

# Case Study: Searching for Patterns

Problem: find all occurrences of pattern  $P$  of length  $m$  inside the text  $T$  of length  $n$ .

$\Rightarrow$  *Exact matching problem*



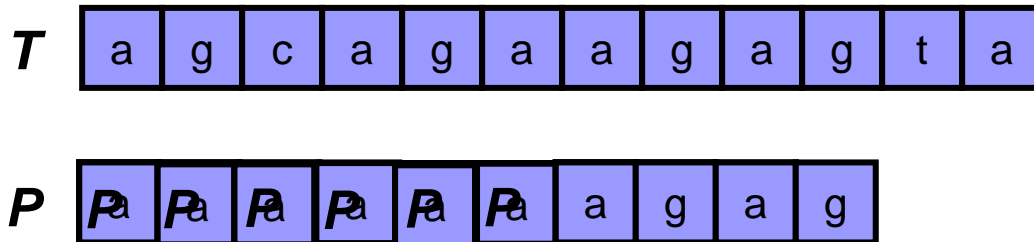
# String Matching - Applications

- ☐ Text editing
- ☐ Term rewriting
- ☐ Lexical analysis
- ☐ Information retrieval
- ☐ And, bioinformatics

# Exact matching problem

Given a string ***P*** (pattern) and a longer string ***T*** (text).  
Aim is to find all occurrences of pattern ***P*** in text ***T***.

The naive method:



If  $m$  is the length of ***P***, and  $n$  is the length of ***T***, then

Time complexity =  $O(m.n)$ ,

Space complexity =  $O(m + n)$

# Can we be more clever ?

- When a mismatch is detected, say at position  $k$  in the pattern string, we have already successfully matched  $k-1$  characters.
- We try to take advantage of this to decide where to restart matching

<i>T</i>	a	g	c	a	g	a	a	g	a	g	t	a
<i>P</i>	a	<del>g</del>	<del>a</del>	a	<del>g</del>	<del>a</del>	a	g	a	g		



# The Knuth-Morris-Pratt Algorithm

Observation: when a mismatch occurs, we may not need to restart the comparison all way back (from the next input position).

What to do:

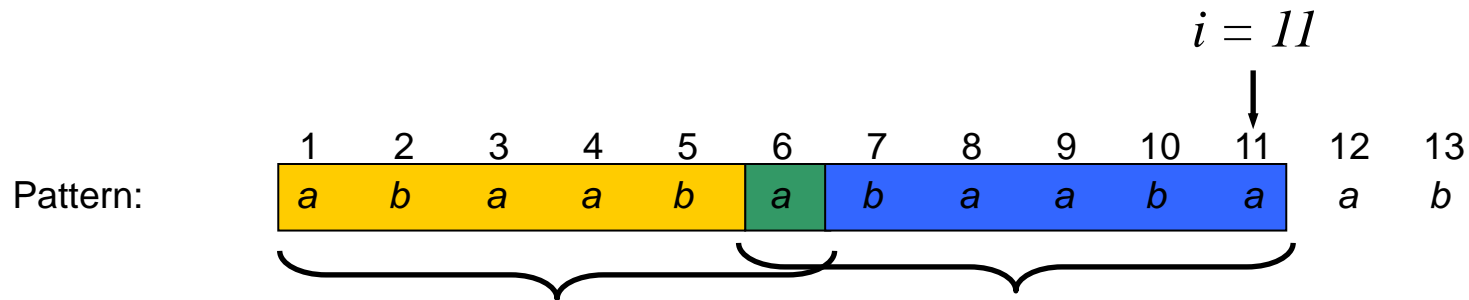
Constructing an array  $h$ , that determines how many characters to shift the pattern to the right in case of a mismatch during the pattern-matching process.

## KMP (2)

The **key idea** is that if we have successfully matched the prefix  $P[1 \dots i-1]$  of the pattern with the substring  $T[j-i+1, \dots, j-1]$  of the input string and  $P(i) \neq T(j)$ , then **we do not need to reprocess any of the suffix  $T[j-i+1, \dots, j-1]$**  since we know this portion of the text string is the **prefix** of the pattern that we have just matched.

# The failure function $h$

For each position  $i$  in pattern  $P$ , define  $h_i$  to be the length of the longest proper suffix of  $P[1, \dots, i]$  that matches a prefix of  $P$ .

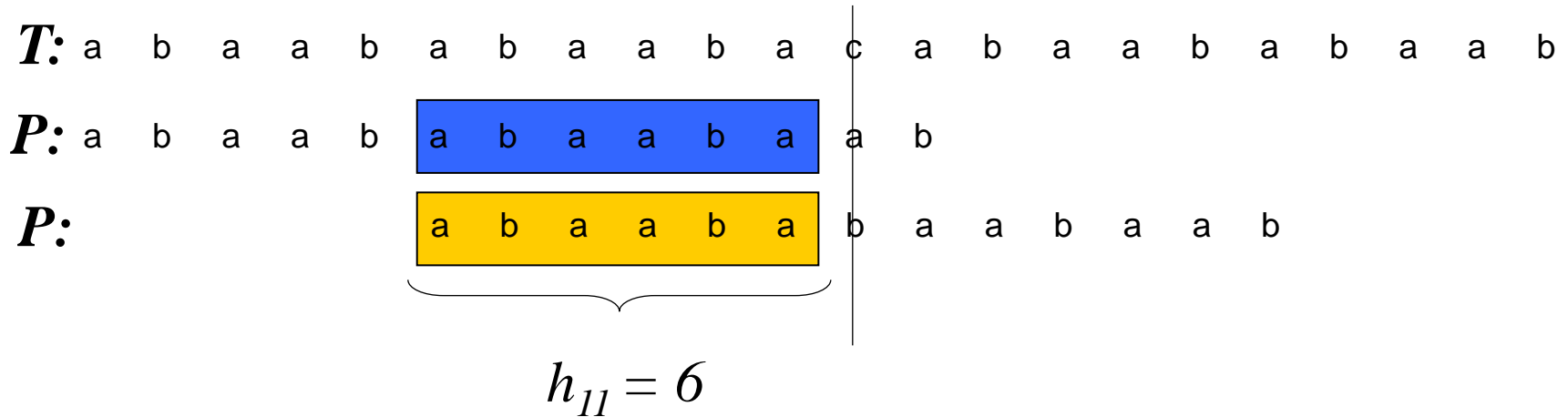


Hence,  $h(11) = 6$ .

If there is no proper suffix of  $P[1, \dots, i]$  with the property mentioned above, then  $h(i) = 0$

# The KMP shift rule

The first mismatch in position  
 $k=12$  of  $T$  and in pos.  $i+1=12$  of  $P$ .



Shift  $P$  to the right so that  $P[1, \dots, h(i)]$  aligns with the suffix  $T[k - h(i), \dots, k - 1]$ .

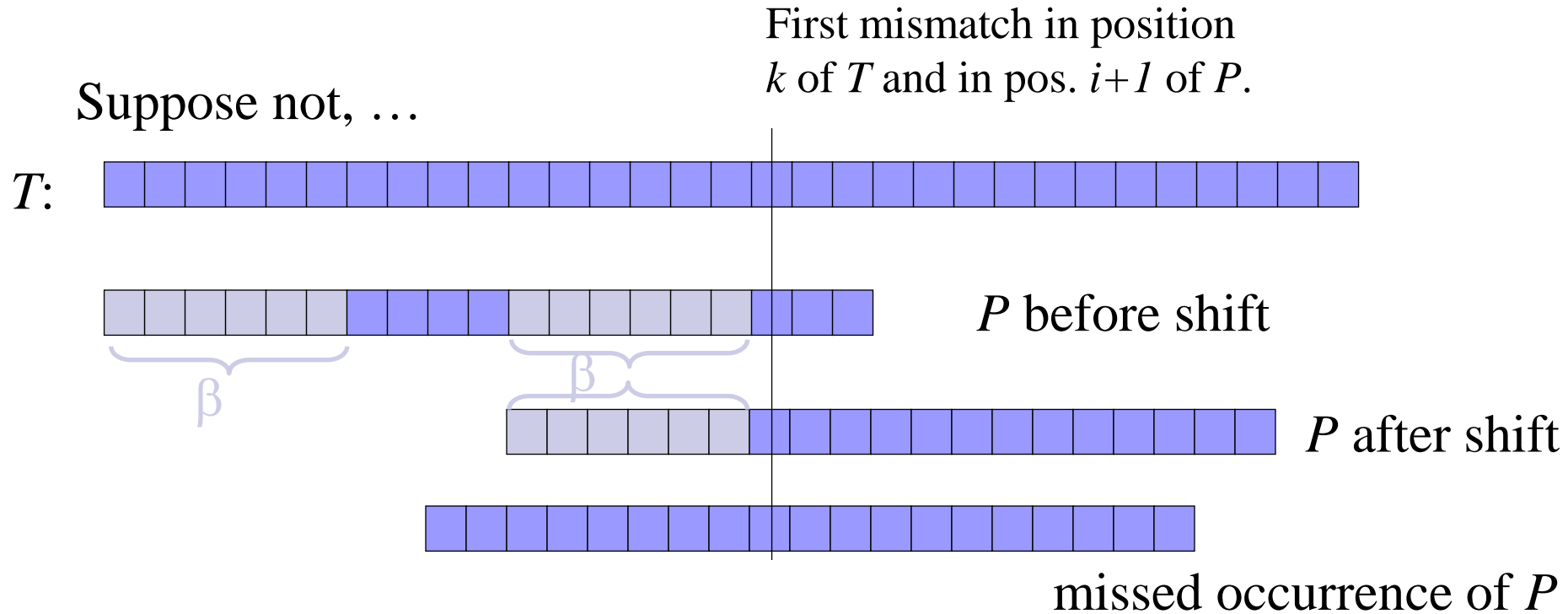
They must match because of the definition of  $h$ .

In other words, shift  $P$  exactly  $i - h(i)$  places to the right.

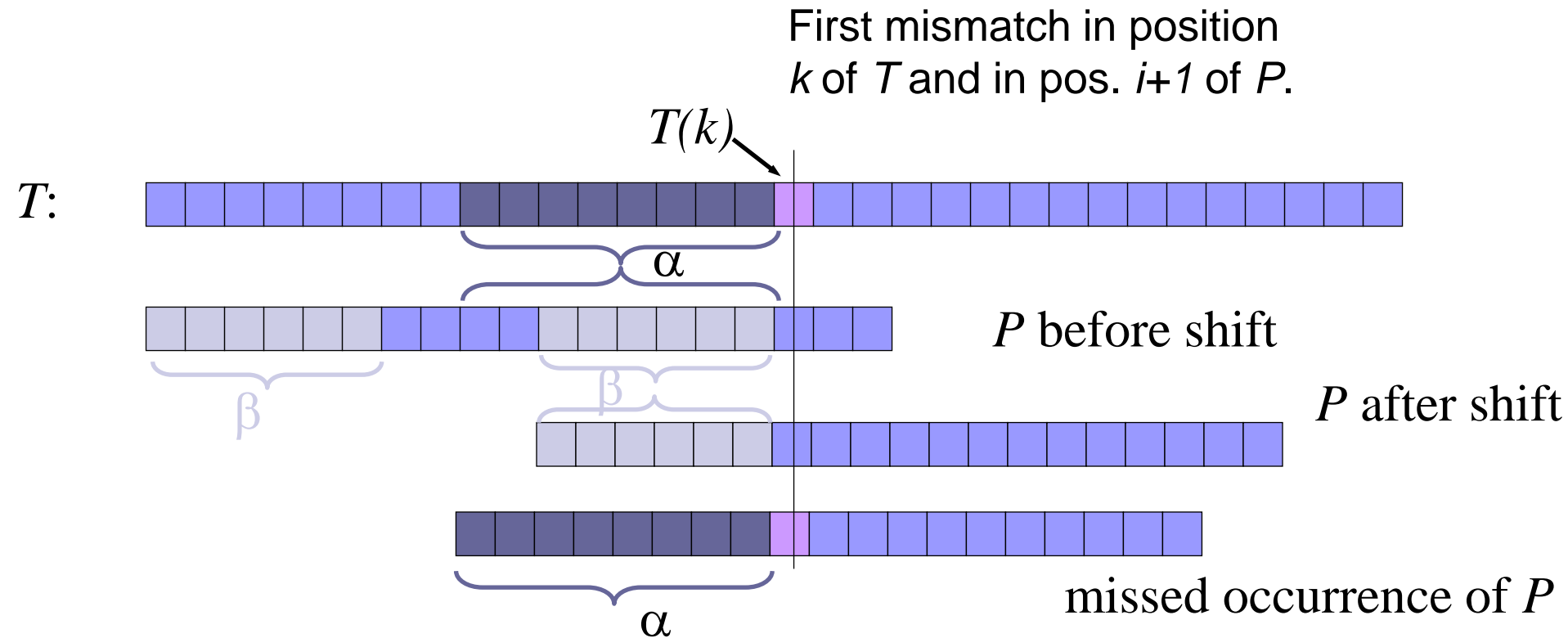
If there is no mismatch, then shift  $P$  by  $m - h(m)$  places to the right.



# The KMP algorithm finds all occurrences of $P$ in $T$ .



# Correctness of KMP.



$$|\alpha| > |\beta| = h(i)$$

It is a contradiction.

# An Example

**Given:**

	1	2	3	4	5	6	7	8	9	10	11	12	13
Pattern:	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>
Array <i>h</i> :	0	0	1	1	2	3	2	3	4	5	6	4	5

**Input string:**

abaababaabacabaababaabaab.

**Scenario 1:**

$i+1 = 12$

↓

a	b	a	a	b	a	b	a	a	b	a	a	b										
a	b	a	a	b	a	b	a	a	b	a	c	a	b	a	a	b	a	a	b	a	a	b

↑

$k = 12$

What is  $h(i) = h(11) = ?$

$$h(11) = 6 \Rightarrow i - h(i) = 11 - 6 = 5$$

**Scenario 2:**

$$i = 6, h(6) = 3$$

$i+1$

↓

a	b	a	a	b	a	b	a	a	b	a	a	b												
a	b	a	a	b	a	b	a	a	b	a	c	a	b	a	a	b	a	b	a	a	b	a	a	b

↑

$k$

# An Example

**Scenario 3:**

$$i = 3, h(3) = 1$$

$i+1$   
↓  
a b a a b a b a a b  
a b a a b a b a a b a c a b a a b a b a a b a a b  
↑  
 $k$

**Subsequently**

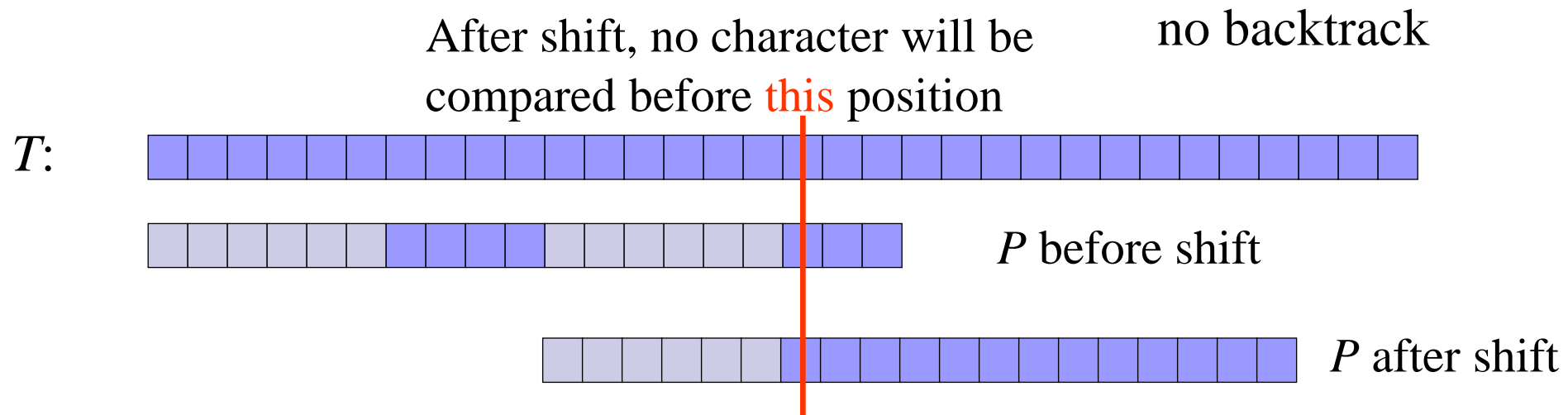
$$i = 2, 1, 0$$

**Finally, a match is found:**

$i+1$   
↓  
a b a a b a b a a b  
a b a a b a b a a b a c a b a a b a b a a b a a b  
↑  
 $k$

# Complexity of KMP

In the KMP algorithm, the number of character comparisons is at most  $2n$ .



In any shift at most one comparison involves a character of *T* that was previously compared.

Hence  $\# \text{comparisons} \leq \# \text{shifts} + |T| \leq 2|T| = 2n$ .

# Computing the failure function

- We can compute  $h(i+1)$  if we know  $h(1)..h(i)$
- To do this we run the KMP algorithm where the text is the pattern with the first character replaced with a \$.
- Suppose we have successfully matched a prefix of the pattern with a suffix of  $T[1..i]$ ; the length of this match is  $h(i)$ .
- If the next character of the pattern and text match then  $h(i+1)=h(i)+1$ .
- If not, then in the KMP algorithm we would shift the pattern; the length of the new match is  $h(h(i))$ .
- If the next character of the pattern and text match then  $h(i+1)=h(h(i))+1$ , else we continue to shift the pattern.
- Since the no. of comparisons required by KMP is length of pattern+text, time taken for computing the failure function is  $O(n)$ .

# Computing h: an example

**Given:**

	1	2	3	4	5	6	7	8	9	10	11	12	13
Failure function $h$ :	0	0	1	1	2	3	2	3	4	5	6	4	5

	1	2	3	4	5	6	7	8	9	10	11	12	13
Text:	\$	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>
Pattern:						<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>	<i>a</i>	<i>b</i>	
Pattern									<i>a</i>	<i>b</i>	<i>a</i>	<i>a</i>	<i>b</i>

h(11)=6  
h(6)=3

- $h(12) = 4 = h(6) + 1 = h(h(11)) + 1$
- $h(13) = 5 = h(12) + 1$

# KMP - Analysis

- The KMP algorithm never needs to backtrack on the text string.

*preprocessing*      *searching*

Time complexity =  $O(m + n)$

Space complexity =  $O(m + n)$ ,

where  $m = |P|$  and  $n = |T|$ .