Lec18-suffix-trees

Sunday, October 22, 2023 7:51 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 a long, known, and fixed text string (like a genome) and many different unknown & changing query strings,

Can we come up with a good preprocessing data structure?

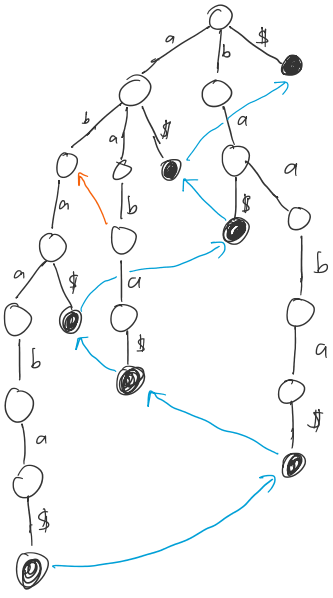
Suffix tries (not yet suffix trees)

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

Ex. $S = abaaba\$$



- special suffix links from $x\alpha$ to α. (only drew ones from suffixes, but all nodes have them) ab aaba$

Why are all solid nodes leaves? (because end-char was unique)

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

Idea: Every substring x is a prefix of a suffix.
↳ 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 number of leaves under current node

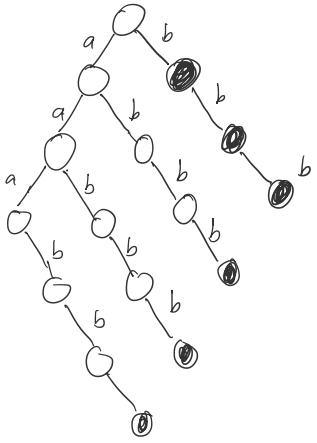
- what is the longest repeat in S?
  Find deepest node w/ at least 2 leaves

- 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 t & s.

Suffix tries are memory inefficient.

$S = aaabbb$



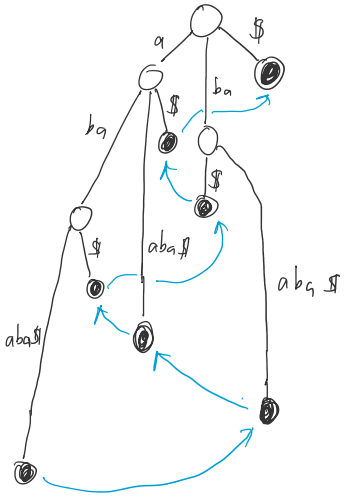
- $S = a^n b^n$ will have
  - 1 root node
  - n nodes in a path of b's
  - n paths of $n+1$ b nodes
- Total $= n(n+1) + n + 1 = O(n^2)$

Can we compress it?

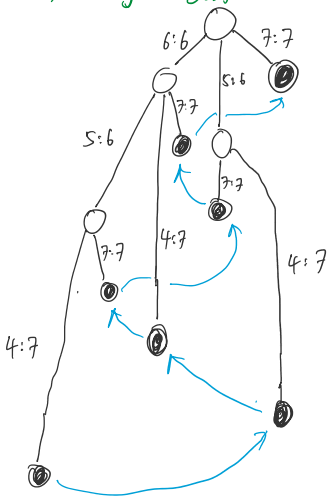
Space-efficient suffix trees

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

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)

Compressed representation:

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

$\Rightarrow$ # internal nodes $\cong$ # leaves (think number of internal nodes in binary tree)

edges = number of 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.

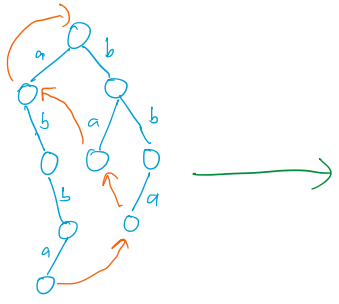
suffixTrie (s[0..i-1]) $\longrightarrow$ suffixTrie (s[0,..,i]), add s[i] to all suffixes already in suffTrie ([0..i-1])

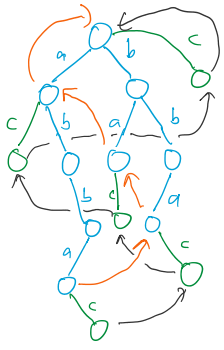
abbacabaa
i=4

abbac
bbac
bac
ac
c

need new suffix links too

or just add s[i] to everything, including empty suffix





Pseudocode:

CurrentSuffix = longest (deepest suffix)

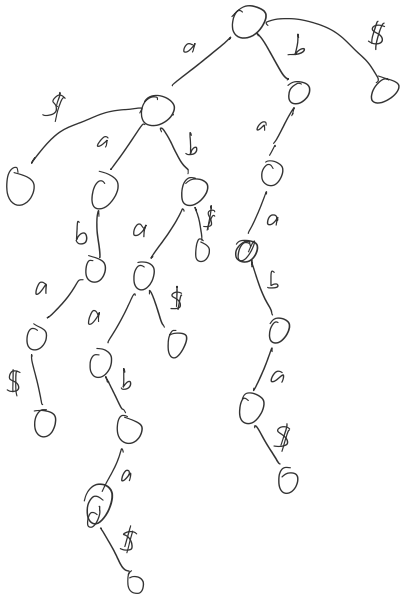
Repeat: (until reach root of if the current node already has s[i] edge)
  Add child labeled s[i] to CurrentSuffix
  Follow suffix link to set CurrentSuffix to next suffix

Add suffix links connecting nodes in order we added them.

In practice, do in same pass as above

Since if already a node for suffix of s[i] then a node for every smaller suffix exists

Example S = abaaba#



Constructing suffix tree (Ukkonen's Algorithm)

- Same idea as with suffix trie
- main idea is not every trie node explicitly represented
- Solution: represent trie nodes as pairs (u, α)
  - u: real node in tree
  - α: string leading to it
- 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)