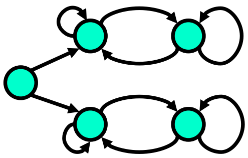
Algorithm Design and Analysis
Victor Adamchik CS 15-451 Spring 2014
Lecture 40 Apr 28, 2014 Carnegie Mellon University

String Matching - II

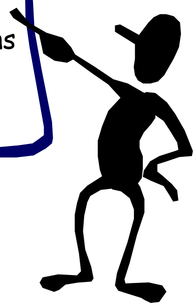
Morris



Knuth

The KMP Algorithm

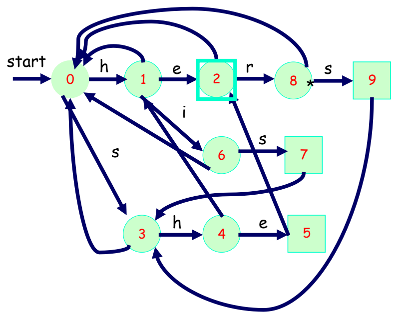
Theorem:
At most 2N comparisons in total



The Aho-Corasick Algorithm
(1986)

The algorithm preprocesses the set of patterns.

Patterns {he, she, his, hers}

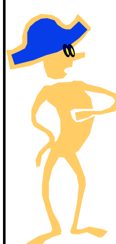


The Aho-Corasick Algorithm

We still use the longest suffix rule. If we fail on making a transition from a node N to its child, we transition to a node M, where the string that defines M is the farthest node (longest prefix) from the root which is also a suffix of the string we had matched when we failed (removing the first transition).

The only difference is that instead of traversing a single string left-to-right we now have to traverse a trie.

The Rabin-Karp Algorithm
(1981)



The algorithm uses the idea of hashing

The main idea

pattern = 4848
text = 16180339887498948482045

We do not match a string against a given pattern, but rather compare their hash codes.

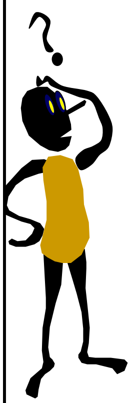
The main idea

pattern = 4848 % 71 = 20

1 6 1 8 0 3 3 9 8 8 7 4 9 8 9 4 8 4 8 2 0 4 5

1 6 1 8	1618 % 71 = 56
6 1 8 0	6180 % 71 = 3
1 8 0 3	1803 % 71 = 28

We read the text in the number of characters equal to the length of the pattern, compute its hash code and compare with the pattern hash code.



What is its complexity?

M = pattern.length()
N = text.length();

Similar to a brute-force matching…

The key idea of improving the algorithm is in computing a hash code in O(1).

Computing a hash code

How can we get from 145 to 456?

We will do this by creating a chain of operations

145 - 45 - 450 - 456

Remove the leading digit, multiply by a base, add a single digit. It takes O(1) to compute a hash code from the previous value.

Example

Given: a hash code for 31729

31729 mod 41 = 36

Task: compute a hash code for 17295.

Example

Given: a hash code for 31729

31729 mod 41 = 36

Task: compute a hash code for 17295.

Observe,

$$17295 = (31729 - 3*10^4) * 10 + 5$$

Example

17295%41 = [(31729%41-3*10^4%41)*10 + 5]%41

31729%41 is already computed.

3*10^4 % 41 will be precomputed

17295%41 = [(36 - 29)*10 + 5]%41
= 75 % 41 = 34

Rabin-Karp formalized

Let P[1 ... m] be a pattern and T[1 ... n] be a text. We define a pattern

$$P = 10^{m-1} P[1] + 10\ P[m-1] + \dots + P[m]$$

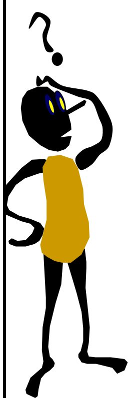
and a shift in the text:

$$t_s = 10^{m-1} T[s+1] + 10\ T[s+m-1] + \dots + T[s+m]$$

The value t_{s+1} can be obtained from t_s by

$$t_{s+1} = (t_s - 10^{m-1} T[s+1])\ 10 + T[s+m+1]$$

Exercise



We said
"31729%41 is already computed"

How would you compute it fast?

Horner's Rule

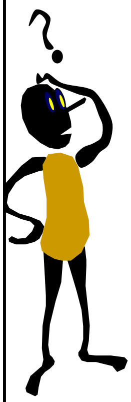
$$a\,x^4 + b\,x^3 + c\,x^2 + d\,x + e$$
$$=$$
$$e + x\,(d + x\,(c + x\,(b + a\,x)$$

Implementation

```
public int search(String T, String P){
    int M = P.length(), N = T.length();

    int dM = 1, h1 = 0, h2 = 0;
    int q = 3355439; /*pick it at random */
    int d = 256;       /* radix */

    for(int j = 1; j < M; j++)   dM = (d*dM) % q;

    for(int j = 0; j < M; j++){
       h1 = (h1*d + P.charAt(j)) % q;
       h2 = (h2*d + T.charAt(j)) % q;
    }
```

Implementation (cont.)

```
    if(h1 == h2) return 0;

    for(int i = M; i < N; i++) {
      h2 = h2 - T.charAt(i - M) * dM % q;
      h2 = (h2*d + T.charAt(i)) % q;
      if(h1 == h2) return  i - M + 1;
    }

    return -1;
}
```



False match

$T == P \bmod q$

What do we do in a case of false match?

When we found a match we can check the match by char comparison.

TRIES = "retrieval"

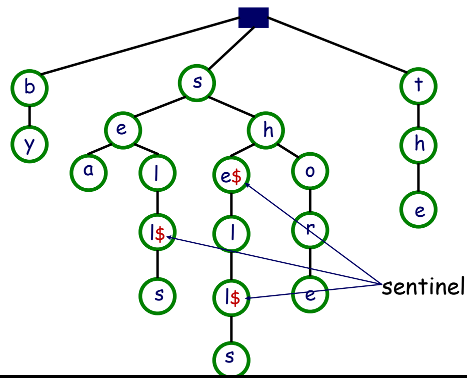
Fredkin (1960)

Main idea: based on the digits of the keys!

TRIES

- Each node (or edge) is labeled with a character
- Children of node are ordered (alphabetically)
- Paths from root to leaves yield all input strings

sells sea shells by the sea shore



Applications

Auto completion
Spell checkers
Data compression
Computational biology
Google's inverted tables

Node Structure

Often wasteful of space because many of the child fields are null.

Possible node representations:

- Array
- Hash Table
- Linked List
- Binary Tree

Search

```
public boolean find (TrieNode node, String key)
{
  if (key.length()==0) return node.isWord();

  char ch = key.getChar(0);
  String rest = key.substring(1);
  TrieNode child = node.getChild(ch);
  if(child == null) return false;
  else
    return find (child, rest);
}
```

Runtime complexity -?

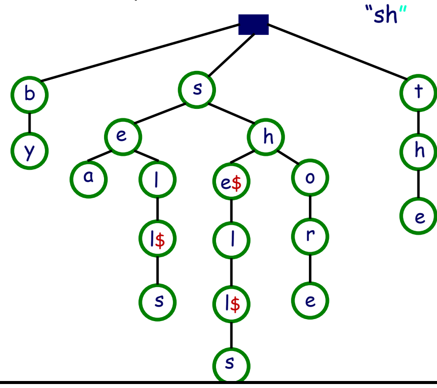
Insert

```
public void insert (TrieNode node, String key)
{
  if (key.length()==0) node.setWord(true);
  char ch = key.getChar(0);
  String rest = key.substring(1);
  TrieNode child = node.getChild(ch);
  if(child == null) {
    node.setChild(new TrieNode(ch), ch);
    insert (newChild, rest);
  }
  else
    insert (child, rest);
}
```

Runtime complexity -?

Prefix Match

Find all string starting with "sh"



Advantages, relative to BST

Search is faster !
It does not depend on the number of elements in the tree.

Trie helps with prefix-matching.

Advantages, relative to hashing

No collisions.

No hash function.

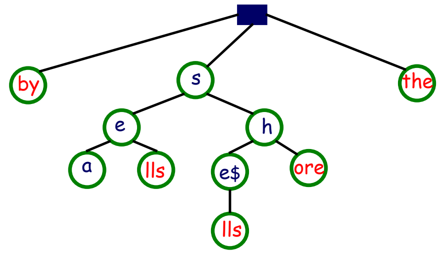
Alphabetical sorting. How?

Compressed Tries

- Each non-leaf node (except root) has at least two children
- Replace a chain of one-child nodes with a single node labeled with a string

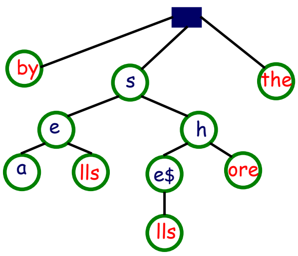
Compressed Tries

sells sea shells by the sea shore



Compact Tries (PATRICIA)

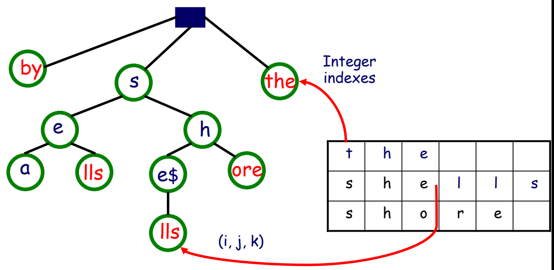
A more compact representation of compressed tries



t	h	e			
s	h	e	l	l	s
s	h	o	r	e	

Compact Tries (PATRICIA)

A more compact representation of compressed tries



t	h	e			
s	h	e	l	l	s
s	h	o	r	e	