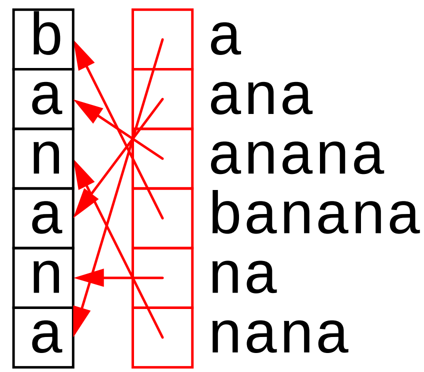
The Inverse Suffix Array

Suffix Array: $SA[i]$ is the starting pos of the i-th smallest suffix of input S

Inverse Suffix Array $\overline{SA}$: $SA[i] = j \Rightarrow \overline{SA}[j] = i$,
i.e., $\overline{SA}[j]$ is the rank of the j-th suffix $S[j:n)$

Example:

$$S \quad = [\mathtt{b}, \mathtt{a}, \mathtt{n}, \mathtt{a}, \mathtt{n}, \mathtt{a}]$$
$$SA = [5, 3, 1, 0, 4, 2]$$
$$\overline{SA} = [3, 2, 5, 1, 4, 0]$$



Longest Common Prefix Length

$\mathrm{lcp}(S, S') := \max\{i : S[0:i) = S'[0:i)\}$ for any two strings S and S'.

We focus on

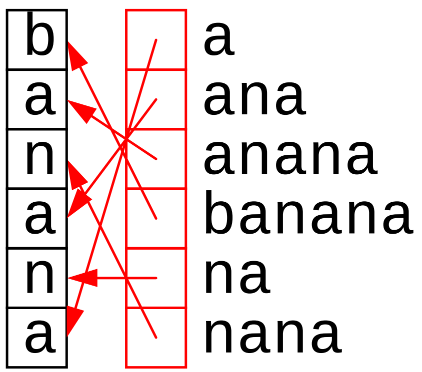
$$\mathrm{lcp}[i] := \begin{cases} 0 & \text{if } i = 0 \\ \mathrm{lcp}(S[SA[i-1]:n), S[SA[i]:n)) & \text{if } i > 0 \end{cases}$$

Example:

$S \quad = [\mathtt{b}, \mathtt{a}, \mathtt{n}, \mathtt{a}, \mathtt{n}, \mathtt{a}]$

$SA = [5, 3, 1, 0, 4, 2]$

$\mathrm{lcp} = [0, 1, 3, 0, 0, 2]$



A Continuity Lemma

Lemma 1. $\mathrm{lcp}[\overline{SA}[i]] \geq \mathrm{lcp}[\overline{SA}[i-1]] - 1$

Proof. Obvious if $\mathrm{lcp}[\overline{SA}[i-1]] \leq 1$.

Otherwise, suppose $\mathrm{lcp}[\overline{SA}[i-1]] = \ell > 1$,

i.e., $\exists S[j:n) : S[j:j+\ell) = S[i-1:i+\ell-1)$. Hence

$S[j+1:j+\ell) = S[i:i+\ell-1)$

$\ldots$ $\square$

Computing lcp-Information

$\mathrm{lcp}[0]$:= $[0, \ldots, 0]$

ℓ:= 0 // lower bound for lcp

foreach $i \in [0 : n)$ **do** // find $\mathrm{lcp}[\overline{SA}[i]]$

  if $\overline{SA}[i] \geq 1$ **then**

    j:= $SA[\overline{SA}[i] - 1]$ // $S[j : n)$ precedes $S[i : n)$ in the suffix array

    while $S[i + \ell] = S[j + \ell]$ **do** ℓ++

    $\mathrm{lcp}[\overline{SA}[i]]$:= ℓ

    ℓ:= $\max\{\ell - 1, 0\}$

$\ell \leq n$

n iterations of main loop

total decrease of ℓ is $\leq n$

---

time $\mathcal{O}(n)$

Pattern Matching Using lcp-Information

Theorem 2. lcp-*Information, all occurences of a pattern can be found in time* $\mathcal{O}(\log n + |P| + \#\textit{occurences})$

Trick: ignore the common prefix of the strings to be compared P

This information can be precomputed in time $\mathcal{O}(n)$

But we want to go for an even faster approach:

Suffix Tree $T = (V, E)$ for S

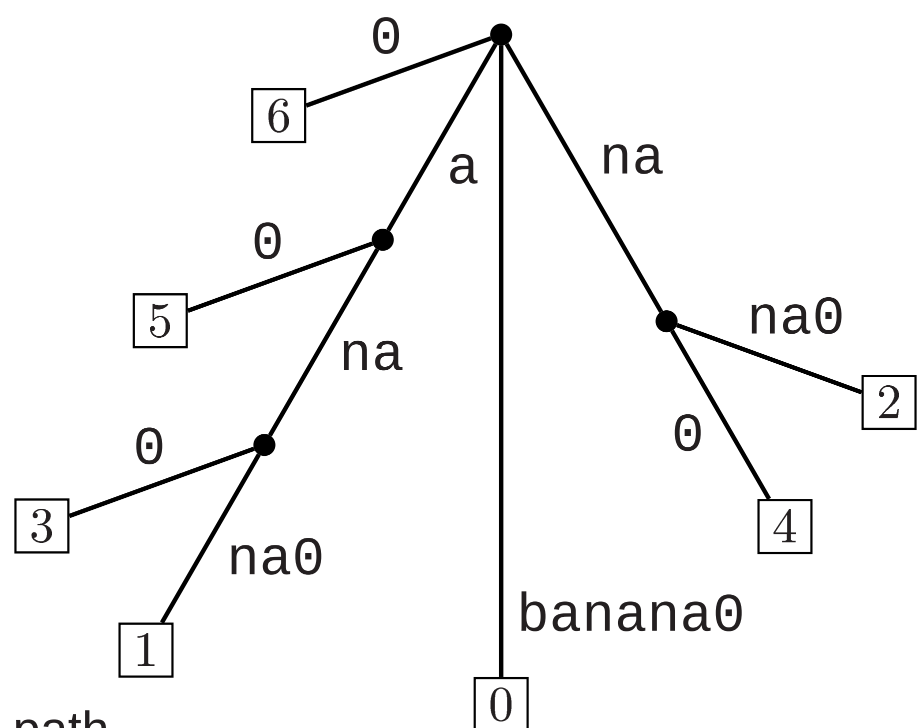
- Edges are labeled with substrings of S (just pointers)
- One leaf for each suffix and the $label$s on the root-leaf path are equal to the suffix.
- $\forall v \in T$:

  $\forall e = (v, u) \in E$:

  $\forall e' = (v, u') \in E \setminus \{e\}$:

  $e.label[0] \neq e'.label[0]$
- $v.d$:= total length of labels on root-v path.

$S =$ banana0



Suffix Array $\rightarrow$ Suffix Tree

Function *suffixTree*(SA, lcp : $\mathbf{Array}\ [0:n)$ **of** $[0:n)$) : *Tree*

T:= empty suffix tree (root only).

v:= T // current node

d:= 0 // distance from root in characters

foreach $i \in [0:n)$ **do** // insert $S[SA[i]:n)$ into T

while $v.parent.d \geq \mathrm{lcp}[i]$ **do** v:= $v.parent$

if $v.d > \mathrm{lcp}[i]$ **then** v:= $splitEdge(v)$

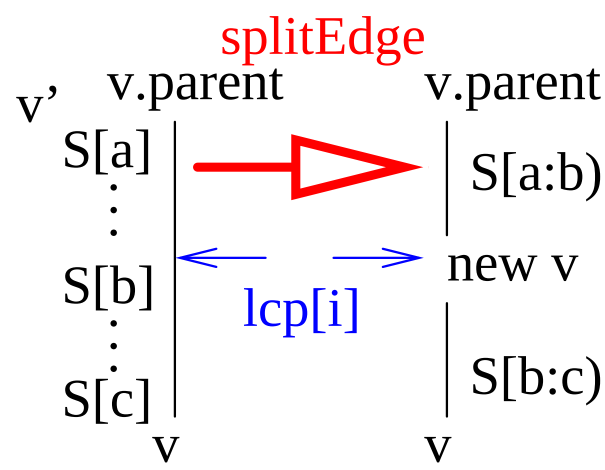
assert $v.d = \mathrm{lcp}[i]$

make a new leaf v'

$(v, v').label$:= $S[SA[i] + \mathrm{lcp}[i] : n)$

v:= v'

return T



Analysis

- $\mathcal{O}(n)$ Iterations of the main loop
- $\mathcal{O}(1)$ time for an iteration of the main loop
- at most 2 edges per iteration
- Only one backtrack per edge

————————————————————————-

$\mathcal{O}(n)$ time

Pattern Matching Using Suffix Trees

Function *match*($P[0:m)$: *String*; $T=(V,E)$: *SuffixTree*) :

i:= 0 // first unmatched character

v:= $T.root$

while $i<m$ **do**

if $\exists e=(v,u)\in E : e.label=P[i:i+|e.label|)$ **then**

i:= $i+|e.label|$

v:= w

else return $\emptyset$

report all leaves rooted at v as occurences

Time $\mathcal{O}(m+\#\text{occurences})$