CS 124 Lecture 24 Monday, April 27, 2020

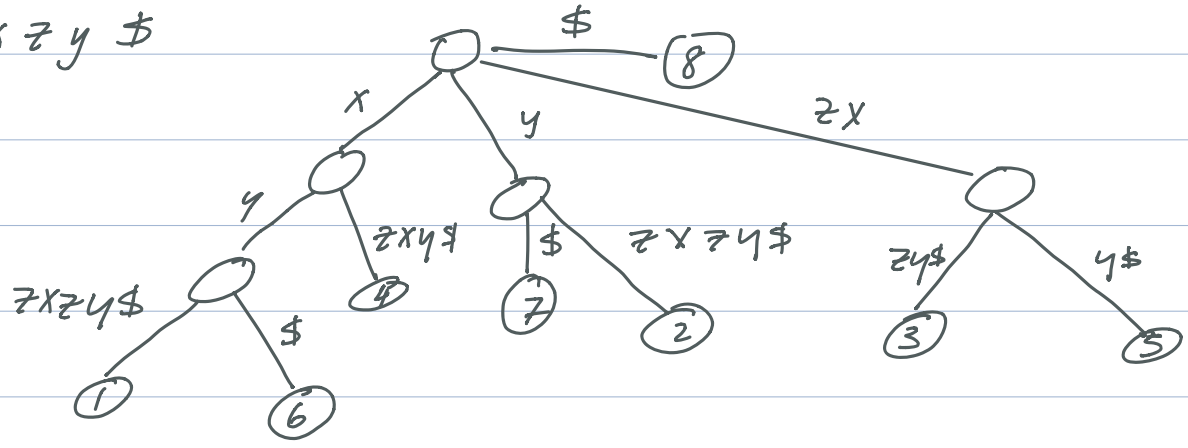
~ Suffix Trees and Palindromes ~

Suffix Trees

- String processing (pattern matching)
- built from a bit string $S[1 \ldots m]$
- rooted, directed, m leaves
- leaves labeled $1 \to m$
- edges labeled

follow path from root → leaf : suffix $S[1 \ldots m]$ on edges

$x\ y\ z\ x\ z\ y\ \$$



$\$ = EOF$

① unique end of string character $

② every internal node has 2 children

Quadratic Alg: build character by character + adjust tree

Linear-time Alg in $O(m)$ time w/ $O(m)$ space

nodes bounded by $m + m/2 + m/4 + \ldots \leq 2m$

$O(m)$ edges → represented by index pair

Pattern Matching on a Suffix Tree

Assume alphabet size, $|\Sigma|$ is constant

$O(1)$ to figure out which edge to follow

To find pattern P: $O(|P| + \#\text{matches})$ ← not in terms of document!

"walk" on edges to find pattern: $O(|P|)$

to find # of matches: DFS, or $O(\#\text{leaves})$

$= O(\#\text{matches})$

To find single match:

"walk" on edges

BFS to find first match / leaf node

Generalized Suffix Tree

$\{S_1, S_2, \ldots S_k\}$ collection of strings

constructed in time $O(\sum_i |S_i|)$

- each leaf node contains pairs of numbers of form

  (pos, string#) ← the suffix corresponds to pos in string #

- each edge may have triple identity

  (start, end, string)

Longest Common Extension

Given S_1, S_2

Given (i, j): longest matching substring starting at i in S_1 and j in S_2

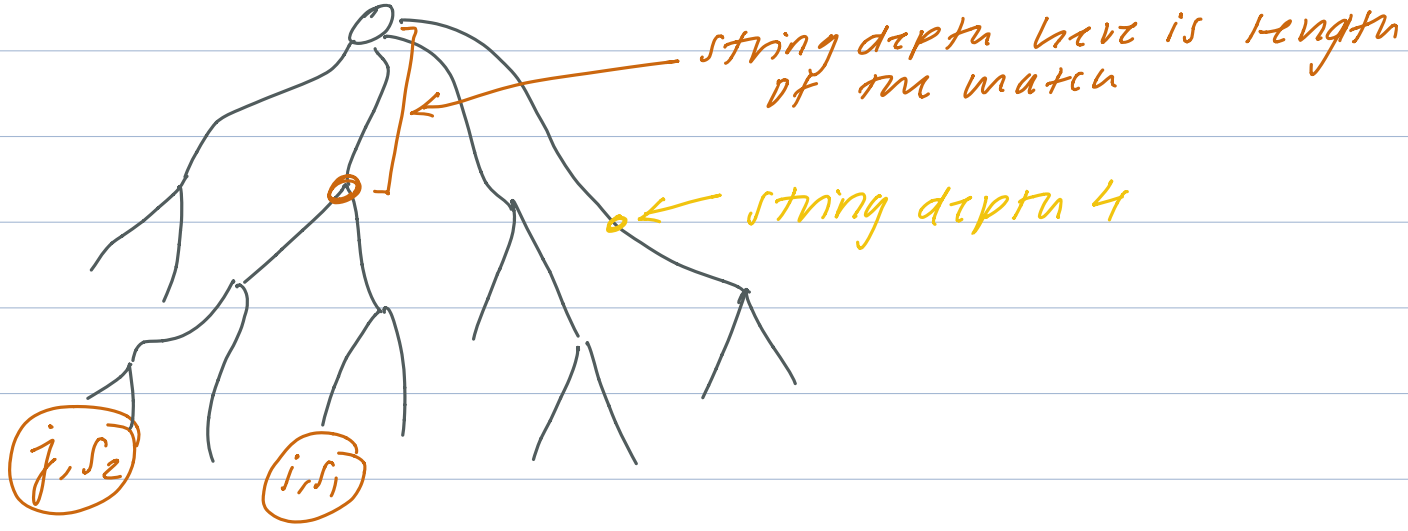
Preprocess: (linear time + space)

Answer queries in $O(1)$

Alg:

1. Put S_1, S_2 in generalized suffix tree $O(n)$
2. Compute string depth of each node $O(n)$

   easy via DFS
3. Make LCA data structure on the generalized suffix tree $O(n)$



Maximal Palindrome

RACECAR

WAS I ERE I SAW

A MAN A PLAN A CANAL PANAMA

MADAM IM ADAM

S = axbccbbbaa

maximal odd length

starting from center

maximal even length

start between

Find all (even-)length palindromes in linear time

S and S^R

max palindrome @ pos X:

S = axbccbbbaa

S^R = aabbbccbxa

longest common extension

LCE (X, n-X)

S S^R

Build a LCE data structure for S, S^R

Ask queries LCE(1, n-1), LCE(2, n-2), etc

↑

each is O(1), make O(n) queries