Lec16-suffix-trees

Monday, October 7, 2024 11:08 PM

Often, we want to search for things other than just numbers
(of course, can encode as numbers, but that may lose structure)
What about strings? (e.g. natural language or genomic strings)

Problem: We are given large, known, & fixed text string (like a genome) and many different unknown & changing query strings.

Can we come up with a good preprocessing data structure?

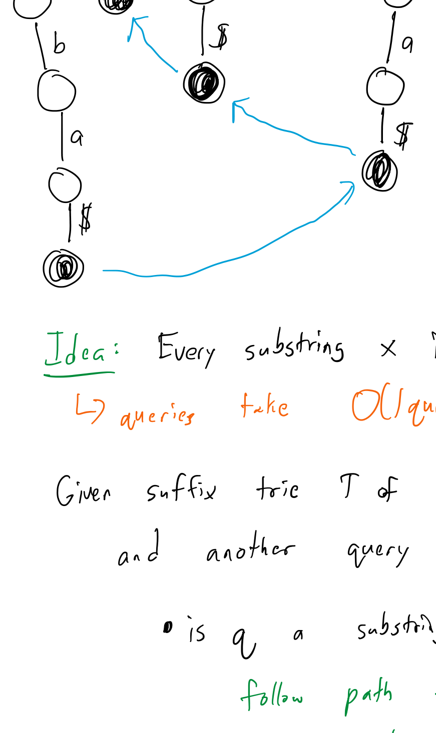
Suffix tries (not yet suffix trees)

- A trie is a tree that exploits internal structure of keys (e.g. strings & their characters)
- not space efficient, but can check substring, suffix, & occurences, longest repeat, lexicographically first suffix, etc.

(later, we'll see space-efficient suffix trees, which are used in genomic search software like Mummer)

- edges are labelled with alphabet (e.g. $\Sigma = \{A, C, G, T\}$)
- every path from root to solid node is a suffix of string, & vice versa.
- nodes can be thought of as the substring corresponding to the path from root to node.

Ex. $s = abaaba\$$



- Special suffix links from $x\alpha$ to α (drawn for all suffixes in blue.)

(one example of suffix link b/t non-suffix nodes abaaba$)

Why are all solid nodes leaves in this example?
(because end-character $ is unique)

How many leaves will there be?
(length of string in this case)

Idea: Every substring x is a prefix of a suffix of the string.
↳ queries take $O(|query|)$ time regardless of $|S|$.

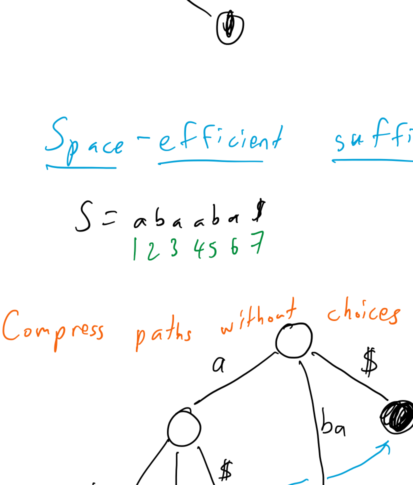
Given suffix trie T of a reference string S with end char, and another query string q, answer:

- is q a substring of S?
  follow path for q from root.
  If q exhausts, $q \in S$. If path stops, $q \notin S$.
- is q a suffix of S?
  follow path for q from root.
  If q exhausts at a leaf, then it is a suffix.
- how many times does q appear in S?
  follow path for q until it exhausts
  Count # leaves under current node.
- what is the largest repeat in S?
  Find deepest node with at least 2 leaves
- what is the lexicographically first suffix?
  Start root, always follow smallest letter.

We didn't use the suffix links in any of the examples above, but it is essential for things like finding the longest common substring between q & S.

Suffix tries are memory inefficient

$S = aaabbb$



$S = a^n b^n$ will have
1 root node
n nodes in a path of b's
n paths of n+1 b nodes

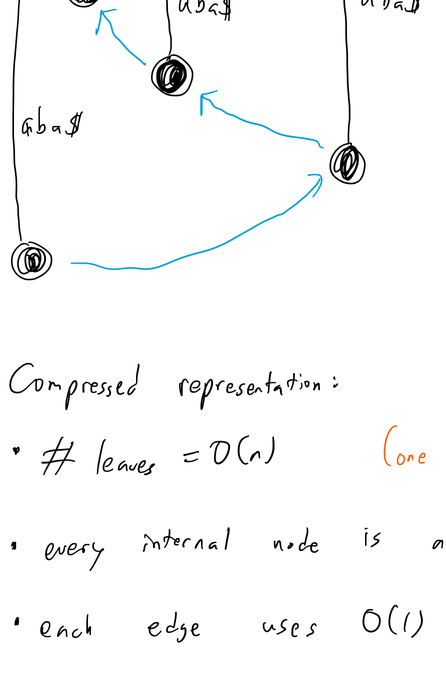
- Total = $n(n+1) + n + 1 = O(n^2)$ nodes

Can we compress it?

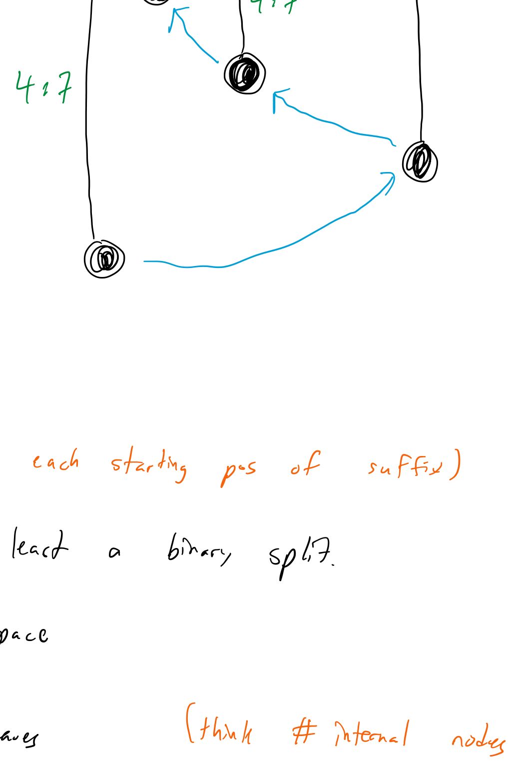
Space-efficient suffix trees

$S = abaaba\$$
1 2 3 4 5 6 7

Compress paths without choices



represent sequence along path by range $[i,j]$ in input string



Compressed representation:

- # leaves $= O(n)$ (one for each starting pos of suffix)
- every internal node is at least a binary split
- each edge uses $O(1)$ space

$\Rightarrow$ # internal nodes $\approx$ # leaves (think # internal nodes of binary tree)

edges = # nodes $= O(n)$

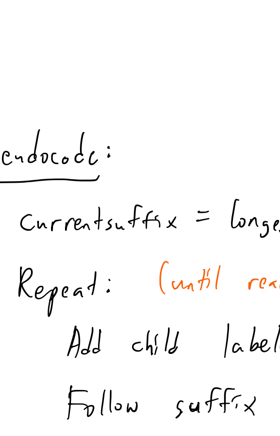
Building suffix tries

$S = abbacabaa$

Walk down string from left to right, building suffix trie for $s[0], s[0..1], s[0...2], ..., s[0...n]$
Can build next trie from previous one

$suffixTrie(s[0...i-1]) \longrightarrow suffixTrie(s[0,...,i])$, add $s[i]$ to all suffixes.

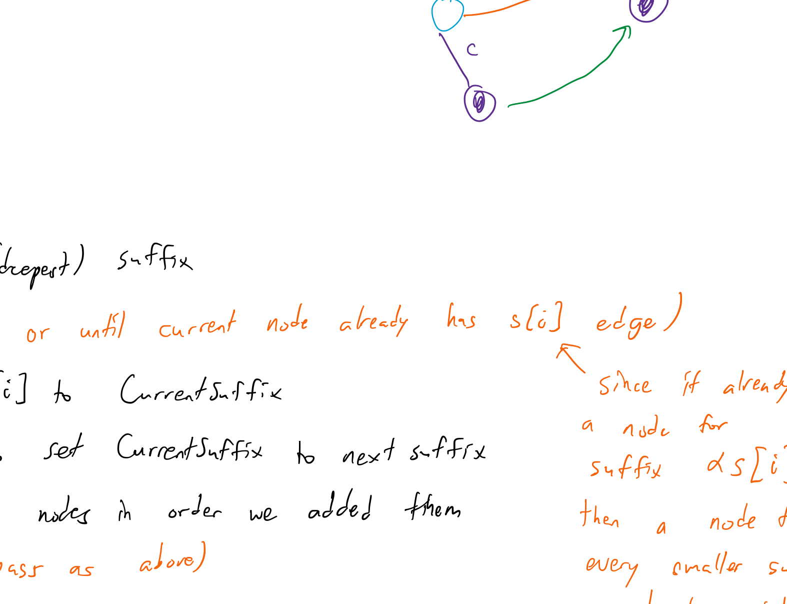
abbacabaa
i=4



already in sufTrie $([0...i-1])$

abbac
bbac
bac
ac
c

add $s[i]$ to old suffixes, incl empty suffix



Pseudocode:

CurrentSuffix = longest (deepest) suffix
Repeat: (until reach root or until current node already has $s[i]$ edge)
Add child labeled $s[i]$ to CurrentSuffix
Follow suffix link to set CurrentSuffix to next suffix
Add suffix links connecting nodes in order we added them
(in practice in same pass as above)

since if already a node for suffix $\alpha s[i]$ then a node for every smaller suffix already exists

Constructing suffix tree (Ukkonen's Alg)

- same idea as with suffix tree
- main idea is not every trie node explicitly represented
- solution: represent trie nodes as pairs (u, α)
  - u ↑ real node in tree
  - α ↑ string leaving node
- Some additional tricks needed for $O(n)$

Summary

- suffix tries natural way to store string, but $O(n^2)$
- suffix trees space optimal $O(n)$, but a little more subtle.
- Can also store sets of strings (next time)