String Matching

String Matching

- T = abcabaabcabac
- P = abaa
- Goal: find all occurrences of pattern P in text T. Assume T is length n and P is length m.
- Assume all characters are from an alphabet Σ
- P occurs with shift s=3

Naïve String Matcher

- Naïve-String Matcher(T,P)
  - n= length of T
  - m= length of P
  - For s = 0 to n-m
    - If P = T[s+1,…s+m]
      - Print pattern occurs at shift s
- Complexity
  - To check if takes m steps
  - For repeats n-M+1 steps
  - O((n-m+1)m) steps

Algorithms

Algorithm	Preprocessing time	Machine Time
Naïve	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\|\Sigma\|)$	$\Theta(n)$
Knuth-Morris Pratt	$\Theta(m)$	$\Theta(n)$

Knuth-Morris-Pratt

- P = abababaca
- P_5= ababa
- P_3= aba
- P_4= abab
- $\pi(P_5)=3$
- The longest prefix of P that is a proper suffix of P_5 is P_3
- $\pi(P_6)=0$
  - P_6 =ababac

Compute-Prefix, π

```
Compute-Prefix-Function (P)
1  m ≈ length[P]
2 π[0] ≈ 0
3  k ≈ 0
4      for q ≈ 1 to m-1
5              while k > 0 and P[k] ≠ P[q]
6                      k ≈ π[k-1]
7              If P[k] = P[q]
8                    then k ≈ k +1
9              π[q] ≈ k
10     return π
```

Compute-Prefix, π

```
Compute-Prefix-Function (P)
1  m ≈ length[P]
2 π[0] := 0
3      k := 0
4      q :=1
4       while q < m
5                   If k= 0 and P[k] != P[q]
6                           then
7                              π[q] := 0 and q++
6
                  else If P[k] == P[q]
8                   then
9                      k ++
9                      π[q] := k
10                     q ++
11                  else k := π[k-1]
10    return π
```

Example: compute π for the pattern P:

p	a	b	a	b	a	c	a

Initially: m = length[p] = 7
π[1] = 0
k = 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0					

Step 1: q = 2, k=0

- P[k+1] ≠ P[q]
- π[2] := 0

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1				

Step 2: q = 3, k = 0

- P[k+1] = P[q], set k=k+1
- π[3] = 1

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	A
π	0	0	1	2			

Step 3: q = 4, k = 1
π[4] = 2

Step 4: q = 5, k =2

- P[k+1] = P[q]
  - set k=3 and $\pi[5] = 3$

Step 5: q = 6, k = 3

- P[k+1] ≠ P[q], set k= $\pi[3]=1$
- P[k+1] ≠ P[q], set k= $\pi[1]=0$
- Set $\pi[6] = 0$

Step 6: q = 7, k = 0

$\pi[7] = 1$

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3		

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	

q	1	2	3	4	5	6	7
p	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

KMP-Matcher(T,P)

```
KMP-Matcher(T,P)
1 n ≈ length[T]
2 m ≈ length[P]
3 π ≈ Compute-Prefix-Function(P)
4 q ≈ 0                              //number of characters matched
5 for i ≈ 1 to n                     //scan text from left to right
6       while  q > 0 and P[q+1] ≠ T[i]
7                 q ≈ π[q]           //next character does not match
8         if P[q+1] = T[i]
9            then q ≈ q + 1          //next character matches
10        if q = m                   //is all of P matched?
11           then print "Pattern occurs with shift" i – m
12                q ≈ π[ q]          // look for the next match
```

T

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

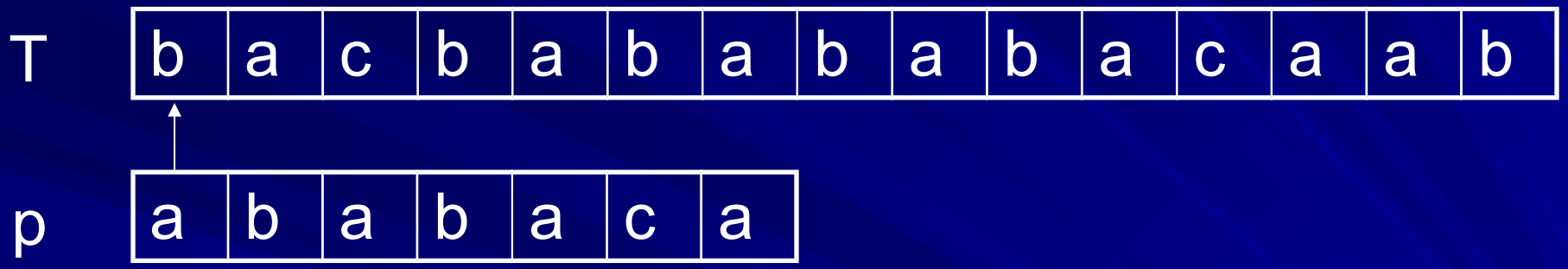
p

a	b	a	b	a	c	a

q	1	2	3	4	5	6	7
p	a	b	A	b	a	c	a
π	0	0	1	2	3	0	1

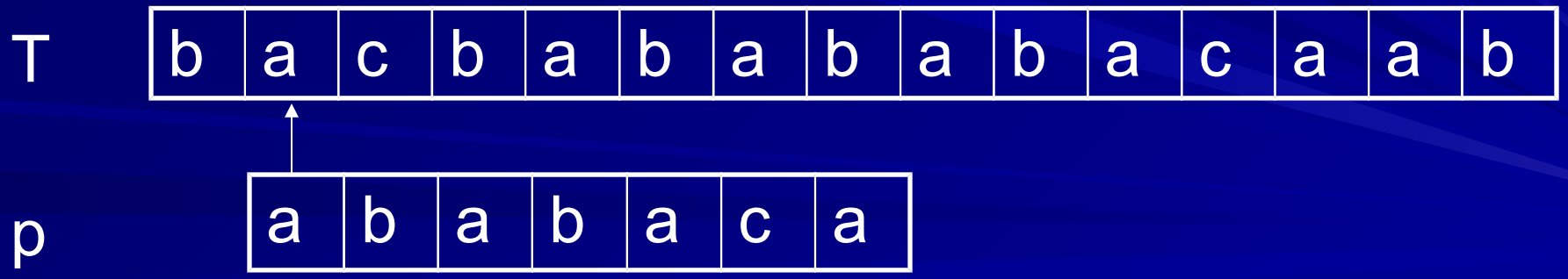
n = 15;
m = 7

Step 1: i = 1, q = 0



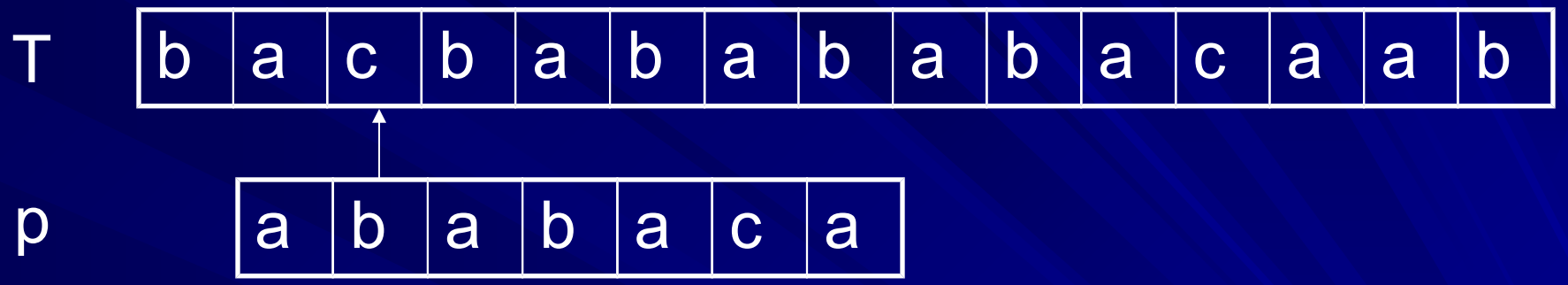
P[1] does not match with T[1]. P shifted one position to the right.

Step 2: i = 2, q = 0

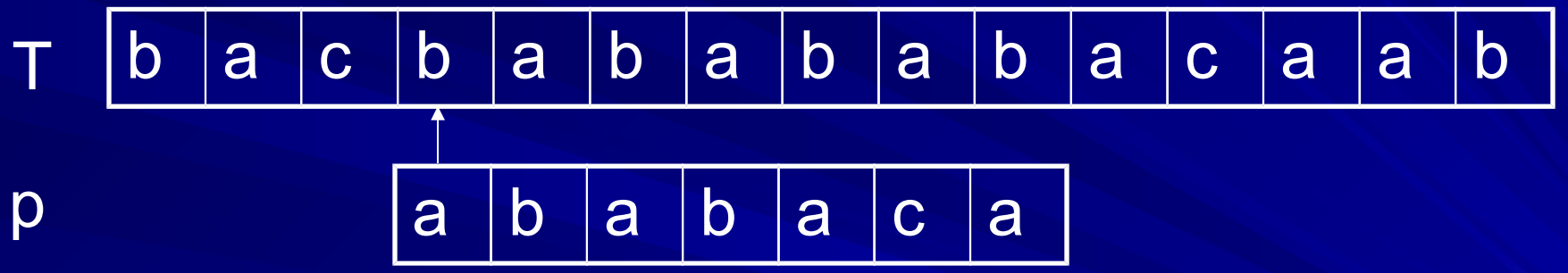


P[1] matches T[2]. Since there is a match, p is not shifted.

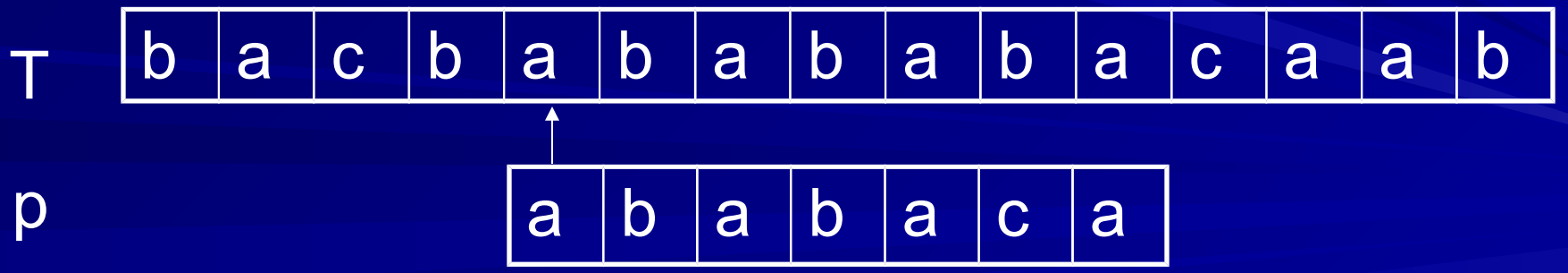
Step 3: i = 3, q = 1



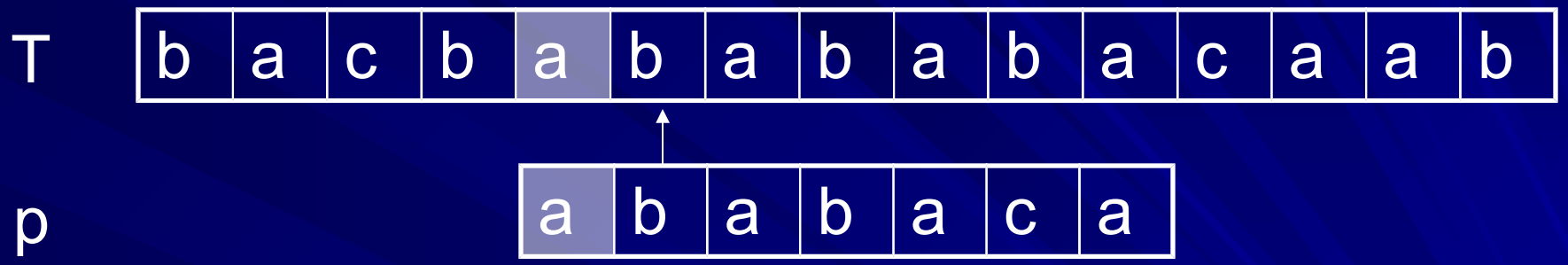
Step 4: i = 4, q = 0



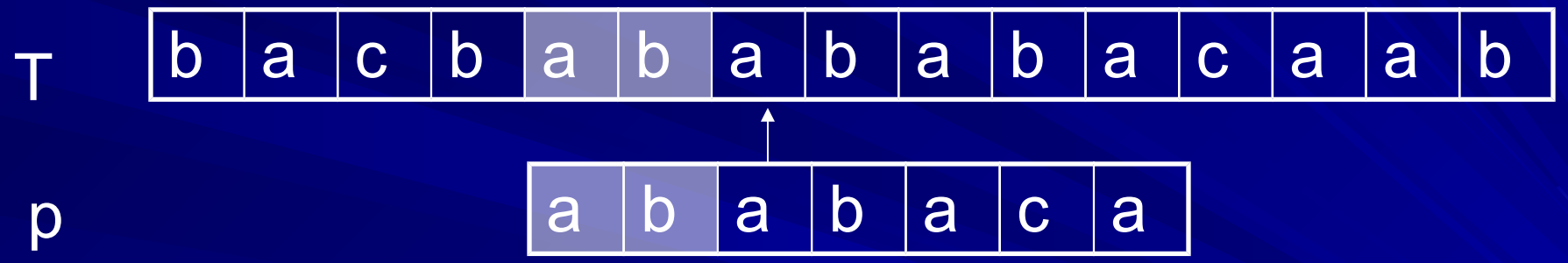
Step 5: i = 5, q = 0



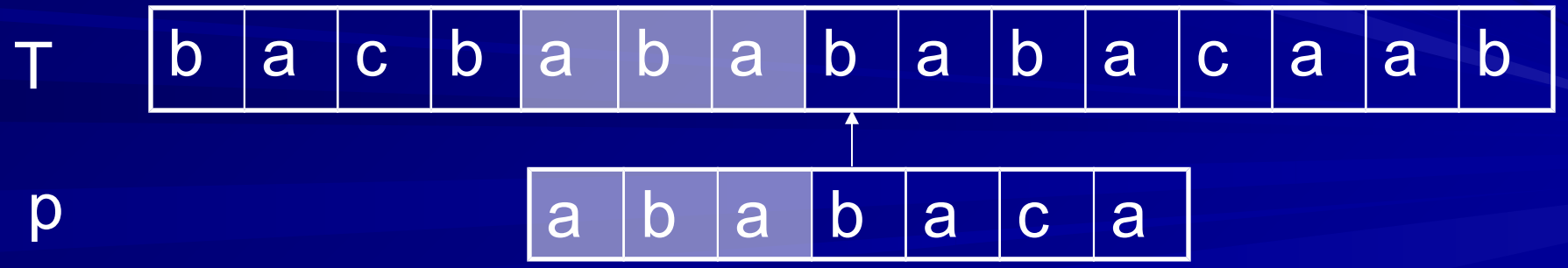
Step 6: i = 6, q = 1



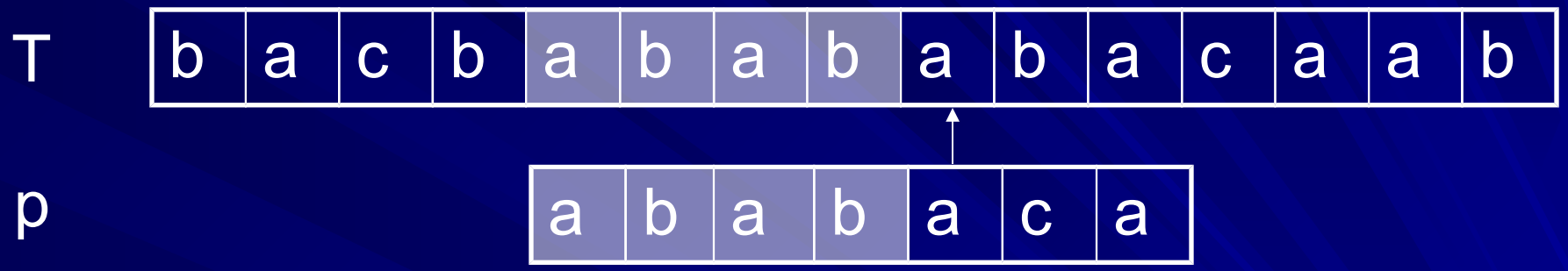
Step 7: i = 7, q = 2



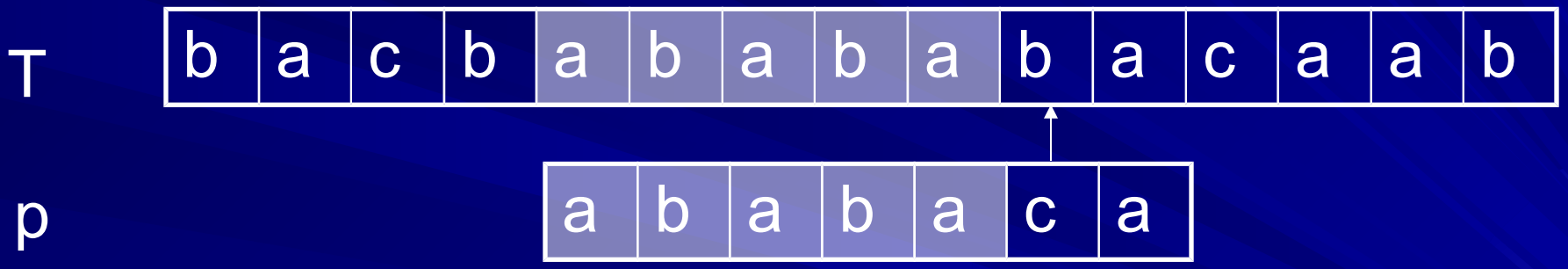
Step 8: i = 8, q = 3



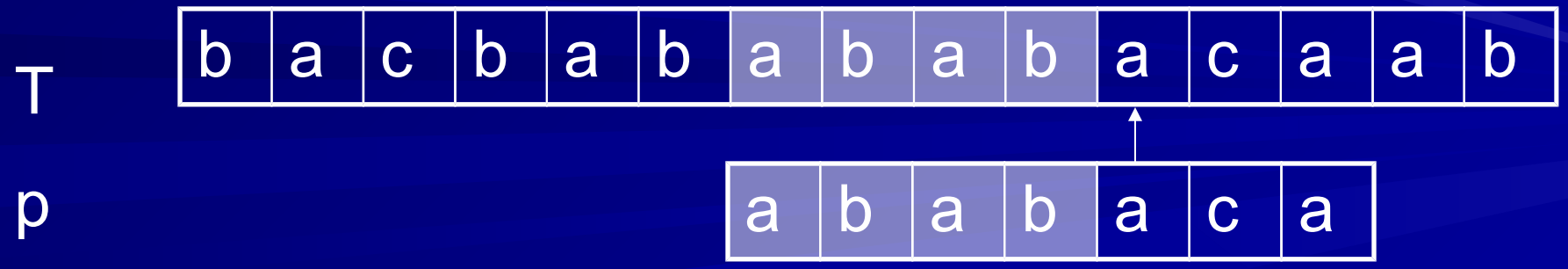
Step 9: i = 9, q = 4



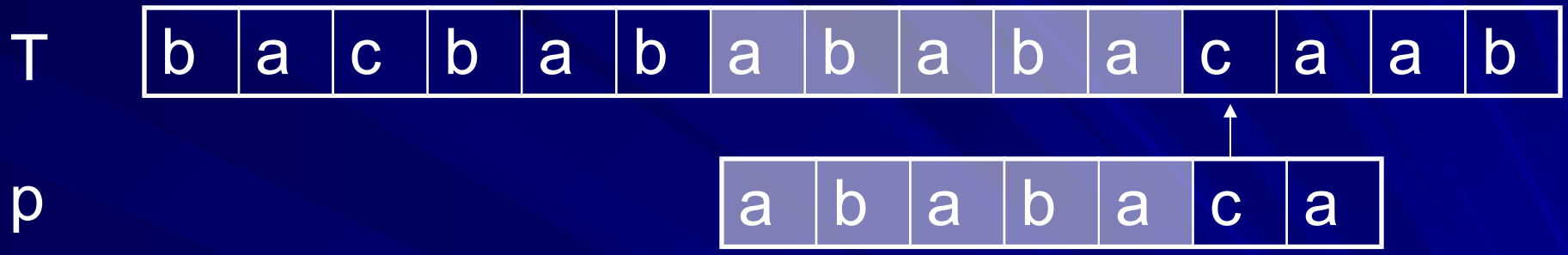
Step 10: i = 10, q = 5



Step 11: i = 11, q = 4



Step 12: i = 12, q = 5



Step 13: i = 13, q = 6