

# **COL 351: Analysis and Design of Algorithms**

## **Lecture 13**

# String Matching Problem

**Given:** String  $S = [s_1, \dots, s_n]$  and a pattern  $P = [p_1, \dots, p_k]$ , represented as arrays of size  $n, k$ . (Here  $k < n$ ).

**Find:** Does there exists a **sub-string of S** that is identical to P.

Example:

S = “cuckoo hashing is efficient”

P = “hash”

Yes

S = “cuckoo hashing is efficient”

P = “hash-table”

No

# String Matching Problem

**Given:** String  $S = [s_1, \dots, s_n]$  and a pattern  $P = [p_1, \dots, p_k]$ , represented as arrays of size  $n, k$ . (Here  $k < n$ ).

**Find:** Does there exists a **sub-string of  $S$**  that is identical to  $P$ .

Example:

$S =$  “cuckoo hashing is efficient”

$P =$  “hash”

Yes



*Can you have an  
 $O(nk)$  time algorithm?*

# String Matching Problem

**Given:** String  $S = [s_1, \dots, s_n]$  and a pattern  $P = [p_1, \dots, p_k]$ , represented as arrays of size  $n, k$ . (Here  $k < n$ ).

**Find:** Does there exists a **sub-string of S** that is identical to P.

**For**  $i = 1$  to  $n$ :

Flag = **True**

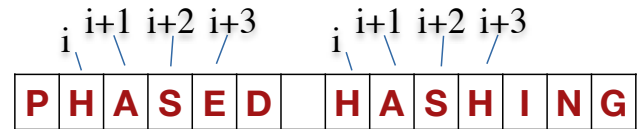
**For**  $j = 1$  to  $k$ :

**If**  $S[i + j - 1] \neq P[j]$  **then**

Flag = **False**

**If** (Flag) **Return** **True**

**Return** **False**



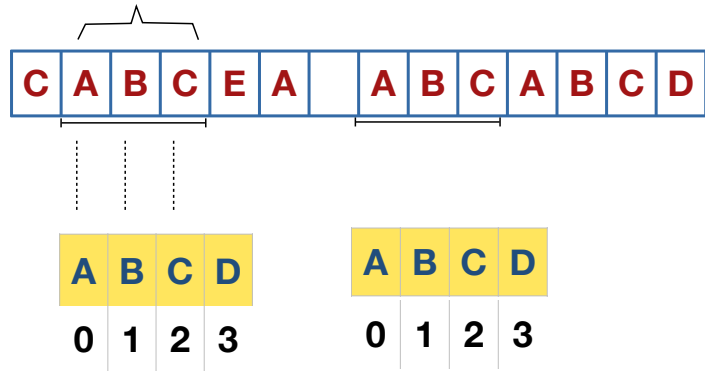
$O(nk)$  time algorithm

# Special Scenario: All characters in “ pattern P ” are different!



*Can you have an  
 $O(n)$  time algorithm?*

The substring matched  
with prefix of P contains  
only one copy of  $P[1]=A$   
as no characters are  
repeated in P

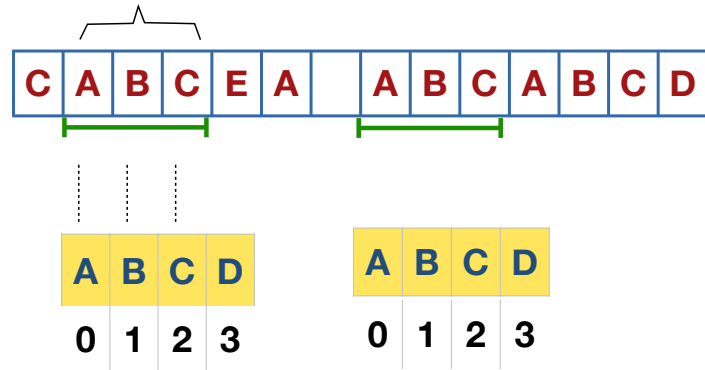


# Special Scenario: All characters in “ pattern P ” are different!

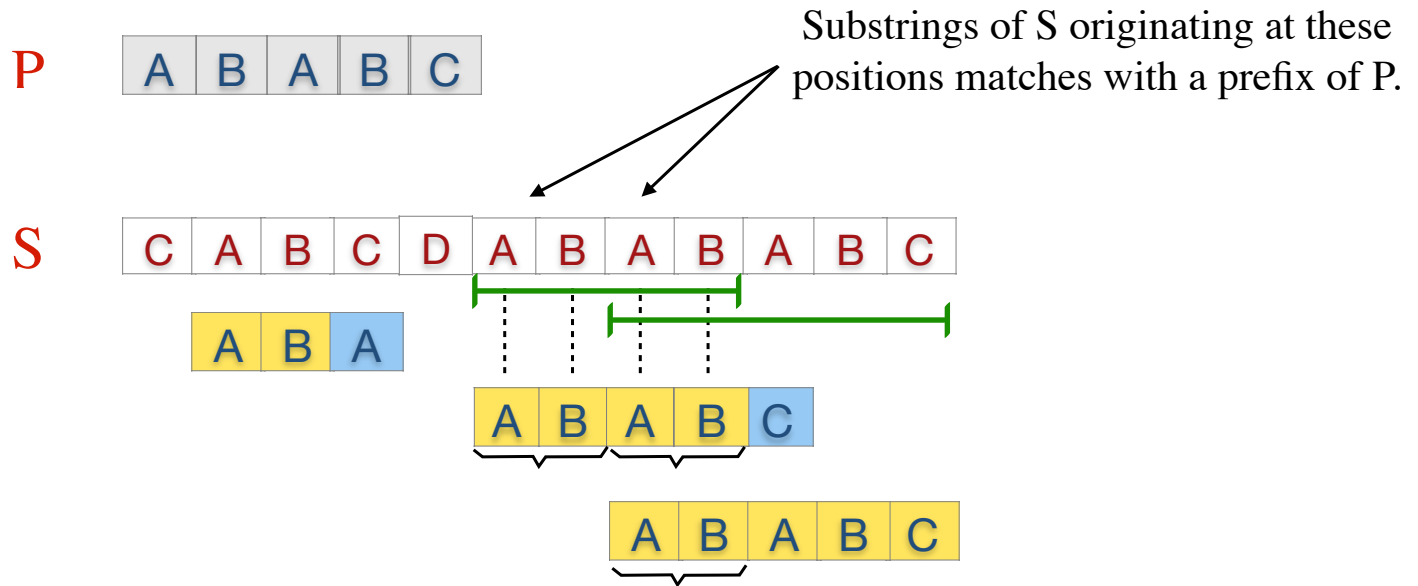
```
i, j ← 1;  
While (i ≤ n):  
  If S[i] = P[j]:  
    If (j = k): Return True  
    Increment i and j by 1;  
  Else :  
    j ← 1;  
    Increment i by 1;  
Return False;
```

$O(n)$  time algorithm  
for special scenario

The substring matched  
with prefix of P contains  
only one copy of  $P[1]=A$   
as no characters are  
repeated in P



## An Example where P has repeated characters



**Remark:** Key Idea to obtain Linear time algorithm is Pattern preprocessing.

# Sub-Problem

**Given:** Pattern  $P = [p_1, \dots, p_k]$ .

**Find:** Table of size  $k$  satisfying

$\text{Table}[i] =$  Length of longest ~~non-trivial~~ prefix of  $P[1, i]$  that is also a suffix of  $P[1, i]$

PROPER

**Examples:**

$i$	1	2	3	4	5	6
$P[i]$	A	B	C	A	B	B
$\text{Table}[i]$	0	0	0	1	2	0

$i$	1	2	3	4	5	6	7	8	9
$P[i]$	A	A	B	A	A	B	A	A	A
$\text{Table}[i]$	0	1	0	1	2	3	4	5	2



# Sub-Problem

$\text{Table}[i]$  = Length of longest (non-trivial) common prefix and suffix of  $P[1, i]$

**Lemma:** For any  $i \geq 1$ , we have  $\text{Table}[i + 1] \leq 1 + \text{Table}[i]$ .

Proof Sketch:

(H.W.)

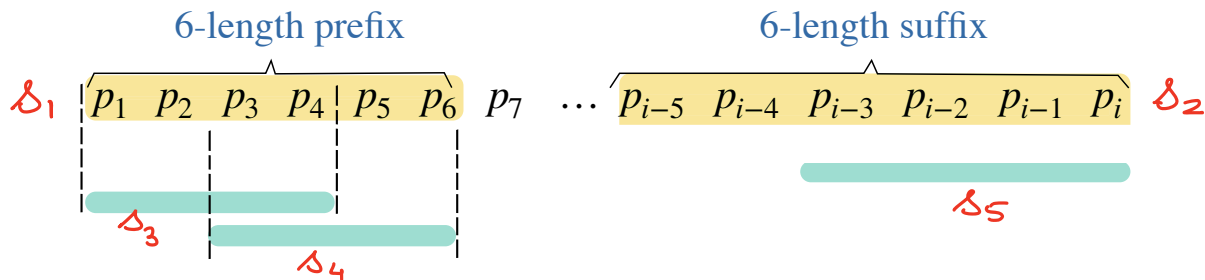
# Prefix Suffix subproblem

$\text{Table}[i]$  = Length of longest (non-trivial) common prefix and suffix of  $P[1, i]$

**Lemma:** Suppose  $L \geq 1$  satisfy that  $L$ -length prefix and suffix of  $P[1, i]$  are identical.

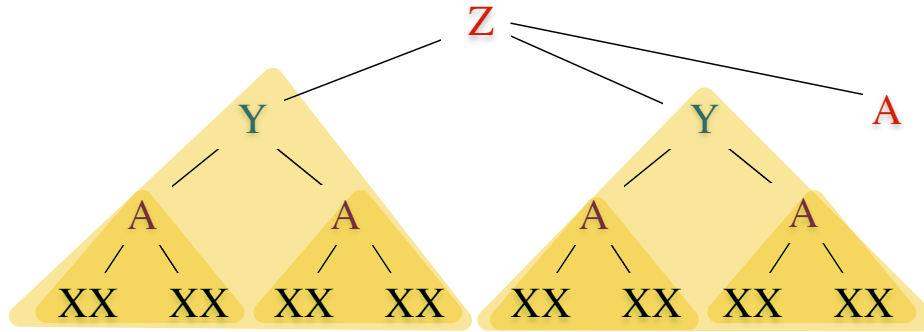
Then, the length of longest common prefix-suffix of  $P[1, i]$  of size just smaller than  $L$  is “Table[L]”.

## Proof Sketch:



If  $\delta_1 = \delta_2$  and  $\delta_3 = \delta_4$ , then  $\delta_3 = \delta_5$

We will study this example to understand the intuition for an  $O(k)$  time algorithm to solve our sub-problem.



$i$	1	2	3	4	5	6	7	8	9	10	11	12												
$P[i]$	X	X	A	X	X	Y	X	X	A	X	X	Z	X	X	A	X	X	Y	X	X	A	X	X	A
Table[ $i$ ]	0	1	0	1	2	0	1	2	3	4	5	0	1	2	3	4	5	6	7	8	9	10	11	?

## Example

$i$	1	2	3	4	5	6	7	8	9	10	11	12	...								22	$i$	$i+1$	
$P[i]$	X	X	A	X	X	Y	X	X	A	X	X	Z	X	X	A	X	X	Y	X	X	A	X	X	A
Table[ $i$ ]	0	1	0	1	2	0	1	2	3	4	5	0	1	2	3	4	5	6	7	8	9	10	11	

- 1) We have computed Table values upto an index  $i = 23$ .
- 2) Note  $\text{Table}[23] = 11$ . Thus, 11 is length of longest identical suffix-prefix of  $P[1, 23]$ .
- 3) Now,  $Z = P[11+1] \neq P[23+1] = A$ , so,  $\text{Table}[23+1]$  cannot be 12.  
We compute length of longest identical suffix-prefix of  $P[1, 23]$  smaller than 11.  
This is just  $P[11] = 5$ .
- 4) Again,  $Y = P[5+1] \neq P[23+1] = A$ .  
Therefore, we compute length of longest identical suffix-prefix of  $P[1, 23]$  smaller than 5.  
This is just  $P[5] = 2$ .
- 5) Compare  $P[2+1]$  and  $P[23+1]$ . Both are A. Thus,  $\text{Table}[23+1]$  is  $2+1=3$ .

# Prefix Suffix subproblem

$\text{Table}[i] :=$  Length of longest (non-trivial) common prefix and suffix of  $P[1, i]$

**Lemma:** Suppose Table is computed for indices 1 to  $i$ .

Let  $len = \text{Table}[i]$  and suppose  $P[i+1] \neq P[len+1]$ .

Then,  $\text{Table}[i+1]$  can be computed as follows:

Recursively update  $len = \text{Table}[len]$  until either  $P[i+1] = P[len+1]$  or  $len = 0$ .

— If  $len$  becomes 0, then  $\text{Table}[i+1]$  must be 0.

— If  $len > 0$ , then  $\text{Table}[i+1]$  must be one larger than current value of  $len$ .