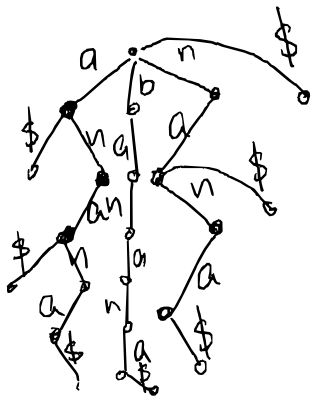
Suffix trees & Suffix arrays

Suffix trie

A trie over alphabet $\bar{\Sigma}$ is a rooted tree with edge labels in $\bar{\Sigma}$. Each node has at most one edge to its child with each symbol $\sigma \in \bar{\Sigma}$. Each path from the root corresponds to a string over $\bar{\Sigma}$.

Given a string $T \in \bar{\Sigma}^n$ over a (small) alphabet $\bar{\Sigma}$ (think of $|\bar{\Sigma}| = O(1)$), the suffix trie is the trie that contains all suffixes of T.

Ex: T = "banana$" w = "ana"



A suffix trie has $O(n^2)$ nodes.

Text searching and counting.

Given a (short) pattern w, suffix trie for T allows for

- determining if w is a substring of T
- count #occurrence of w in T
- find first/last occur ...

in time $O(|w|)$.

Traverse down the tree following the edge corresponding to the symbol of w.

- w is a substring of T iff the node corresponding to w exists.
- #occ = #leaves in the subtree. (can preprocess)

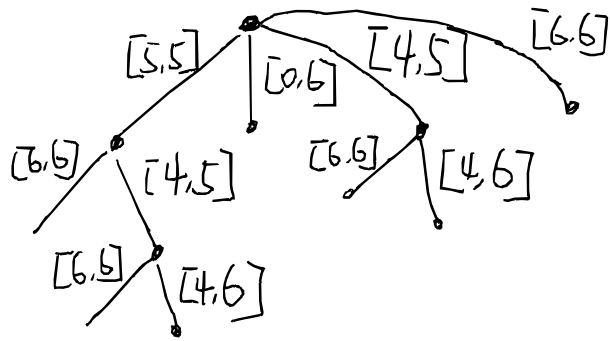
Suffix tree

Compressed version of the suffix trie.

- suffix trie has exactly n leaves, hence, $\leq n-1$ branching nodes (deg ≥ 2)
- from any branching node to a descendant that is also branching, the path must correspond to a substring of T.
- suffix tree compresses such non-branching path into a single edge, "labeled" with the indices of the substring.

$T =$ "banana$"

0 1 2 3 4 5 6



Suffix trees can be constructed suffix arrays + LCP array in $O(n)$ time.

Suffix array (SA)

T = "banana$"
0123456

sort all suffixes in lexicographic order

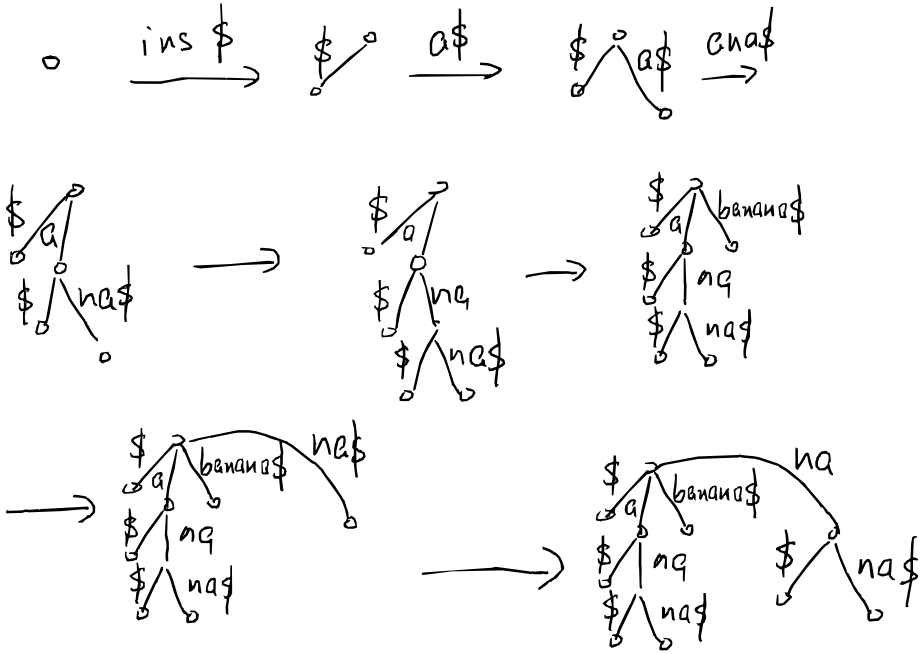
6: $
5: a$
3: ana$
1: anana$
0: banana$
4: na$
2: nana$

SA: 6 5 3 1 0 4 2
LCP: 0 1 3 0 0 2

LCP array (longest common prefix): store the length of the longest common prefix of T[SA[i]] and T[SA[i-1]].

Inserting suffixes in the order of SA = traversing through the suffix tree

- insert the suffixes, using compressed representation of the nonbranching paths
- break a path if it branches
  the depth of the branch is exactly LCP[i]



Construction of SA

This can be done in $O(n)$ time.

We cover an $O(n \log n)$ time algo.

The algo runs in $O(\log n)$ iterations.

- goal of iteration k is to sort all suffixes but only "compare" up to the first 2^k symbols.

  break ties arbitrarily.

group all 2^k-substrings into equiv classes

[compute for each suffix the rank of its first 2^k symbols among all distinct 2^k-substrings

Iteration 0:

sort $i=0,\dots,n-1$ by $T[i]$.

$O(n+|\Sigma|)$ time by bucket sort / counting sort

compute the rank of $T[i]$ among all distinct symbols

Iteration k,

each suffix can be viewed as two giant symbols

$T[i,\dots,i+2^k-1]$

$= \underbrace{T[i,\dots,i+2^{k-1}-1]}_{2^{k-1}} \underbrace{T[i+2^{k-1},\dots,i+2^k-1]}_{2^{k-1}}$

we have sorted all first symbols
- as the first 2^{k-1} symbols of $T[i,\dots,n]$

have sorted all second symbols
- as the first 2^{k-1} symbols of $T[i+2^{k-1},\dots,n]$

both mapped to a number $\le n$ by the "rank"

Then we can apply radix sort in $O(n)$ time.

Total time is $O(n \log n)$.