Source: http://mathcs.emory.edu/~cheung/Courses/323/Syllabus/Text/Matching-KMP1.html

Knuth-Morris-Pratt (KMP)

S = given string $len(S) = n$

p = pattern $len(p) = m$

Goal: Find if p occurs in S as a substring. If it does, find where it occurs in S

Naive:

S = `abcd abc abcdf'

p = `abcdf'

check if $s[i : i + |p| - 1] = p$ for each i

$O(mn)$

KMP: $O(m+n)$ solution

→ ⊛ solve a simpler problem: Given string l, find longest prefix of l which is also a suffix of l efficiently.

l = abcdabC

prefix:
- a
- ab
- abc
- abcd
- abcda
- abcdab

suffix:
- c
- bc
- abc
- dabc
- cdabc
- bcdabc

KMP algorithm

Step 1 : Prepare `prefix - suffix' table (PS) for pattern p

pattern $p = p_0 \; p_1 \; \cdots \; p_{m-1}$

Define PS[i] = length of longest prefix which is also a suffix for $p_0 p_1 \cdots p_i$

[we will see how to do this efficiently via the solved simpler subproblem] ⊛

Granting this for now,

Example

	0	1	2	3	4	5	6	7
P =	d	s	w	a	d	s	s	z
PS =	0	0	0	0	1	2	3	0

PS[5] = 2 ie, in d s w a d s

length of longest prefix which is also suffix is 2

Step 2 : pointer i running across string s
pointer j running across pattern p

i_0 ← potential start of a match

(eg) s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

Start pointer i at pos 0, in s; j at pos 0 in p, $i_0 = 0$

Find longest possible match (by moving i, j forward)

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

a b a b d

Mis match at pos 4.

$i = 4, \; j = 4$

Naive strategy

Resets pointer $i = i_0$ to pos 1,

Resets pointer j to pos 0

Repeat

KMP

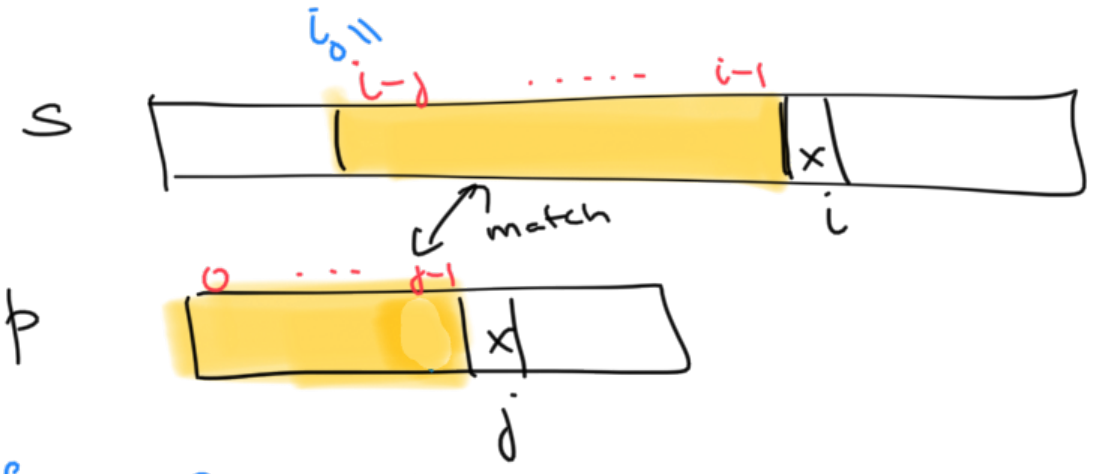
AVOID MOVING POINTER i backwards!

	0	1	2	3	4	i
s	a	b	a	b	x	
p	a	b	a	b	x	

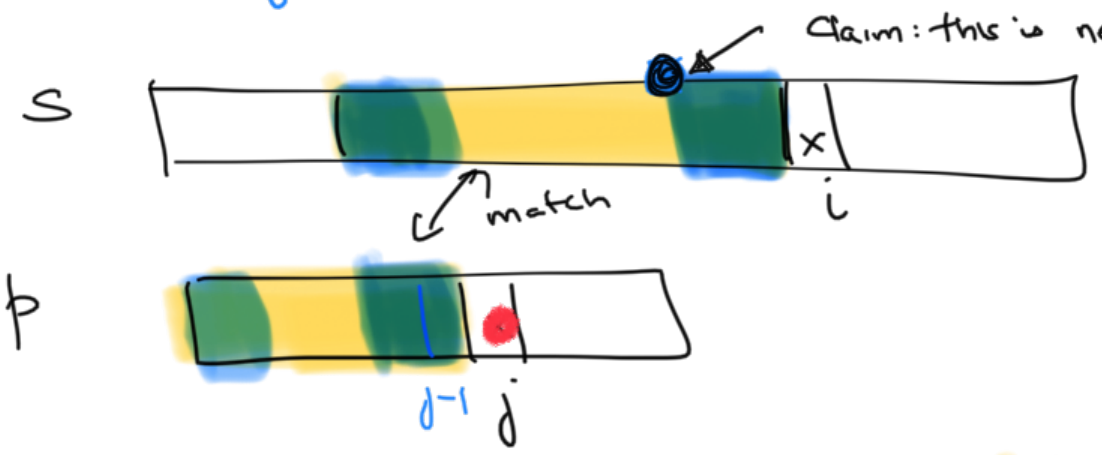
j

Know $i_0 = 0$ DOESN'T WORK.

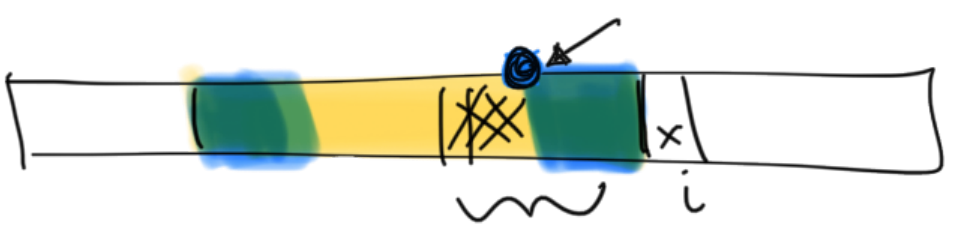
Also know $p[0 : j]$ matches with $s[i-j : i]$ (not included)



Further, if $ps(j-1) = k$, then what is the next i_0 position?



(if something before this works,

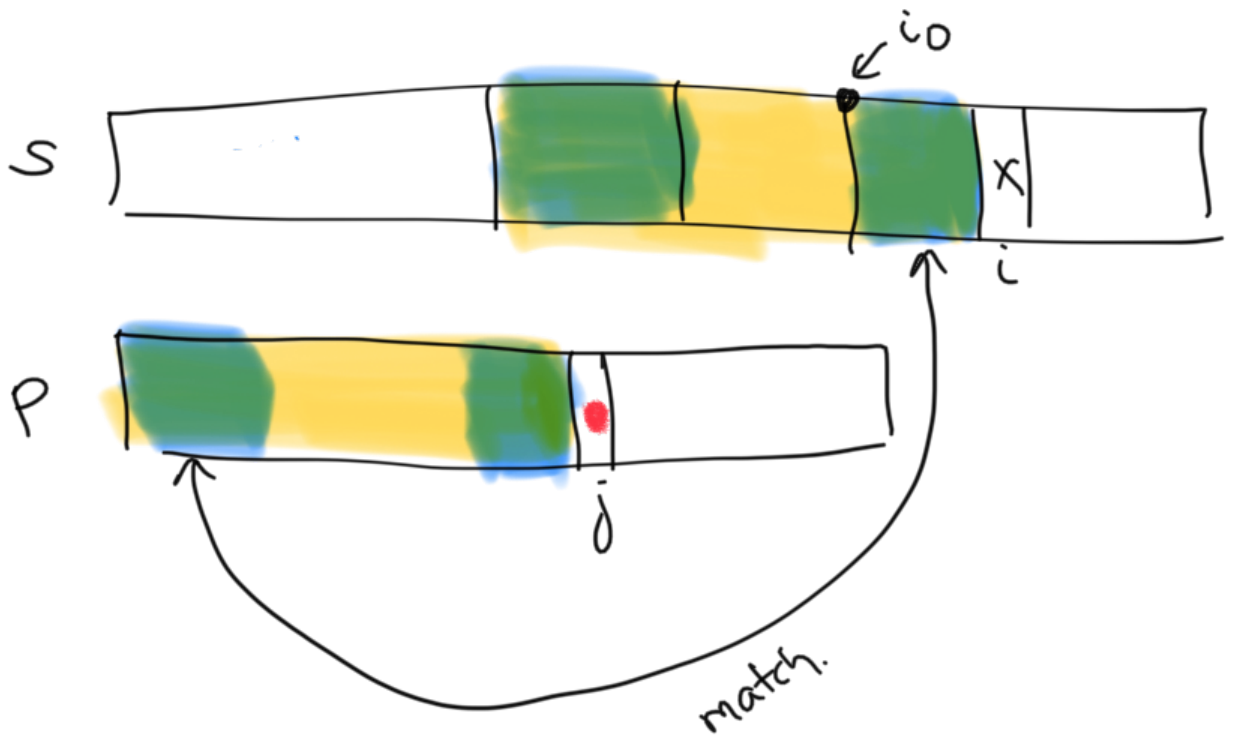


then

would have been the longest prefix of $p[0 : j]$ which is also a suffix

$\rightarrow \leftarrow$

And also, we already know



→ so need to compare s[i] with p[k]

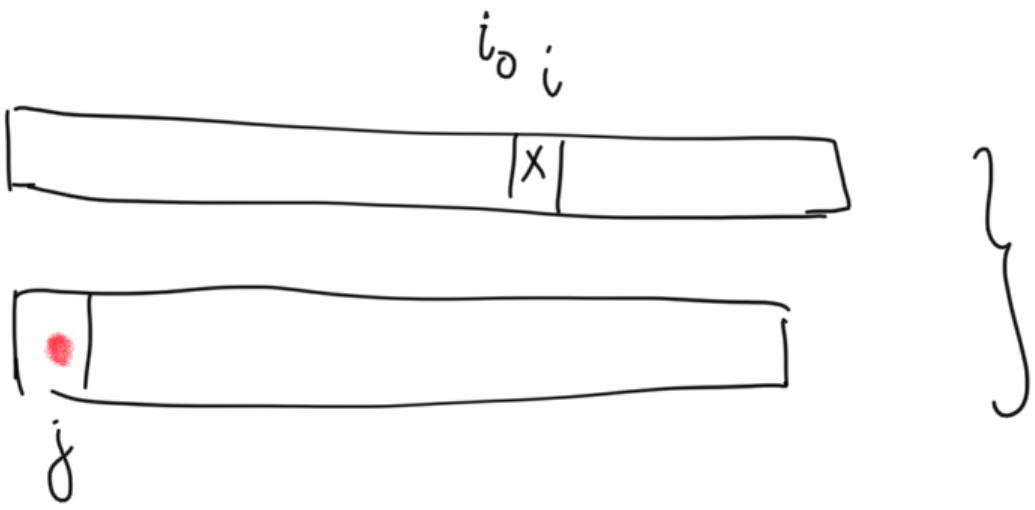
→ so we reset j = ps(k) !!

→ Reset $i_0 = i - j$

i never moves back

and we compare s[i] with p[j] ...

▷ what happens if j = 0 when mismatch occurs ?



$i_0 = i_0 + 1$

$i = i_0$

$j = 0$

Dry run

0)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$i_0 = 0$

$i = 0$

$j = 0$

match

1)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$i_0 = 0$

$i = 1$

$j = 1$

match

2)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$i_0 = 0$

$i = 2$

$j = 2$

match

3)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$i_0 = 0$

$i = 3$

$j = 3$

match

4)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$i_0 = 0$

$i = 4$

$j = 4$

mismatch

5)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$k = ps[j-1] = ps[3] = 2$

$j = k = 2$

$i_0 = i - j = 4 - 2 = 2$

$i = 4$

mismatch

6)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$k = ps[j-1] = ps[1] = 0$

$j = k = 0$

$i_0 = i - j = 4 - 0 = 4$

$i = 4$

mismatch

7)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

As $j = 0$

$i_0 = i_0 + 1 = 5$

$i = i_0 = 5$

match

8-10)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$j=1$, $j=2$, $j=3$,

$i=6$, $i=7$, $i=8$,

$i_0=5$, $i_0=5$, $i_0=5$,

matches

11)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

$j=4$

$i=9$

$i_0=5$

mismatch

12)

s = a b a b c a b a b a b d

p = a b a b d

ps = 0 0 1 2 0

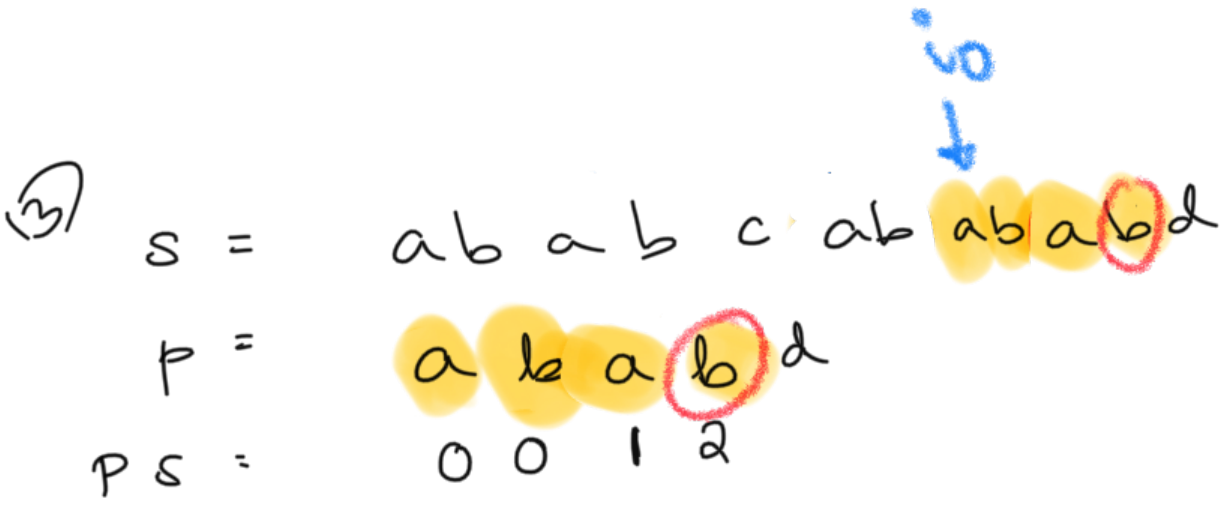
$k = ps[j-1] = ps[3] = 2$

$j = k = 2$

$i_0 = i - j = 9 - 2 = 7$

match

(3)



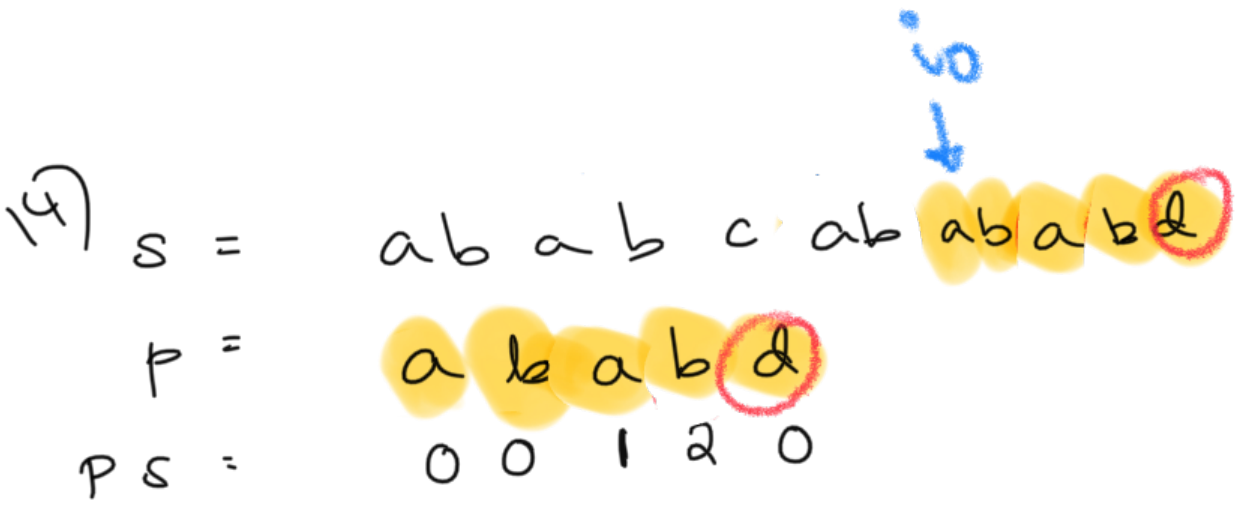
$i = 10$

$i_0 = 7$

$j = 3$

match

(4)



$i = 11$

$i_0 = 7$

$j = 4$

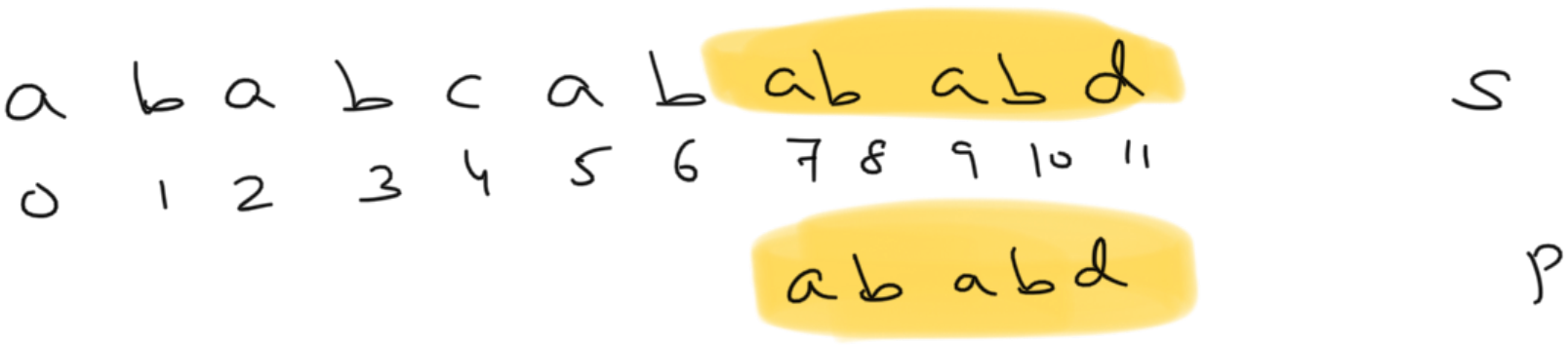
match

(j is max possible)

↓

Terminate

pattern found, starting at $i_0 = 7^{th}$ position in S



How many steps?

→ i only inc from 0 to len(S) − 1

→ either i, j both inc by 1 (match) and i_0 stays same

→ or i stays same, j resets to $PS(j-1)$, $i_0 = i - j$

Note $PS(j-1) < j-1$ as $PS(j-1)$ = length of longest proper prefix of $P_0 P_1 \cdots P_{j-1}$ which is also suffix

so j decreases.

But $i_0 = i - j$, so i_0 inc

So 2 kinds of steps

$i = i+1$
$j = j+1$
i_0 same

I

i same
j dec to $PS[j-1]$
i_0 inc by $j - PS[j-1]$

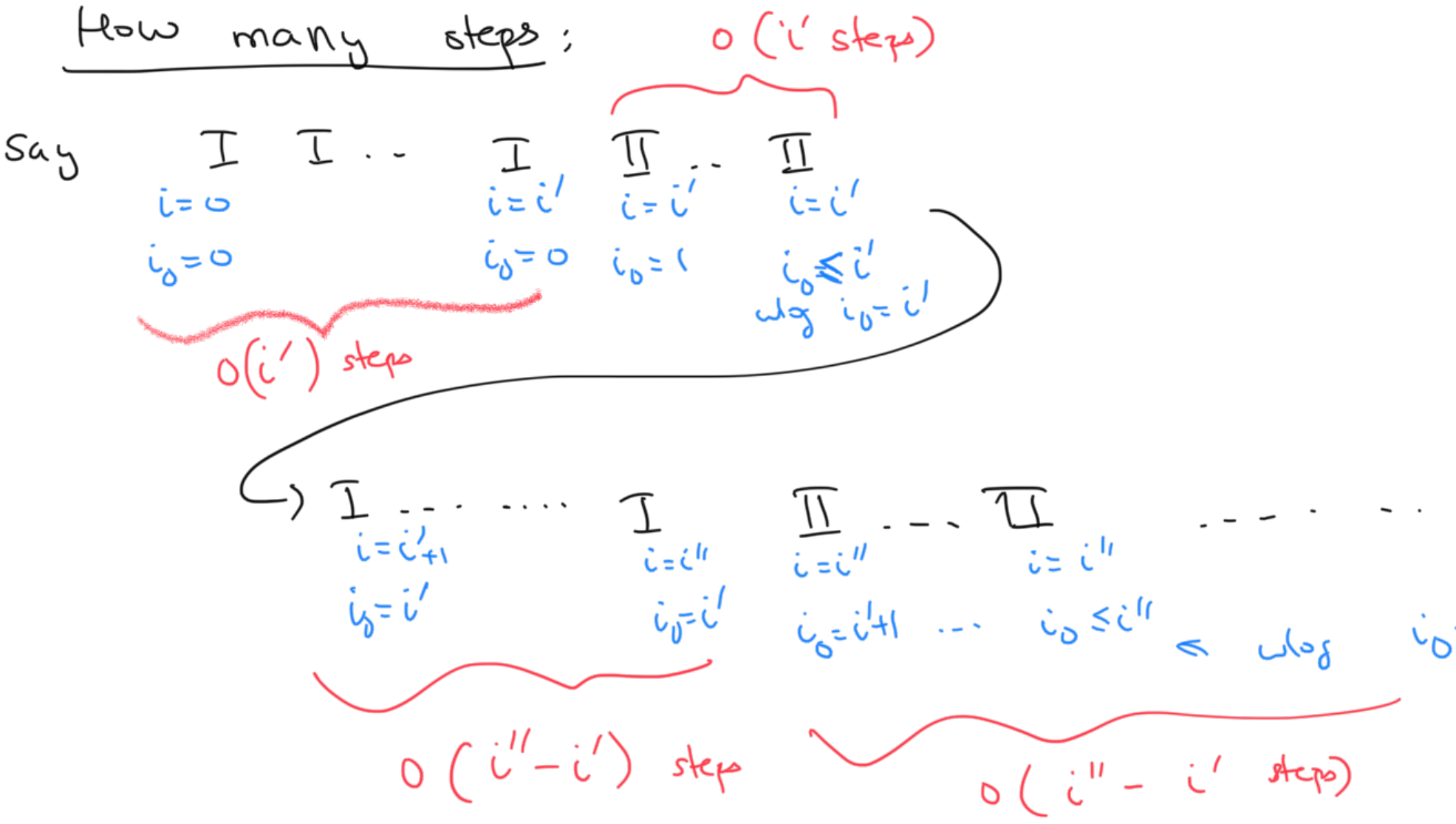
II

≥ 2

$i \leq len(s)$

$i_0 \leq len(s)$ and $i_0 \leq i$ also in fact

How many steps:



So overall

$O(2i'') = O(i'' \text{ steps})$ where i'' ← value of i at last instance possible

$i \leq len(s) \Rightarrow O(len(s)) \text{ steps}$

Kmp algo: $O(PS\text{-table computation}) + O(len(s))$