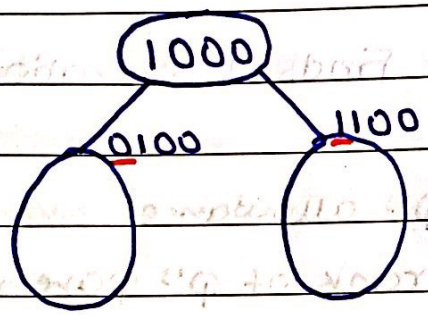
* Collection of Strings:
→ Search
→ Insert
→ Delete

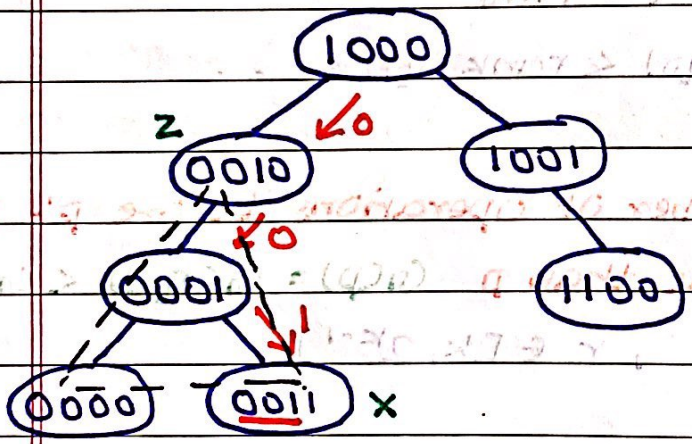
① Digital Search Tree:



Level 0

0th bit is zero

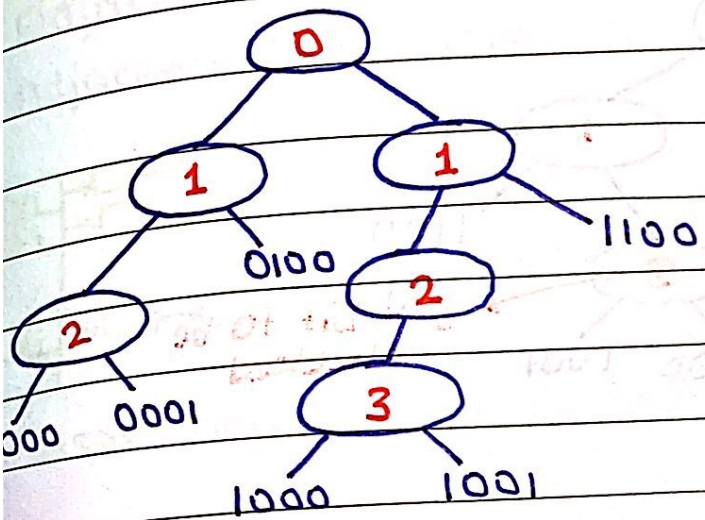
0th bit is 1



* Insert 0011

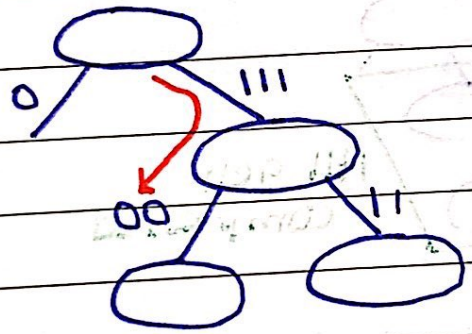
* Delete z, x goes to z's place
↓
leaf in a subtree

Binary Trie: retrieval



- : bit no
if 0 go left
if 1 go right

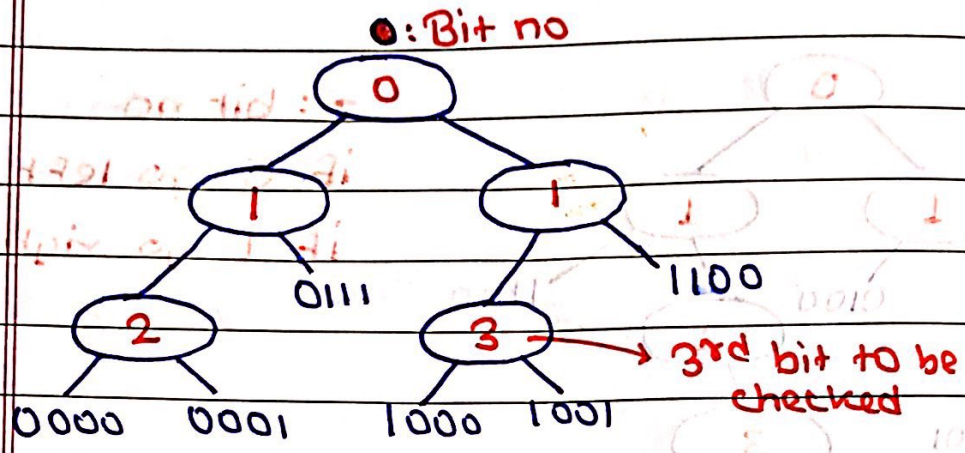
→ Variation



11100

→ Prefix : 000$
00011$

③ Compressed Binary Tree:



★ Space is getting saved:

→ Every internal node has 2 children

★ n: leaves ⇒ n-1 internal nodes

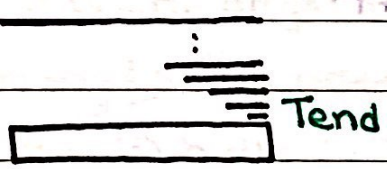
G.P

* Trie:
  Insert
  Search
  Delete

* Given a Text T & pattern P check whether P occurs in T.
* If P ∈ T
  then P is a prefix of some suffix of T

T: aababcb
P: abc

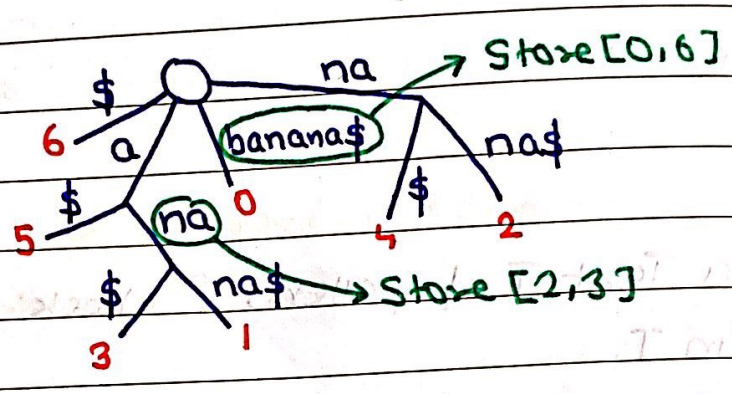
∀i Ti



* Store suffixes in compressed multiweight trie

0 1 2 3 4 5 6

* banana$



* 7 leaves : stored in lexographic order
* Suffix Tree :
* If indices stored in edges radix tree

→ r: maximum size (of alphabet)

|T| : no. of leaves

$1 + r + r^2 + \ldots + r^{n-1}$

$\frac{r^n - 1}{r - 1}$

* No. of internal nodes $\frac{|T| - 1}{r - 1}$

* Total no. of nodes $= |T| + \frac{|T| - 1}{r - 1} = O(|T|)$

classmate
Date
Page

* Space: O(|T|r)
* querytime: O(|P|logr) → Binary search (# sorted array)
* Preprocessing Time:

→ each node has an array of size r

* Instead store the possible alphabets in a BST for each node

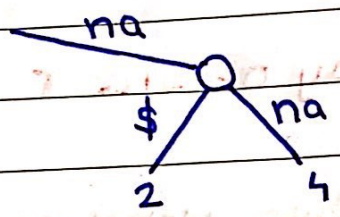
* 4 possible alphabets stored as BST in node v
* Space : O(|T|) (# no. of edges |T| - 1)
* Querytime: O(|P| log (no. of worst case branches))
* Preprocessing Time: To build suffix tree
  By building BST

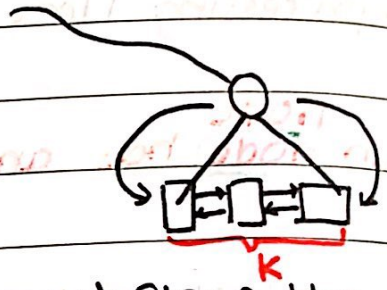
* Various applications of suffix tree

① P is present or not

② Find all occurances of P

To find na





* Store the pointers to leaves of the node

Build: O(T)

Find all occurances : O(|P| + k) time

* Connect all the leaves by doubly linked list in O(|T|) # no. of leaves

③ Find all texts $T_1, T_2, \ldots, T_k$ which contain pattern P

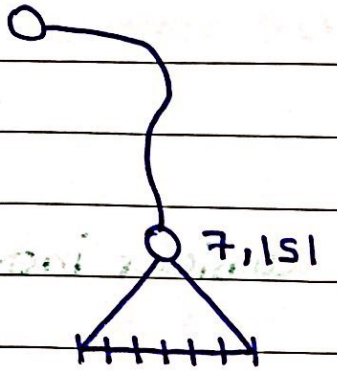
$T_1\$T_2\$T_3 \ldots \$T_k\#$

Build a suffix tree

$O(|T_1|+|T_2|+\ldots+|T_k|)$

* Whenever we encounter $ Prune the subsequent part

④ Longest substring of T
that appears at least $m > 1$ times



* Store no. of children & length of string as a pair as a satelite data of each node

⑤ Longest common substring of two strings T_1, T_2
$T_1\$T_2\#$

→ check one string from T_1 & other from T_2 (# T_1 & T_2 have diff delimiters)

* Suffix array

→ Preprocessing space is upperbounded in Preprocessing time

0 1 2 3 4 5 6
b a n a n a $

* To construct suffix array consider inorder traversal of suffix tree.

6 5 3 1 0 4 2 → Space O(|T|)
Lexographical order

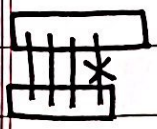
* Once suffix array is ready destroy suffix tree

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

We need log|T| to binary search
To compare we need |P|
Hence to check P ∈ T
O(|P|log|T|)

* LCP array

→ Longest common prefix

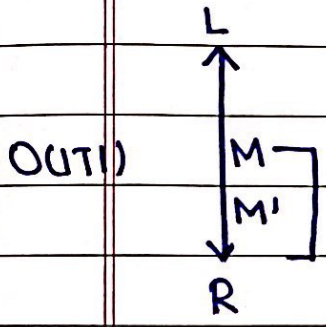


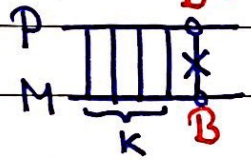
Common prefix of length 3

Suffix array: 6 5 3 1 0 4 2

0 1 3 0 0 2 : LCP array

Preprocessing time : $O(|T|^2)$



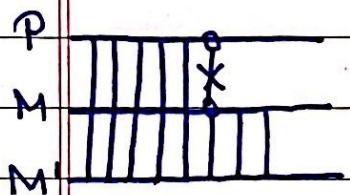


D → go to M to R (# lexographic)
else L to M

LCP(P, M) = k

if k = |P| then P is found

IF k < |P|



LCP(M, M') = k'

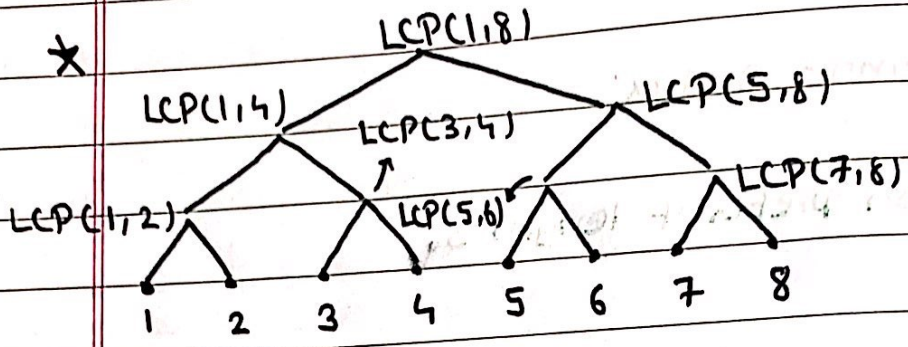
If k' > k

(k+1)th chrs of M' & P donot match

If k' = k

→ Compare from k+1

Hence : $O(|P| + \log|T|)$



* $LCP(a,b) = \min(LCP(a,c), LCP(c,b))$ $c \in [a,b]$
* LCP(a,b) can be calculated in O(log T)

Seg Tree