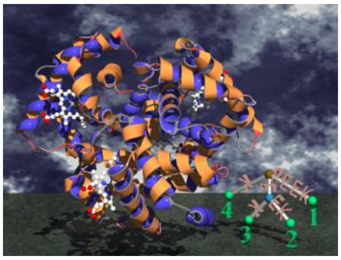
Algorithm Design and Analysis
Victor Adamchik CS 15-451 Spring 2014
Lecture 41 Apr 30, 2014 Carnegie Mellon University

Suffix Trees and Arrays



Purpose – pattern matching

Given a very long text T, preprocess it, so that once a query text P is given, we can efficiently find if P appears in T.

SUFFIX TREES (Weiner ,1973)

Used to search the human genome.

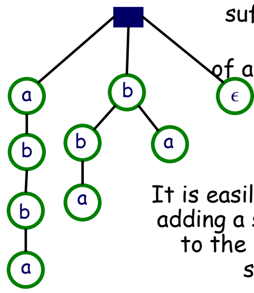
Suffix Tree

A suffix tree of a string s is a compressed trie that stores all <u>suffixes</u> of s.

A compressed trie is a trie in which non-branching paths are stored as single node labeled with a string.

Suffix Tree (uncompressed)

s = abba



suffix can be a prefix of another suffix

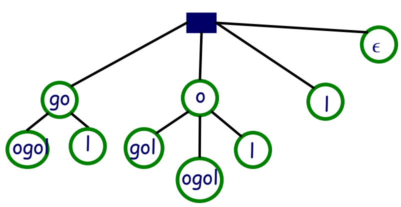
It is easily overcome by adding a special char $ to the end of each suffix.

Suffix Tree

Draw a suffix tree for GOOGOL

Here are all suffixes
ϵ, L, OL, GOL, OGOL, OOGOL, GOOGOL

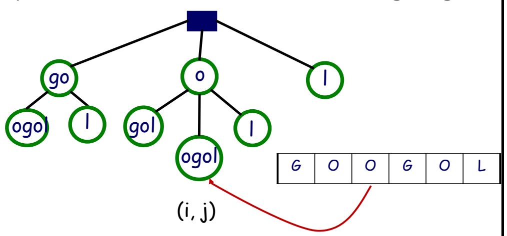
Suffix Tree - GOOGOL



Space complexity- ?
uncompressed vs. compressed...

Space Complexity

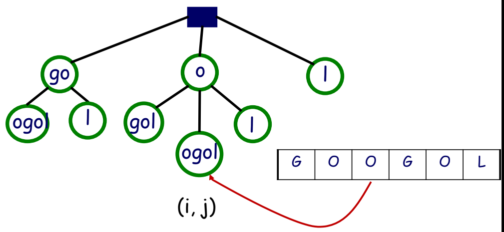
Uncompressed: $O(s^2)$, where s is the string length



To reduce space we store just indices

Space Complexity

Fact: compressed tree requires a linear space



Proof:

1) #_leaves = #_suffixes
2) #_internal nodes < #_suffixes

Searching

Build a suffix tree for a text.

Traverse the tree according to the pattern.

If we did not get stuck traversing the pattern then the pattern occurs in the text.

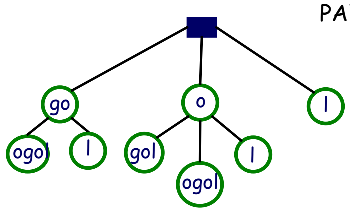
The complexity is the pattern length.

How can we count occurrences of the pattern?

Occurrences of the pattern

TEXT: googol

PATTERN: go



The algorithm returns a subtree with all occurrences of a pattern (just count leaves)

Find the longest substring that appears more than once

We label each node with its depth.

Then we find the internal node with the largest depth.

Longest common substring (of two strings)

Example: ALOHA and HELLO

Construct a new string a&b, where & is a special char.

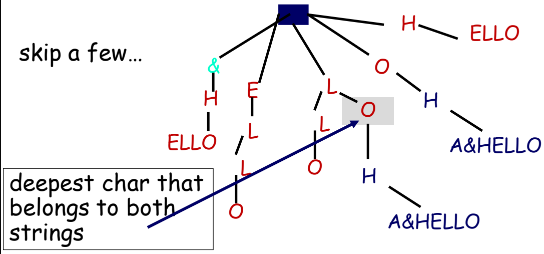
Construct a suffix tree for a&b.

Each leaf repres. a suffix that begins in a or in b.

Find the deepest node that belongs to both strings.

ALOHA&HELLO

skip a few…



So, we can find a longest substring in a linear time

Find the Longest Palindrome

Example: bananas

Construct a suffix tree for a string S_f and its reverse S_r.

For every suffix k in S_f, find the lowest common ancestor with the suffix n - k + 1 in S_r.
Here n is the length of the string.

The path from the root to the LCA is a palindrome.

Building Suffix Trees in O(n) Time

We need some extra preprocessing:
a) Suffix array
b) Longest common prefix array

Suffix array is just the lexicographically sorted array of all its suffixes (indexes)

0	1	2	3	4	5	6
B	A	N	A	N	A	$

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

k-th suffix begins at position k.

6,5,3,1,0,4,2 is a suffix array.

Building Suffix Trees in O(n) Time

For every two adjacent suffixes, we compute the longest common prefix

6	5	3	1	0	4	2
0	*1*	*3*	*0*	*0*	*2*	

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

We add suffixes into a tree in the order they appear in the suffix array.

To add the next suffix into a tree we use the LCP values.

Complexity of building is the same as traversing a tree in preorder, which is linear.

Suffix Arrays

were introduced by Manber and Myers in 1989 (and published in 1993).

They take of a factor 4 less space then suffix trees.

They can be used for searching.
O(P + log T)

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

How would you search for a pattern?

Searching Suffix Arrays

Since suffix array is sorted we can use a binary search…

Let P be a pattern, and A[k] is a suffix array. Compute

$L_p = \mathit{min}\{k \mid P \le A[k] \text{ or } k = n\}$
$R_p = \mathit{max}\{k \mid P \ge A[k] \text{ or } k = -1\}$
as the left/right bounds.
At the start, $L_p=0$ and $R_p=n$.

6	$
5	A$
3	ANA$
1	ANANA$
0	BANANA$
4	NA$
2	NANA$

Pattern matches some A[k] for $k \in [L_p, R_p]$