Text Retrieval (uncompacted trie)

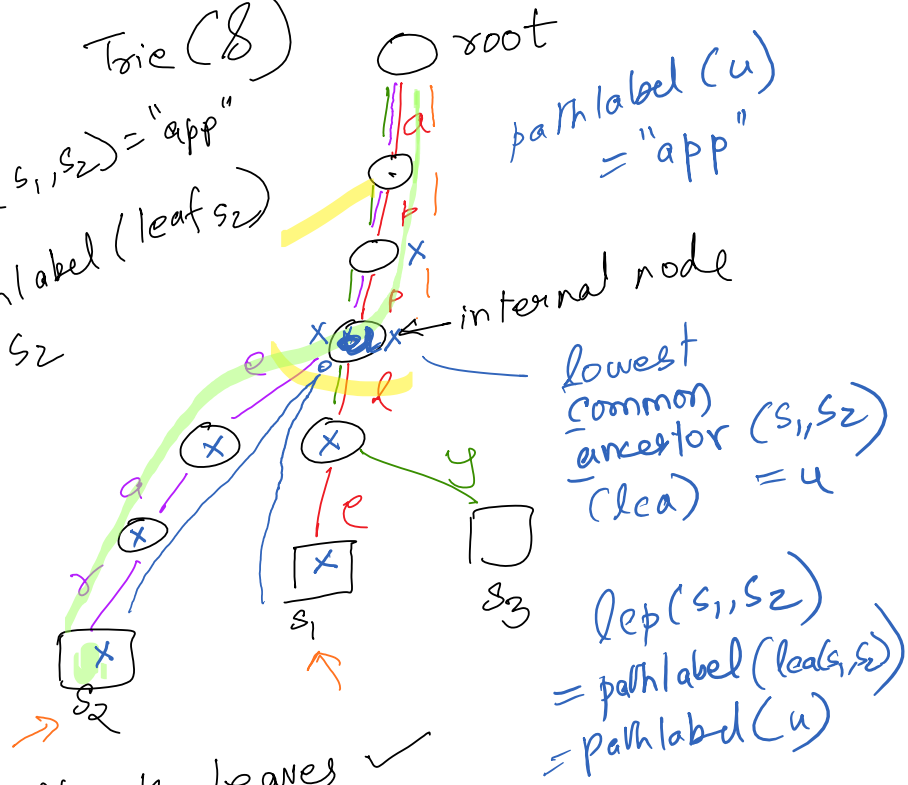
Tries

Friday, February 26, 2021 10:50 AM

Motivation: To capture variable-length matches (directly).

$S = \{s_1, s_2, s_3, \ldots s_k\}$ Trie(S)

	1	2	3	4	5	6
s_1:	a	p	p	l	e	
s_2:	a	p	p	e	a	r
s_3:	a	p	p	l	y	
s_4:	a	p	p			



lcp(s_1, s_2) = "app"

pathlabel(leaf s_2) = s_2

pathlabel(u) = "app"

lowest common ancestor (s_1, s_2) (lca) = u

lcp(s_1, s_2) = pathlabel(leaf s_2) = pathlabel(u)

Definition:

Given $S = \{s_1, \ldots s_k\}$ from alphabet Σ, the trie for S is:

a) rooted tree with exactly k leaves ✓

b) each edge is labeled with a (single) character.

c) no two outgoing edges out of an internal node can start with the same character label

d) the path label from the root to any leaf should spell out one of the input strings.

→ For a compacted trie, this rule is relaxed so edge labels can be strings (of length ≥ 1).

Appending $ symbol to make the leaves unique

Monday, March 1, 2021 11:14 AM

$\Sigma = \{a, c, g, t\}$

Add a new symbol $\$ \notin \Sigma$

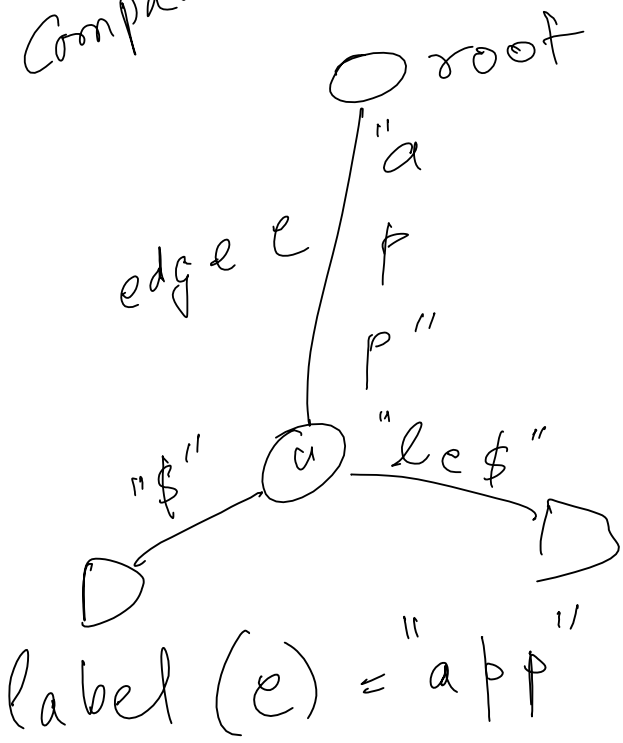
Append $ to all strings in the input set S.

$S = \{s_1\$, s_2\$, s_3\$, \ldots, s_k\$\}$

$s_1 =$ apple$

$s_2 =$ app$

Compacted trie



Trie

compact

to

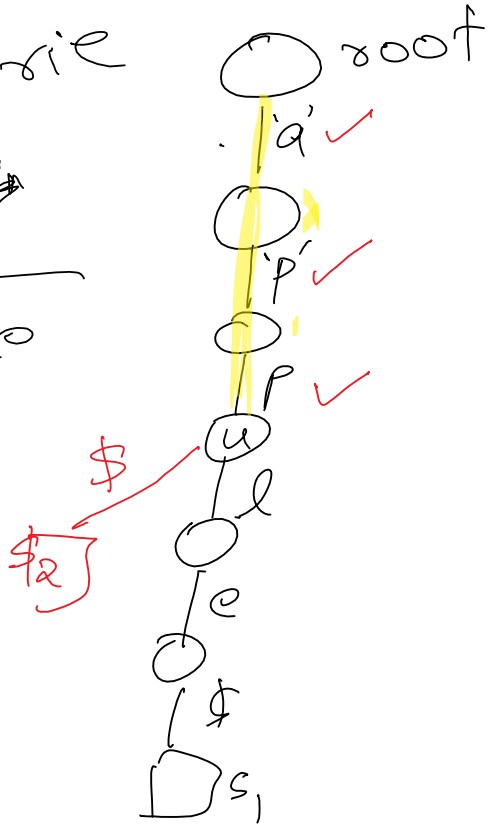
edges

eliminate

int. nodes

with only

one child



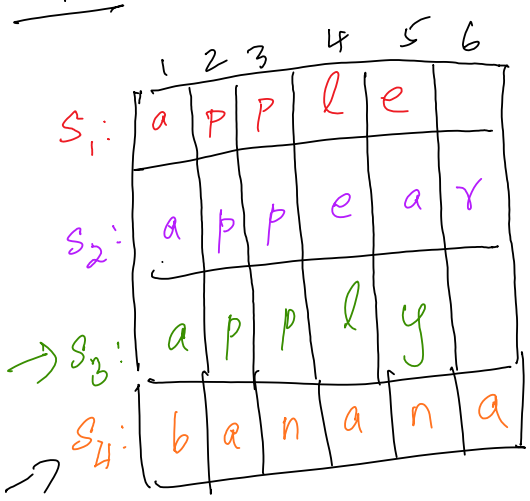
PATRICIA trees

Friday, February 26, 2021 10:50 AM

Practical Algorithms for Text Retrieval for Info. Coded in Alphanumeric.

PATRICIA tree: = the compacted trie for a set of strings S.

Input: $S = \{s_1, s_2, \ldots s_k\}$

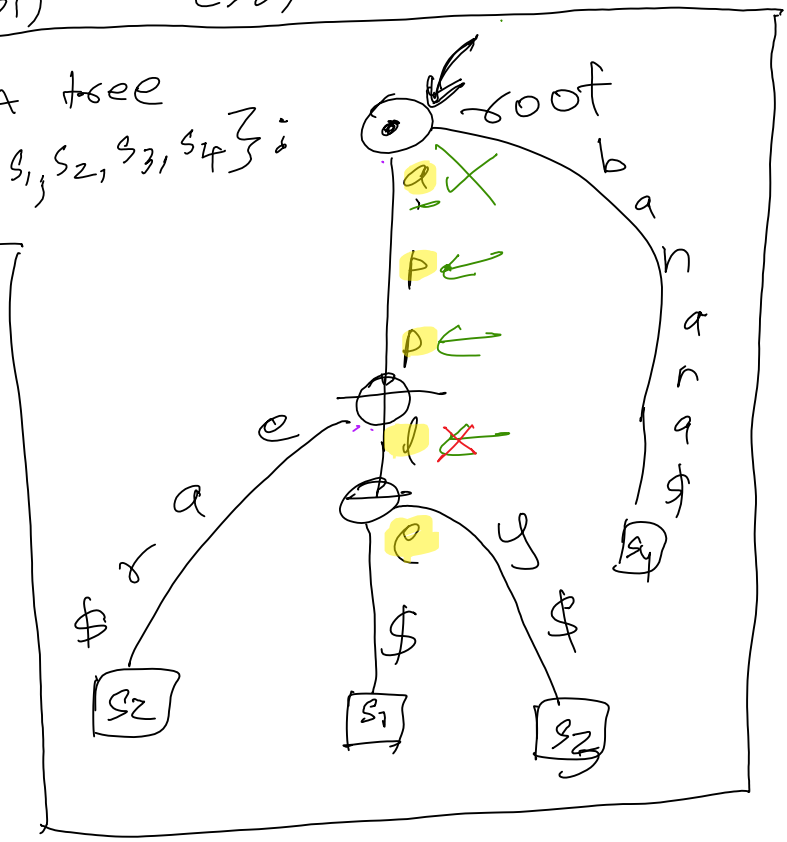


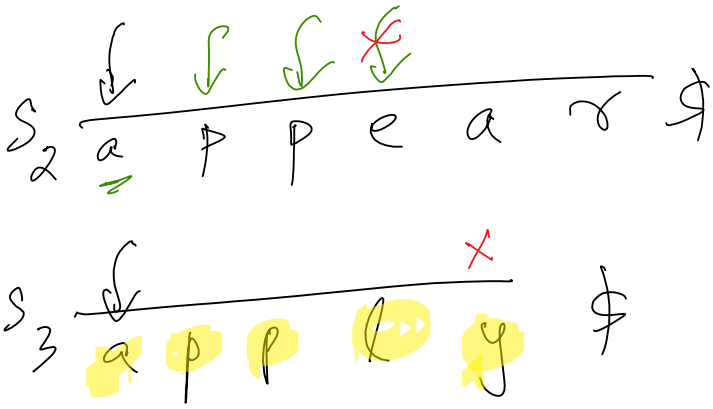
Construction: // insert one string at a time

$T_0 \Rightarrow T_1 \Rightarrow T_2 \Rightarrow T_3 \Rightarrow T_4$

T_0 —insert(s_1)→ T_1 —insert(s_2)→

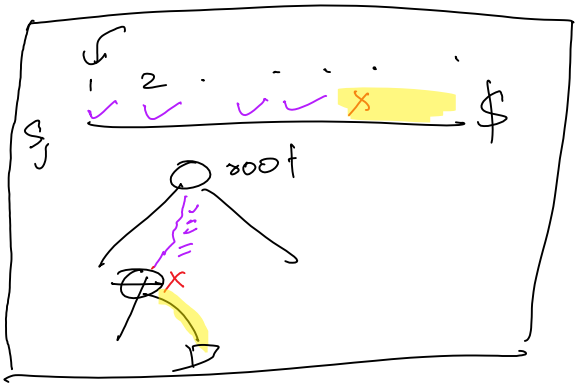
PATRICIA tree for $\{s_1, s_2, s_3, s_4\}$:





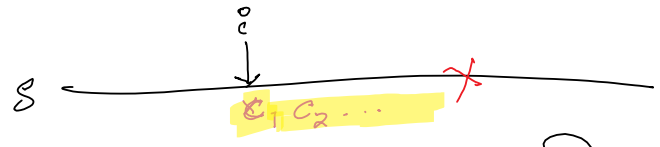
Insert of s_j into T

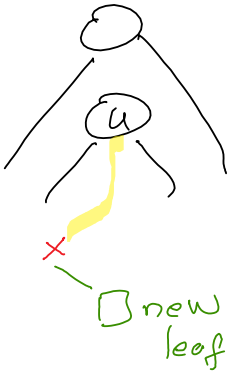
↳ Find Path (root, $<s_j, 1>$)



FindPath function

Monday, March 1, 2021 10:50 AM





FindPath (Tree node * u, <string s, index i>)

//Goal: to insert suffix S[i...] under node u.

{

0) Initialize: $v \leftarrow u$, $x \leftarrow S[i..end]$

1) REPEAT {

a) Find the child branch that starts with x[1]

b) if no such branch exists {

- insert new leaf for s under u
- return.

}

c) Compare characters of the edge label against x[1...] until first mismatch or edge label exhausted

d) if mismatch {

- break edge
- create a new internal node
- create new leaf for s under that node
- return

}

else { $v \leftarrow$ next internal node

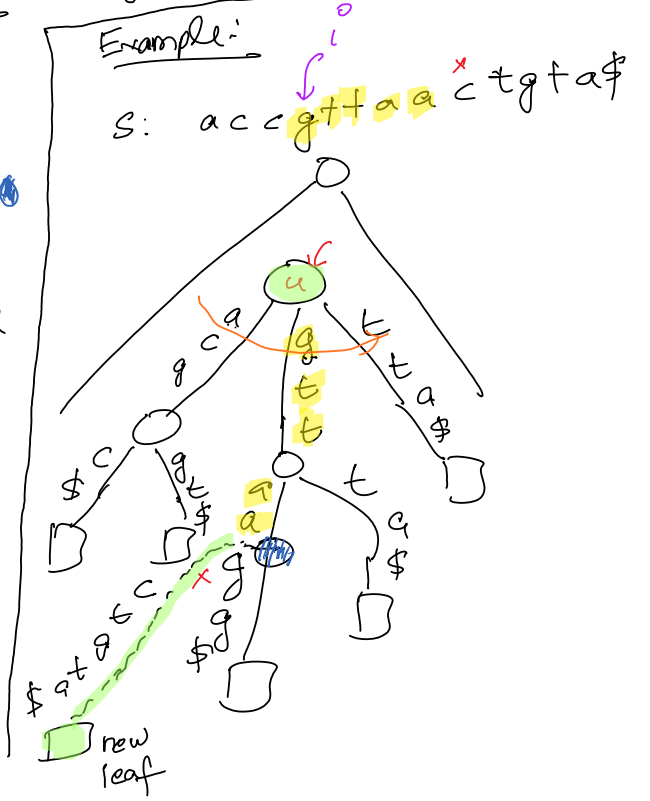
$x \leftarrow x[i+r...]$

}

UNTIL(leaf inserted)

}

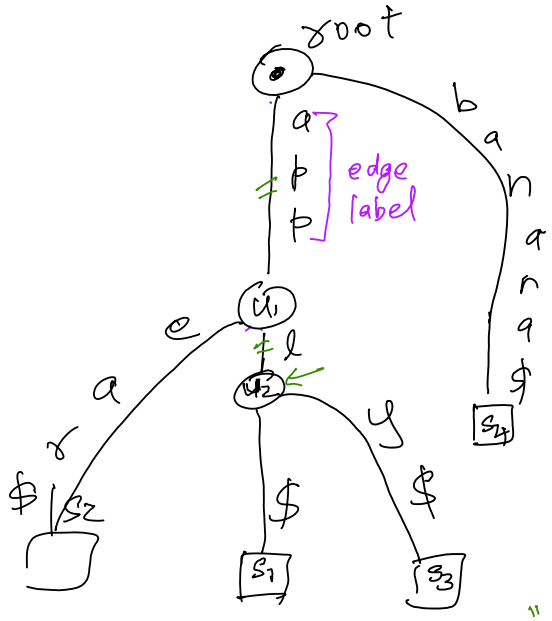
Example:



Trie Definitions

Wednesday, March 3, 2021 10:42 AM

string-depth(u_2) = 4
node-depth(u_2) = 2



path label(u_2) = "appl".

Node: either internal node or leaf

Internal node: node with ≥ 2 children

Leaf: node with $\emptyset$ children

Root: an internal node (special case w/ 0 or more children)

Edge-label: string label of an edge

path-label(u): The concatenation of all edge labels from the root to u.

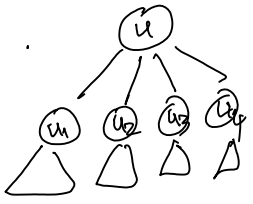
String-depth(u): Length of path-label(u)

Node-depth(u): Number of edges between the root and u.

Parent(u): The parent node of node u (if it exists) (root has no parent)

Ancestor(u): Any node on the path from u to the root.

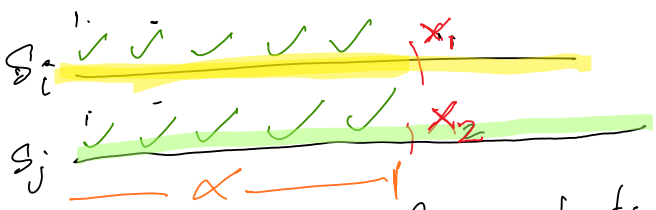
child(u): Any (immidiate) child node of u.



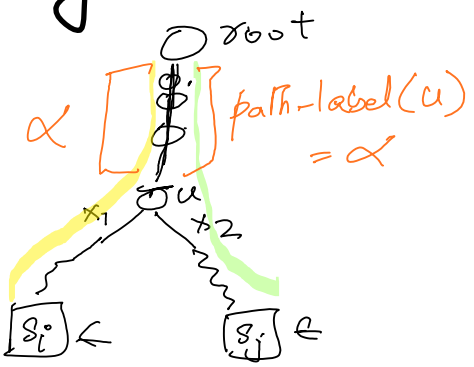
LCP vs. LCA

Friday, February 26, 2021 10:50 AM

Q) What is the longest common prefix (lcp) between any two strings? $\langle s_i, s_j \rangle$



$x_1 \neq x_2$



Let α be the lcp between s_i & s_j.

⇒ Then there has to exist an internal node u, with path-label α under which, leaf(s_i) and leaf(s_j) are present in two different branches (as shown)

⇒ In other words, look for the **lowest common ancestor (lca)** of the leaf(s_i) and leaf(s_j) and output its path-label.

Input: $S = \{s_1, s_2, \ldots s_i \ldots s_j \ldots s_k\}$

Algorithm:

1) Build the PATRICIA tree for S // one time preprocessing
2) Query time: // querying for lots of lcp by user
   a) user provides $\langle s_i, s_j \rangle$ pair
   b) Compute lca (leaf(s_i), leaf(s_j)) → node u
   c) Output path-label (u).

Longest Common Prefix ==> Lowest Common Ancestor

Monday, February 24, 2020 10:53 AM

LCP problem:

Input: $S = \{S_1, S_2, \ldots S_k\}$

Output: lcp(S_i, S_j) query

Algo: → preprocessing step

1. Build PATRICIA(S) ✓

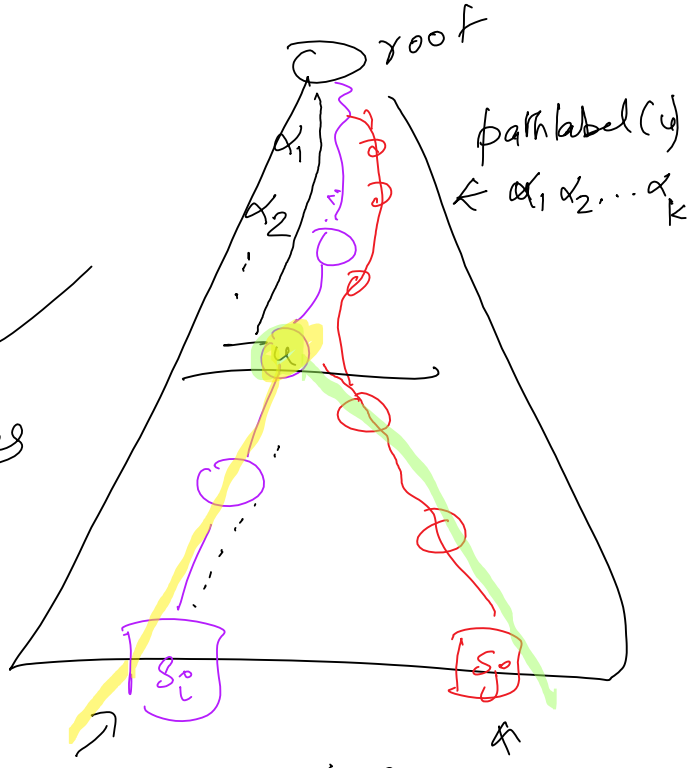
2. locate S_i & S_j leaves

color S_i ← purple

color S_j ← red

color all ancestors of S_i w/ purple

" " " S_j w/ red



Time = O(m+n)

3. lca(S_i, S_j) ← lowest/deepest node which has both colors ("mixed node")

Let $u \leftarrow$ lca(S_i, S_j)

4. Output pathlabel(u).

PATRICIA : $S = \{s_1, s_2 \dots s_k\}$

$\$ \notin \Sigma$

Trie properties & implementation

Monday, March 1, 2021 11:45 AM

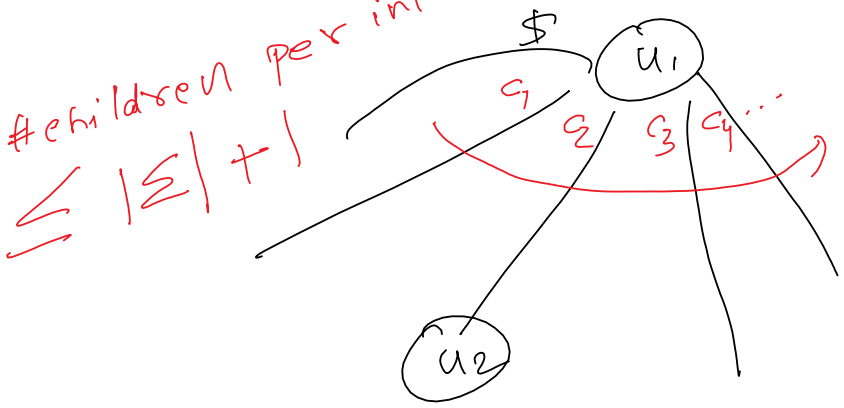
compacted trie (or PATRICIA tree) is a $(|\Sigma|+1)$-way tree.

Tree: root
internal nodes
leaves
edges

edge labels (strings over Σ)

$\Sigma = \{a, c, g, t\}$

#children per internal node $\leq |\Sigma| + 1$

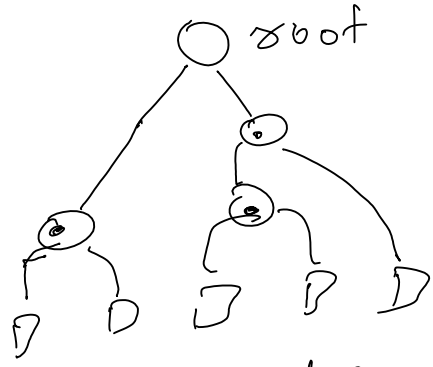


1) 1 root (special case of an internal node)
2) Number of leaves = k
3) Number of internal nodes

$1 \leq$ #int. nodes (incl. root) $\leq k - 1$

binary tree

5 leaves

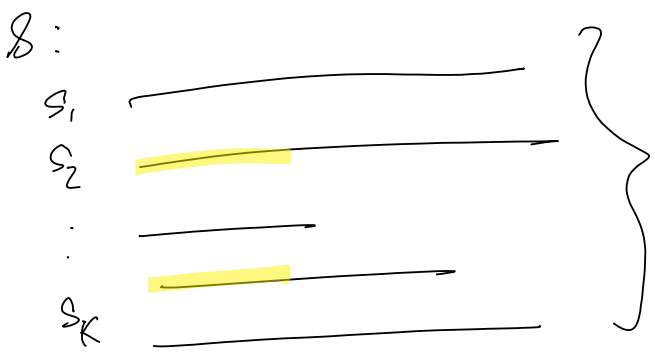


4 internal nodes

Exact matching use-cases: LCP, LCS (and a reason to build a trie for suffixes)

Wednesday, March 3, 2021 10:55 AM

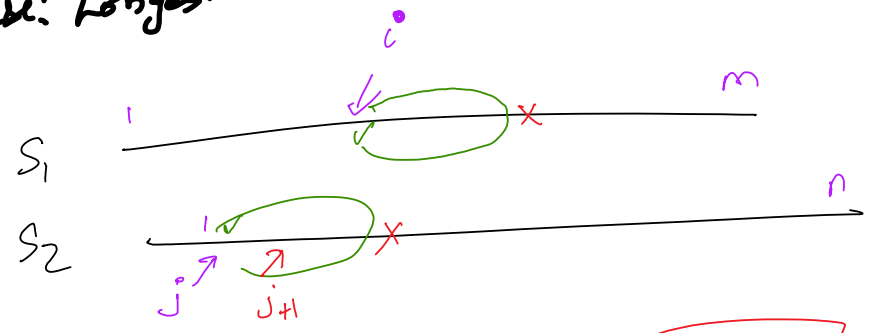
Usecase: Longest common Prefix (LCP)



$lcp(S_i, S_j) = ?$

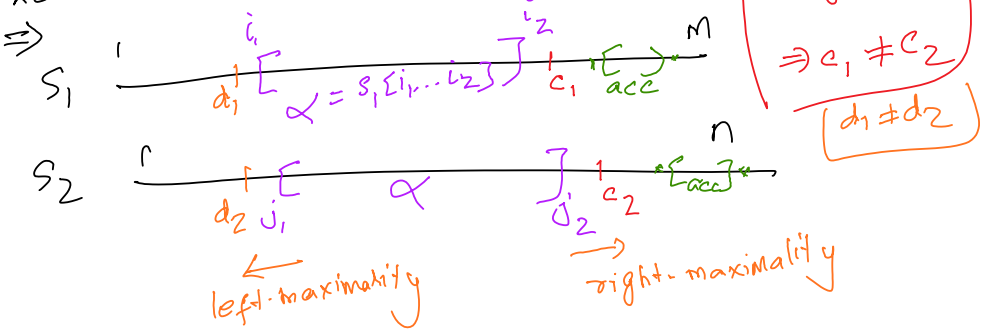
1) Build PATRICIA tree
2) $lca(S_i, S_j) \rightarrow u$
output pathlabel(u)

Usecase: Longest Common Substring (LCS)



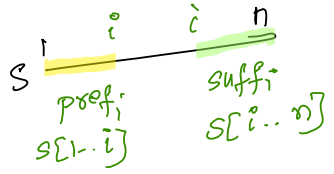
Challenge:
But PATRICIA tree S can only help capture matches that are prefixes.

Let $\alpha = lcs(S_1, S_2)$

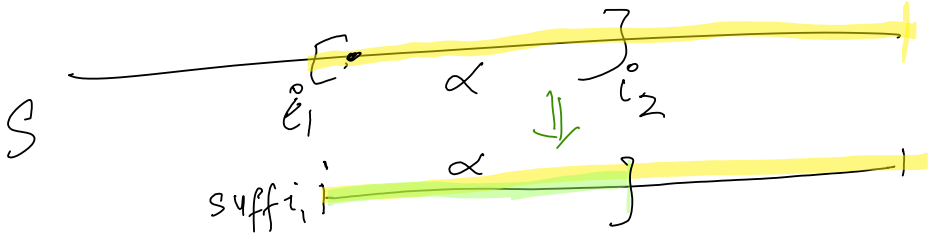


longest
$\Rightarrow c_1 \neq c_2$
$d_1 \neq d_2$

lcs
$\Rightarrow$ right & left maximal.
$\Rightarrow$ "maximal"



Q) What is a substring? $\equiv$ a prefix of a suffix



$\Rightarrow$ So let us build a PATRICIA tree for all suffixes!