i

$s_0 \; s_1 \; s_2 \; s_3 \; - - - \; s_n$

$t_0 \; t_1 \; t_2$

$s_i \; s_{i+1} \; s_{i+2} \; - - - \; s_j$

$t_0 \; t_1 \; t_3$

$a \; a \; b \; b \; a \; b \; c \; - - - \; - -$

$a \; b \; c$

Naive:

for all i, j s.t. $i \leq j$

$S_i \sim S_j$, check if

$t_0 \sim t_n$ is in $S_i \sim S_j$

Idea: for each s_i, we want to find the smallest j s.t. contains $t_0 \sim t_{n-1}$

Ex.

$$\begin{array}{cccccccc} & i & & & & j & & \\ & \downarrow & & & & \downarrow & & \\ s_0 & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & \end{array}$$

$$\begin{array}{c} t_0\ t_1\ t_2\ t_3 \\ \uparrow \\ k \end{array}$$

observe that:

① if $s_i \mathrel{!}= t_0$,

then $s_{i+1} \sim s_j$ must contain $t_0 \sim t_n$

and j will be $i+1$'s smallest

index too

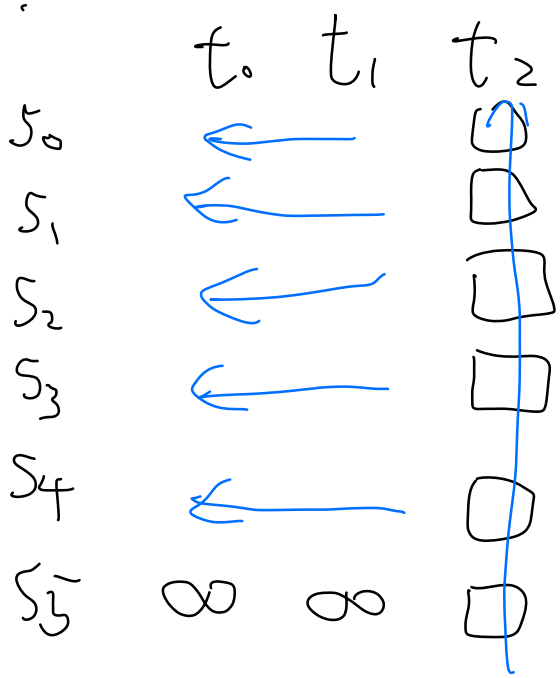
② if $s_i = t_0$,

the $s_{i+1} \sim s_j$ only needs to contain $t_1 t_2 \sim t_{n-1}$

Define: $dp(s_i, t_k)$ = the smallest j s.t. $s_i \sim s_j$ contains $t_k \sim t_{n-1}$

$$dp(s_i, t_k) = \begin{cases} dp(s_{i+1}, t_{k+1}) & \text{if } s_i = t_k \\ dp(s_{i+1}, t_k) & \text{else} \end{cases}$$

Ex.



order of computation

$s_0\ s_1\ s_2\ s_3\ s_4\ s_5\ s_6\ s_7$

$t_0\ t_1\ t_2$

a b a b c

a d d d d d d b c d d d d d d d d b c

a b c

Two pointers:

$$S = s_0\ s_1\ \underbrace{s_2\ s_3\ s_4}\ s_5\ s_6\ s_7 \Rightarrow s_0 \sim s_{m-1} \text{ in general}$$

$$t = t_0\ t_1\ t_2 \Rightarrow t_0 \sim t_{n-1} \text{ in general}$$

If $s_l \sim s_r$ is the first window starting from l that can contain $t_0\ t_1\ t_2$

We can observe several properties:

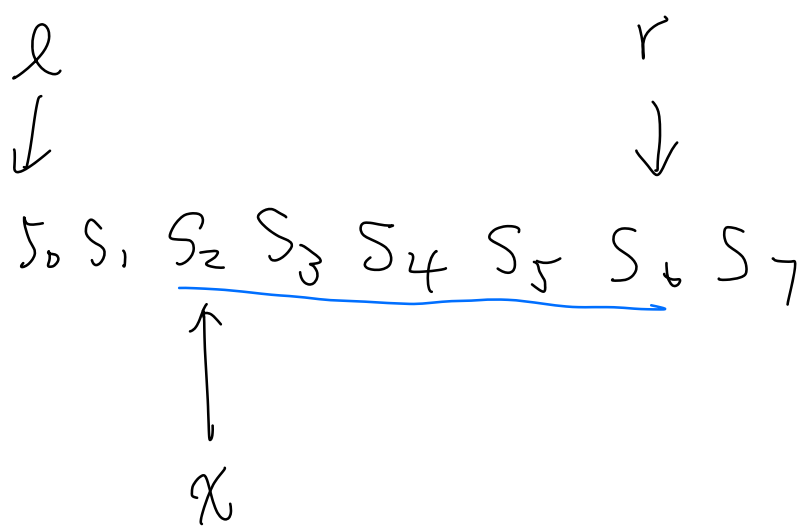
① if $x \in [l+1, r)$, $y \in [x, \boxed{r})$

$s_x \sim s_y$ does not contain t

Because if $S_x \sim S_y$ contains t, $[l, r]$ window can be set to $[l, y]$

$\Rightarrow$ contradicts that $[l, r]$ is the smallest r

② $x \in [l+1, r] \Rightarrow S_x \sim S_r$ <u>might</u> contain t



ex. $s = \overset{l}{a}\,a\,a\,b\,a\,b\,a\,b\,\overset{r}{c}$

$t = abc$

Usually, in a standard two pointers problem, we want to move l to the right until $[l, r]$ does not contain t anymore.

However, in this problem, when we move $l \leftarrow l+1$, we are not sure if $[l, r]$ still contains t

Ex.

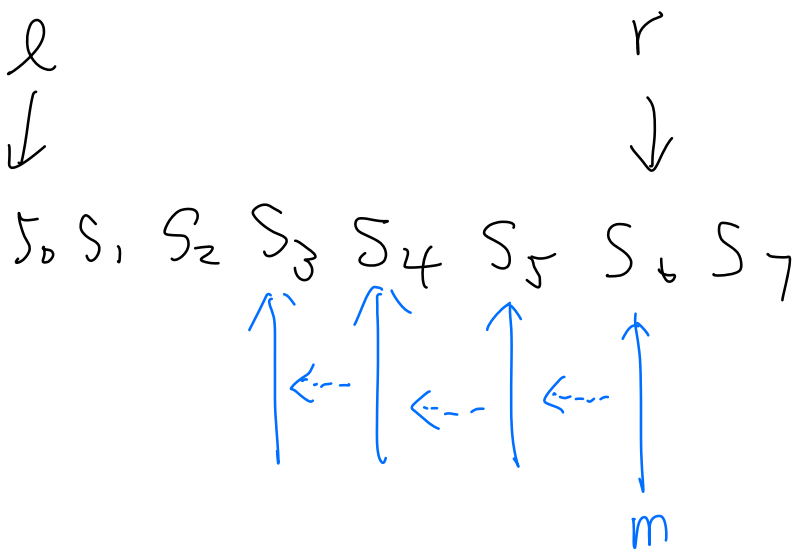
$s = \overset{l}{a}\ \overset{l+1}{\boxed{bbbbbb\ ?}}\ bc$

$t = abc$

You can see that ? can be a,

but we don't know how to store this information.

Therefore, we think "backward",



We instead set $m=r$, and move m left until we find $[m, r]$ contains t

This m will tells us:

(i) if $x \in [m+1, r]$, then

$S_x \sim S_r$ does not contain t

$\Rightarrow$ every $x \in [m+1, r]$, must keep expanding

r to the right in order to contain t

(ii) if $x \in [l, m)$, $S_x \sim S_r$ contains t

but $S_x \sim S_r$ length $>$ $S_m \sim S_r$,
(worse)

so every substring starting from x,

we don't need to check!

by (i), (ii), after getting m, we will set

$l \leftarrow m+1$

and expand r to the right until

$S_l \sim S_r$ contains t

$\boxed{\text{Complexity:}}$ ← hard to prove

[$O(\text{len}(t))$ times]

S_i will be visited at most $\text{len}(t)$ times

Proof:

$s = S_0\, S_1\, S_2 \cdots S_k \cdots S_i \cdots S_{m-1}$

l ↑ mid ↑ r ↑

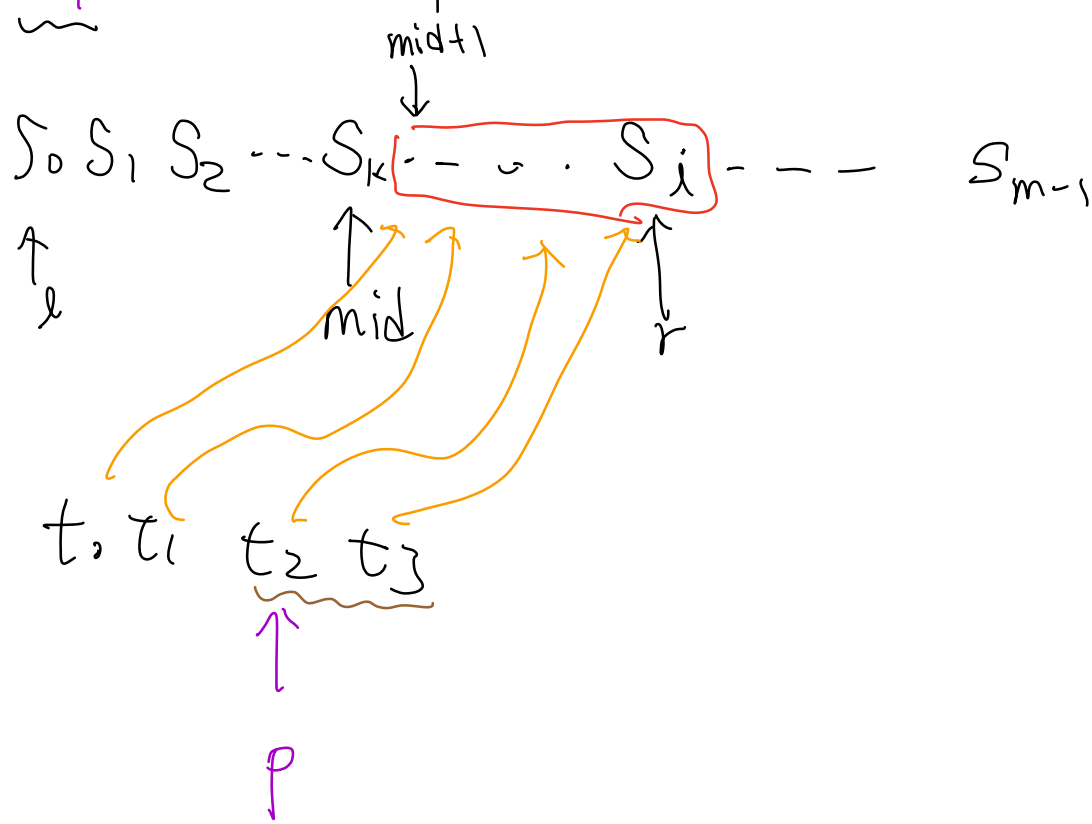
$t = t_0\, t_1\, t_2\, t_3$

When we move mid backward from r

t_0 is mapped to mid

When we set $l \leftarrow mid+1$ and expand r to the right

All the current mapping will be broken (every t_p) because if say any t_p still maps to the same $S[q]$



then obviously, every t_x where $x \geq p$ can still map to same characters in S

And because t_p map to the same character.

Every $x < p$, t_x must map to character in front of t_p's corresponding charater

$\Rightarrow$ However, this contradicts that $\boxed{}$ part does not contain t !

Thus, all t_p must map to different (on the right of its corresponding charater)

$\Rightarrow$ Every $S[i]$ at most be mapped $O(\text{len}(t))$