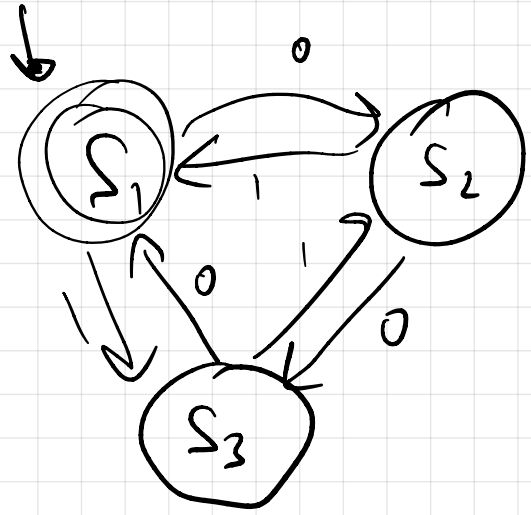
Suffix Automaton

Automata Theory



↓ - starting state

◎ accepting st.

→ transitions

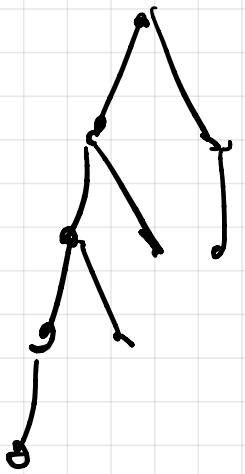
usual convention — if link is missing — it goes to "garbage" node

$$Aut = (\underbrace{S}_{\text{states}}, s_0 \in S, \delta: S \times \Sigma \to S, T \subseteq S)$$

$$L_{Aut} = \{ t \in \Sigma^* : \delta(s_0, t) \in T \}$$

Suffix automaton ($s \in \Sigma^*$) — smallest automata accepting suffixes of s & only them

Some automata accepting only suffixes of S



trie

Right Contexts:

$$R_{Aut}(t \in \Sigma^*) = \{\alpha \in \Sigma^* : t\alpha \in L_{Aut}\}$$

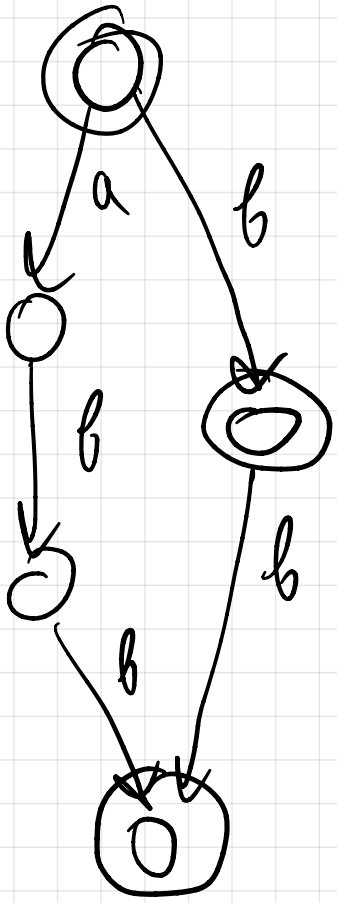
$$R_{Aut}(v \in S) = \{\alpha \in \Sigma^* : \delta(v, \alpha) \in T\}$$

Claim: it's possible to compress states of some $R(v)$

$\Rightarrow$ So smallest automata consists of exactly $\{R(s) \mid s \in \Sigma^*\}$ states

Automaton edges: $R(s) \xrightarrow{c} R(sc)$

Suffix Automaton

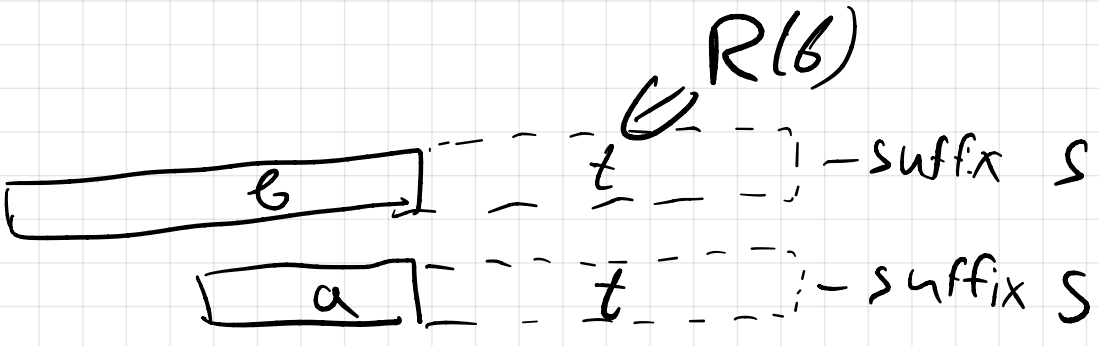


$SA(s)$ – suffix aut. of s.

Lm α – suffix β

$\Rightarrow R(\alpha) \supseteq R(\beta)$

Proof



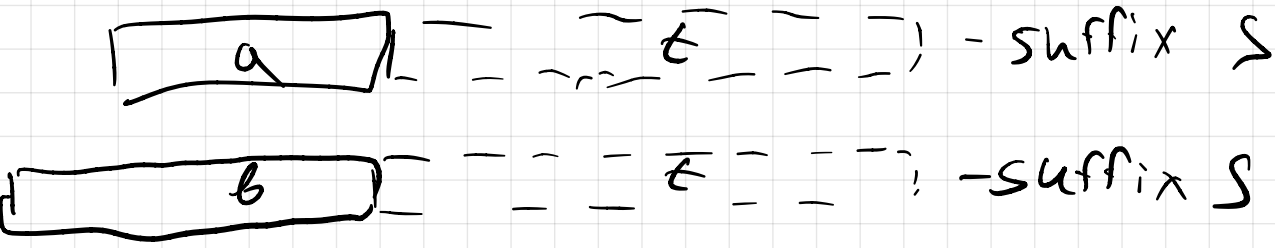
$\Rightarrow t \in R(\alpha)$

Lm $R(\alpha) \cap R(\beta) \neq \emptyset$

$\Rightarrow R(\alpha) \supseteq R(\beta) \vee R(\beta) \supseteq R(\alpha)$

(α - suffix β) (β - suffix α)

Proof Suppose $|\alpha| \leq |\beta|$, $t \in R(\alpha) \cap R(\beta)$

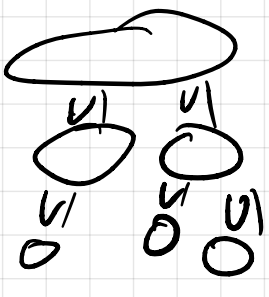


$\Rightarrow \alpha$ - suffix β

$\Rightarrow R(\alpha) \supseteq R(\beta)$ □

i.e. R sets are either nested or don't intersect.

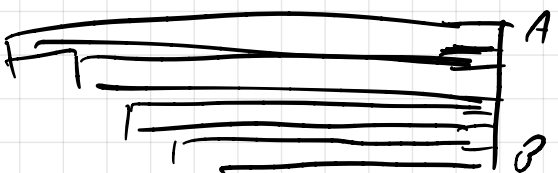
We can consider tree of R sets (excl. empty set)



Lm For any $R \neq \emptyset$

Consider $Str_R = \{\alpha \mid R(\alpha) = R\}$

Then Str_R = Continuous slt of suffixes



i.e. if A is longest in StR_R
and β is shortest in StR_R

Then $StR_R = \{ t \mid t\text{-suffix } A,$
$\beta\text{-suffix } t\}$

and β - suffix A

Proof ① $R(A) = R(\beta)$

$\Rightarrow A$ suffix β
or β suffix A

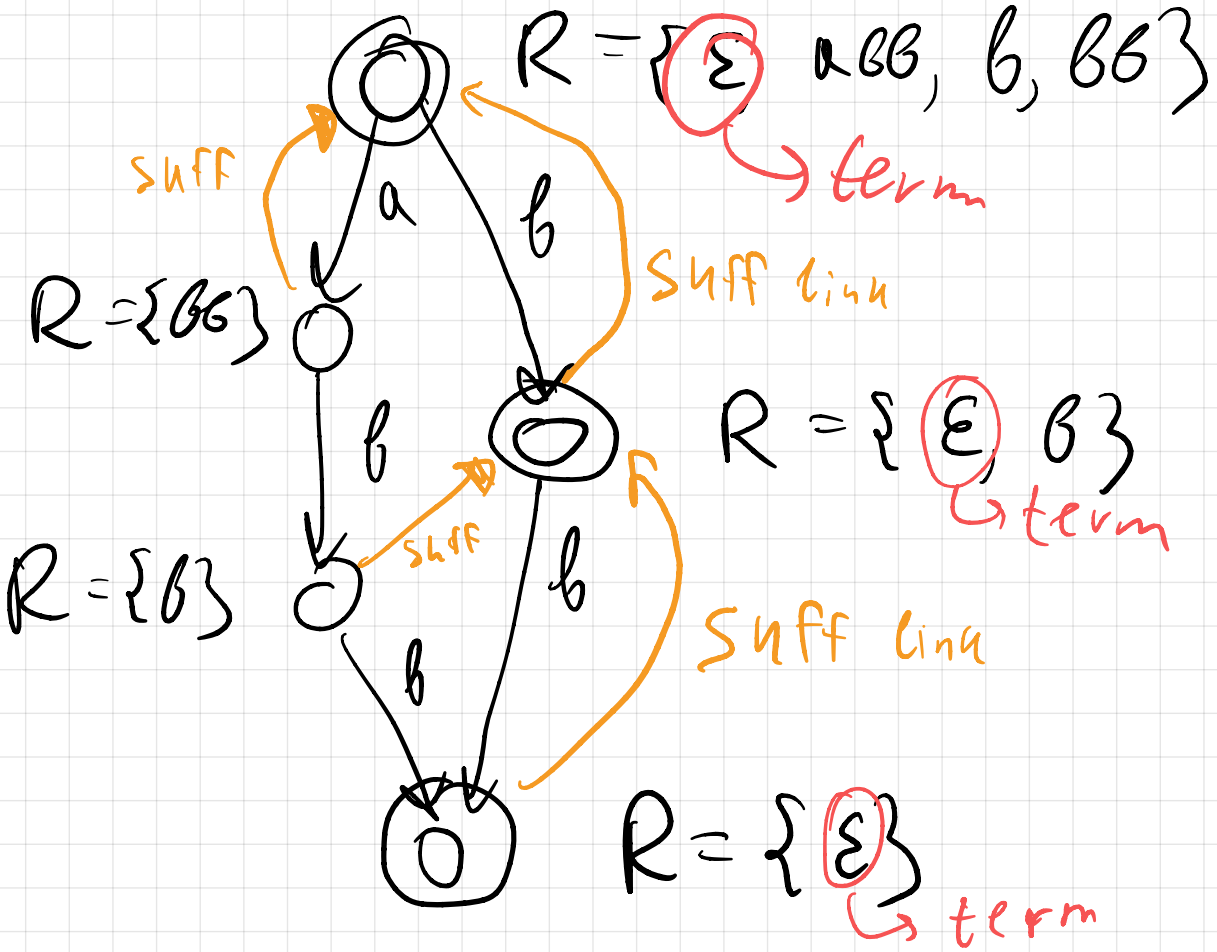
Since $|\beta| \leq |A| \Rightarrow$ β - suffix A

② $R(t) = R(\beta) = R(A)$

$\Rightarrow t$ suffix A,
β - suffix t

③ if t - suffix A,
β - suffix t

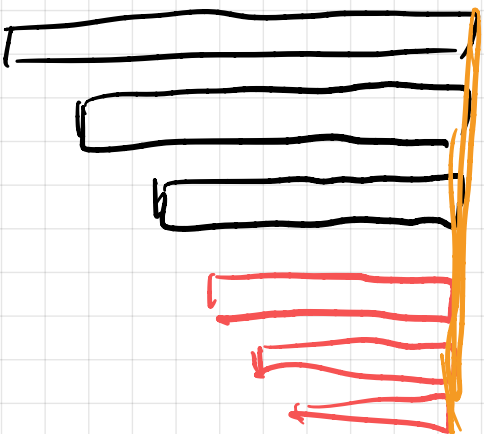
$\Rightarrow R(A) \subseteq R(t) \leq R(\beta)$
$\Rightarrow R(t) = R$



Def: Let len(v) = length of longest t s.t. R(t) corresponds to v

Let minlen(v) = length of shortest t s.t. R(t) corresponds to v

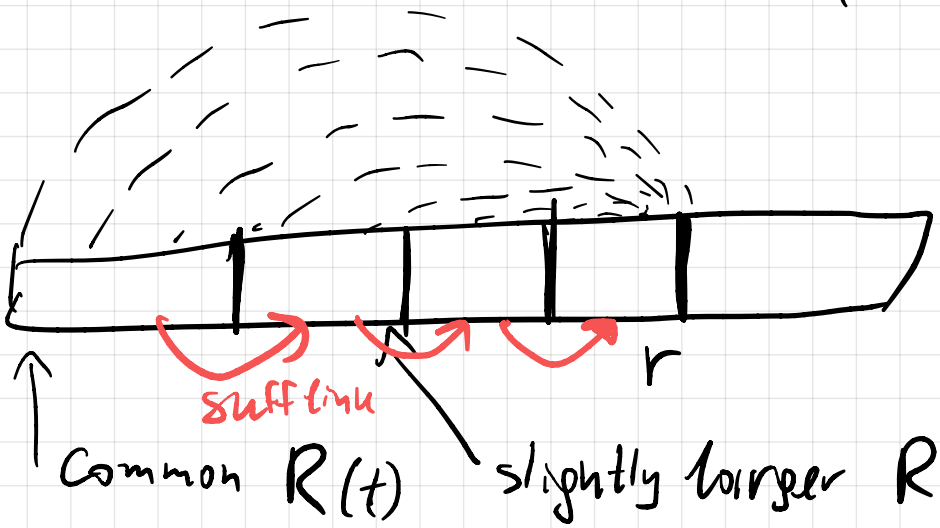
Lm: minlen(v) = len(suf(v)) + 1



EDCAB
DCAB
not CAB
~~XYAB~~
AB !

The journey of shorter strings

Consider strings with fixed r & different ℓ



SA(S) vs SA(Sc) ?

Reminder:

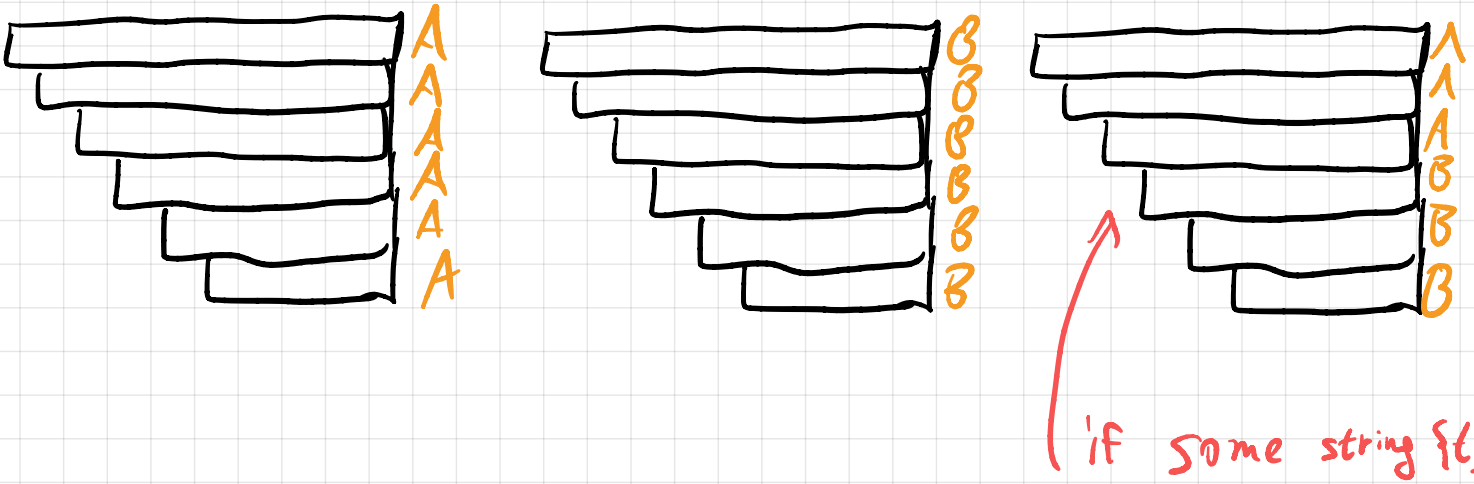
$$R_S(\alpha) = \{\beta \mid \alpha\beta \text{- suffix of } S\}$$

Then

$$R_{Sc}(\alpha) = \begin{cases} \{\beta c \mid \beta \in R_S(\alpha)\} & \text{type A} \\ \{\beta c \mid \beta \in R_S(\alpha)\} \cup \{\varepsilon\} & \text{type B} \end{cases}$$

type B $\Leftrightarrow$ iff α - suffix Sc

Classification $\{R_S(t) \mid t\text{-substr } S\}$



if some string $\{t\}$ is type-B, then t-suffix sc

$\Rightarrow$ ALL SHORTER suffixes ARE TYPE-B too.

Remember edges:

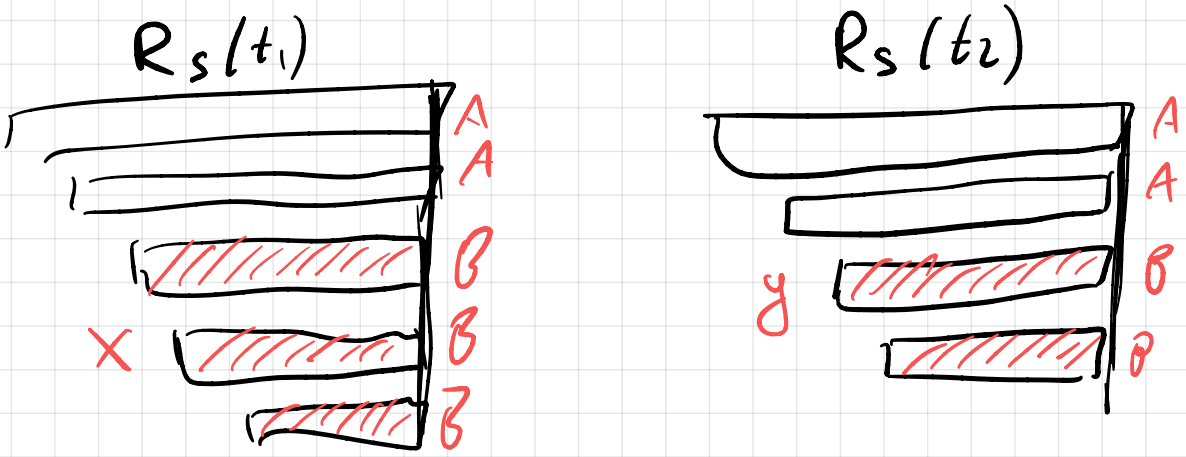
$$R_S(t) \xrightarrow{\beta} R_S(t\beta)$$

$\Rightarrow$ if states $R_S(t)$ & $R_S(t\beta)$ change by rule A only or rule B only, no need to do anything.

Otherwise we have to split vertex.

Lm: There is at most one vertex which splits

Proof Suppose contrary



Let x, y - any type-B string from $R_S(t_1), R_S(t_2)$ ← ($SA(s) \to SA(sc)$)

x - suffix sc

y - suffix sc

WLOG assume $|x| < |y|$

$\Rightarrow$ x - suffix y

$\Rightarrow$ $R_S(t_1) \supseteq R_S(t_2)$

$\Rightarrow$ $R_S(t_2) \to R_S(t_1)$ by chain of suf links

$\Rightarrow$ all strings in t_1 are suffixes of y

$\Rightarrow$ type-B

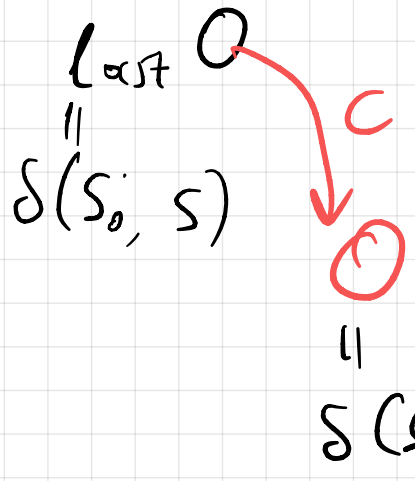
Implementation Note

```
struct Node {
    Node* suf;
    (unordered_)map<char, Node*> go;
    int len;
};
```

Algorithm

extend(c)

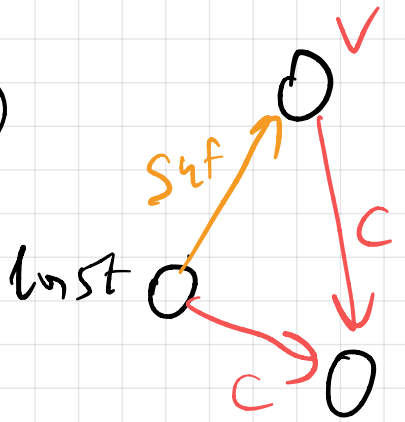
① last = $\delta(s_0, s)$



$\delta(s_0, sc)$

have to add this

$R(s) \to R(sc)$

② 

(while there is no edge c, add it)

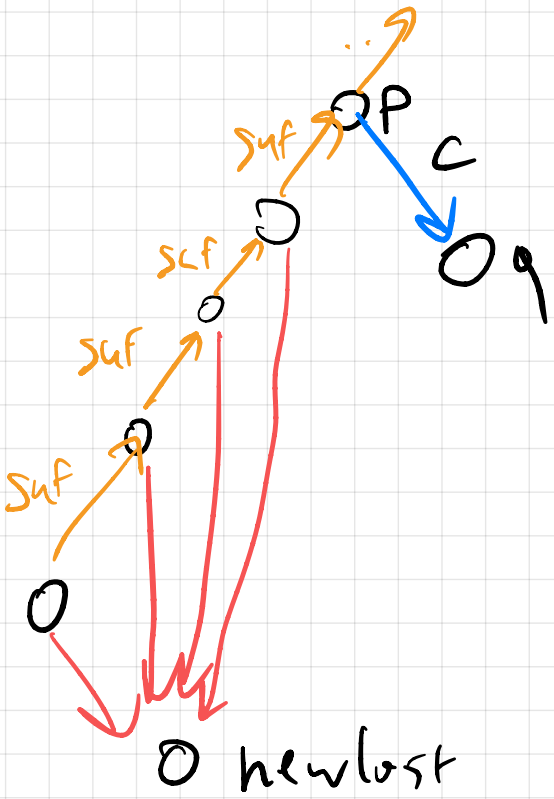
$R_{sc}(newlast) = \{\varepsilon\}$,

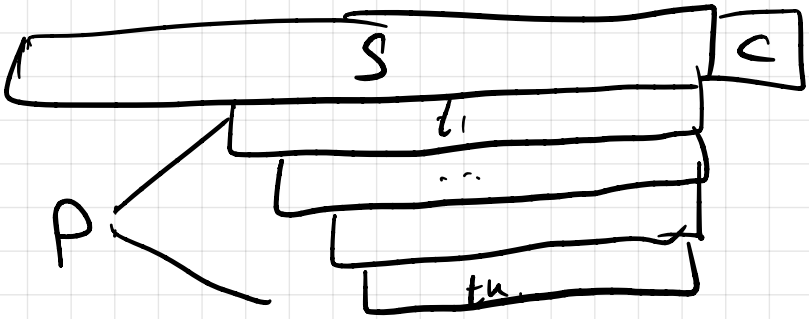
let $t : \delta(s_0, t) = v$

$R_{sc}(tc) = \{\varepsilon\}$, otherwise

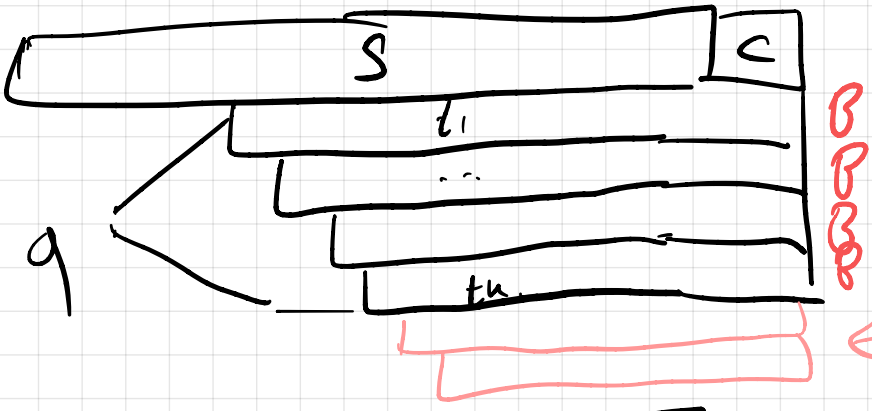
$R_s(t)$ should have edge c.

III

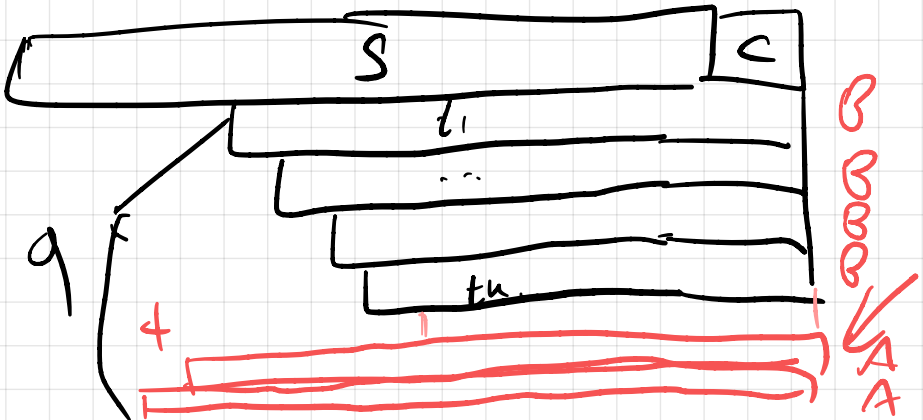




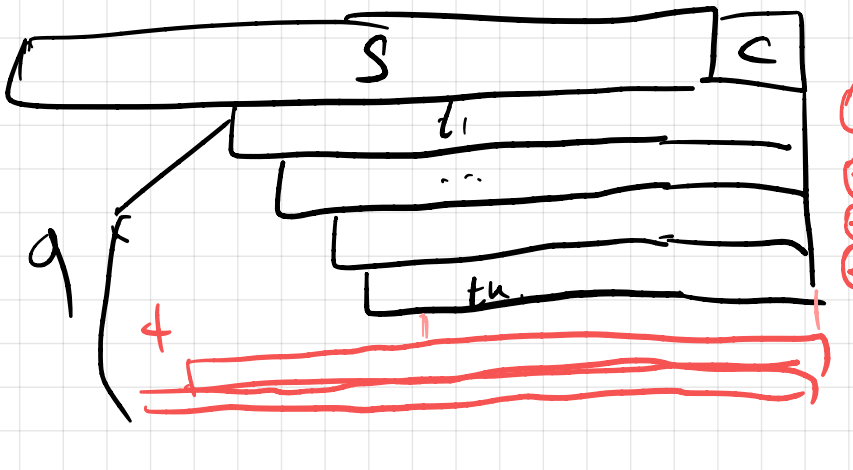
OPT 1



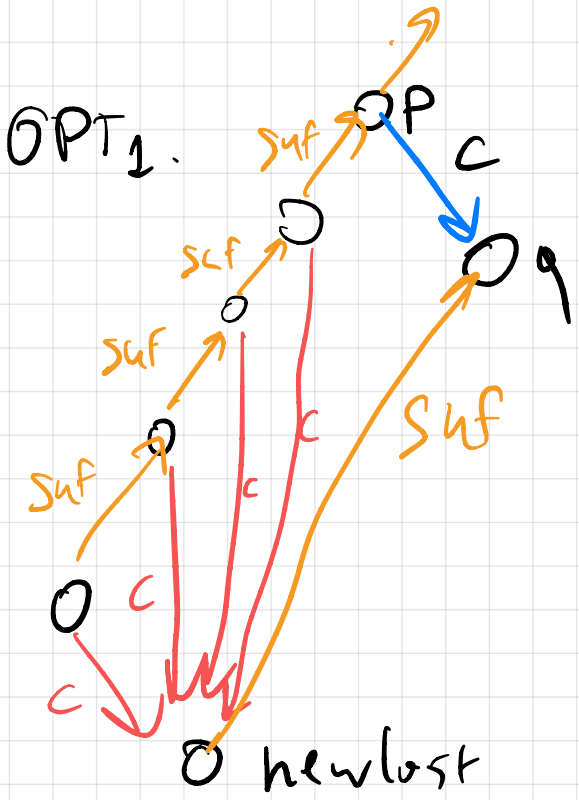
OPT 2:



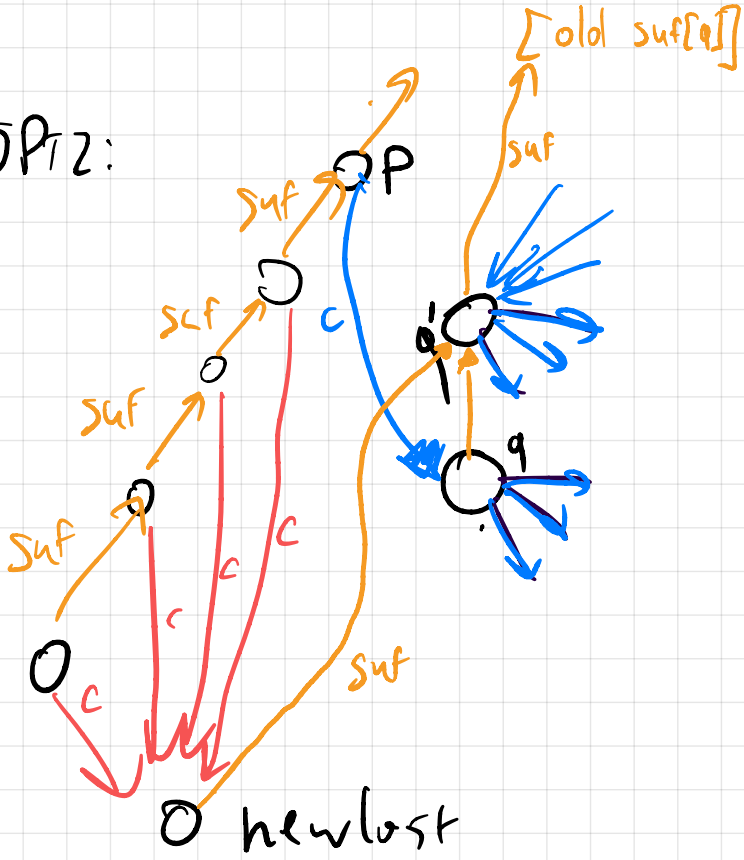
SC automaton:



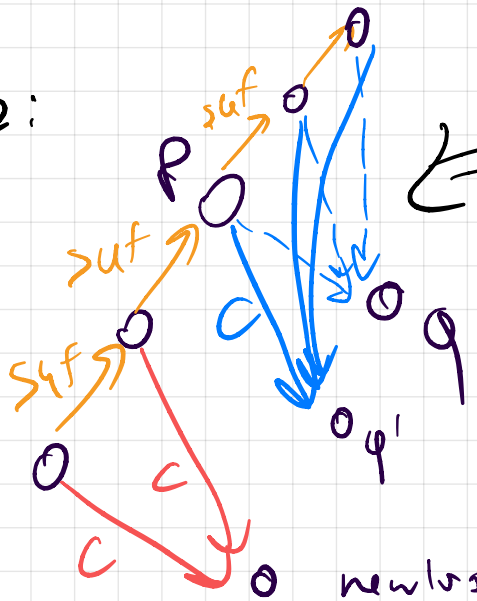
OPT1.



OPT2:



OPT 2:



⇐ might need to redirect shorter strings in suf(p) suf(suf(p)) ...

```
Extend(c)
  newlast = new node()
  newlast.len = last.len+1

  p=last
  while p && !(c∈p.go):
    p.go[c]=newlast                    (*)
    p=p→suf.

  if p==null:
    newlast.suf=root
    return

  q=p.go[c]
  if p.len+1==q.len
    newlast.suf=q
  else:
    q'=new node(len=p.len+1
                go=p.go                (**)
                suf=q.suf)
    newlast.suf=q.suf=q'
    while p && p.go[c]=q:              (***)
      p.go[c]=q'; p=p.suf
```

Linearity:

Goals:

# Vertex:	$\leq 2n-1$	$(n \geq 2)$
# Edges	$\leq 3n-4$	$(n \geq 3)$
Complexity:	$O(n)$	

Vertices:

$V(n) =$ MAX vert. for $|s|=n$.

$V(0)=1$

$V(1)=2$

$V(2)=3 \leq 2 \cdot n-1$

$V(n : n \geq 2) \leq 2+V(n-1) \leq 2 \cdot n-1$

Edges

def: $p \to q$ continuous edge if $p.len+1 = q.len$

$\#Edges = \#Cont\ Edges + \#Non\ Cont\ Edges$

$= V(n)-1$

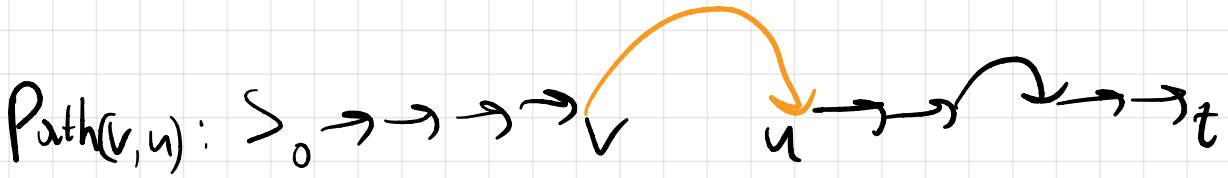
(Each vertex has exactly 1 incoming edge)

Non Cont Edges:

Let $v \to u$ non cont.

Take $s_0 \to v$ path with Cont. edges

$u \to t$ any path



There $\leq n$ paths $s_0 \to t$

note $\{path(v,u)\}$ are all diff. since (v,u) - first non cont edge on path(v,u)

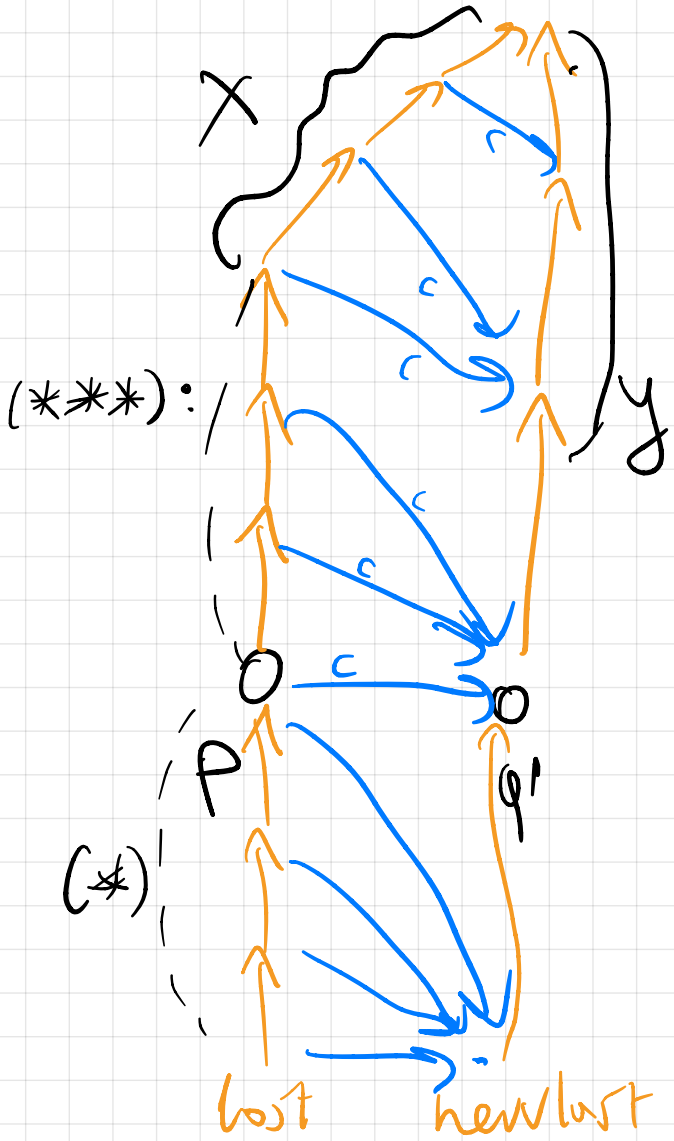
$$\#Edges = \#Cont\ Edges + \#Non\ Cont\ Edges$$
$$= V(n) - 1 \qquad \leq n$$
$$\leq 2n - 1$$
$$\leq 3n - 2$$

To improve to $\leq 3n-3$, note that $s_0 \to t$ path $s \to \cdots \to t$ doesn't correspond to any long edge. $\leq 3n-4$ skipped proof

Time Complexity

There are 3 places in algorithm's loop not working in $O(1)$

(*), (**) ← upper bounded by #Edges



φ: len of suff links path last → root

$\varphi_{old} = X + (***) + (*) + O(1)$

$\varphi_{new} = y + O(1)$

$(***) + (*) + \varphi_{new} - \varphi_{old} =$

$= y - x + O(1)$

$y \leq x + 1$

due to surjectivity of mapping

Applications

[0] Check if t - substr s

Compute $\delta(\text{State}_0, t)$.

[1] Number of substrings

#substrings $+ =$ #new substrings $=$

$= last \to len - last \to suf \to len$

[2] Find t : t has 2 non-overlapping occurr.

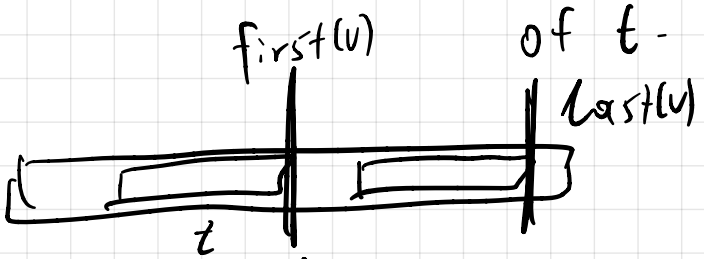
Compute for each v:

$first(v)$, $last(v)$

i.e. if $R(t)$ corresponds to v

$first(v)$ = smallest end of occurr. of t

$last(v)$ = largest end of occurr of t



ans $\leftarrow \min(last(v) - first(v), len(v))$