Occ}(\text{str}(v), T) \neq \emptyset$.

Proposition 7.18. *Let $P \in [0\,..\,\sigma)^+$ be a periodic pattern satisfying $e(P) > |P|$. Given any $c \in [0\,..\,\sigma)$, the structure from Section 7.3.1, and the values* L-head(P), L-root(P), *and* $|P|$, *we can in $\mathcal{O}(\log\log n)$ time compute the pair* $(\text{RangeBeg}(Pc,T), \text{RangeEnd}(Pc,T))$.

Proof. Denote $P' = Pc$ and $m = |P| + 1 = |P'|$. Observe, that since P is periodic and it is a prefix of P', by Lemma 6.3, P' is also periodic and it holds L-head(P) = L-head(P') and L-root(P) = L-root(P'). Let us denote $s = \text{L-head}(P) = \text{L-head}(P')$ and $H = \text{L-root}(P) = \text{L-root}(P')$. By the assumption, we have $e(P) = m$. First, in $\mathcal{O}(1)$ time we compute $t := \text{L-tail}(P) = (m - 1 - s) \bmod |H|$. We then check if $e(P') \leq |P'|$ by comparing c to $H[t+1]$. If $c = H[t+1]$, then we have $e(P') > |P'|$. Since we have L-head$(P') = s$, L-root$(P') = H$, and $|P'| = m$, in $\mathcal{O}(\log\log n)$ time we thus compute and return $(\text{RangeBeg}(P',T), \text{RangeEnd}(P',T))$ using Proposition 7.17. Let us thus assume $c \neq H[t+1]$, i.e., $e(P') \leq |P'|$. We then compute $\text{type}(P')$ by comparing c with $H[t+1]$. Let us assume that $c \prec H[t+1]$, i.e., $\text{type}(P) = -1$ (the case $\text{type}(P) = +1$ is handled symmetrically). We now execute the modified algorithm from Proposition 6.12 for P'. The modification is to replace implementation of operations taking

$\Theta(m/\log_\sigma n)$ time with faster alternatives, exploiting the fact that by $e(P') = e(P)$, L-head$(P') =$ L-head(P), and L-root$(P') =$ L-root(P) it follows that $e^{\text{full}}(P') = e^{\text{full}}(P) = e(P) - \text{L-tail}(P) = m - t$ and thus $P'[e^{\text{full}}(P')\,..\,m]$ is of length $t+1 \le \tau$ (importantly, the modified algorithm will not use the components of the data structures in Section 6.3 which are not part of the structure from Section 7.3.1). More precisely:

- First, using L_{pref}, in $\mathcal{O}(1)$ time we compute $X = \text{Pref}_{3\tau-1}(s, H) = P'[1\,..\,3\tau{-}1]$.
- We then compute $|\text{Occ}(P', T)|$. First, note that since $e(P') \le |P'|$ and type$(P') = -1$, it follows by Lemma 6.4(2) that $\text{Occ}(P', T) \subseteq \mathsf{R}^-$, and that for every $j \in \text{Occ}(P', T)$ it holds L-exp$(j) =$ L-exp(P'). Thus, the sets $\text{Occ}^{\mathsf{a}}(P', T)$ and $\text{Occ}^{\mathsf{s}+}(P', T)$ are empty, and hence it remains to explain the computation of $|\text{Occ}^{\mathsf{s}-}(P', T)|$ (Proposition 6.8). Observe, that the expensive operations are the computation of L-head(P'), L-root(P'), L-exp(P'), and the pair $(b_{\text{pre}}, e_{\text{pre}})$. Observe, however, that here we already have $s =$ L-head(P'), $H =$ L-root(P'), and $e(P') = m$. This lets us deduce $k :=$ L-exp$(P') = \lfloor \frac{m-1-s}{|H|} \rfloor$ in $\mathcal{O}(1)$ time. As for the computation of $(b_{\text{pre}}, e_{\text{pre}})$, we first in $\mathcal{O}(1)$ time compute $H' := P'[e^{\text{full}}(P')\,..\,m] = H[1\,..\,t+1]$, and then obtain $(b_{\text{pre}}, e_{\text{pre}})$ by looking up the pair associated with the key (H, H') in the lookup table L_{runs}. The rest of the algorithm in Proposition 6.8 takes $\mathcal{O}(\log\log n)$ time.
- Finally, we compute $\delta(P', T)$. By type$(P') = -1$ and Lemma 6.9, it holds $\delta(P', T) = \delta^{\mathsf{a}}(P', T) - \delta^{\mathsf{s}}(P', T)$. To compute $\delta^{\mathsf{a}}(P', T)$, we proceed as in the proof of Proposition 6.10. The string X was already obtained above. The expensive step in Proposition 6.10 is the computation of L-root(P') and L-exp(P'). As noted above, here we already have L-root$(P') = H$, and in $\mathcal{O}(1)$ time we can compute L-exp$(P') = \lfloor \frac{e(P')-1-\text{L-head}(P')}{|\text{L-root}(P')|} \rfloor = \lfloor \frac{m-1-s}{|H|} \rfloor$. The rest of the algorithm in Proposition 6.10 takes $\mathcal{O}(1)$ time. We then compute $\delta^{\mathsf{s}}(P', T)$ using a modified Proposition 6.11. The expensive part is the computation of x and x'. After those are computed, the rest takes $\mathcal{O}(\log\log n)$ time. Here, we obtain x by observing that it is equal to b_{pre} (which was computed above), and then obtain x' using L_{runs} (this only requires knowing L-root(P'), which we already have).

Note that all components of the structure from Propositions 6.8, 6.10, and 6.11 that we used are also components of the structure from Section 7.3.1. Using the above values, we now obtain $(\text{RangeBeg}(P', T), \text{RangeEnd}(P', T))$ as follows. By Lemma 6.8, it holds $\text{RangeBeg}(P', T) = \text{RangeBeg}(X, T) + \delta(P', T)$, where $X = P'[1\,..\,3\tau{-}1]$. The value $\text{RangeBeg}(X, T)$ is obtained in $\mathcal{O}(1)$ time using the lookup table L_{range}. We thus obtain $\text{RangeBeg}(P', T)$. By definition, we then compute $\text{RangeEnd}(P', T) = \text{RangeEnd}(P', T) + |\text{Occ}(P', T)|$. □

Remark 7.8. Note that, analogously to Proposition 7.17 (see Remark 7.7), the above result holds even if $\text{Occ}(Pc, T) = \emptyset$.

7.3.3 Implementation of LCA(u, v)

Lemma 7.16. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\text{st}}$ such that* LCA(v_1, v_2) *is periodic and it holds* $e(\text{LCA}(v_1, v_2)) \le |\text{str}(\text{LCA}(v_1, v_2))|$ *and* type$(\text{LCA}(v_1, v_2)) = -1$. *Then, v_1 and v_2 are periodic and it holds $e(v_1) \le |\text{str}(v_1)|$, $e(v_2) \le |\text{str}(v_2)|$, and* type$(v_1) =$ type$(v_2) = -1$. *Moreover,*

$$\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(\text{LCA}(v_1, v_2)) = \text{LCA}(\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_1), \text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_2)).$$

Proof. Denote $v = \text{LCA}(v_1, v_2)$, $Q = \text{str}(v)$, $H =$ L-root(v), and $s =$ L-head(v). Let also $Q_1 = \text{str}(v_1)$. By $|Q| \ge 3\tau - 1$ and since v is an ancestor of v_1, we have $\text{lcp}(Q, Q_1) \ge 3\tau - 1$. Consequently, by Lemma 6.3, the node v_1 is periodic and it holds L-root$(v_1) =$ L-root$(Q_1) =$ L-root$(Q) =$ L-root$(v) = H$ and L-head$(v_1) =$ L-head$(Q_1) =$ L-head$(Q) =$ L-head$(v) = s$. Furthermore, by $e(Q) \le |Q|$ and type$(Q) = -1$, it holds $Q[e(Q)] \prec Q[e(Q) - p]$. Since Q is a prefix of Q_1, this immediately implies $e(Q_1) = e(Q) \le |Q| \le |Q_1|$ and $Q_1[e(Q_1)] = Q[e(Q)] \prec Q[e(Q) - p] = Q_1[e(Q_1) - p]$, i.e., type$(Q_1) = -1$. We have thus shown $e(v_1) \le |\text{str}(v_1)|$ and type$(v_1) = -1$. Analogously, we obtain that v_2 is periodic and it holds $e(v_2) = e(v)$, L-root$(v_2) = H$, L-head$(v_2) = s$, $e(v_2) \le |\text{str}(v_2)|$, and type$(v_2) = -1$. We have thus shown that $u_1 = \text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_1)$ and $u_2 = \text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_2)$ are well-defined (see Section 7.3.2).

Let $u = \text{LCA}(u_1, u_2)$, $\ell' = \text{sdepth}(u)$, and $\ell = \text{sdepth}(v)$. By Observation 4.1, we have $\ell = \text{lcp}(\text{str}(v_1), \text{str}(v_2))$, $\ell' = \text{lcp}(\text{str}(u_1), \text{str}(u_2))$, $\text{str}(v) = \text{str}(v_1)[1\,..\,\ell]$, and $\text{str}(u) = \text{str}(u_1)[1\,..\,\ell']$. Denote $\delta = e^{\text{full}}(v)$. As observed above, $e(v_1) = e(v)$, L-head$(v_1) =$ L-head(v), and L-root$(v_1) =$ L-root(v). Thus, $e^{\text{full}}(v_1) = 1 + \text{L-head}(v_1) + |\text{L-root}(v_1)| \cdot \lfloor \frac{e(v_1)-1-\text{L-head}(v_1)}{|\text{L-root}(v_1)|} \rfloor = 1 + \text{L-head}(v) + |\text{L-root}(v)| \cdot \lfloor \frac{e(v)-1-\text{L-head}(v)}{|\text{L-root}(v)|} \rfloor = e^{\text{full}}(v) = \delta$. Analogously, $e^{\text{full}}(v_2) = \delta$. By definition of $\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_1)$ and $\text{map}_{\mathcal{T}_{\text{st}}, \mathcal{T}_{\mathsf{z}}}(v_2)$, we have $\text{str}(u_1) = \text{pow}(H) \cdot \text{str}(v_1)[e^{\text{full}}(v_1)\,..\,|\text{str}(v_1)|] = \text{pow}(H) \cdot \text{str}(v_1)[\delta\,..\,|\text{str}(v_1)|]$ and $\text{str}(u_2) = \text{pow}(H) \cdot \text{str}(v_2)[e^{\text{full}}(v_2)\,..\,|\text{str}(v_2)|] = \text{pow}(H) \cdot \text{str}(v_2)[\delta\,..\,|\text{str}(v_2)|]$.

Thus, $\ell' = \mathrm{lcp}(\mathrm{str}(u_1), \mathrm{str}(u_2)) = |\mathrm{pow}(H)| + (\mathrm{lcp}(\mathrm{str}(v_1), \mathrm{str}(v_2)) - \delta + 1) = |\mathrm{pow}(H)| + \ell - \delta + 1$. Consequently, $\mathrm{str}(u) = \mathrm{str}(u_1)[1\,..\,\ell'] = \mathrm{pow}(H){\cdot}\mathrm{str}(v_1)[\delta\,..\,\delta{+}\ell'{-}|\mathrm{pow}(H)|{-}1] = \mathrm{pow}(H){\cdot}\mathrm{str}(v_1)[\delta\,..\,\ell] = \mathrm{pow}(H){\cdot}\mathrm{str}(v)[\delta\,..\,\ell] = \mathrm{pow}(H) \cdot \mathrm{str}(v)[e^{\mathrm{full}}(v)\,..\,|\mathrm{str}(v)|]$. Thus, by definition of $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(u)$, and since no two nodes of $\mathcal{T}_{\mathsf{Z}}$ have the same value of str, we therefore obtain $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v) = u$. □

LEMMA 7.17. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{LCA}(v_1, v_2)$ is periodic. Denote $v = \mathrm{LCA}(v_1, v_2)$, $i_{\min} = \min(\mathrm{lrank}(v_1), \mathrm{lrank}(v_2)) + 1$, and $i_{\max} = \max(\mathrm{rrank}(v_1), \mathrm{rrank}(v_2))$. Then, it holds:*

1. *$\mathrm{SA}[i_{\min}], \mathrm{SA}[i_{\max}] \in \mathsf{R}$ and $e(v) = 1 + \min(e(\mathrm{SA}[i_{\min}]) - \mathrm{SA}[i_{\min}], e(\mathrm{SA}[i_{\max}]) - \mathrm{SA}[i_{\max}])$.*
2. *$e(v) \leq |\mathrm{str}(v)|$ holds if and only if $T[\mathrm{SA}[i_{\min}] + e(v) - 1] = T[\mathrm{SA}[i_{\max}] + e(v) - 1]$.*

Proof. Denote $i_1 = \mathrm{lrank}(v_1) + 1$, $i_2 = \mathrm{rrank}(v_1)$, $i_3 = \mathrm{lrank}(v_2) + 1$, and $i_4 = \mathrm{rrank}(v_2)$.

1. Let $H = \text{L-root}(v)$, and $p = |H|$. We start by noting that $\mathrm{SA}[i_{\min}], \mathrm{SA}[i_{\max}] \in \mathsf{R}$ follows by Lemma 7.15(1). Next, we prove the formula for $e(v)$.

First, we show that $e(v) = \min(e(v_1), e(v_2))$. Observe that if P is a prefix of S, both P and S and periodic, and $\text{L-root}(S) = H$, then, $e(S) = 1 + p + \mathrm{lcp}(S(0\,..\,|S| - p], S(p\,..\,|S|]) \geq 1 + p + \mathrm{lcp}(S(0\,..\,|P| - p], S(p\,..\,|P|]) = 1 + p + \mathrm{lcp}(P(0\,..\,|P| - p], P(p\,..\,|P|]) = e(P)$. Since $\mathrm{str}(v)$ is a prefix of $\mathrm{str}(v_1)$ and $\mathrm{str}(v_2)$, we thus obtain $e(v_1) \geq e(v)$ and $e(v_2) \geq e(v)$. It remains to show that there exists $t \in \{1, 2\}$ such that $e(v_t) = e(v)$. Consider two cases:

- If $e(v) = |\mathrm{str}(v)| + 1$, then there are two possibilities. Either for some $t \in \{1, 2\}$, we have $|\mathrm{str}(v_t)| = |\mathrm{str}(v)|$, in which case $\mathrm{str}(v_t) = \mathrm{str}(v)$ and thus $e(v_t) = e(v)$ follows. The other possibility is that $|\mathrm{str}(v_1)| > |\mathrm{str}(v)|$ and $|\mathrm{str}(v_2)| > |\mathrm{str}(v)|$. Since $\mathrm{str}(v)$ is the longest common prefix of $\mathrm{str}(v_1)$ and $\mathrm{str}(v_2)$, we then have $\mathrm{str}(v_1)[e(v)] = \mathrm{str}(v_1)[|\mathrm{str}(v)| + 1] \neq \mathrm{str}(v_2)[|\mathrm{str}(v) + 1] = \mathrm{str}(v_2)[e(v)]$. Thus, there exists $t \in \{1, 2\}$ such that $\mathrm{str}(v_t)[e(v)] \neq \mathrm{str}(v)[e(v) - p]$. By definition, for such t we have $e(v_t) = e(v)$.
- Let us now assume $e(v) \leq |\mathrm{str}(v)|$. This implies that $\mathrm{str}(v)[e(v)] = \mathrm{str}(v_1)[e(v)] = \mathrm{str}(v_2)[e(v)]$ and $\mathrm{str}(v)[e(v)] \neq \mathrm{str}(v)[e(v) - p]$. Thus, by $\mathrm{str}(v)[1\,..\,e(v)] = \mathrm{str}(v_1)[1\,..\,e(v)] = \mathrm{str}(v_2)[1\,..\,e(v)]$ we obtain $e(v_1) = e(v_2) = e(v)$.

We have thus shown that there exists $t \in \{1, 2\}$ such that $e(v) = e(v_t)$. Combined with $e(v_1) \geq e(v)$ and $e(v_2) \geq e(v)$, this yields $\min(e(v_1), e(v_2)) = \min(e(v_t), e(v_{3-t})) = e(v)$.

By the above and Lemma 7.15(1) for v_1 and v_2, it holds $e(v) = 1 + \min_{t\in[1\,..\,4]}\{e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t]\}$. To show that this is equal to the expression for $e(v)$ from the claim, we first observe that letting $X = \mathrm{str}(v)[1\,..\,3\tau - 1]$ and $(b, e) = (\mathrm{RangeBeg}(X, T), \mathrm{RangeEnd}(X, T))$, we have $i_t \in (b\,..\,e]$ for all $t \in [1\,..\,4]$. Observe that by Lemma 5.4, the sequence $(e(\mathrm{SA}[i]) - \mathrm{SA}[i])_{i=b+1}^{e}$ is bitonic, i.e., there exists $m \in (b\,..\,e]$ such that $e(\mathrm{SA}[b + 1]) - \mathrm{SA}[b + 1] \leq e(\mathrm{SA}[b + 2]) - \mathrm{SA}[b + 2] \leq \cdots \leq e(\mathrm{SA}[m]) - \mathrm{SA}[m]$ and $e(\mathrm{SA}[m]) - \mathrm{SA}[m] \geq e(\mathrm{SA}[m + 1]) - \mathrm{SA}[m + 1] \geq \cdots \geq e(\mathrm{SA}[e]) - \mathrm{SA}[e]$. This implies that for every triple $k_1, k_2, k_3 \in (b\,..\,e]$, the inequalities $k_1 \leq k_2 \leq k_3$ imply $\min(e(\mathrm{SA}[k_1]) - \mathrm{SA}[k_1], e(\mathrm{SA}[k_3]) - \mathrm{SA}[k_3]) = \min_{t\in[1\,..\,3]}\{e(\mathrm{SA}[k_t]) - \mathrm{SA}[k_t]\}$. For a proof, consider two cases:

- If $k_2 < m$, then by the bitonic property, we have $e(\mathrm{SA}[k_2]) - \mathrm{SA}[k_2] \geq e(\mathrm{SA}[k_1]) - \mathrm{SA}[k_1]$. Thus, the expression $e(\mathrm{SA}[k_2]) - \mathrm{SA}[k_2]$ has no effect on the minimum.
- If $k_2 \geq m$, then by the bitonic property, we have $e(\mathrm{SA}[k_2]) - \mathrm{SA}[k_2] \geq e(\mathrm{SA}[k_3]) - \mathrm{SA}[k_3]$. Thus, the expression $e(\mathrm{SA}[k_2]) - \mathrm{SA}[k_2]$ can again be excluded in the minimum.

By the above, letting $i'_{\min} = \min_{t\in[1\,..\,4]}\{i_t\}$ and $i'_{\max} = \max_{i\in[1\,..\,4]}\{i_t\}$, we thus have $e(v) = 1 + \min_{t\in[1\,..\,4]}\{e(\mathrm{SA}[i_t]) - \mathrm{SA}[i_t]\} = 1 + \min(e(\mathrm{SA}[i'_{\min}]) - \mathrm{SA}[i'_{\min}], e(\mathrm{SA}[i'_{\max}]) - \mathrm{SA}[i'_{\max}])$.

It remains to show that $i'_{\min} = i_{\min}$ and $i'_{\max} = i_{\max}$. For this, it suffices to note that by definition, we have $i_1 \leq i_2$ and $i_3 \leq i_4$, thus, $i'_{\min} = \min_{t\in[1\,..\,4]}\{i_t\} = \min(i_1, i_3) = i_{\min}$ and analogously, $i'_{\max} = \max_{t\in[1\,..\,4]}\{i_t\} = \max(i_2, i_4) = i_{\max}$.

2. As observed in the proof of Lemma 7.15(2), it holds $\mathrm{SA}[i_1] + e(v_1) - 1 \leq n$, $\mathrm{SA}[i_2] + e(v_1) - 1 \leq n$, $\mathrm{SA}[i_3] + e(v_2) - 1 \leq n$, and $\mathrm{SA}[i_4] + e(v_2) - 1 \leq n$. Thus, by $e(v) = \min(e(v_1), e(v_2))$, for every $t \in [1\,..\,4]$, we have $\mathrm{SA}[i_t] + e(v) - 1 \leq n$. In particular, $\mathrm{SA}[i_{\min}] + e(v) - 1 \leq n$ and $\mathrm{SA}[i_{\max}] + e(v) - 1 \leq n$. We now prove the equivalence. Let us first assume $e(v) \leq |\mathrm{str}(v)|$. Then, since $\mathrm{str}(v)$ is a prefix of both $\mathrm{str}(v_1)$ and $\mathrm{str}(v_2)$ and $\mathrm{SA}[i_{\min}], \mathrm{SA}[i_{\max}] \in \mathrm{Occ}(\mathrm{str}(v_1), T) \cup \mathrm{Occ}(\mathrm{str}(v_2), T)$, it follows that $\mathrm{SA}[i_{\min}], \mathrm{SA}[i_{\max}] \in \mathrm{Occ}(\mathrm{str}(v), T)$. Therefore, $T[\mathrm{SA}[i_{\min}] + e(v) - 1] = T[\mathrm{SA}[i_{\max}] + e(v) - 1]$ follows immediately. To show the opposite implication, assume by contraposition that $e(v) = |\mathrm{str}(v)| + 1$. Then, there are two possibilities. Either for some $t \in \{1, 2\}$ we

have $\mathrm{str}(v_t) = \mathrm{str}(v)$, in which case $\mathrm{str}(v_t)$ is a prefix of $\mathrm{str}(v_{3-t})$, which in turn implies $i_{\min} = \mathrm{lrank}(v_t) + 1$ and $i_{\max} = \mathrm{rrank}(v_t)$. Then, $e(v) = e(v_t)$ and by applying Lemma 7.15(2) to v_t, we obtain $T[\mathrm{SA}[i_{\min}] + e(v) - 1] = T[\mathrm{SA}[\mathrm{lrank}(v_t) + 1] + e(v_t) - 1] \neq T[\mathrm{SA}[\mathrm{rrank}(v_t)] + e(v_t) - 1] = T[\mathrm{SA}[i_{\max}] + e(v) - 1]$. The other possibility is that $|\mathrm{str}(v_1)| > |\mathrm{str}(v)|$ and $|\mathrm{str}(v_2)| > |\mathrm{str}(v)|$. Then, since $\mathrm{str}(v)$ is the longest common prefix of $\mathrm{str}(v_1)$ and $\mathrm{str}(v_2)$, neither of v_1 or v_2 is an ancestor of the other, and hence either it holds $i_{\min} = i_1 \leq i_2 < i_3 \leq i_4 = i_{\max}$ or $i_{\min} = i_3 \leq i_4 < i_1 \leq i_2 = i_{\max}$. In the first case $\mathrm{SA}[i_{\min}] \in \mathrm{Occ}(\mathrm{str}(v_1), T)$ and $\mathrm{SA}[i_{\max}] \in \mathrm{Occ}(\mathrm{str}(v_2), T)$, and in the second case $\mathrm{SA}[i_{\min}] \in \mathrm{Occ}(\mathrm{str}(v_2), T)$ and $\mathrm{SA}[i_{\max}] \in \mathrm{Occ}(\mathrm{str}(v_1), T)$. Therefore, in both cases we have $T[\mathrm{SA}[i_{\min}] + e(v) - 1] = T[\mathrm{SA}[i_{\min}] + |\mathrm{str}(v)|] \neq T[\mathrm{SA}[i_{\max}] + |\mathrm{str}(v)|] = T[\mathrm{SA}[i_{\max}] + e(v) - 1]$. □

REMARK 7.9. Observe that in Lemma 7.17, it does not necessarily hold that $\mathrm{lrank}(\mathrm{LCA}(v_1, v_2)) = i_{\min}$ or $\mathrm{rrank}(\mathrm{LCA}(v_1, v_2)) = i_{\max}$. Thus, the lemma does not immediately follow as a corollary from Lemma 7.15.

PROPOSITION 7.19. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{LCA}(v_1, v_2)$ is periodic. Given the data structure from Section 7.3.1 and the pairs $\mathrm{repr}(v_1)$ and $\mathrm{repr}(v_2)$, we can in $\mathcal{O}(\log\log n)$ time compute $\mathrm{repr}(\mathrm{LCA}(v_1, v_2))$.*

Proof. Denote $v = \mathrm{LCA}(v_1, v_2)$, $\mathrm{repr}(v_1) = (b_1, e_1)$ and $\mathrm{repr}(v_2) = (b_2, e_2)$ (recall that for $i \in \{1, 2\}$, we have $b_i = \mathrm{lrank}(v_i)$ and $e_i = \mathrm{rrank}(v_i)$).

First, in $\mathcal{O}(1)$ time we compute $i_{\min} = \min(b_1, b_2) + 1$ and $i_{\max} = \max(e_1, e_2)$. By Lemma 7.17(1), we have $\mathrm{SA}[i_{\min}], \mathrm{SA}[i_{\max}] \in \mathsf{R}$. Using Proposition 5.14, in $\mathcal{O}(\log\log n)$ time we compute $j_{\min} = \mathrm{SA}[i_{\min}]$ and $j_{\max} = \mathrm{SA}[i_{\max}]$. Next, using Proposition 5.7 in $\mathcal{O}(1)$ time we compute $H := \text{L-root}(j_{\min})$, $s := \text{L-head}(j_{\min})$, $k_{\min} = \text{L-exp}(j_{\min})$, $k_{\max} = \text{L-exp}(j_{\max})$, $t_{\min} = \text{L-tail}(j_{\min})$, and $t_{\max} = \text{L-tail}(j_{\max})$. Observe that since v is periodic, and $j_{\min}, j_{\max} \in \mathrm{Occ}(\mathrm{str}(v_1), T) \cup \mathrm{Occ}(\mathrm{str}(v_2), T) \subseteq \mathrm{Occ}(\mathrm{str}(v), T)$, it follows by Lemmas 5.4 and 6.2, that $\text{L-root}(v) = \text{L-root}(j_{\max}) = H$ and $\text{L-head}(v) = \text{L-head}(j_{\max}) = s$. In $\mathcal{O}(1)$ time we thus compute $e_{\min} := e(j_{\min}) = j_{\min} + s + k_{\min}|H| + t_{\min}$ and $e_{\max} := e(j_{\max}) = j_{\max} + s + k_{\max}|H| + t_{\max}$. Next, in $\mathcal{O}(1)$ time we compute $e_v := e(v) = 1 + \min(e_{\min} - j_{\min}, e_{\max} - j_{\max})$ (see Lemma 7.17(1)). Using Lemma 7.17(2), we then in $\mathcal{O}(1)$ time check if it holds $e(v) \leq |\mathrm{str}(v)|$ by comparing $T[j_{\min} + e_v - 1]$ with $T[j_{\max} + e_v - 1]$. Consider two cases:

- Let $T[j_{\min} + e_v - 1] = T[j_{\max} + e_v - 1]$, i.e., $e(v) \leq |\mathrm{str}(v)|$. Recall now that $j_{\min} \in \mathrm{Occ}(\mathrm{str}(v), T)$. In $\mathcal{O}(1)$ time we thus compute $\mathrm{type}(v)$ by comparing $T[j_{\min} + e_v - 1]$ with $T[j_{\min} + e_v - 1 - |H|]$. Let us assume that $T[j_{\min} + e_v - 1] \prec T[j_{\min} + e_v - 1 - |H|]$, i.e., $\mathrm{type}(v) = -1$ (the case $\mathrm{type}(v) = +1$ is handled symmetrically, using the part of the structure from Section 7.3.1 adapted according to Lemma 5.4). By Lemma 7.16, we now have that v_1 and v_2 are periodic and it holds $e(v_1) \leq |\mathrm{str}(v_1)|$, $e(v_2) \leq |\mathrm{str}(v_2)|$, and $\mathrm{type}(v_1) = \mathrm{type}(v_2) = -1$. Using Proposition 7.15, in $\mathcal{O}(\log\log n)$ time we compute pointers to $u_1 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v_1)$ and $u_2 = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v_2)$. Using the representation of $\mathcal{T}_{\mathsf{Z}}$ stored as part of the structure in Section 7.3.1, and Proposition 4.1, in $\mathcal{O}(1)$ time we compute a pointer to $u = \mathrm{LCA}(u_1, u_2)$. By Lemma 7.16, it holds $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v) = u$. Our goal is to exploit this connection to compute $\mathrm{repr}(v)$. In $\mathcal{O}(1)$ time we compute $k := \text{L-exp}(v) = \lfloor \frac{e_v - 1 - s}{|H|} \rfloor$ and $\ell := e^{\mathrm{full}}(v) - 1 = s + k|H|$. Using Proposition 7.16, in $\mathcal{O}(\log\log n)$ time we then compute the pair $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u)$. As noted above, it holds $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v) = u$. Thus, by Lemma 7.14, we have $\mathrm{repr}(v) = (b, e)$.
- Let $T[j_{\min} + e_v - 1] \neq T[j_{\max} + e_v - 1]$, i.e., $e(v) > |\mathrm{str}(v)|$. Letting $P = \mathrm{str}(v)$, we then have $e(P) > |P|$, $\text{L-head}(P) = s$, $\text{L-root}(P) = H$, and $|P| = e_v - 1$. Using Proposition 7.18, we thus compute $(b, e) = (\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T))$ in $\mathcal{O}(\log\log n)$ time, and return $\mathrm{repr}(v) = (b, e)$. □

7.3.4 Implementation of $\mathrm{child}(v, c)$

LEMMA 7.18. *Let $c \in [0\,..\,\sigma)$ and v be an explicit periodic internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $e(v) \leq |\mathrm{str}(v)|$ and $\mathrm{type}(v) = -1$. Let $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v)$. If $\mathrm{child}(u, c) = \perp$ then $\mathrm{child}(v, c) = \perp$. Otherwise, letting $u' = \mathrm{child}(u, c)$, it holds*

$$\mathrm{repr}(\mathrm{child}(v, c)) = \begin{cases} (b, e) & \text{if } b \neq e, \\ (0, 0) & \text{otherwise,} \end{cases}$$

where $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$ and $\ell = e^{\mathrm{full}}(v) - 1$.

Proof. Let $H = \text{L-root}(v)$, $s = \text{L-head}(v)$, $k = \text{L-exp}(v)$, and $\mathsf{P} = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\}$. We first show that $\mathsf{P} \neq \emptyset$. Consider any $j \in \mathrm{Occ}(\mathrm{str}(v), T)$. By Lemma 6.2, $j \in \mathsf{R}$, $\text{L-root}(j) = \text{L-root}(v) = H$, and $\text{L-head}(j) = \text{L-head}(v) = s$, i.e., $j \in \mathsf{R}_{s,H}$. Furthermore, by $e(v) \leq |\mathrm{str}(v)|$ and $\mathrm{type}(v) = -1$ we obtain from

Lemma 6.4(2) that L-exp(j) = L-exp$(v) = k$ and type(j) = type$(v) = -1$. Thus, $j \in \mathsf{P}$, and hence $\mathsf{P} \neq \emptyset$. Let $b_\mathsf{P}, e_\mathsf{P} \in [0\,..\,n]$ be such that $\{\mathrm{SA}[i]\}_{i\in(b_\mathsf{P}\,..\,e_\mathsf{P}]} = \mathsf{P}$, and $b_H, e_H \in [0\,..\,q]$ be such that $\{r_i^{\mathrm{lex}}\}_{i\in(b_H\,..\,e_H]} = \mathsf{R}'^{-}_H$.

Denote $P = \mathrm{str}(v)c$ and $P' = \mathrm{pow}(H) \cdot P[e^{\mathrm{full}}(P)\,..\,|P|]$. Using the above notation, we now establish the characterization of $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$ with the help of Lemma 7.13. First, we observe that since $\mathrm{str}(v)$ is periodic, it follows by Lemma 6.3 that P is periodic and it holds L-root(P) = L-root$(v) = H$ and L-head(P) = L-head$(v) = s$. Moreover, since $e(v) \leq |\mathrm{str}(v)|$ and type$(v) = -1$, it follows by Lemma 6.4(1), that $e(P) = e(v)$, $e^{\mathrm{full}}(P) - 1 = e^{\mathrm{full}}(v) - 1 = \ell$, L-exp$(P)$ = L-exp$(v) = k$, and type(P) = type$(v) = -1$. In particular, this implies that the assumptions of Lemma 7.13 as satisfied. More precisely, $e(P) = e(v) \leq |\mathrm{str}(v)| \leq |P|$. On the other hand, as shown above, $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\} \neq \emptyset$. Observe also that by $e^{\mathrm{full}}(P) - 1 = \ell$, we have $P' = \mathrm{pow}(H) \cdot P(\ell\,..\,|P|]$. Putting all this together, by Lemma 7.13 we obtain that $(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) = (e_\mathsf{P} - c_1, e_\mathsf{P} - c_2)$, where $c_1 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, b_{\mathrm{pre}})$, $c_2 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_{\mathrm{pre}})$, $b_{\mathrm{pre}} = |\{i \in [1\,..\,q] : T[A_\mathsf{Z}[i]\,..\,n] \prec P'\}|$, and $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,q] : P'$ is a prefix of $T[A_\mathsf{Z}[i]\,..\,n]\}$.

We are now ready to show the first claim. Recall, that by definition of $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{Z}}(v)$, we have $\mathrm{str}(u) = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. Thus, it holds $\mathrm{str}(u)c = P'$. By definition of $\mathcal{T}_\mathsf{Z}$ and $\mathrm{child}(u,c)$, we thus obtain that $\mathrm{child}(u,c) = \bot$ implies $e_{\mathrm{pre}} - b_{\mathrm{pre}} = 0$. Consequently, by the above characterization, it holds

$$\begin{aligned}|\mathrm{Occ}(P,T)| &= \mathrm{RangeEnd}(P,T) - \mathrm{RangeBeg}(P,T)\\ &= (e_\mathsf{P} - c_2) - (e_\mathsf{P} - c_1)\\ &= \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_{\mathrm{pre}}) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, b_{\mathrm{pre}})\\ &= 0.\end{aligned}$$

Thus, $\mathrm{child}(v,c) = \bot$.

Let us now assume $\mathrm{child}(u,c) = u' \neq \bot$. Using the above notation, we first show the characterization of $\mathrm{pseudoinv}_{\mathcal{T}_\mathsf{Z}}(\ell, u')$. First, note that $\mathrm{pow}(H)$ is a prefix of $\mathrm{str}(u')$ (since it is a prefix of $\mathrm{str}(u)$). Next, note that $\ell \bmod |H| = (e^{\mathrm{full}}(v) - 1) \bmod |H| =$ L-head$(v) = s$ and $\lfloor \frac{\ell}{|H|} \rfloor = \lfloor \frac{e^{\mathrm{full}}(v)-1}{|H|} \rfloor =$ L-exp$(v) = k$. As shown above, the set $\{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\}$ is nonempty. This implies that $\mathrm{pseudoinv}_{\mathcal{T}_\mathsf{Z}}(\ell, u) = (e_\mathsf{P} - c'_1, e_\mathsf{P} - c'_2)$, where $c'_1 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(u'))$ and $c'_2 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(u'))$. It remains to observe that by definition of $\mathcal{T}_\mathsf{Z}$ and the facts that $\mathrm{child}(u,c) = u'$ and $\mathrm{str}(u)c = P'$, we have $\mathrm{lrank}(u') = b_{\mathrm{pre}}$ and $\mathrm{rrank}(u') = e_{\mathrm{pre}}$. Thus, we have $c'_1 = c_1$ and $c'_2 = c_2$, and consequently

$$\begin{aligned}(\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T)) &= (e_\mathsf{P} - c_1, e_\mathsf{P} - c_2)\\ &= (e_\mathsf{P} - c'_1, e_\mathsf{P} - c'_2)\\ &= \mathrm{pseudoinv}_{\mathcal{T}_\mathsf{Z}}(\ell, u')\\ &= (b, e).\end{aligned}$$

By the above, if $b \neq e$, then $\mathrm{Occ}(P,T) \neq \emptyset$. This implies $\mathrm{child}(v,c) \neq \bot$ and $\mathrm{repr}(\mathrm{child}(v,c)) = (\mathrm{RangeBeg}(P,T), \mathrm{RangeEnd}(P,T))$. We thus indeed have $\mathrm{repr}(\mathrm{child}(v,c)) = (b,e)$. Otherwise (i.e., if $b = e$), by the above we have $\mathrm{Occ}(P,T) = \emptyset$. This implies $\mathrm{child}(v,c) = \bot$ and hence indeed we also have $\mathrm{repr}(\mathrm{child}(v,c)) = (0,0)$. □

Remark 7.10. Note that, similarly as in Lemma 7.9 (see Remark 7.3), even though in the above result we have $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{Z}}(v) = u$ and $\mathrm{child}(u,c)$ contains information used to determine $\mathrm{child}(v,c)$, it does not necessarily hold that $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_\mathsf{Z}}(\mathrm{child}(v,c)) = \mathrm{child}(u,c)$.

Proposition 7.20. *Let v be an explicit periodic internal node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.3.1,* $\mathrm{repr}(v)$*, and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log\log n)$ time we can compute* $\mathrm{repr}(\mathrm{child}(v,c))$.

Proof. Denote $i_1 = \mathrm{lrank}(v) + 1$ and $i_2 = \mathrm{rrank}(v)$. By Lemma 7.15(1), it holds $\mathrm{SA}[i_1], \mathrm{SA}[i_2] \in \mathsf{R}$. Using Proposition 5.14, in $\mathcal{O}(\log\log n)$ time we compute $j_1 = \mathrm{SA}[i_1]$ and $j_2 = \mathrm{SA}[i_2]$. Next, using Proposition 5.7 in $\mathcal{O}(1)$ time we compute $H =$ L-root(j_1), $s =$ L-head(j_1), $k_1 =$ L-exp(j_1), $k_2 =$ L-exp(j_2), $t_1 =$ L-tail(j_1), and $t_2 =$ L-tail(j_2). Observe that since v is periodic, and $j_1, j_2 \in \mathrm{Occ}(\mathrm{str}(v), T)$, it follows by Lemmas 5.4

and 6.2 that L-root(v) = L-root$(j_2) = H$ and L-head(v) = L-head$(j_2) = s$. In $\mathcal{O}(1)$ time we thus compute $e_1 := e(j_1) = j_1 + s + k_1|H| + t_1$ and $e_2 := e(j_2) = j_2 + s + k_2|H| + t_2$. Next, in $\mathcal{O}(1)$ time we compute $e_v := e(v) = 1 + \min(e_1 - j_1, e_2 - j_2)$ (see Lemma 7.15(1)). Using Lemma 7.15(2), we then in $\mathcal{O}(1)$ time check if it holds $e(v) \le |\mathrm{str}(v)|$ by comparing $T[j_1 + e_v - 1]$ with $T[j_2 + e_v - 1]$. Consider two cases:

- Let $T[j_1 + e_v - 1] = T[j_2 + e_v - 1]$, i.e., $e(v) \le |\mathrm{str}(v)|$. In $\mathcal{O}(1)$ time we compute type(v) by comparing $T[j_1 + e_v - 1]$ with $T[j_1 + e_v - 1 - |H|]$. Let us assume that $T[j_1 + e_v - 1] \prec T[j_1 + e_v - 1 - |H|]$, i.e., type$(v) = -1$ (the case type$(v) = +1$ it handled symmetrically, using the part of the structure from Section 7.3.1 adapted according to Lemma 5.4). Using Proposition 7.15, in $\mathcal{O}(\log\log n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$. Using the representation of $\mathcal{T}_{\mathsf{Z}}$ stored as part of the structure in Section 7.3.1, and Proposition 4.1, in $\mathcal{O}(\log\log n)$ time we check if child$(u, c) = \bot$. If so, by Lemma 7.18 we have child$(v, c) = \bot$, and thus we return repr(child$(v, c)) = (0, 0)$. Otherwise (child$(u, c) \neq \bot$), we obtain a pointer to $u' = \mathrm{child}(u, c)$. In $\mathcal{O}(1)$ time we now compute $k := \text{L-exp}(v) = \lfloor \frac{e_v - 1 - s}{|H|} \rfloor$ and $\ell := e^{\mathrm{full}}(v) - 1 = s + k|H|$. Using Proposition 7.16, in $\mathcal{O}(\log\log n)$ time we then compute the pair $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$. If $b = e$ then by Lemma 7.18 it holds child$(v, c) = \bot$ and hence we return repr(child$(v, c)) = (0, 0)$. Otherwise, by Lemma 7.18, it holds repr(child$(v, c)) = (b, e)$. We thus return (b, e).
- Let $T[j_1 + e_v - 1] \neq T[j_2 + e_v - 1]$, i.e., $e(v) > |\mathrm{str}(v)|$. Denote $P = \mathrm{str}(v)$. We then have $e(P) > |P|$, L-head$(P) = s$, L-root$(P) = H$, and $|P| = e_v - 1$. Using Proposition 7.18, in $\mathcal{O}(\log\log n)$ time we compute $(b, e) = (\mathrm{RangeBeg}(Pc, T), \mathrm{RangeEnd}(Pc, T))$. If $b = e$, then $\mathrm{Occ}(P, T) = \mathrm{Occ}(\mathrm{str}(v)c, T) = \emptyset$, and hence child$(v, c) = \bot$. We thus return repr(child$(v, c)) = (0, 0)$. Otherwise, we return that repr(child$(v, c)) = (b, e)$.

□

7.3.5 Implementation of pred(v, c)

Lemma 7.19. *Let $c \in [0\,..\,\sigma)$ and v be an explicit periodic internal node of $\mathcal{T}_{\mathrm{st}}$ satisfying $e(v) \le |\mathrm{str}(v)|$ and* type$(v) = -1$. *Let $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$. If* pred$(u, c) = \bot$ *then* RangeBeg$(\mathrm{str}(v)c, T) = \mathrm{RangeBeg}(\mathrm{str}(v), T)$. *Otherwise, letting $u' = \mathrm{pred}(u, c)$, it holds*

$$\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e,$$

where $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$ and $\ell = e^{\mathrm{full}}(v) - 1$.

Proof. We start by characterizing RangeBeg$(\mathrm{str}(v), T)$. Let $H = \text{L-root}(v)$, $s = \text{L-head}(v)$, $k = \text{L-exp}(v)$, and $\mathsf{P} = \{j \in \mathsf{R}^-_{s,H} : \text{L-exp}(j) = k\}$. In the proof of Lemma 7.18, we showed that $\mathsf{P} \neq \emptyset$. Let $b_{\mathsf{P}}, e_{\mathsf{P}} \in [0\,..\,n]$ be such that $\{\mathrm{SA}[i]\}_{i \in (b_{\mathsf{P}}\,..\,e_{\mathsf{P}}]} = \mathsf{P}$, and $b_H, e_H \in [0\,..\,q]$ be such that $\{r_i^{\mathrm{lex}}\}_{i \in (b_H\,..\,e_H]} = \mathsf{R}'^-_H$. We now additionally note that by definition of $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$, we have $\mathrm{str}(u) = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. Thus, by definition of $\mathcal{T}_{\mathsf{Z}}$, it holds $|\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]\}| = \mathrm{lrank}(u)$. By Lemma 7.13 for pattern str(v), we thus obtain $\mathrm{RangeBeg}(\mathrm{str}(v), T) = e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(u)))$.

Next, we characterize RangeBeg$(\mathrm{str}(v)c, T)$. Denote $P = \mathrm{str}(v)c$ and $P' = \mathrm{pow}(H) \cdot P[e^{\mathrm{full}}(P)\,..\,|P|]$. In the proof of Lemma 7.18, we observed that P is periodic and it holds L-root(P) = L-root(v) = H, L-head(P) = L-head(v) = s, $e(P) = e(v)$, $e^{\mathrm{full}}(P) - 1 = e^{\mathrm{full}}(v) - 1 = \ell$, L-exp$(P)$ = L-exp(v) = k, and type(P) = type(v) = -1. Moreover, we noted that $e(P) \le |P|$ and $P' = \mathrm{pow}(H) \cdot P(\ell\,..\,|P|]$. Finally, putting all this together, we observed that $(\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T)) = (e_{\mathsf{P}} - c_1, e_{\mathsf{P}} - c_2)$, where $c_1 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, b_{\mathrm{pre}})$, $c_2 = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_{\mathrm{pre}})$, $b_{\mathrm{pre}} = |\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec P'\}|$, and $(b_{\mathrm{pre}}\,..\,e_{\mathrm{pre}}] = \{i \in [1\,..\,q] : P' \text{ is a prefix of } T[A_{\mathsf{Z}}[i]\,..\,n]\}$.

Let us first assume pred$(u, c) = \bot$. By definition, this implies $|\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec \mathrm{str}(u)c\}| = \mathrm{lrank}(u)$. Recall, however, that $\mathrm{str}(u) = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. Thus, $\mathrm{str}(u)c = P'$ and consequently $b_{\mathrm{pre}} = \mathrm{lrank}(u)$. Using the above characterization, we thus have $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(u)))$. Since above we also established that $\mathrm{RangeBeg}(\mathrm{str}(v), T) = e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(u)))$, we have thus proved that pred$(u, c) = \bot$ implies $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = \mathrm{RangeBeg}(\mathrm{str}(v), T)$.

Let us now assume pred$(u, c) = u' \neq \bot$. By definition of pred(u, c), this implies $|\{i \in [1\,..\,q] : T[A_{\mathsf{Z}}[i]\,..\,n] \prec \mathrm{str}(u)c\}| = \mathrm{rrank}(u')$. By recalling again that $\mathrm{str}(u)c = P'$, we thus have $b_{\mathrm{pre}} = \mathrm{rrank}(u')$. By the above characterization, we thus have $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(u')))$. On the other hand, by definition of $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$, we have $e = e_{\mathsf{P}} - (\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(u')))$. We thus obtain $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e$. □

PROPOSITION 7.21. *Let v be an explicit periodic internal node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.3.1, $\mathrm{repr}(v)$, and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log\log n)$ time we can compute* $\mathrm{RangeBeg}(\mathrm{str}(v)c, T)$.

Proof. Denote $i_1 = \mathrm{lrank}(v) + 1$ and $i_2 = \mathrm{rrank}(v)$. By Lemma 7.15(1), it holds $\mathrm{SA}[i_1], \mathrm{SA}[i_2] \in \mathsf{R}$. Using Proposition 5.14, in $\mathcal{O}(\log\log n)$ time we compute $j_1 = \mathrm{SA}[i_1]$ and $j_2 = \mathrm{SA}[i_2]$. Next, using Proposition 5.7 in $\mathcal{O}(1)$ time we compute $H = \text{L-root}(j_1)$, $s = \text{L-head}(j_1)$, $k_1 = \text{L-exp}(j_1)$, $k_2 = \text{L-exp}(j_2)$, $t_1 = \text{L-tail}(j_1)$, and $t_2 = \text{L-tail}(j_2)$. Observe that since v is periodic, and $j_1, j_2 \in \mathrm{Occ}(\mathrm{str}(v), T)$, it follows by Lemmas 5.4 and 6.2 that $\text{L-root}(v) = \text{L-root}(j_2) = H$ and $\text{L-head}(v) = \text{L-head}(j_2) = s$. In $\mathcal{O}(1)$ time we thus compute $e_1 := e(j_1) = j_1 + s + k_1|H| + t_1$ and $e_2 := e(j_2) = j_2 + s + k_2|H| + t_2$. Next, in $\mathcal{O}(1)$ time we compute $e_v := e(v) = 1 + \min(e_1 - j_1, e_2 - j_2)$ (see Lemma 7.15(1)). Using Lemma 7.15(2), we then in $\mathcal{O}(1)$ time check if it holds $e(v) \le |\mathrm{str}(v)|$ by comparing $T[j_1 + e_v - 1]$ with $T[j_2 + e_v - 1]$. Consider two cases:

- Let $T[j_1 + e_v - 1] = T[j_2 + e_v - 1]$, i.e., $e(v) \le |\mathrm{str}(v)|$. In $\mathcal{O}(1)$ time we compute $\mathrm{type}(v)$ by comparing $T[j_1 + e_v - 1]$ with $T[j_1 + e_v - 1 - |H|]$. Let us assume that $T[j_1 + e_v - 1] \prec T[j_1 + e_v - 1 - |H|]$, i.e., $\mathrm{type}(v) = -1$ (the case $\mathrm{type}(v) = +1$ it handled symmetrically, using the part of the structure from Section 7.3.1 adapted according to Lemma 5.4). Using Proposition 7.15, in $\mathcal{O}(\log\log n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$. Using the representation of $\mathcal{T}_{\mathsf{Z}}$ stored as part of the structure in Section 7.3.1, and Proposition 4.1, in $\mathcal{O}(\log\log n)$ time we check if $\mathrm{pred}(u, c) = \bot$. If so, by Lemma 7.19 we have $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = \mathrm{RangeBeg}(\mathrm{str}(v), T)$, and thus we return $\mathrm{rrank}(v)$ as the answer. Otherwise ($\mathrm{pred}(u, c) \neq \bot$), we obtain a pointer to $u' = \mathrm{pred}(u, c)$. In $\mathcal{O}(1)$ time we now compute $k := \text{L-exp}(v) = \lfloor \frac{e_v - 1 - s}{|H|} \rfloor$ and $\ell := e^{\mathrm{full}}(v) - 1 = s + k|H|$. Using Proposition 7.16, in $\mathcal{O}(\log\log n)$ time we then compute the pair $(b, e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$. By Lemma 7.19, we then have $\mathrm{RangeBeg}(\mathrm{str}(v)c, T) = e$. Thus, we return e as the answer.
- Let $T[j_1 + e_v - 1] \neq T[j_2 + e_v - 1]$, i.e., $e(v) > |\mathrm{str}(v)|$. Denote $P = \mathrm{str}(v)$. We then have $e(P) > |P|$, $\text{L-head}(P) = s$, $\text{L-root}(P) = H$, and $|P| = e_v - 1$. Using Proposition 7.18, in $\mathcal{O}(\log\log n)$ time we compute $(b, e) = (\mathrm{RangeBeg}(Pc, T), \mathrm{RangeEnd}(Pc, T))$. We then return b as the answer. ☐

7.3.6 Implementation of $\mathrm{WA}(v, d)$

LEMMA 7.20. *Let v be an explicit periodic node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{type}(v) = -1$ and d be an integer satisfying $e(v) \le d \le |\mathrm{str}(v)|$. Then, letting $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$, it holds*

$$\mathrm{repr}(\mathrm{WA}(v, d)) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, \widehat{u}),$$

where $\ell = e^{\mathrm{full}}(v) - 1$, $H = \text{L-root}(v)$, and $\widehat{u} = \mathrm{WA}(u, d - \ell + |\mathrm{pow}(H)|)$.

Proof. As in the proof of Lemma 7.11, let us denote $f^{(0)}(x) = x$ and $f^{(i)}(x) = f(f^{(i-1)}(x))$ for $i \in \mathbb{Z}_+$. Let

$$\mathcal{V} := \{\mathrm{parent}^{(i)}(v) : i \in \mathbb{Z}_{\ge 0} \text{ and } \mathrm{sdepth}(\mathrm{parent}^{(i)}(v)) \ge e(v)\} \text{ and}$$
$$\mathcal{U} := \{\mathrm{parent}^{(i)}(u) : i \in \mathbb{Z}_{\ge 0} \text{ and } \mathrm{sdepth}(\mathrm{parent}^{(i)}(u)) \ge |\mathrm{pow}(H)| + \text{L-tail}(v) + 1\}$$

By $e(v) \le |\mathrm{str}(v)|$, $\mathrm{type}(v) = -1$, and Lemma 6.4(1), for every $v' \in \mathcal{V}$ it holds that $\mathrm{str}(v')$ is periodic, and we have $e(v') = e(v) \le |\mathrm{str}(v')|$, $\mathrm{type}(v') = \mathrm{type}(v) = -1$, and $e^{\mathrm{full}}(v') = e^{\mathrm{full}}(v) = \ell + 1$. Thus, for every $v' \in \mathcal{V}$, the node $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v')$ is well-defined and satisfies $\mathrm{str}(u') = \mathrm{pow}(H) \cdot \mathrm{str}(v')[e^{\mathrm{full}}(v')\,..\,|\mathrm{str}(v')|] = \mathrm{pow}(H) \cdot \mathrm{str}(v')(\ell\,..\,|\mathrm{str}(v')|]$. In particular, $\mathrm{str}(u) = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. Since for any $v' \in \mathcal{V}$, $\mathrm{str}(v') = \mathrm{str}(v)[1\,..\,|\mathrm{str}(v')|]$, we thus obtain that for $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v')$ it holds $\mathrm{str}(u') = \mathrm{pow}(H) \cdot \mathrm{str}(v')(\ell\,..\,|\mathrm{str}(v')|] = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v')|] = \mathrm{str}(u)[1\,..\,|\mathrm{str}(u')|]$. i.e., u' is an ancestor of u. Moreover, $\mathrm{sdepth}(u') = |\mathrm{pow}(H)| + |\mathrm{str}(v')| - e^{\mathrm{full}}(v') + 1 = |\mathrm{pow}(H)| + |\mathrm{str}(v')| - e^{\mathrm{full}}(v) + 1 \ge |\mathrm{pow}(H)| + e(v) - e^{\mathrm{full}}(v) + 1 = |\mathrm{pow}(H)| + \text{L-tail}(v) + 1$. Consequently, $\mathcal{U}' := \{\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v') : v' \in \mathcal{V}\}$ satisfies $\mathcal{U}' \subseteq \mathcal{U}$. Note also, that $v' \neq v''$ implies $\mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v') \neq \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v'')$.

For any $u' \in \mathcal{U}$, denote $(s(u'), t(u')) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$. We prove the following property of $\mathcal{U}'$. Let $w, w' \in \mathcal{U}$ be such that $w = \mathrm{parent}(w')$. We claim, that $(s(w), t(w)) \neq (s(w'), t(w'))$ implies $w \in \mathcal{U}'$. The proof consists of five steps:

1. For any node y of $\mathcal{T}_{\mathsf{Z}}$ such that $\mathrm{pow}(H)$ is a prefix of $\mathrm{str}(y)$, by S_y we denote a string such that $\mathrm{str}(y) = \mathrm{pow}(H) \cdot S_y$. Let P and P' be such that PP' is a prefix of $\mathrm{str}(v)$, and it holds $|P| = \ell$ and $|P'| = \text{L-tail}(v) + 1$ (which is defined by $e(v) \leq |\mathrm{str}(v)|$). Consider any node y of $\mathcal{T}_{\mathsf{Z}}$ such that $\mathrm{pow}(H) \cdot P'$ is a prefix of $\mathrm{str}(y)$ (note that although this includes all nodes in $\mathcal{U}$, it is possible that $y \notin \mathcal{U}$). We prove that for any such y, it holds $|\mathrm{Occ}(PS_y, T)| = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(y)) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(y))$. First, observe that since $e(\mathrm{str}(v)) = |PP'|$, we obtain by $\mathrm{lcp}(PS_y, \mathrm{str}(v)) \geq |PP'|$ and Lemma 6.4(1), that PS_y is periodic, $e(PS_y) = e(\mathrm{str}(v)) = |PP'| \leq |PS_y|$, $e^{\mathrm{full}}(PS_y) = e^{\mathrm{full}}(\mathrm{str}(v)) = \ell + 1$, and $\mathrm{type}(PS_y) = \mathrm{type}(\mathrm{str}(v)) = -1$. By Lemma 6.5, $\mathrm{Occ}(PS_y, T)$ is thus a disjoint union of $\mathrm{Occ}^{\mathsf{a}}(PS_y, T)$ and $\mathrm{Occ}^{\mathsf{s}}(PS_y, T)$ (see the beginning of Section 6.3.4 for definitions). By $e(PS_y) \leq |PS_y|$, Lemma 6.6 and its symmetric version (adapted according to Lemma 6.2) moreover imply that $\mathrm{Occ}^{\mathsf{a}}(PS_y, T) = \emptyset$. Finally, by $\mathrm{type}(PS_y) = -1$ and Lemma 6.4(2), it follows that $\mathrm{Occ}(PS_y, T) \subseteq \mathsf{R}^{-}$. Thus, $\mathrm{Occ}^{\mathsf{s}}(PS_y, T) = \mathrm{Occ}^{\mathsf{s}-}(PS_y, T)$ and consequently $\mathrm{Occ}(PS_y, T) = \mathrm{Occ}^{\mathsf{s}-}(PS_y, T)$. It thus remains to prove $|\mathrm{Occ}^{\mathsf{s}-}(PS_y, T)| = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(y)) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(y))$. Recall that the set $\{\mathrm{pow}(H) : H \in \mathrm{Roots}\}$ is prefix-free. Letting $H_j = \text{L-root}(j)$ (where $j \in \mathsf{R}$), it follows by definition of $\mathcal{T}_{\mathsf{Z}}$ that:

$$\begin{aligned}\{r_i^{\mathrm{lex}}\}_{i \in (\mathrm{lrank}(y)\,..\,\mathrm{rrank}(y)]} &= \{j \in \mathsf{R}'^{-} : \mathrm{pow}(H) \cdot S_y \text{ is a prefix of } T[e^{\mathrm{full}}(j) - |\mathrm{pow}(H_j)|\,..\,n]\} \\ &= \{j \in \mathsf{R}'^{-} : \mathrm{pow}(H) \cdot S_y \text{ is a prefix of } \mathrm{pow}(H_j) \cdot T[e^{\mathrm{full}}(j)\,..\,n]\} \\ &= \{j \in \mathsf{R}'^{-}_H : S_y \text{ is a prefix of } T[e^{\mathrm{full}}(j)\,..\,n]\}.\end{aligned}$$

   Finally, note that by $e^{\mathrm{full}}(PS_y) = \ell + 1$, we have $(PS_y)[e^{\mathrm{full}}(PS_y)\,..\,|PS_y|] = S_y$. Thus, by the above and Lemma 6.7, $|\mathrm{Occ}^{\mathsf{s}-}(PS_y, T)| = |\{i \in (\mathrm{lrank}(y)\,..\,\mathrm{rrank}(y)] : A_{\mathrm{len}}[i] \geq e^{\mathrm{full}}(PS_y) - 1\}| = |\{i \in (\mathrm{lrank}(y)\,..\,\mathrm{rrank}(y)] : A_{\mathrm{len}}[i] \geq \ell\}| = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(y)) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(y))$.
2. We prove that there exists $c' \in [0\,..\,\sigma)$ such that $|\mathrm{Occ}(PS_w c', T)| > 0$ (where S_w is defined as above). First, note that by $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v)$, it holds $\mathrm{str}(u) = \mathrm{pow}(H) \cdot \mathrm{str}(v)[e^{\mathrm{full}}(v)\,..\,|\mathrm{str}(v)|] = \mathrm{pow}(H) \cdot \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. On the other hand, by definition of S_u, we have $\mathrm{str}(u) = \mathrm{pow}(H) \cdot S_u$. Thus, $S_u = \mathrm{str}(v)(\ell\,..\,|\mathrm{str}(v)|]$. By $P = \mathrm{str}(v)[1\,..\,\ell]$, we thus obtain $\mathrm{str}(v) = PS_u$. Consequently, since v is a node of $\mathcal{T}_{\mathrm{st}}$, we have $|\mathrm{Occ}(PS_u, T)| = |\mathrm{Occ}(\mathrm{str}(v), T)| > 0$. Observe now that since w' is an ancestor of u, the string $\mathrm{str}(w') = \mathrm{pow}(H) \cdot S_{w'}$ is a prefix of $\mathrm{str}(u) = \mathrm{pow}(H) \cdot S_u$. This implies that $S_{w'}$ is a prefix of S_u, and hence $PS_{w'}$ is a prefix of PS_u. Consequently, $|\mathrm{Occ}(PS_{w'}, T)| \geq |\mathrm{Occ}(PS_u, T)| > 0$. In particular, since PS_w is a prefix of $PS_{w'}$, letting $c' \in [0\,..\,\sigma)$ be such that $\mathrm{child}(w, c') = w'$, we have $|\mathrm{Occ}(PS_w c', T)| > 0$.
3. Let $s = \ell \bmod |H|$, $k = \lfloor \frac{\ell}{|H|} \rfloor$. Let also $\mathsf{P} := \{j \in \mathsf{R}^{-}_{s,H} : \text{L-exp}(j) = k\}$, $b_{\mathsf{P}}, e_{\mathsf{P}} \in [0\,..\,n]$ be such that $\{\mathrm{SA}[i]\}_{i \in (b_{\mathsf{P}}\,..\,e_{\mathsf{P}}]} = \mathsf{P}$, and $b_H, e_H \in [0\,..\,q]$ be such that $\{r_i^{\mathrm{lex}}\}_{i \in (b_H\,..\,e_H]} = \mathsf{R}'^{-}_H$. Note that $\text{L-head}(v) = s$, $\text{L-root}(v) = H$, $\text{L-exp}(v) = k$, $e(v) \leq |\mathrm{str}(v)|$, and $\mathrm{type}(v) = -1$ imply that $\mathsf{P} \neq \emptyset$ (it suffices to take $j = \mathrm{SA}[i]$ for any $i \in (\mathrm{lrank}(v)\,..\,\mathrm{rrank}(v)]$, and apply Lemma 6.2 and Lemma 6.4(2)). Therefore, $(b_{\mathsf{P}}, e_{\mathsf{P}})$ and (b_H, e_H) are well-defined. Denote $\delta = e_{\mathsf{P}} - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, e_H)$. By definition of $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, w)$ and $\mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, w')$, we then have $(s(w), t(w)) = (\delta + \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w)), \delta + \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w)))$ and $(s(w'), t(w')) = (\delta + \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w')), \delta + \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w')))$. Thus, the assumption $(s(w), t(w)) \neq (s(w'), t(w'))$, or equivalently, $s(w) \neq s(w')$ or $t(w) \neq t(w')$, implies

$$\begin{aligned}&\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w)) \neq \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w')) \text{ or} \\ &\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w)) \neq \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w')).\end{aligned}$$

4. By definition, the values $\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(\widehat{w})) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(\widehat{w}))$ over all children $\widehat{w}$ of w sum up to $\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w)) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w))$. Thus, it follows by Step 3 that there exists a child $w'' \neq w'$ of w such that $\mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w'')) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w'')) > 0$. By Step 1, for such w'', we thus have $|\mathrm{Occ}(PS_{w''}, T)| = \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{rrank}(w'')) - \mathsf{rcount}_{A_{\mathrm{len}}}(\ell, \mathrm{lrank}(w'')) > 0$. In particular, letting $c'' \in [0\,..\,\sigma)$ be such that $\mathrm{child}(w, c'') = w''$, it holds $|\mathrm{Occ}(PS_w c'', T)| > 0$. Note that $w'' \neq w'$ implies $c'' \neq c'$.
5. We have thus proved (Steps 2 and 4) that there exist $c', c'' \in [0\,..\,\sigma)$ such that $c' \neq c''$, $|\mathrm{Occ}(PS_w c', T)| > 0$, and $|\mathrm{Occ}(PS_w c'', T)| > 0$. This implies that there exists a node v' in $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{str}(v') = PS_w$. As observed in Step 1, PS_w is periodic, and it holds $e(PS_w) \leq |PS_w|$, $\mathrm{type}(PS_w) = -1$, and $e^{\mathrm{full}}(PS_w) = |P| + 1$. Thus, the node $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v')$ is defined and satisfies $\mathrm{str}(u') = \mathrm{pow}(H) \cdot S_w$. This implies $u' = w$, and consequently, $w \in \mathcal{U}'$.

We are now ready to prove the main claim. Let $v' = \mathrm{WA}(v,d)$ and $v'' = \mathrm{parent}(v')$. We then have $\mathrm{sdepth}(v'') < d \leq \mathrm{sdepth}(v')$. Moreover, by $e(v) \leq d$, we have $v' \in \mathcal{V}$. Let $u' = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v')$. As observed earlier, we then have $u' \in \mathcal{U}'$, and hence $d - \ell + |\mathrm{pow}(H)| \leq \mathrm{sdepth}(u')$. This implies that $\widehat{u} = \mathrm{WA}(u', d - \ell + |\mathrm{pow}(H)|)$ satisfies $d - \ell + |\mathrm{pow}(H)| \leq \mathrm{sdepth}(\widehat{u}) \leq \mathrm{sdepth}(u')$. By $d - \ell + |\mathrm{pow}(H)| \geq e(v) - \ell + |\mathrm{pow}(H)| = e(v) - (e^{\mathrm{full}}(v) - 1) + |\mathrm{pow}(H)| = \text{L-tail}(v) + 1 + |\mathrm{pow}(H)|$, this implies that $\widehat{u} \in \mathcal{U}$. Let $k \in \mathbb{Z}_{\geq 0}$ be such that $\widehat{u} = \mathrm{parent}^{(k)}(u')$. This implies that $\mathrm{parent}^{(i)}(u') \notin \mathcal{U}'$ holds for $i \in [1\,..\,k]$, since otherwise it would contradict $v' = \mathrm{WA}(v,d)$. If $k = 0$ then we trivially have $(s(u'), t(u')) = (s(\widehat{u}), t(\widehat{u}))$. Otherwise, by (the contraposition of) the above property of $\mathcal{U}'$ we have

$$\begin{aligned}(s(u'), t(u')) &= (s(\mathrm{parent}(u')), t(\mathrm{parent}(u')))\\ &= \dots\\ &= (s(\mathrm{parent}^{(k)}(u')), t(\mathrm{parent}^{(k)}(u')))\\ &= (s(\widehat{u}), t(\widehat{u})).\end{aligned}$$

Recall now that $e(v') \leq |\mathrm{str}(v')|$, $\mathrm{type}(v') = -1$, and $e^{\mathrm{full}}(v') = e^{\mathrm{full}}(v) = \ell + 1$. Thus, by Lemma 7.14, we have $\mathrm{repr}(v') = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(e^{\mathrm{full}}(v') - 1, u') = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u')$. Consequently, $\mathrm{repr}(\mathrm{WA}(v,d)) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, u') = (s(u'), t(u')) = (s(\widehat{u}), t(\widehat{u})) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, \widehat{u})$. □

Proposition 7.22. *Let v be an explicit periodic node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.3.1, $\mathrm{repr}(v)$, and an integer d satisfying $3\tau - 1 \leq d \leq |\mathrm{str}(v)|$, in $\mathcal{O}(\log\log n)$ time we can compute $\mathrm{repr}(\mathrm{WA}(v,d))$.*

Proof. Denote $i_1 = \mathrm{lrank}(v) + 1$ and $i_2 = \mathrm{rrank}(v)$. By Lemma 7.15(1), it holds $\mathrm{SA}[i_1], \mathrm{SA}[i_2] \in \mathsf{R}$. Using Proposition 5.14, in $\mathcal{O}(\log\log n)$ time we first compute $j_1 = \mathrm{SA}[i_1]$ and $j_2 = \mathrm{SA}[i_2]$. Next, using Proposition 5.7, in $\mathcal{O}(1)$ time we compute $H = \text{L-root}(j_1)$, $s = \text{L-head}(j_1)$, $k_1 = \text{L-exp}(j_1)$, $k_2 = \text{L-exp}(j_2)$, $t_1 = \text{L-tail}(j_1)$, and $t_2 = \text{L-tail}(j_2)$. Since v is periodic, and $j_1, j_2 \in \mathrm{Occ}(\mathrm{str}(v), T)$, it follows by Lemmas 5.4 and 6.2 that $\text{L-root}(v) = \text{L-root}(j_2) = H$ and $\text{L-head}(v) = \text{L-head}(j_2) = s$. In $\mathcal{O}(1)$ time we thus compute $e_1 := e(j_1) = j_1 + s + k_1|H| + t_1$ and $e_2 := e(j_2) = j_2 + s + k_2|H| + t_2$. Next, in $\mathcal{O}(1)$ time we compute $e_v := e(v) = 1 + \min(e_1 - j_1, e_2 - j_2)$ (see Lemma 7.15(1)). We then consider two cases:

- Assume $e_v \leq d$. Then, to obtain $\mathrm{repr}(\mathrm{WA}(v,d))$ we follow Lemma 7.20. First, in $\mathcal{O}(1)$ time we compute $\mathrm{type}(v)$ by comparing $T[j_1 + e_v - 1]$ with $T[j_1 + e_v - 1 - |H|]$. Let us assume that $T[j_1 + e_v - 1] \prec T[j_1 + e_v - 1 - |H|]$, i.e., $\mathrm{type}(v) = -1$ (the case $\mathrm{type}(v) = +1$ it handled symmetrically, using the part of the structure from Section 7.3.1 adapted according to Lemma 5.4). Using Proposition 7.15, in $\mathcal{O}(\log\log n)$ time we compute a pointer to $u = \mathrm{map}_{\mathcal{T}_{\mathrm{st}},\mathcal{T}_{\mathsf{Z}}}(v)$. In $\mathcal{O}(1)$ time we also calculate $|\mathrm{pow}(H)| = |H|\lceil \frac{\tau}{|H|} \rceil$. Using the representation of $\mathcal{T}_{\mathsf{Z}}$ stored as part of the structure in Section 7.3.1, and Proposition 4.1, in $\mathcal{O}(\log\log n)$ time we compute a pointer to $\widehat{u} = \mathrm{WA}(u, d - \ell + |\mathrm{pow}(H)|)$. In $\mathcal{O}(1)$ time we now compute $k := \text{L-exp}(v) = \lfloor \frac{e_v - 1 - s}{|H|} \rfloor$ and $\ell := e^{\mathrm{full}}(v) - 1 = s + k|H|$. Using Proposition 7.16, in $\mathcal{O}(\log\log n)$ time we then compute the pair $(b,e) = \mathrm{pseudoinv}_{\mathcal{T}_{\mathsf{Z}}}(\ell, \widehat{u})$. By Lemma 7.20, it holds $\mathrm{repr}(\mathrm{WA}(v,d)) = (b,e)$. We thus return (b,e).
- Assume $e_v > d$. Let $v' = \mathrm{WA}(v,d)$, $S = \mathrm{str}(v')$, and $S' = S[1\,..\,d]$. Since, by definition, v' does not have an ancestor v'' in $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v'') \geq d$, it holds $\mathrm{repr}(v') = (\mathrm{RangeBeg}(S,T), \mathrm{RangeEnd}(S,T)) = (\mathrm{RangeBeg}(S',T), \mathrm{RangeEnd}(S',T))$. We thus focus on computing the latter pair. First, we observe that since v' is an ancestor v, we have $S' = \mathrm{str}(v)[1\,..\,d]$. Therefore, since $\mathrm{str}(v)$ is periodic, and it holds $3\tau - 1 \leq d$, we obtain by Lemma 6.3 that S' is periodic, and it holds $\text{L-root}(S') = \text{L-root}(v) = H$ and $\text{L-head}(S') = \text{L-head}(v) = s$. To show $e(S') > |S'|$, let us denote $Q = \mathrm{str}(v)[1\,..\,e(v))$. By definition, we have $e(Q) = 1 + p + \mathrm{lcp}(Q, Q(p\,..\,|Q|]) = |Q| + 1$. Thus, we must have $\mathrm{lcp}(Q, Q(p\,..\,|Q|]) = |Q| - p$. Consequently, since by $e_v = e(v) > d$ the string S' is a prefix of Q, we have $\mathrm{lcp}(S', S'(p\,..\,|S'|]) = |S'| - p$, and hence $e(S') = 1 + p + \mathrm{lcp}(S', S'(p\,..\,|S'|]) = |S'| + 1$. Considering all the above properties of S', the next step of the algorithm is therefore to compute and return the pair $(b,e) = (\mathrm{RangeBeg}(S',T), \mathrm{RangeEnd}(S',T))$ in $\mathcal{O}(\log\log n)$ time using Proposition 7.17. As observed above, it holds $\mathrm{repr}(\mathrm{WA}(v,d)) = (b,e)$. □

7.3.7 Construction Algorithm

Proposition 7.23. *Given $\mathrm{C}_{\mathrm{ST}}(T)$, we can in $\mathcal{O}(n/\log_\sigma n)$ time we can augment it into a data structure from Section 7.3.1.*

Proof. First, we combine Propositions 5.3 and 5.15 (recall that the packed representation of T is a component of $\mathrm{C}_{\mathrm{ST}}(T)$) to construct the data structure from Section 5.3.2 in $\mathcal{O}(n/\log_\sigma n)$ time. In particular, this constructs $(r_i^{\mathrm{lex}})_{i\in[1\,..\,q]}$. Using Proposition 5.7, we can now compute $A_{\mathsf{Z}}[i]$ for any $i \in [1\,..\,q]$ in $\mathcal{O}(1)$ time. Then, in $\mathcal{O}(n/\log_\sigma n)$ time we construct the data structure $\mathcal{T}_{\mathsf{Z}}$ using Proposition 4.1.

After the above components are constructed, we then analogously construct their symmetric counterparts (adapted according to Lemma 5.4). □

7.4 The Final Data Structure

In this section, we put together Sections 7.1 to 7.3 to obtain a data structure that performs suffix tree operations in $\mathcal{O}(\log^\epsilon n)$ time.

The section is organized as follows. First, we introduce the components of the data structure (Section 7.4.1). We then describe the query algorithms for all operations in Table 1 (Sections 7.4.2 to 7.4.20). Finally, we show the construction algorithm (Section 7.4.21).

7.4.1 The Data Structure

Definitions Recall (Section 2), that we assumed $T[n] = 0$, and that 0 that not appear anywhere else in T. We define T^{rev} as a text obtained by first reversing T, and then moving the symbol 0 from the beginning to the end. Formally, for every $i \in [1\,..\,n]$:

$$T^{\mathrm{rev}}[i] = \begin{cases} T[n-i] & \text{if } i \neq n, \\ T[n] & \text{if } i = n. \end{cases}$$

Observe that for every P not containing the symbol 0, $j \in \mathrm{Occ}(P,T)$ holds if and only if $j' \in \mathrm{Occ}(\overline{P}, T^{\mathrm{rev}})$, where $j' = n - (j + |P| - 1)$.

Remark 7.11. The motivation for defining T^{rev} is that the standard reverse operation on T (denoted $\overline{T}$) does not preserve a unique sentinel at the end.

Components The data structure consists of two parts. The first part is constructed for T and consists of the following two components:

1. The structure from Section 7.2.1 (used to handle nonperiodic nodes).
2. The structure from Section 7.3.1 (used to handle periodic nodes). Note that similarly as the first component it also includes $\mathrm{C}_{\mathrm{ST}}(T)$. It suffices, however, to only store one copy.

The second part contains the analogous two components for the text T^{rev}. In this section, unless specified otherwise, we refer to the part of the structure for text T.

In total, the data structure takes $\mathcal{O}(n/\log_\sigma n)$ space.

7.4.2 Implementation of $\mathrm{sdepth}(v)$

Proposition 7.24. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and $\mathrm{repr}(v)$, we can in $\mathcal{O}(\log^\epsilon n)$ time compute $\mathrm{sdepth}(v)$.*

Proof. Denote $i_1 = \mathrm{lrank}(v) + 1$ and $i_2 = \mathrm{rrank}(v)$. Let v_1 and v_2 be the i_1th and i_2th (respectively) leftmost leaf of $\mathcal{T}_{\mathrm{st}}$. Then, $v = \mathrm{LCA}(v_1, v_2)$. By Observation 4.1, we thus have $\mathrm{sdepth}(v) = \mathrm{lcp}(\mathrm{str}(v_1), \mathrm{str}(v_2)) = \mathrm{LCE}(\mathrm{SA}[i_1], \mathrm{SA}[i_2])$. Consequently, to compute $\mathrm{sdepth}(v)$ we proceed as follows. First, in $\mathcal{O}(\log^\epsilon n)$ time we compute $j_1 = \mathrm{SA}[i_1]$ and $j_2 = \mathrm{SA}[i_2]$ using Proposition 5.17. Then, using the structure to answer LCE queries (stored as part of the structure in Section 5.3.2), in $\mathcal{O}(1)$ time we compute and return $\mathrm{sdepth}(v) = \mathrm{LCE}(j_1, j_2)$. □

7.4.3 Implementation of $\mathrm{LCA}(u, v)$

Proposition 7.25. *Let v_1 and v_2 be explicit nodes of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and the pairs $\mathrm{repr}(v_1)$ and $\mathrm{repr}(v_2)$, we can in $\mathcal{O}(\log^\epsilon n)$ time compute the pair $\mathrm{repr}(\mathrm{LCA}(v_1, v_2))$.*

Proof. First, using Proposition 7.3, in $\mathcal{O}(1)$ time we check if $\mathrm{sdepth}(\mathrm{LCA}(v_1, v_2)) < 3\tau - 1$. If so, in $\mathcal{O}(1)$ time we additionally obtain $\mathrm{repr}(\mathrm{LCA}(v_1, v_2))$. Let us thus assume $\mathrm{sdepth}(\mathrm{LCA}(v_1, v_2)) \geq 3\tau - 1$. Then, Proposition 7.3 additionally indicates whether $\mathrm{LCA}(v_1, v_2)$ is periodic. If not, we use Proposition 7.10 to compute $\mathrm{repr}(\mathrm{LCA}(v_1, v_2))$ in $\mathcal{O}(\log^\epsilon n)$ time. Otherwise, we obtain $\mathrm{repr}(\mathrm{LCA}(v_1, v_2))$ in $\mathcal{O}(\log\log n)$ time using Proposition 7.19. □

7.4.4 Implementation of child(v, c)

PROPOSITION 7.26. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1,* repr(v), *and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* repr(child(v, c)).

Proof. First, using Proposition 7.2, in $\mathcal{O}(1)$ time we check if v is periodic. If so, we obtain repr(child(v, c)) in $\mathcal{O}(\log\log n)$ time using Proposition 7.20. Otherwise (i.e., if v is not periodic), Proposition 7.2 additionally return the information on whether it holds sdepth$(v) < 3\tau - 1$. If so, then we obtain repr(child(v, c)) in $\mathcal{O}(1)$ time using Proposition 7.4. Otherwise, we obtain repr(child(v, c)) in $\mathcal{O}(\log^{\epsilon} n)$ time using Proposition 7.11. □

7.4.5 Implementation of pred(v, c)

PROPOSITION 7.27. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1,* repr(v), *and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* RangeBeg(str$(v)c, T$).

Proof. First, using Proposition 7.2, in $\mathcal{O}(1)$ time we check if v is periodic. If so, we obtain RangeBeg(str$(v)c, T$) in $\mathcal{O}(\log\log n)$ time using Proposition 7.21. Otherwise (i.e., if v is not periodic), Proposition 7.2 additionally return the information on whether it holds sdepth$(v) < 3\tau - 1$. If so, then we obtain RangeBeg(str$(v)c, T$) in $\mathcal{O}(1)$ time using Proposition 7.5. Otherwise, we obtain RangeBeg(str$(v)c, T$) in $\mathcal{O}(\log^{\epsilon} n)$ time using Proposition 7.12. □

PROPOSITION 7.28. *Let v be an explicit internal node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1,* repr(v), *and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* repr(pred(v, c)).

Proof. Denote $(b, e) =$ repr(v). First, using Proposition 7.27, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $i =$ RangeBeg(str$(v)c, T$). Observe that by definition of pred(v, c) we then have pred$(v, c) = \perp$ if and only if $i = b$. If $i = b$, we thus return repr(pred(v, c)) $= (0, 0)$. Let us thus assume $i \neq b$. Observe that we then have SA$[i] \in$ Occ(str(pred(v, c)), T), and moreover, pred$(v, c) =$ child(v, c'), where $c' = T[\mathrm{SA}[i] + \mathrm{sdepth}(v)]$. We thus proceed as follows. First, using Proposition 5.17, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $j = \mathrm{SA}[i]$. Next, using Proposition 7.24, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\ell = \mathrm{sdepth}(v)$. In $\mathcal{O}(1)$ time we then obtain $c' = T[j + \ell]$. Finally, using Proposition 7.26, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute and return repr(child(v, c')) = repr(pred(v, c)). □

7.4.6 Implementation of WA(v, d)

PROPOSITION 7.29. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1,* repr(v), *and an integer d satisfying $0 \leq d \leq |\mathrm{str}(v)|$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* repr(WA(v, d)).

Proof. If $d < 3\tau - 1$, we obtain repr(WA(v, d)) in $\mathcal{O}(1)$ time using Proposition 7.6. Let us thus assume $d \geq 3\tau - 1$. This implies sdepth$(v) \geq 3\tau - 1$. First, using Proposition 7.2, in $\mathcal{O}(1)$ time we determine whether v is periodic. If not, then in $\mathcal{O}(\log^{\epsilon} n)$ time we compute repr(WA(v, d)) using Proposition 7.13. Otherwise, we obtain repr(WA(v, d)) using Proposition 7.22 in $\mathcal{O}(\log\log n)$ time. □

7.4.7 Implementation of wlink(v, c)

PROPOSITION 7.30. *Let $P \in [0\,..\,\sigma)^m$. Given the data structure from Section 7.4.1, the value $|P|$, any $j \in$ Occ(P, T), and any $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can check if* Occ$(Pc, T) \neq \emptyset$, *and if so, return some position $j' \in$ Occ(Pc, T).*

Proof. We start by checking if P contains the symbol 0. For this, we simply check if $j + |P| = n + 1$. If so, we return that Occ$(Pc, T) = \emptyset$. Let us thus assume $j + |P| \leq n$.

Using Proposition 5.16, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $i = \mathrm{ISA}[j]$. Let $(b, e) = (i - 1, i)$, and observe that we then have $(b, e) =$ repr(v), where v is a leaf of $\mathcal{T}_{\mathrm{st}}$ satisfying str$(v) = T[j\,..\,n]$. Next, using Proposition 7.29, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute the pair $(b', e') =$ repr(WA$(v, |P|)$) (we can use it, since $|P| \leq n - j + 1 = \mathrm{sdepth}(v)$). We then have:

$$(b', e') = (\mathrm{RangeBeg}(\mathrm{str}(v)[1\,..\,|P|], T), \mathrm{RangeEnd}(\mathrm{str}(v)[1\,..\,|P|], T))$$

$$= (\text{RangeBeg}(T[j\,..\,j+|P|), T), \text{RangeEnd}(T[j\,..\,j+|P|), T))$$
$$= (\text{RangeBeg}(P, T), \text{RangeEnd}(P, T)).$$

Next, note that it holds $(b', e') = \text{repr}(v')$ for some node v' such that $\text{sdepth}(v') \geq |P|$. To check if $\text{sdepth}(v') = |P|$, in $\mathcal{O}(\log^\epsilon n)$ time we compute $j_1 = \text{SA}[b'+1]$ and $j_2 = \text{SA}[e']$ using Proposition 5.17. As explained in the proof of Proposition 7.24, we then have $\text{sdepth}(v') = |P|$ if and only if $T[j_1 + |P|] \neq T[j_2 + |P|]$, which we can check in $\mathcal{O}(1)$ time (note that $T[j_1 + |P|]$ and $T[j_2 + |P|]$ are well-defined, since $j_1, j_2 \in \text{Occ}(P, T)$ and we assumed that P does not contain symbol $T[n] = 0$). Consider two cases:

- If $T[j_1 + |P|] = T[j_2 + |P|]$, then v' satisfies $\text{sdepth}(v') > |P|$. In that case we check if $c = T[j_1 + |P|]$. If so, we have $j_1 \in \text{Occ}(Pc, T)$ and hence we return j_1. Otherwise, we return that $\text{Occ}(Pc, T) = \emptyset$.
- Otherwise (i.e., if $T[j_1 + |P|] \neq T[j_2 + |P|]$), we have $\text{sdepth}(v') = |P|$. Using Proposition 7.26, we then compute the pair $(b'', e'') = \text{repr}(\text{child}(v', c))$ in $\mathcal{O}(\log^\epsilon n)$ time. If $(b'', e'') = (0, 0)$, then we return that $\text{Occ}(Pc, T) = \emptyset$. Otherwise, we have $\text{Occ}(Pc, T) \neq \emptyset$. We then use Proposition 5.17 to compute $j' = \text{SA}[e''] \in \text{Occ}(Pc, T)$ in $\mathcal{O}(\log^\epsilon n)$ time. □

PROPOSITION 7.31. *Let v be an explicit node of $\mathcal{T}_{\text{st}}$. Given the data structure from Section 7.4.1,* $\text{repr}(v)$*, and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^\epsilon n)$ time we can compute* $\text{repr}(\text{wlink}'(v, c))$.

Proof. Denote $(b, e) = \text{repr}(v)$ and $P = \text{str}(v)$. The algorithm consists of two steps:

1. The first step is to determine if $\text{Occ}(cP, T) \neq \emptyset$, and if so, to compute some $j' \in \text{Occ}(cP, T)$. First, using Proposition 7.24, in $\mathcal{O}(\log^\epsilon n)$ time we compute $\ell := \text{sdepth}(v) = |P|$. Using Proposition 5.17, in $\mathcal{O}(\log^\epsilon n)$ time we also compute $j = \text{SA}[e]$. We then have $j \in \text{Occ}(P, T)$. We now check if $j + \ell - 1 = n$. If so, then by the uniqueness of $T[n]$, we have $\text{Occ}(P, T) = \{j\}$. In that case, we have $\text{Occ}(cP, T) \neq \emptyset$ if and only if $T[j-1] = c$, which we can check in $\mathcal{O}(1)$ time. If $T[j-1] = c$, in $\mathcal{O}(1)$ time we then obtain $j' \in \text{Occ}(cP, T)$, where $j' = j - 1$. Let us now assume that $j + \ell - 1 \neq n$. We now check if $c = 0$. If so, then $\text{Occ}(cP, T) \neq \emptyset$ holds if and only if $\ell = 0$. We can again check this condition in $\mathcal{O}(1)$ time. Moreover, if $\ell = 0$, then we have $j' \in \text{Occ}(cP, T)$, where $j' = n$. Let us thus assume that $c \neq 0$. Observe that then, letting $j^{\text{rev}} := n - (j + \ell - 1)$, it holds $j^{\text{rev}} \in \text{Occ}(\overline{P}, T^{\text{rev}})$. Denote $\ell' = \ell + 1$. Using Proposition 7.30 for the text T^{rev}, in $\mathcal{O}(\log^\epsilon n)$ time we check if $\text{Occ}(\overline{P}c, T^{\text{rev}}) = \emptyset$ (note that we have $|\overline{P}c| = \ell'$). If so, we have $\text{Occ}(cP, T) = \emptyset$, since $\overline{cP} = \overline{P}c$ and hence $\text{Occ}(\overline{P}c, T^{\text{rev}}) = \emptyset$ holds if and only if $\text{Occ}(cP, T) = \emptyset$. Otherwise (i.e., if $\text{Occ}(\overline{P}c, T^{\text{rev}}) \neq \emptyset$), Proposition 7.30 returns some position $j_c^{\text{rev}} \in \text{Occ}(\overline{P}c, T^{\text{rev}})$. Letting $j' := n - (j_c^{\text{rev}} + \ell' - 1)$, we then have $j' \in \text{Occ}(cP, T)$.
2. If in the first step we found that $\text{Occ}(cP, T) = \emptyset$, then by definition it holds $\text{wlink}'(v, c) = \perp$, and hence we return $\text{repr}(\text{wlink}'(v, c)) = (0, 0)$. Let us thus assume that $\text{Occ}(cP, T) \neq \emptyset$ and $j' \in \text{Occ}(cP, T)$. We now compute the SA range containing all elements of $\text{Occ}(cP, T)$. For this, we first compute $i = \text{ISA}[j']$ using Proposition 5.16 in $\mathcal{O}(\log^\epsilon n)$ time. Letting $(b', e') = (i-1, i)$, we then have $(b', e') = \text{repr}(v')$, where v' is a leaf of $\mathcal{T}_{\text{st}}$ satisfying $\text{str}(v') = T[j'\,..\,n]$. Using Proposition 7.29, in $\mathcal{O}(\log^\epsilon n)$ time, we compute $(b'', e'') = \text{repr}(\text{WA}(v', \ell'))$. We then have

$$\begin{aligned}(b'', e'') &= (\text{RangeBeg}(\text{str}(v')[1\,..\,\ell'], T), \text{RangeEnd}(\text{str}(v')[1\,..\,\ell'], T)) \\ &= (\text{RangeBeg}(cP, T), \text{RangeEnd}(cP, T)) \\ &= \text{repr}(\text{wlink}'(v, c)).\end{aligned}$$

In total, the query takes $\mathcal{O}(\log^\epsilon n)$ time. □

PROPOSITION 7.32. *Let v be an explicit node of $\mathcal{T}_{\text{st}}$. Given the data structure from Section 7.4.1,* $\text{repr}(v)$*, and $c \in [0\,..\,\sigma)$, in $\mathcal{O}(\log^\epsilon n)$ time we can compute* $\text{repr}(\text{wlink}(v, c))$.

Proof. As observed at the beginning of Section 7, $\text{wlink}(v, c) \neq \perp$ holds if and only if $\text{wlink}'(v, c) \neq \perp$ and $\text{sdepth}(\text{wlink}'(v, c)) = \text{sdepth}(v) + 1$. Therefore, we can use $\text{wlink}'(v, c)$ to compute $\text{wlink}(v, c)$. First, using Proposition 7.31, in $\mathcal{O}(\log^\epsilon n)$ time we compute $(b, e) = \text{repr}(\text{wlink}'(v, c))$. If $(b, e) = (0, 0)$, then by the above we have $\text{wlink}(v, c) = \perp$, and hence return $\text{repr}(\text{wlink}(v, c)) = (0, 0)$. Otherwise, using Proposition 7.24, in $\mathcal{O}(\log^\epsilon n)$ time we compute $\ell = \text{sdepth}(v)$ and $\ell' = \text{sdepth}(\text{wlink}'(v, c))$. If $\ell' = \ell + 1$, then we return that $\text{repr}(\text{wlink}(v, c)) = (b, e)$. Otherwise, we have $\text{wlink}(v, c) = \perp$ and we return $\text{repr}(\text{wlink}(v, c)) = (0, 0)$. □

7.4.8 Implementation of $\mathrm{slink}(v)$

PROPOSITION 7.33. *Let $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$ be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and $\mathrm{repr}(v)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{slink}(v))$.*

Proof. Denote $(b, e) = \mathrm{repr}(v)$, $P = \mathrm{str}(v)$, and $P' = P[2\,..\,|P|]$. Recall, that for every $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$, $\mathrm{slink}(v)$ is an explicit node of $\mathcal{T}_{\mathrm{st}}$. Thus, to compute $\mathrm{repr}(\mathrm{slink}(v))$, we need to determine $(\mathrm{RangeBeg}(P', T), \mathrm{RangeEnd}(P', T))$.

First, using Proposition 7.24, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\ell := \mathrm{sdepth}(v) = |P|$. Next, using Proposition 5.17, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $j = \mathrm{SA}[e]$. We then have $j \in \mathrm{Occ}(P, T)$. Then, $j' := j + 1$ satisfies $j' \in \mathrm{Occ}(P', T)$. Using Proposition 5.16, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $i = \mathrm{ISA}[j']$. Letting $(b', e') = (i - 1, i)$, we then have $(b', e') = \mathrm{repr}(v')$, where v' is a leaf of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{str}(v') = T[j'\,..\,n]$. Using Proposition 7.29, in $\mathcal{O}(\log^{\epsilon} n)$ time, we compute $(b'', e'') = \mathrm{repr}(\mathrm{WA}(v', \ell - 1))$. We then have

$$\begin{aligned}(b'', e'') &= (\mathrm{RangeBeg}(\mathrm{str}(v')[1\,..\,\ell-1], T), \mathrm{RangeEnd}(\mathrm{str}(v')[1\,..\,\ell-1], T))\\ &= (\mathrm{RangeBeg}(T[j'\,..\,j'+\ell-1), T), \mathrm{RangeEnd}(T[j'\,..\,j'+\ell-1), T))\\ &= (\mathrm{RangeBeg}(P', T), \mathrm{RangeEnd}(P', T))\\ &= \mathrm{repr}(\mathrm{slink}(v)).\end{aligned}$$

In total, the query takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

7.4.9 Implementation of $\mathrm{slink}(v, i)$

PROPOSITION 7.34. *Let $i \in \mathbb{Z}_+$ and let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{sdepth}(v) \geq i$. Given the data structure from Section 7.4.1, $\mathrm{repr}(v)$, and the value i, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{slink}(v, i))$.*

Proof. Denote $(b, e) = \mathrm{repr}(v)$, $P = \mathrm{str}(v)$, and $P' = P[i{+}1\,..\,|P|]$. Note that since for every $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$, $\mathrm{slink}(v)$ is an explicit node of $\mathcal{T}_{\mathrm{st}}$ (Proposition 7.33), it follows that for every explicit node v of $\mathcal{T}_{\mathrm{st}}$ that satisfies $\mathrm{sdepth}(v) \geq i$, $\mathrm{slink}(v, i)$ is an explicit node of $\mathcal{T}_{\mathrm{st}}$. Thus, to compute $\mathrm{repr}(\mathrm{slink}(v))$, we need to determine $(\mathrm{RangeBeg}(P', T), \mathrm{RangeEnd}(P', T))$.

The procedure is a generalization of the one explained in the proof of Proposition 7.34. First, using Proposition 7.24, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\ell := \mathrm{sdepth}(v) = |P|$. Next, using Proposition 5.17, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $j = \mathrm{SA}[e]$. We then have $j \in \mathrm{Occ}(P, T)$. Then, $j' := j + i$ satisfies $j' \in \mathrm{Occ}(P', T)$. Using Proposition 5.16, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $i' = \mathrm{ISA}[j']$. Letting $(b', e') = (i' - 1, i')$, we then have $(b', e') = \mathrm{repr}(v')$, where v' is a leaf of $\mathcal{T}_{\mathrm{st}}$ satisfying $\mathrm{str}(v') = T[j'\,..\,n]$. Using Proposition 7.29, in $\mathcal{O}(\log^{\epsilon} n)$ time, we compute $(b'', e'') = \mathrm{repr}(\mathrm{WA}(v', \ell - i))$. We then have $(b'', e'') = (\mathrm{RangeBeg}(\mathrm{str}(v')[1\,..\,\ell - i], T), \mathrm{RangeEnd}(\mathrm{str}(v')[1\,..\,\ell - i], T)) = (\mathrm{RangeBeg}(T[j'\,..\,j' + \ell - i), T), \mathrm{RangeEnd}(T[j'\,..\,j' + \ell - i), T)) = (\mathrm{RangeBeg}(P', T), \mathrm{RangeEnd}(P', T)) = \mathrm{repr}(\mathrm{slink}(v, i))$. □

7.4.10 Implementation of $\mathrm{parent}(v)$

LEMMA 7.21. *Let $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$ be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Let $\mathrm{repr}(v) = (b, e)$. If $b \neq 0$ (resp. $e \neq n$) then, letting v_1 and v_2 be the bth and $(b+1)$st (resp. eth and $(e+1)$st) leftmost leaves of $\mathcal{T}_{\mathrm{st}}$, the following conditions are equivalent:*

1. $\mathrm{leftsibling}(v) \neq \bot$ *(resp.* $\mathrm{rightsibling}(v) \neq \bot$*),*
2. $\mathrm{parent}(v) = \mathrm{LCA}(v_1, v_2)$.

Proof. Assume $\mathrm{leftsibling}(v) = v_s \neq \bot$ (resp. $\mathrm{rightsibling}(v) = v_s \neq \bot$). By $\mathrm{repr}(v) = (b, e)$, we have $\mathrm{repr}(v_1) = (b - 1, b)$ (resp. $\mathrm{repr}(v_1) = (e - 1, e)$), and $\mathrm{repr}(v_2) = (b, b + 1)$ (resp. $\mathrm{repr}(v_2) = (e, e + 1)$). This implies that v_1 is in the subtree rooted in v_s (resp. v) and v_2 in the subtree rooted in v (resp. v_s). Consequently, $\mathrm{LCA}(v_1, v_2) = \mathrm{LCA}(v, v_s)$. On the other hand, since v_s is a sibling of v, we have $\mathrm{LCA}(v, v_s) = \mathrm{parent}(v)$. Thus, $\mathrm{parent}(v) = \mathrm{LCA}(v_1, v_2)$.

We show that $\mathrm{parent}(v) = \mathrm{LCA}(v_1, v_2)$ implies $\mathrm{leftsibling}(v) \neq \bot$ (resp. $\mathrm{rightsibling}(v) \neq \bot$) by contraposition. Assume $\mathrm{leftsibling}(v) = \bot$ (resp. $\mathrm{rightsibling}(v) = \bot$) and denote $v_p = \mathrm{parent}(v)$. Observe that then $\mathrm{repr}(v_p) = (b, e_p)$ for some $e_p > b$ (resp. $\mathrm{repr}(v_p) = (b_p, e)$ for some $b_p < e$). By $\mathrm{repr}(v_1) = (b - 1, b)$ (resp.

$\mathrm{repr}(v_2) = (e, e+1)$), the node v_1 (resp. v_2) is thus not in the subtree rooted in v_p. On the other hand, $\mathrm{repr}(v_2) = (b, b+1)$ (resp. $\mathrm{repr}(v_1) = (e-1, e)$) implies that v_p is an ancestor of v_2 (resp. v_1). Therefore, the node $\mathrm{LCA}(v_1, v_2) = \mathrm{LCA}(v_1, v_p)$ (resp. $\mathrm{LCA}(v_1, v_2) = \mathrm{LCA}(v_p, v_2)$) is a proper ancestor of v_p. In particular, $\mathrm{LCA}(v_1, v_2) \neq v_p$. □

PROPOSITION 7.35. *Let $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$ be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and $\mathrm{repr}(v)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{parent}(v))$.*

Proof. Let $\mathrm{repr}(v) = (b, e)$. We first construct a set of pairs $\mathcal{P}$ as follows.

- If $b = 0$, we skip this step. Otherwise, let v_1 and v_2 be the leftmost bth and $(b+1)$st leaves of $\mathcal{T}_{\mathrm{st}}$, $(b_1, e_1) = (b-1, b)$, and $(b_2, e_2) = (b, b+1)$. We then have $\mathrm{repr}(v_1) = (b_1, e_1)$ and $\mathrm{repr}(v_2) = (b_2, e_2)$. Using Proposition 7.25, in $\mathcal{O}(\log^{\epsilon} n)$ we obtain $\mathrm{repr}(v')$, where $v' = \mathrm{LCA}(v_1, v_2)$. Note that v' is an ancestor of v. We add $\mathrm{repr}(v')$ to $\mathcal{P}$.
- If $e = n$, we skip this step. Otherwise, let v_1' and v_2' be the leftmost eth and $(e+1)$st leaves of $\mathcal{T}_{\mathrm{st}}$, $(b_1', e_1') = (e-1, e)$, and $(b_2', e_2') = (e, e+1)$. We repeat the same procedure as above, again adding $\mathrm{repr}(v'')$ (where $v'' = \mathrm{LCA}(v_1', v_2')$) to $\mathcal{P}$.

Recall now that we assumed $|T| \geq 2$ and that $T[n]$ is unique in T. This implies that the root of $\mathcal{T}_{\mathrm{st}}$ has at least two children. On the other hand, any other non-leaf node has at least two children by definition. This implies that for every explicit node $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$, it holds that either $\mathrm{leftsibling}(v) \neq \perp$ or $\mathrm{rightsibling}(v) \neq \perp$. Therefore, by Lemma 7.21, there exists $(b_p, e_p) \in \mathcal{P}$ such that $(b_p, e_p) = \mathrm{repr}(\mathrm{parent}(p))$. Since each of the nodes u corresponding to an element in $\mathcal{P}$ is an ancestor of v, to compute $\mathrm{parent}(v)$, it suffices to compute $\mathrm{sdepth}(u)$ for all candidates u and return the pair $\mathrm{repr}(u)$ corresponding to u with the largest value. We obtain $\mathrm{sdepth}(u)$ using Proposition 7.24 in $\mathcal{O}(\log^{\epsilon} n)$ time. By $|\mathcal{P}| \leq 2$, the whole procedure takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

REMARK 7.12. It might appear that the computation of $\mathrm{parent}(v)$ could be implemented by modifying the definition of the $\mathrm{WA}(v, d)$ to instead return the deepest ancestor v' of v satisfying $\mathrm{sdepth}(v') \leq d$ (rather than the most shallow ancestor v' of v satisfying $\mathrm{sdepth}(v') \geq d$). Observe, however that as shown in the proof of Lemma 7.11 (resp. Lemma 7.20), $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{S}}}(v)$ (resp. $\mathrm{map}_{\mathcal{T}_{\mathrm{st}}, \mathcal{T}_{\mathsf{Z}}}(v)$) always returns the *lowest* of all nodes u' of $\mathcal{T}_{\mathsf{S}}$ (resp. $\mathcal{T}_{\mathsf{Z}}$) satisfying $(s(u'), t(u')) = \mathrm{repr}(v)$. This enforces the current definition of $\mathrm{WA}(v, d)$ and implies that the implementation of $\mathrm{parent}(v)$ with $\mathrm{WA}(v, d)$ would require a binary search. Thus, to achieve faster time, $\mathrm{parent}(v)$ is implemented as above.

7.4.11 Implementation of $\mathrm{firstchild}(v)$

PROPOSITION 7.36. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and $\mathrm{repr}(v)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{firstchild}(v))$.*

Proof. Denote $(b, e) = \mathrm{repr}(v)$ and $P = \mathrm{str}(v)$. First, we check if $b + 1 = e$. If so, then v is a leaf and hence we return $\mathrm{repr}(\mathrm{firstchild}(v)) = (0, 0)$ (note that here we used that $|T| \geq 2$ and that $T[n]$ is unique in T, since this implies that every non-leaf node of $\mathcal{T}_{\mathrm{st}}$, including the root, has at least two children).

Let us thus assume $b + 1 \neq e$. Denote $v' = \mathrm{firstchild}(v)$ and $P' = \mathrm{str}(v')$. We then have $v' \neq \perp$. Observe that letting $(b', e') = (b, b+1)$, it holds $(b', e') = \mathrm{repr}(v'')$, where v'' is a leaf of $\mathcal{T}_{\mathrm{st}}$ such that $\mathrm{str}(v')$ is a prefix of $\mathrm{str}(v'')$. On the other hand, by definition, we have $\mathrm{sdepth}(v') \geq \mathrm{sdepth}(v) + 1$, and there is no ancestor of v' at depth $d \in (\mathrm{sdepth}(v) \,..\, \mathrm{sdepth}(v'))$. Therefore, we must have $v' = \mathrm{WA}(v'', \mathrm{sdepth}(v) + 1)$. We thus proceed as follows. First, using Proposition 7.24, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\ell := \mathrm{sdepth}(v) = |P|$. Next, using Proposition 7.29, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $(b'', e'') = \mathrm{repr}(\mathrm{WA}(v'', \ell + 1))$. We then have $\mathrm{repr}(\mathrm{firstchild}(v)) = (b'', e'')$. In total, the query takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

7.4.12 Implementation of $\mathrm{lastchild}(v)$

PROPOSITION 7.37. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and $\mathrm{repr}(v)$, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute $\mathrm{repr}(\mathrm{lastchild}(v))$.*

Proof. Denote $(b, e) = \mathrm{repr}(v)$ and $P = \mathrm{str}(v)$. The algorithm is symmetrical to the one presented in the proof of Proposition 7.36, i.e., rather than setting $(b', e') = (b, b+1)$, we set $(b', e') = (e-1, e)$. For such pair, it holds $(b', e') = \mathrm{repr}(v'')$, where v'' is a leaf of $\mathcal{T}_{\mathrm{st}}$ such that, letting $v' = \mathrm{lastchild}(v)$, the string $\mathrm{str}(v')$ is a prefix of $\mathrm{str}(v'')$. □

7.4.13 Implementation of rightsibling(v)

PROPOSITION 7.38. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and* repr(v)*, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* repr(rightsibling(v)).

Proof. Denote $(b, e) = \mathrm{repr}(v)$. We start by checking if $(b, e) = (0, n)$. If so, then $v = \mathrm{root}(\mathcal{T}_{\mathrm{st}})$. In that case, we have $\mathrm{rightsibling}(v) = \bot$ and hence we return $\mathrm{repr}(\mathrm{rightsibling}(v)) = (0, 0)$.

Let us thus assume that $(b, e) \neq (0, n)$, i.e., $v \neq \mathrm{root}(\mathcal{T}_{\mathrm{st}})$. Using Proposition 7.35, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $(b', e') = \mathrm{repr}(\mathrm{parent}(v))$. We then have $b' \leq b < e \leq e'$. Next, we compare e and e'. If $e = e'$ then, by definition, v is the rightmost child of its parent and hence we return $\mathrm{repr}(\mathrm{rightsibling}(v)) = (0, 0)$. Let us thus assume $e < e'$. We then have $\mathrm{rightsibling}(v) \neq \bot$. Moreover, letting $v' = \mathrm{rightsibling}(v)$ and $P' = \mathrm{str}(v')$, it then holds $\mathrm{SA}[e + 1] \in \mathrm{Occ}(P', T)$. This implies that, letting $(b'', e'') = (e, e + 1)$, we have $(b'', e'') = \mathrm{repr}(v'')$, where v'' is a leaf of $\mathcal{T}_{\mathrm{st}}$ such that P' is a prefix of $\mathrm{str}(v'')$. Moreover, it holds $\mathrm{sdepth}(v') \geq \mathrm{sdepth}(\mathrm{parent}(v)) + 1$, and the node v' does not have any ancestors at depth $d \in (\mathrm{sdepth}(\mathrm{parent}(v)) \,..\, \mathrm{sdepth}(v'))$. Consequently, $v' = \mathrm{WA}(v'', \mathrm{sdepth}(\mathrm{parent}(v)) + 1)$. We thus proceed as follows. First, using Proposition 7.24, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $\ell := \mathrm{sdepth}(\mathrm{parent}(v))$. Then, using Proposition 7.29, in $\mathcal{O}(\log^{\epsilon} n)$ time we compute $(b''', e''') = \mathrm{repr}(\mathrm{WA}(v'', \ell + 1))$. By the above discussion, we have $(b''', e''') = \mathrm{repr}(\mathrm{rightsibling}(v))$. □

7.4.14 Implementation of leftsibling(v)

PROPOSITION 7.39. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and* repr(v)*, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* repr(leftsibling(v)).

Proof. Denote $(b, e) = \mathrm{repr}(v)$. The algorithm is symmetrical to the one presented in the proof of Proposition 7.38. More precisely, we replace the check $e = e'$ with $b = b'$. We also set $(b'', e'') = (b-1, b)$ instead of $(b'', e'') = (e, e+1)$. □

7.4.15 Implementation of isleaf(v)

PROPOSITION 7.40. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given* repr($v$)*, we can check if v is a leaf in $\mathcal{O}(1)$ time.*

Proof. Recall, that $|T| \geq 2$ and that $T[n]$ is unique in T. This implies that the root of $\mathcal{T}_{\mathrm{st}}$ has at least two children. On the other hand, any other non-leaf node has at least two children by definition. Thus, letting $(b, e) = \mathrm{repr}(v)$, and recalling that $\mathrm{repr}(v) = (\mathrm{lrank}(v), \mathrm{rrank}(v))$, the node v is a leaf if and only if $b + 1 = e$, which we can check in $\mathcal{O}(1)$ time. □

7.4.16 Implementation of index(v)

PROPOSITION 7.41. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given the data structure from Section 7.4.1 and* repr(v)*, in $\mathcal{O}(\log^{\epsilon} n)$ time we can compute* index(v).

Proof. Denote $(b, e) = \mathrm{repr}(v)$ and $P = \mathrm{str}(v)$. Recall, that it holds $\mathrm{repr}(v) = (\mathrm{RangeBeg}(P, T), \mathrm{RangeEnd}(P, T))$, and hence $\mathrm{Occ}(P, T) = \{\mathrm{SA}[i]\}_{i \in (b\,..\,e]}$. Thus, to obtain index($v$), it suffices to compute $j = \mathrm{SA}[i]$ for any $i \in (b\,..\,e]$. Using Proposition 5.17, this takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

7.4.17 Implementation of count(v)

PROPOSITION 7.42. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$. Given* repr($v$)*, we can compute* count(v) *in $\mathcal{O}(1)$ time.*

Proof. Denote $(b, e) = \mathrm{repr}(v)$. Since by definition we have $\mathrm{Occ}(\mathrm{str}(v), T) = \{\mathrm{SA}[i]\}_{i \in (b\,..\,e]}$, in $\mathcal{O}(1)$ time we return $\mathrm{count}(v) = |\mathrm{Occ}(\mathrm{str}(v), T)| = e - b$. □

7.4.18 Implementation of letter(v, i)

PROPOSITION 7.43. *Let v be an explicit node of $\mathcal{T}_{\mathrm{st}}$ and $i \in [1\,..\,|\mathrm{str}(v)|]$. Given the data structure from Section 7.4.1,* repr(v)*, and the value i, we can compute* letter(v, i) *in $\mathcal{O}(\log^{\epsilon} n)$ time.*

Proof. It suffices to find any $j \in \mathrm{Occ}(\mathrm{str}(v), T)$ and return $T[j + i - 1]$. Using Proposition 7.41, we find j in $\mathcal{O}(\log^{\epsilon} n)$ time. We then return the output symbol in $\mathcal{O}(1)$ time (recall, that the packed representation of T is stored as part of $\mathrm{C}_{\mathrm{ST}}(T)$). □

7.4.19 Implementation of isancestor(u, v)

PROPOSITION 7.44. *Let u and v be explicit nodes of $\mathcal{T}_{\text{st}}$. Given $\text{repr}(u)$ and $\text{repr}(v)$, we can check if u is an ancestor of v in $\mathcal{O}(1)$ time.*

Proof. Denote $(b, e) = \text{repr}(u)$, $(b', e') = \text{repr}(v)$. The node u is an ancestor of v if and only if $\text{Occ}(\text{str}(v), T) \subseteq \text{Occ}(\text{str}(u), T)$, which (by definition) holds if and only if $b \leq b' < e' \leq e$. □

7.4.20 Implementation of findleaf(j)

PROPOSITION 7.45. *Let $j \in [1\,..\,n]$. Given the data structure from Section 7.4.1 and the position j, in $\mathcal{O}(\log^{\epsilon} n)$ time we can return $\text{repr}(v)$, where v is a leaf of $\mathcal{T}_{\text{st}}$ satisfying $\text{str}(v) = T[j\,..\,n]$.*

Proof. Let $(b, e) = \text{repr}(v)$. Observe that by definition of v, we have $\text{Occ}(\text{str}(v), T) = \{j\}$. Thus, it must hold $\{\text{SA}[i]\}_{i \in (b\,..\,e]} = \{j\}$. This implies, that it suffices to compute $i = \text{ISA}[j]$ and then return that $\text{repr}(v) = (i - 1, i)$. Using Proposition 5.16, the computation of $\text{ISA}[j]$ takes $\mathcal{O}(\log^{\epsilon} n)$ time. □

7.4.21 Construction Algorithm

PROPOSITION 7.46. *Given the packed representation of a text $T \in [0\,..\,\sigma)^n$, we can construct the data structure from Section 7.4.1 in $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space.*

Proof. The first part of the structure is constructed as follows. First, from a packed representation of T, we construct $\text{C}_{\text{ST}}(T)$ in $\mathcal{O}(n/\log_\sigma n)$ time using Proposition 7.7. Then, using Propositions 7.14 and 7.23, we augment $\text{C}_{\text{ST}}(T)$ in $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ and $\mathcal{O}(n/\log_\sigma n)$ time (respectively) and using $\mathcal{O}(n/\log_\sigma n)$ working space into the two components of the structure from Section 7.4.1. The overall runtime is thus $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$.

Next, we compute T^{rev}. With the help of the lookup table L_{rev}, we first compute $T^{\text{rev}}[1\,..\,n) = \overline{T[1\,..\,n)}$ in $\mathcal{O}(n/\log_\sigma n)$ time. In $\mathcal{O}(1)$ time we then append the sentinel $T^{\text{rev}}[n] := 0$. After that, analogously as above, we construct the structures from Sections 7.2.1 and 7.3.1 for T^{rev}, i.e., the second part of the structure from Section 7.4.1. □

7.5 Summary

By combining Propositions 7.24 to 7.26, 7.28, 7.29, and 7.32 to 7.46 we obtain the following main result of this section.

THEOREM 7.1. *Given any constant $\epsilon \in (0, 1)$ and the packed representation of a text $T \in [0\,..\,\sigma)^n$ with $2 \leq \sigma < n^{1/7}$, in $\mathcal{O}(n \min(1, \log \sigma/\sqrt{\log n}))$ time and $\mathcal{O}(n/\log_\sigma n)$ working space we can construct a representation of the suffix tree of T occupying $\mathcal{O}(n/\log_\sigma n)$ space and supporting all standard operations (see Table 1) in $\mathcal{O}(\log^{\epsilon} n)$ time.*

We also immediately obtain the following general reduction.

THEOREM 7.2. *Consider a data structure answering prefix rank and selection queries that, for any string of length m over alphabet $[0\,..\,\sigma)^{\ell}$, achieves the following complexities:*

1. *Space usage $S(m, \ell, \sigma)$,*
2. *Preprocessing time $P_t(m, \ell, \sigma)$,*
3. *Preprocessing space $P_s(m, \ell, \sigma)$,*
4. *Query time $Q(m, \ell, \sigma)$.*

For every $T \in [0\,..\,\sigma)^n$ with $2 \leq \sigma < n^{1/7}$, there exist $m = \mathcal{O}(n/\log_\sigma n)$ and $\ell = \mathcal{O}(\log_\sigma n)$ such that, given the packed representation of T, we can in $\mathcal{O}(n/\log_\sigma n + P_t(m, \ell, \sigma))$ time and $\mathcal{O}(n/\log_\sigma n + P_s(m, \ell, \sigma))$ working space construct a representation of the suffix tree of T occupying $\mathcal{O}(n/\log_\sigma n + S(m, \ell, \sigma))$ space and supporting all standard operations (see Table 1) in $\mathcal{O}(\log \log n + Q(m, \ell, \sigma))$ time.

References

[1] Donald Adjeroh, Tim Bell, and Amar Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching*. Springer, Boston, MA, USA, 2008. doi:10.1007/978-0-387-78909-5.

[2] Amihood Amir, Gad M. Landau, Moshe Lewenstein, and Dina Sokol. Dynamic text and static pattern matching. *ACM Trans. Algorithms*, 3(2):19, 2007. doi:10.1145/1240233.1240242.

[3] Maxim Babenko, Paweł Gawrychowski, Tomasz Kociumaka, and Tatiana Starikovskaya. Wavelet trees meet suffix trees. In *26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 572–591, 2015. doi:10.1137/1.9781611973730.39.

[4] Jérémy Barbay, Francisco Claude, Travis Gagie, Gonzalo Navarro, and Yakov Nekrich. Efficient fully-compressed sequence representations. *Algorithmica*, 69(1):232–268, 2014. doi:10.1007/s00453-012-9726-3.

[5] Djamal Belazzougui. Linear time construction of compressed text indices in compact space. In *46th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 148–193, 2014. doi:10.1145/2591796.2591885.

[6] Djamal Belazzougui, Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In *Data Compression Conference (DCC)*, pages 83–92, 2015. doi:10.1109/DCC.2015.69.

[7] Djamal Belazzougui and Gonzalo Navarro. Alphabet-independent compressed text indexing. *ACM Trans. Algorithms*, 10(4):23:1–23:19, 2014. doi:10.1145/2635816.

[8] Djamal Belazzougui and Gonzalo Navarro. Optimal lower and upper bounds for representing sequences. *ACM Trans. Algorithms*, 11(4):31:1–31:21, 2015. doi:10.1145/2629339.

[9] Djamal Belazzougui and Simon J. Puglisi. Range predecessor and Lempel-Ziv parsing. In *27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2053–2071, 2016. doi:10.1137/1.9781611974331.ch143.

[10] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *4th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 88–94, 2000. doi:10.1007/10719839_9.

[11] Philip Bille, Mikko Berggren Ettienne, Inge Li Gørtz, and Hjalte Wedel Vildhøj. Time-space trade-offs for Lempel-Ziv compressed indexing. *Theor. Comput. Sci.*, 713:66–77, 2018. doi:10.1016/j.tcs.2017.12.021.

[12] Philip Bille, Inge Li Gørtz, and Frederik Rye Skjoldjensen. Deterministic indexing for packed strings. In *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 6:1–6:11, 2017. doi:10.4230/LIPIcs.CPM.2017.6.

[13] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM J. Comput.*, 44(3):513–539, 2015. doi:10.1137/130936889.

[14] Christina Boucher, Ondrej Cvacho, Travis Gagie, Jan Holub, Giovanni Manzini, Gonzalo Navarro, and Massimiliano Rossi. PFP compressed suffix trees. In *24th Symposium on Algorithm Engineering and Experiments (ALENEX)*, pages 60–72, 2021. doi:10.1137/1.9781611976472.5.

[15] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994. URL: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[16] Manuel Cáceres and Gonzalo Navarro. Faster repetition-aware compressed suffix trees based on block trees. *Inf. Comput.*, 285(Part):104749, 2022. doi:10.1016/j.ic.2021.104749.

[17] Ho-Leung Chan, Wing-Kai Hon, Tak Wah Lam, and Kunihiko Sadakane. Compressed indexes for dynamic text collections. *ACM Trans. Algorithms*, 3(2):21, 2007. doi:10.1145/1240233.1240244.

[18] Timothy M. Chan and Mihai Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 161–173, 2010. doi:10.1137/1.9781611973075.15.

[19] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988. doi:10.1137/0217026.

[20] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Trans. Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.

[21] David R. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1998. URL: https://uwspace.uwaterloo.ca/bitstream/handle/10012/64/nq21335.pdf.

[22] Richard Cole, Tsvi Kopelowitz, and Moshe Lewenstein. Suffix trays and suffix trists: Structures for faster text indexing. *Algorithmica*, 72(2):450–466, 2015. doi:10.1007/s00453-013-9860-6.

[23] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, 2nd edition, 2006. doi:10.1002/047174882X.

[24] Maxime Crochemore, Christophe Hancart, and Thierry Lecroq. *Algorithms on strings*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/cbo9780511546853.

[25] Martin Farach and S. Muthukrishnan. Perfect hashing for strings: Formalization and algorithms. In *7th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 130–140, 1996. doi:10.1007/3-540-61258-0_11.

[26] Martin Farach-Colton, Paolo Ferragina, and S. Muthukrishnan. On the sorting-complexity of suffix tree construction. *J. ACM*, 47(6):987–1011, 2000. doi:10.1145/355541.355547.

[27] Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *41st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

[28] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, 2005. doi:10.1145/1082036.1082039.

[29] Nathan J. Fine and Herbert S. Wilf. Uniqueness theorems for periodic functions. *Proc. Am. Math. Soc.*, 16(1):109–114, 1965. doi:10.2307/2034009.

[30] Johannes Fischer and Paweł Gawrychowski. Alphabet-dependent string searching with Wexponential search trees. In *26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 160–171, 2015. Full version: https://arxiv.org/abs/1302.3347. doi:10.1007/978-3-319-19929-0_14.

[31] Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. Faster entropy-bounded compressed suffix trees. *Theor. Comput. Sci.*, 410(51):5354–5364, 2009. doi:10.1016/j.tcs.2009.09.012.

[32] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In *6th International Conference on Language and Automata Theory and Applications (LATA)*, pages 240–251, 2012. doi:10.1007/978-3-642-28332-1_21.

[33] Travis Gagie, Paweł Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In *11th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 731–742, 2014. doi:10.1007/978-3-642-54423-1_63.

[34] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM*, 67(1):1–54, apr 2020. doi:10.1145/3375890.

[35] Younan Gao, Meng He, and Yakov Nekrich. Fast preprocessing for optimal orthogonal range reporting and range successor with applications to text indexing. In *28th Annual European Symposium on Algorithms (ESA)*, pages 54:1–54:18, 2020. doi:10.4230/LIPIcs.ESA.2020.54.

[36] Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1509–1528, 2018. Full version: https://arxiv.org/abs/1511.02612. doi:10.1137/1.9781611975031.99.

[37] Simon Gog. *Compressed suffix trees: design, construction, and applications*. PhD thesis, University of Ulm, 2011. URL: http://vts.uni-ulm.de/docs/2011/7786/vts_7786_11228.pdf.

[38] Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms (SEA)*, pages 326–337, 2014. doi:10.1007/978-3-319-07959-2_28.

[39] Simon Gog, Juha Kärkkäinen, Dominik Kempa, Matthias Petri, and Simon J. Puglisi. Fixed block compression boosting in FM-indexes: Theory and practice. *Algorithmica*, 81(4):1370–1391, 2019. doi:10.1007/s00453-018-0475-9.

[40] Simon Gog, Alistair Moffat, and Matthias Petri. CSA++: Fast pattern search for large alphabets. In *19th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 73–82, 2017. doi:10.1137/1.9781611974768.6.

[41] Roberto Grossi, Ankur Gupta, and Jeffrey Scott Vitter. High-order entropy-compressed text indexes. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, 2003. URL: https://dl.acm.org/doi/10.5555/644108.644250.

[42] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching (extended abstract). In *32nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 397–406, 2000. doi:10.1145/335305.335351.

[43] Roberto Grossi and Jeffrey Scott Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 35(2):378–407, 2005. doi:10.1137/S0097539702402354.

[44] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997. doi:10.1017/cbo9780511574931.

[45] Torben Hagerup. Sorting and searching on the word RAM. In *15th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 366–398, 1998. doi:10.1007/BFb0028575.

[46] Wing-Kai Hon, Kunihiko Sadakane, and Wing-Kin Sung. Breaking a time-and-space barrier in constructing full-text indices. In *44th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 251–260, 2003. doi:10.1109/SFCS.2003.1238199.

[47] Guy Jacobson. Space-efficient static trees and graphs. In *30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 549–554, 1989. doi:10.1109/SFCS.1989.63533.

[48] Juha Kärkkäinen, Dominik Kempa, and Simon J. Puglisi. Hybrid compression of bitvectors for the FM-index. In *Data Compression Conference (DCC)*, pages 302–311, 2014. doi:10.1109/DCC.2014.87.

[49] Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53(6):918–936, 2006. doi:10.1145/1217856.1217858.

[50] Dominik Kempa and Tomasz Kociumaka. String synchronizing sets: Sublinear-time BWT construction and optimal LCE data structure. In *51st Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 756–767, 2019. doi:10.1145/3313276.3316368.

[51] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler Transform conjecture. In *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1013, 2020. doi:10.1109/FOCS46700.2020.00097.

[52] Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1657–1670, 2022. doi:10.1145/3519935.3520061.

[53] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In *50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 827–840, 2018. doi:10.1145/3188745.3188814.

[54] Tomasz Kociumaka. *Efficient Data Structures for Internal Queries in Texts*. PhD thesis, University of Warsaw, 2018. URL: https://depotuw.ceon.pl/bitstream/handle/item/3614/1000-DR-INF-170341.pdf.

[55] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, 10(3):R25, 2009. doi:10.1186/gb-2009-10-3-r25.

[56] Heng Li and Richard Durbin. Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinform.*, 25(14):1754–1760, 2009. doi:10.1093/bioinformatics/btp324.

[57] Ruiqiang Li, Chang Yu, Yingrui Li, Tak Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. SOAP2: An improved ultrafast tool for short read alignment. *Bioinform.*, 25(15):1966–1967, 2009. doi:10.1093/bioinformatics/btp336.

[58] Veli Mäkinen, Djamal Belazzougui, Fabio Cunial, and Alexandru I. Tomescu. *Genome-scale algorithm design: Biological sequence analysis in the era of high-throughput sequencing*. Cambridge University Press, Cambridge, UK, 2015. doi:10.1017/cbo9781139940023.

[59] Veli Mäkinen and Gonzalo Navarro. Dynamic entropy-compressed sequences and full-text indexes. *ACM Trans. Algorithms*, 4(3):32:1–32:38, 2008. doi:10.1145/1367064.1367072.

[60] Udi Manber and Eugene W. Myers. Suffix arrays: A new method for on-line string searches. *SIAM J. Comput.*, 22(5):935–948, 1993. doi:10.1137/0222058.

[61] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Space-efficient construction of compressed indexes in deterministic linear time. In *28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 408–424, 2017. doi:10.1137/1.9781611974782.26.

[62] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Fast compressed self-indexes with deterministic linear-time construction. *Algorithmica*, 82(2):316–337, 2020. doi:10.1007/s00453-019-00637-x.

[63] J. Ian Munro, Gonzalo Navarro, and Yakov Nekrich. Text indexing and searching in sublinear time. In *31st Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 24:1–24:15, 2020. doi:10.4230/LIPIcs.CPM.2020.24.

[64] J. Ian Munro, Yakov Nekrich, and Jeffrey Scott Vitter. Fast construction of wavelet trees. *Theor. Comput. Sci.*, 638:91–97, 2016. doi:10.1016/j.tcs.2015.11.011.

[65] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/cbo9781316588284.

[66] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surv.*, 54(2), 2021. doi:10.1145/3434399.

[67] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surv.*, 54(2), 2021. doi:10.1145/3432999.

[68] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

[69] Gonzalo Navarro and Yakov Nekrich. Time-optimal top-k document retrieval. *SIAM J. Comput.*, 46(1):80–113, 2017. doi:10.1137/140998949.

[70] Gonzalo Navarro and Nicola Prezza. Universal compressed text indexing. *Theor. Comput. Sci.*, 762:41–50, 2019. doi:10.1016/j.tcs.2018.09.007.

[71] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.

[72] Enno Ohlebusch. *Bioinformatics algorithms: Sequence analysis, genome rearrangements, and phylogenetic reconstruction*. Oldenbusch Verlag, Ulm, Germany, 2013.

[73] Enno Ohlebusch, Johannes Fischer, and Simon Gog. CST++. In *17th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 322–333, 2010. doi:10.1007/978-3-642-16321-0_34.

[74] Nicola Prezza. A framework of dynamic data structures for string processing. In *16th International Symposium on Experimental Algorithms (SEA)*, pages 11:1–11:15, 2017. doi:10.4230/LIPIcs.SEA.2017.11.

[75] Nicola Prezza and Giovanna Rosone. Space-efficient construction of compressed suffix trees. *Theor. Comput. Sci.*, 852:138–156, 2021. doi:10.1016/j.tcs.2020.11.024.

[76] Mihai Pătraşcu. Lower bounds for 2-dimensional range counting. In *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 40–46, 2007. doi:10.1145/1250790.1250797.

[77] Luís M. S. Russo, Gonzalo Navarro, and Arlindo L. Oliveira. Fully compressed suffix trees. *ACM Trans. Algorithms*, 7(4):53:1–53:34, 2011. doi:10.1145/2000807.2000821.

[78] Milan Rŭzić. Constructing efficient dictionaries in close to sorting time. In *35th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 84–95, 2008. doi:10.1007/978-3-540-70575-8_8.

[79] Kunihiko Sadakane. Succinct representations of lcp information and improvements in the compressed suffix arrays. In *13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 225–232, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545410.

[80] Kunihiko Sadakane. Compressed suffix trees with full functionality. *Theory Comput. Syst.*, 41(4):589–607, 2007. doi:10.1007/s00224-006-1198-x.

[81] Peter Weiner. Linear pattern matching algorithms. In *14th Annual Symposium on Switching and Automata Theory (SWAT/FOCS)*, pages 1–11, 1973. doi:10.1109/SWAT.1973.13.

[82] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *Trans. Inf. Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.