Algorithm Seminar – Chomping Strings

Scribe: Jiemin Zeng
Proposer: Steven Skiena

Friday, August 30, 2013

1 Chomping Strings

Suppose we have a text string T and a pattern P. Let us define an operation called "chomp" where we are given T, P, and a location of T where the pattern P exists. When we chomp T, we remove the pattern P from T at the specified location and concatenate the remaining pieces. Note that we remove the pattern only once in one chomp operation.

Given T and P, can we execute a series of chomp operations such that only the empty string remains? Can we chomp the entire string T?

Consider the example $T = AABABA$ and $P = ABA$. We can chomp T at two locations, T[1-3] and T[3-5]. chomp(T, P, [1-3]) $\Rightarrow T' = ABA$ and chomp(T, P, [3-5]) $\Rightarrow T' = AAB$. As you can see, if we are trying to chomp the entire string, we need to use the chomp(T, P, [1-3]) operation first. If we used the chomp(T, P, [3-5]) operation first, then the resulting string will not be chompable.

2 A Solution

We can express any string that is chompable by a pattern as a context-free grammar. For example the CFG for $P = a_1a_2a_3 \dots a_k$ is:

$$T \to \varepsilon$$

$$T \to T\ a_1\ T\ a_2\ T\ a_3\ \dots\ T\ a_k\ T$$

$$T \to T\ T$$

Using dynamic programming, we can parse a string in $O(n^3)$ time. So we have an $O(n^3)$ algorithm to determine if a string is chompable.

3 Circular Strings

If a circular string T is chompable, then there exists a cut such that the linear string is chompable. Below is a sketch of the proof.

Proof. For the case of a circular string consisting of a single instance of the pattern, there exists a cut, between the beginning and end of the pattern, where the resulting linear string is chompable. Note that a sequence of chomps can be represedted in reverse as a series of pattern inserts into an empty string (or a single pattern). For each insert, the pattern is inserted between two adjacent characters. If these two adjacent characters are straddling the cut, the pattern can be inserted to the right or the left of the cut. Therefore, at each stage (including the last), there is a cut that exists that can create a linear chompable string. $\square$

This result immediately implies an $O(n^4)$ algorithm for circular strings, by trying the linear strings resulting from all n possible cuts with the parsing algorithm above. This can be improved to $O(n^3)$ by observing that if we break the circular string in a arbitrary place to create a linear string S, the concatentated string SS has all n-length windows of the original circular string. Thus we need to know whether there exists a parsible region of SS from i to $i+n-1$, which is determined for all $1 \leq i \leq n$ in the course of parsing SS via dynamic programming.

4 Open Questions

1. Taking the most meat off the bone: Given T and P, can you find the largest sequence of chomp operations we can execute starting from T? I (Steve) believe this can be done by dynamic programming, where the state $T[i, j, x_l, y_l, x_r, y_r, P]$ is the most chomps possible of the substring of T from i to j ending up in non-terminal V, where the left of the remaining bone starts with string $T[x_l] \ldots T[y_l]$ and the right of the remaining bone with string $T[x_r] \ldots T[y_r]$ Thus the state table would be $O(n^2 k^4 s)$ in size, where s is the number of non-terminals in the grammar.

   The recurrence would find the cheapst way the non-terminal V can be produced by production $V \varepsilon AB$ between $T[i]$ and $T[j]$ as a consequence of something like

   $$T[i, j, x_l, y_l, x_r, y_r, V] = \min_z T[i, z, x_l, y_l', x_r', y_r', A] + T[z+1, j, x_l'', y_l'', x_r'', x_r, B]$$

   where the allowable indices make the right endstrings.

   So: (1) what is the right recurrence and the exact complexity, and (2) is there a better algorithm, or can we argue this is as hard as more general parsing?

2. Is the ultimate bone unique: Do all paths from T to the point where it does not have any more occurences of P produce the same remaining string? This is interesting in the pseudo-biology application: we would like to know if the state of the string after exposure to the chomper is completely determined, ideally designing a sequence with this property.

3. Leaving pretty patterns on the bone (scrimshaw): Let T' be a second string. Can we chomp from T to remaining bone T' using chomps from pattern P? The earlier problem dealt with the case of T' equals the empty string.

Other generalizations include multiple chomping patterns, or restrictions on how many time each chomping pattern can/must be used.