String matching

String: sequence of characters from an alphabet Σ.

assume $|\Sigma| = O(1)$, constant length.

problems of interest:

1. given a string T and pattern P, find all occurrences of P in T.

   eg. T = banana. P = an. anaa

   an → 2, anaa → 0

   ideally. $O(t+p)$ runtime, $t = |T|$, $p = |P|$

2. given a fixed T, an incoming series of queries for diff. patterns P in T.

   ideally.

   - preprocess T — $O(t)$
   - Search for pattern P — $O(p+k)$ (multi.) ↳ # occurrence

3. given two strings S and T, find the longest substring occuring in both.

   ideally. $O(t+s)$.

Trie "retrieval"

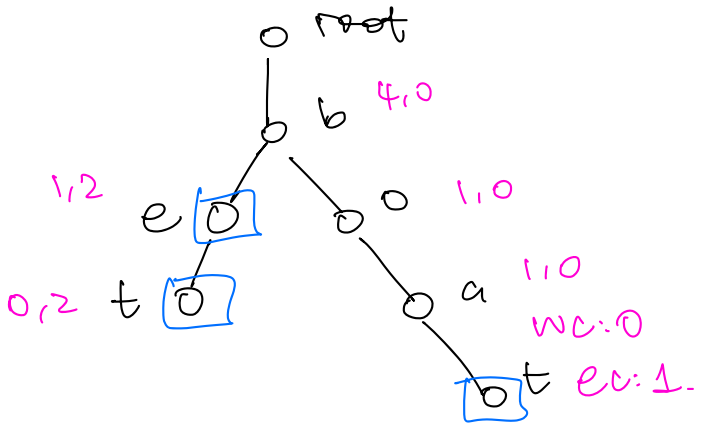
A rooted tree for string storage and retrieval.

branching factor = $|\Sigma|$ · alphabet size

Eg. Σ = English alphabet

strings : be. bet, boat, be

be$ bet$ boat$ be$



Q: "boat"? ✓

"bet"? ✓

"be"? not sure

⇒ Stop ✓

Q: how many "be"s ? ✓

Q: how many strings share (prefix) pattern "be" ? ✓

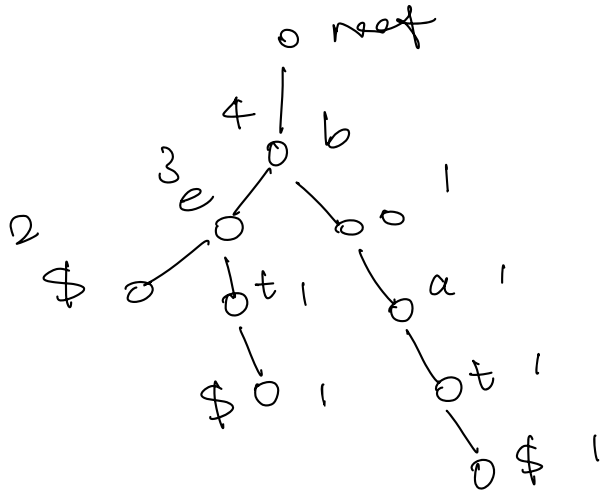
⇒ count

vertex v:

wordcount: # "passing through" v (not including ending)

endcount: # ending at v.

Clearer: end-of-string character. $\notin \Sigma$. conventionally \$



why trie?

insert(T): $O(t)$

lookup(T): $O(t)$

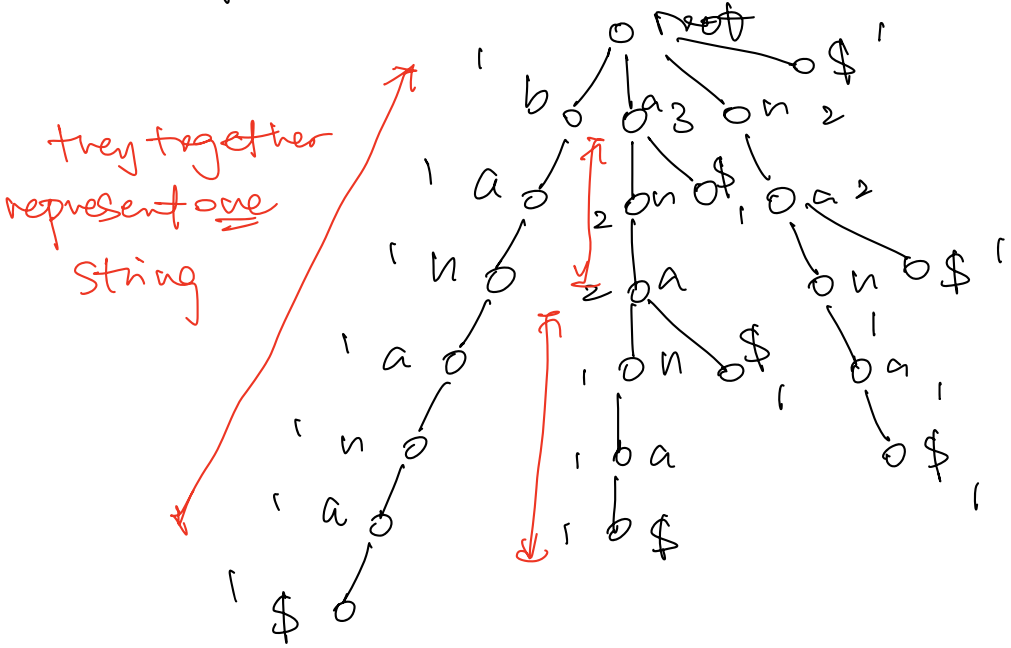
pattern: prefix search. $O(p)$

Suffix Tree

$t = \underline{banana\$}$

→ banana$. anana$. nana$. ana$. na$. a$. $.

create trie:



Q: how many occurrence of a? 3

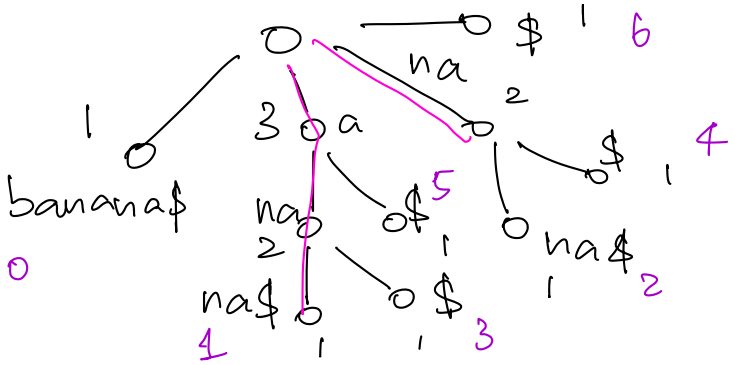
na? 2

ana? 2

How large is the tree in count? $n = 22$

But t is only 7. too much space!

⇒ reduce space by compressing by count on each node.



Q: # pattern anana ? 1 # na?

end with na? ana? 1

↳ na$? 1

Now, how big is this tree?

notice:

1. all leaf nodes represent exactly one suffix string.
   and are not shared / repeats

   ⇒ #leaf nodes = t

2. all internal nodes are shared / repeats and have at least 2 children

   ⇒ #non-leaf nodes ≤ t

   Suppose i non-leaf nodes.

   #edges

   tree with t+i nodes: = t+i-1

   non-leaf (non-root) has ≥ 2 children: ≥ 2(i-1)+1

   = i ≤ t

nodes ≤ 2t. Much more efficient!

Q: given a string T and its suffix tree

1. count # occurrence of a given P in T.
   traverse P down the tree. O(P)

2. output the occurrence of all occurrences of P.
   at leaf, record the start index of the suffix.

$O(p+k)$ → traversing the subtree below it for start indices
↓
find the pattern in tree

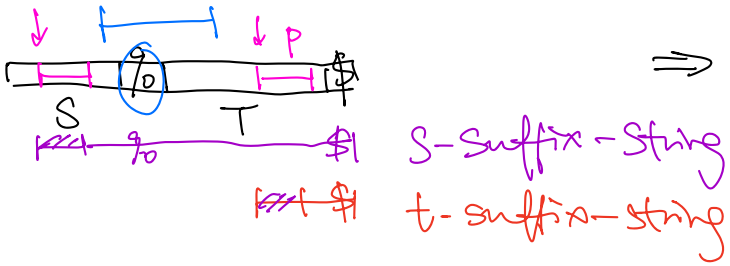
3. find lexicographically first suffix:
   go along the "left" tree until found a leaf

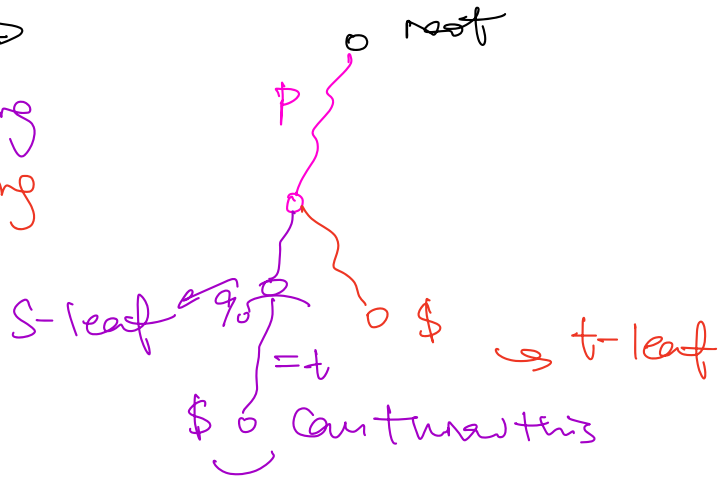
4. find the longest repeat in T:
   find the deepest inner node (all mers are repeats)

Q:
Longest Common Substring

given two strings S, T.

construct $U = S\ \$_0\ T\ \$$. then build a suffix tree for U.





why not S T ? $0
crossing

Now, the question is:

How to construct Suffix tree?

naive Solution: create the "complete" tree then Compress.

more efficient solution: Suffix array.

b a n a n a $

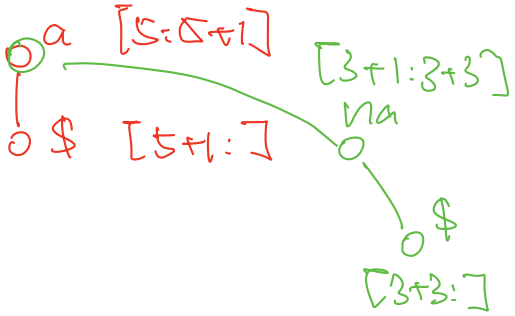
0 1 2 3 4 5 6

1. ~~sorted~~ Suffix strings ($ < $\sum$)

length of common prefix		idx	Suffix
0	<	6	$ → shared
		5	a $ → not shared
1	<	3	a n a $ → not shared
3	<	1	a n a n a $
0	<	0	b a n a n a $
0	<	4	n a $
2	<	2	n a n a $

shared with prev.

shared with next



2. add nodes in order