Text search

CSE 392, Computers Playing Jeopardy!, Fall 2011

Stony Brook University

http://www.cs.stonybrook.edu/~cse392

Today

- 2 parts:
  - theoretical: costs of searching substrings, data structures for string search
  - practical: implementation of text search

Sub-array algorithm example

- Given an array {t,h,i,s,i,s,a,t,e,s,t} and a pattern {t,e,s,t}, write a program that checks whether the pattern is present in the array:

```
public static boolean substring(char[] s, char[] sub){
  for(int i=0; i < s.length - sub.length; i++)
      if(startsWith(s,sub,i)) return true;
  return false;
}
public static boolean startsWith(char[] s, char[] sub,
  int m){
  for(int i=0; i<sub.length; i++)
      if(sub[i] != s[m+i]) return false;
  return true;
}                                    Cost: m x n ☹
```

Suffix arrays and trees

- Idea: preprocess the text, so the search of the substring is fast
- Specialized data structures (e.g., tries)
  - Assumption: no suffix is a prefix of another suffix (can be a substring, but not a prefix)
    - Assure this by adding a character $ to end of S
  - Costs:
    - Build data structure for text (e.g., suffix tree)
      - This is preprocessing O(m)
    - Search time:
      - For example: Suffix trees: O(n+k) where k is the number of occurrences of P in T

Suffix arrays

- An array of integers giving the starting positions of suffixes of a string in lexicographical order

1	2	3	4	5	6	7	8
T	E	S	T	I	N	G	$

  - 8 suffixes: "TESTING$", "ESTING$", "STING$", "TING$", "ING$", "NG$", "G$", "$".

index	Sorted suffix	lcp
8	$	0
2	ESTING$	0
5	ING$	0
7	G$	0
6	NG$	0
3	STING$	0
1	TESTING$	0
4	TING$	1

One-based indexing: {8,2,5,7,6,3,1,4}

Longest common prefix: how many characters one suffix has in common with the one above it

Suffix arrays

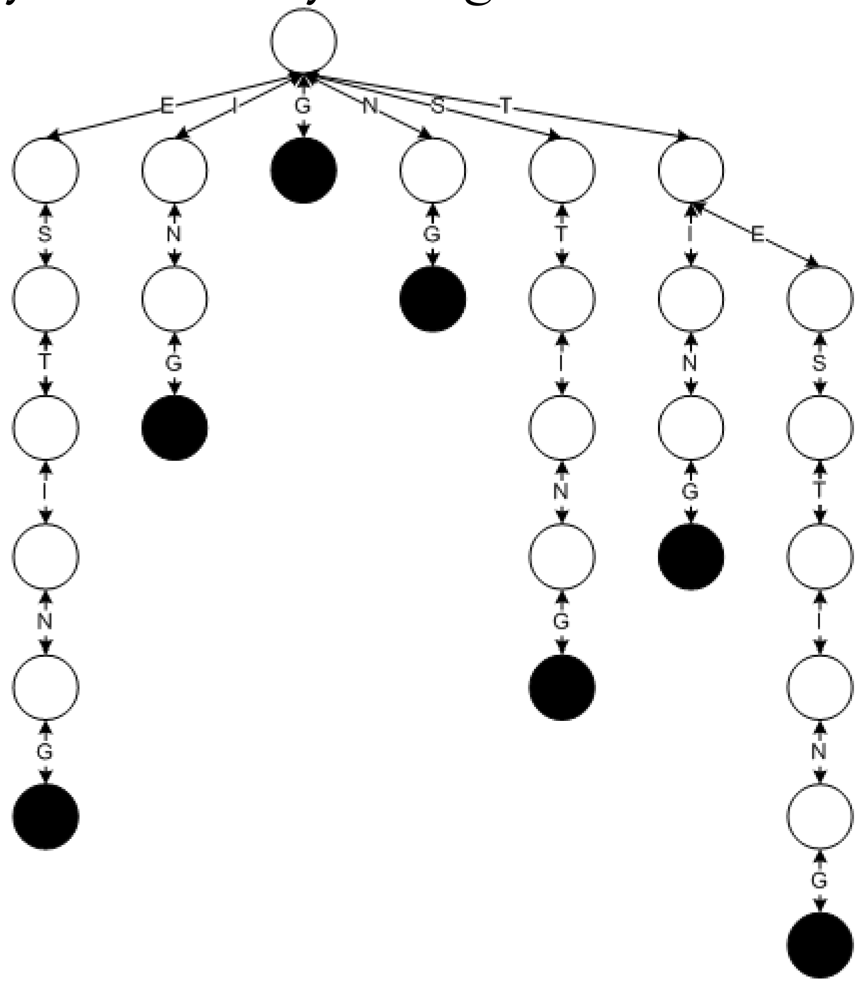
- Construction: comparison sort or suffix trees
- Application: fast search of every occurrence of a substring within a string
  - find every suffix that begins with the substring
  - Cost: O(m log n) time

```
if W <= suffixAt(pos[1]) then
  ans = 1
else if W > suffixAt(pos[n]) then
  ans = n
else{
  L = 1, R = n
  while R-L > 1 do{
    M = (L + R)/2
    if W <= suffixAt(pos[M]) then
      R = M
    else
      L = M
  }
  ans = R
}
```

Suffix tries

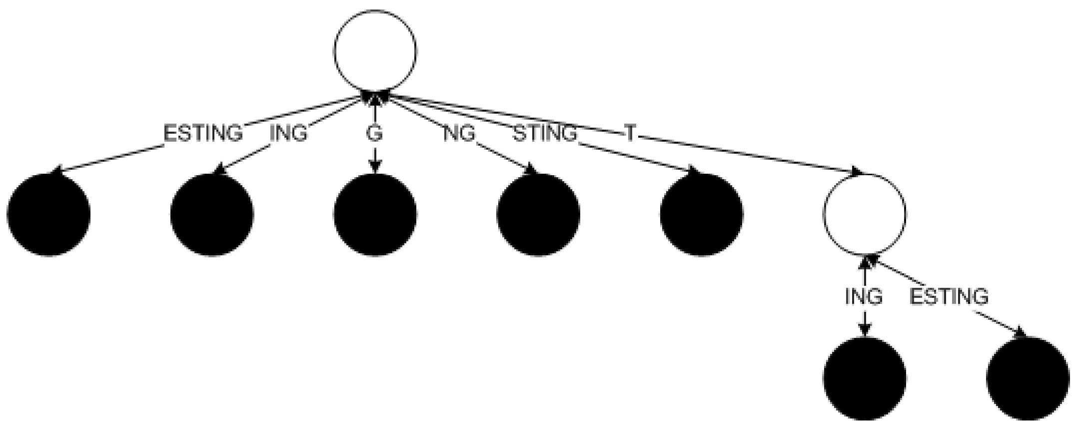
- Tries = ordered tree data structure that is used to store associative arrays where the keys are usually strings

The time to insert, or to delete or to find is identical



Suffix trees

- A data structure that presents the suffixes of a given string in a way that allows for fast implementation of string operations



Building trees: $O(m^2)$ algorithm

- Initialize
  - One edge for the entire string S[1..m]$
- For i = 2 to m
  - Add suffix S[i..m] to suffix tree
    - Find match point for string S[i..m] in current tree
    - If in "middle" of edge, create new node w
    - Add remainder of S[i..m] as edge label to suffix i leaf
- Running Time
  - O(m-i) time to add suffix S[i..m]

Assignment

- The Suffix Array Representing "BANANAS"
- The Suffix Trie Representing "BANANAS“
- The Suffix Tree Representing "BANANAS"

Before search: Tokenization

- Automatically recognize words and sentences
  - identify what constitutes an individual or distinct word, referred to as a token
- Tokenizer or lexer
  - sequences of characters which represent words and other elements, such as punctuation, which are represented by numeric codes,
  - email addresses, phone numbers, and URLs

Other indexes

- Theoretical: Gödel numbering (assigns to each symbol and well-formed formula of some formal language a unique natural number) – not practical
- Hashing: fast, but not unique – collisions, clustering
- B-trees: balanced search trees where every node has between m/2 and m children, where m>1 is a fixed integer

Inverted index

- A mapping from content, such as words or numbers, to its locations in a database file, or in a document or a set of documents
  - T_0 = "it is what it is”
  - T_1 = "what is it“
  - T_2 = "it is a banana"

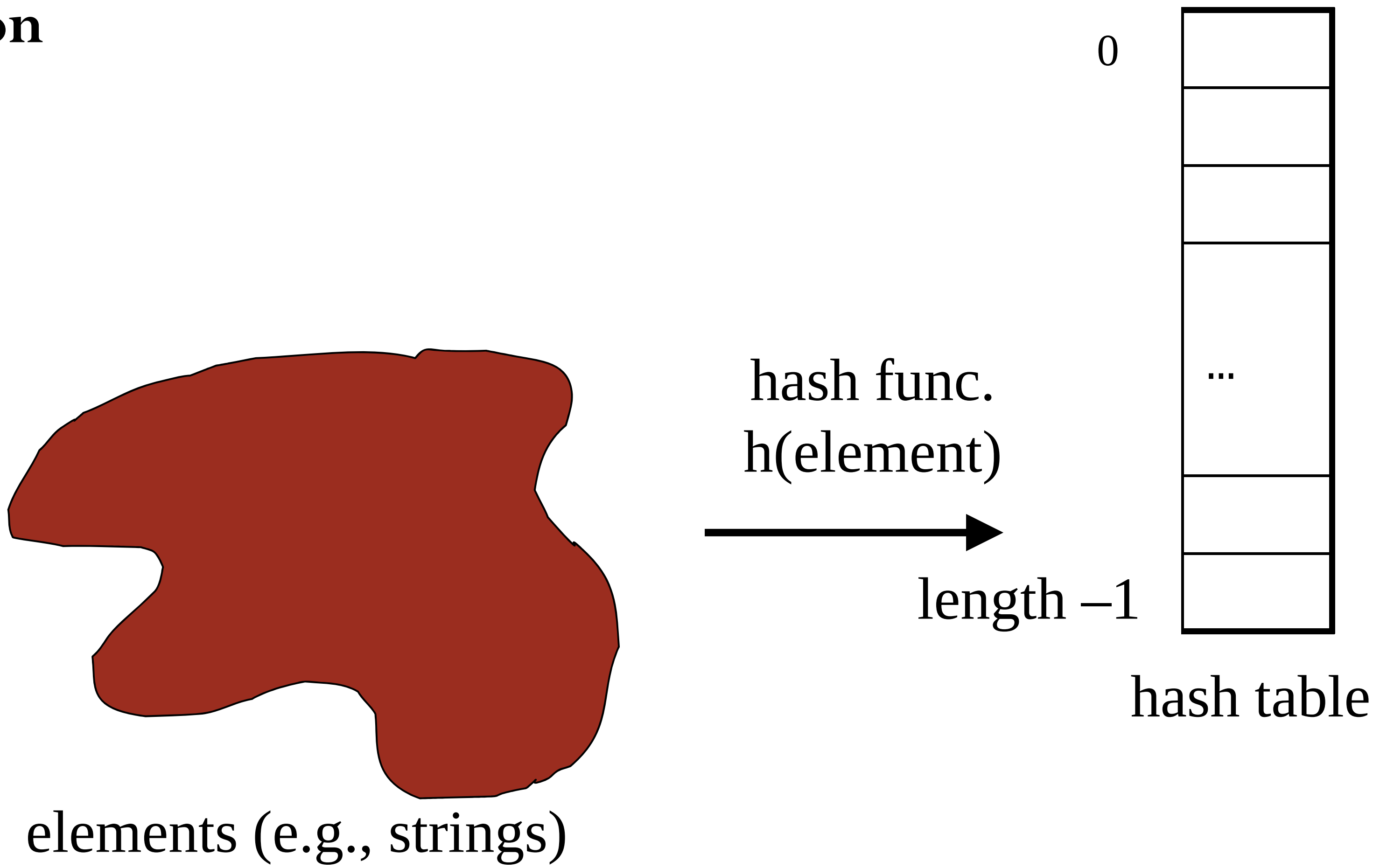
```
"a": {2}
"banana": {2}
"is": {0, 1, 2}
"it": {0, 1, 2}
"what": {0, 1}
```

search for the terms `"what"`, `"is"` and `"it"` would give

$$\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}$$

Hash tables

- **hash table**: an array of some fixed size, that positions elements according to an algorithm called a **hash function**



Hashing, hash functions

- Map every element into some index in the array
  - Lookup becomes ***constant-time***: simply look at that one slot again later to see if the element is there
  - add, remove, contains all become O(1) !
  - Example: *h(i) = i % array.length*

B-trees

- The data items are stored at leaves
- The nonleaf nodes store up to M-1 keys to guide the searching; key I represents the smallest key in subtree I +1.
- The root is either a leaf or has between two and M children.
- All nonleaf nodes (except the root) have between [M/2] and M children
- All leaves are at the same depth and have between [L/2] and L children, for some L (the determination of L is described shortly).

Apache Lucene

- http://lucene.apache.org/
- Tutorial:
- http://www.lucenetutorial.com/lucene-in-5-minutes.html

Parallelism: MapReduce

- Input: a set of key/value pairs
- User supplies two functions:
  - map(k,v) → list(k1,v1)
  - reduce(k1, list(v1)) → v2
- (k1,v1) is an intermediate key/value pair
- Output is the set of (k1,v2) pairs

Hadoop

- An open-source implementation of Map Reduce in Java
  - Uses HDFS for stable storage
- Download from:

http://lucene.apache.org/hadoop/

http://developer.yahoo.com/hadoop/tutorial/module3.html