NCKU Programming Contest Training Course
2013/08/11

Pin-Chieh Huang (free999)

http://myweb.ncku.edu.tw/~p76014143/20130811_KMP.rar

Department of Computer Science and Information Engineering
National Cheng Kung University
Tainan, Taiwan

Outline

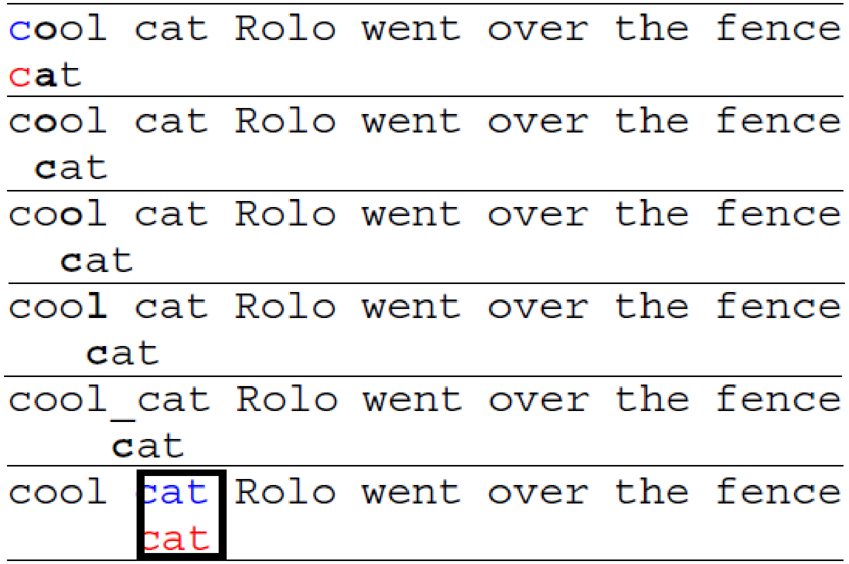
Rabin-Karp Algorithm

KMP Algorithm

String Matching

- Judge if a given string is the substring of another string
  - “abcde” is the substring of “reabcdeef”
  - “abcde” is not the substring of “aibeckdle”
  - “abcde” is the subsequence of “aibeckdle”

- Brute Force Method
  - For loop with O(?)



```
cool cat Rolo went over the fence
cat
cool cat Rolo went over the fence
 cat
cool cat Rolo went over the fence
  cat
cool cat Rolo went over the fence
   cat
cool_cat Rolo went over the fence
    cat
cool cat Rolo went over the fence
     cat
```

Rabin-Karp Algorithm

- Rabin-Karp Algorithm
  - Hash a pattern
  - {A, B, C, …, Z} = {0, 1, 2, …, 25}

- Hash Technique
  - Choose two prime *p* and *q*
  - ABC = {0, 1, 2} → 0*(*p*^2) + 1*(*p*^1) + 2*(*p*^0)
  - ABCDE → 0*(*p*^4) + 1*(*p*^3) + 2*(*p*^2) + 3*(*p*^1) + 4*(*p*^0)

- If too large
  - Mod q

Rabin-Karp Algorithm

- So…Given a string and matching pattern, how to efficiently find the matching?
  - string = “ABCDEFGHIJ”
  - pattern = “EFG”

A	B	C	D	E	F	G	H	I	J

Design complexity ?

Rabin-Karp Algorithm

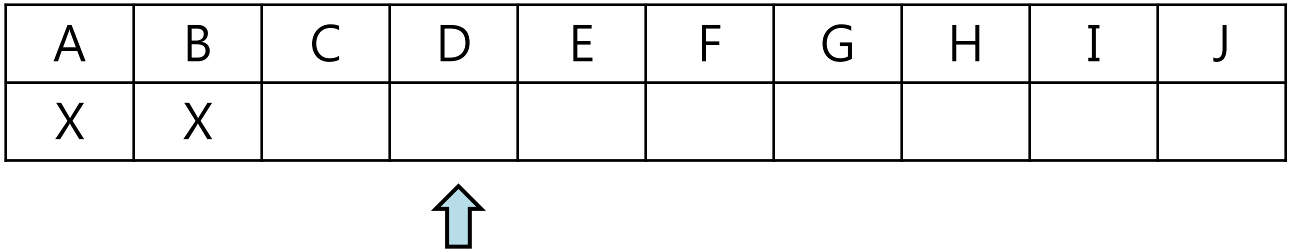
Matching pattern: EFG = 4(p^2) + 5(p^1) + 6(p^0)

A	B	C	D	E	F	G	H	I	J
X	X								

HASH: ABC = 0(p^2) + 1(p^1) + 2(p^0)

Rabin-Karp Algorithm

Matching pattern: EFG = 4(p^2) + 5(p^1) + 6(p^0)

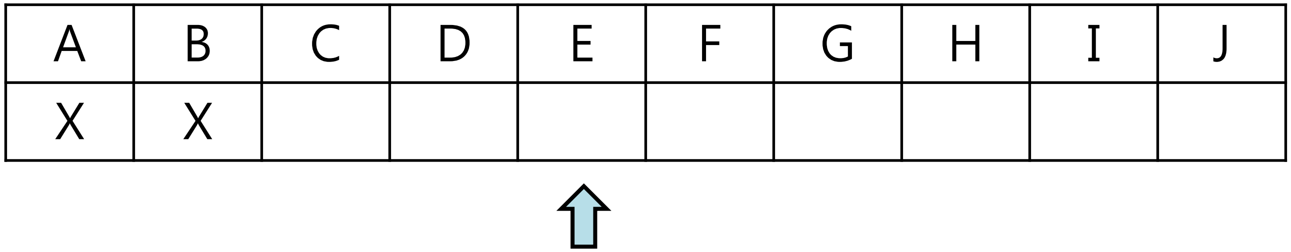


A	B	C	D	E	F	G	H	I	J
X	X								

HASH: BCD = 1(p^2) + 2(p^1) + 3(p^0)

Rabin-Karp Algorithm

Matching pattern: EFG = 4(p^2) + 5(p^1) + 6(p^0)



A	B	C	D	E	F	G	H	I	J
X	X								

HASH: CDE= 2(p^2) + 3(p^1) + 4(p^0)

Rabin-Karp Algorithm

Matching pattern: EFG = 4(p^2) + 5(p^1) + 6(p^0)

A	B	C	D	E	F	G	H	I	J
X	X								

⇧

HASH: DEF= 3(p^2) + 4(p^1) + 5(p^0)

Rabin-Karp Algorithm

HASH: **ABC = 0(p^2) + 1(p^1) + 2(p^0)**

↓

HASH: **BCD = 1(p^2) + 2(p^1) + 3(p^0)**

↓

HASH: **CDE= 2(p^2) + 3(p^1) + 4(p^0)**

↓

HASH: **DEF= 3(p^2) + 4(p^1) + 5(p^0)**

Step 1. k = 0(p^2) + 1(p^1) + 2(p^0) – 0(p^2)
Step 2. k*p
Step 3. k*p + D*(p^0)
Get: 1(p^2) + 2(p^1) + 3(p^0)

Design Complexity: O(?)

Rabin-Karp Algorithm

- Main Algorithm

How to choose the two primes ?

```
RABIN-KARP-MATCHER($T$, $P$, $d$, $q$)
$n \leftarrow length[T]$
$m \leftarrow length[P]$
$h \leftarrow d^{m-1} \bmod q$
$p \leftarrow 0$
$t_0 \leftarrow 0$
for $i \leftarrow 1$ to $m$                          ▹ Preprocessing.
    do $p \leftarrow (dp + P[i]) \bmod q$
       $t_0 \leftarrow (dt_0 + T[i]) \bmod q$
for $s \leftarrow 0$ to $n - m$                      ▹ Matching.
    do if $p = t_s$
          then if $P[1 □ m] = T[s + 1 □ s + m]$
                  then print "Pattern occurs with shift" $s$
       if $s < n - m$
          then $t_{s+1} \leftarrow (d(t_s - T[s + 1]h) + T[s + m + 1]) \bmod q$
```

Example

- POJ 1200

2D Rabin-Karp

- 2D extension (**ural 1486**)

Text:
abcde
edcba
dcbea
abcde
edeca

Pattern:
dc
cb

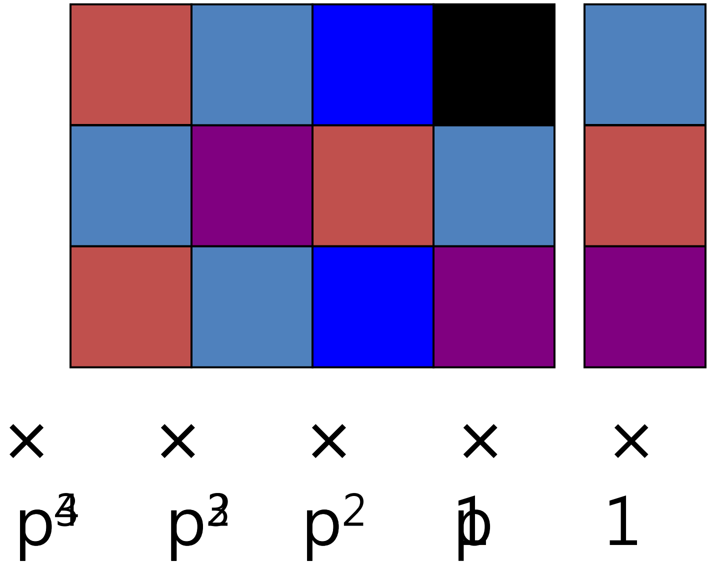
2D Rabin-Karp

- Extend to 2D

$a \times p^2 q^3$	$b \times p^2 q^2$	$a \times p^2 q$	$a \times p^2$
$c \times p q^3$	$b \times p q^2$	$b \times pq$	$c \times p$
$a \times q^3$	$b \times q^2$	$a \times q$	c

2D Rabin-Karp

- Extend to 2D



made by electron & free999

Outline

Rabin-Karp Algorithm

KMP Algorithm

KMP Algorithm

- KMP Algorithm
  - Two-Stage Technique
  - differs from the brute-force algorithm by keeping track of information gained from previous comparisons
  - **Shifting idea: avoid non-necessary moving and comparison**

- First Stage
  - Prefix function

- Second Stage
  - Matching

KMP Algorithm

- First Stage
  - Prefix Function pi[i]
  - The longest prefix of the current suffix

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3	0	1

KMP Algorithm

- First Stage
  - Prefix Function pi[i]
  - The longest prefix of the current suffix

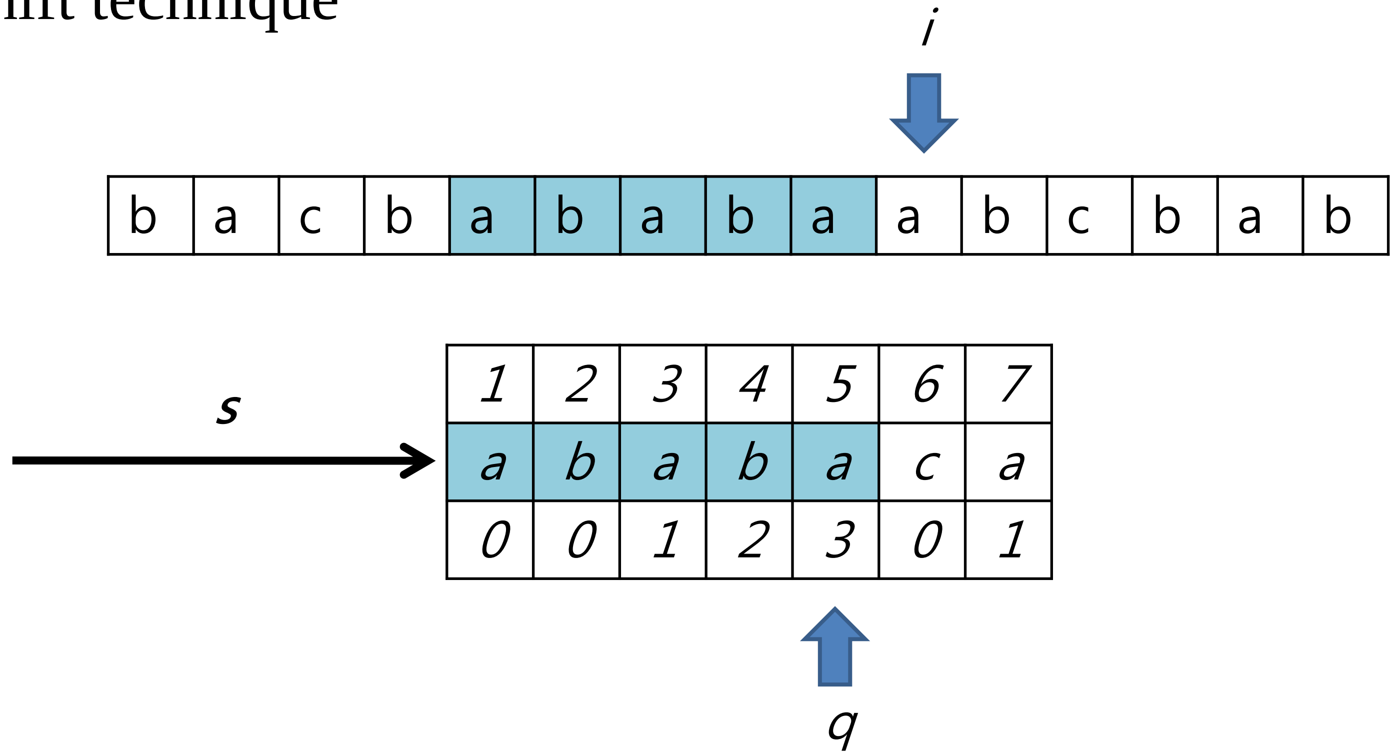
i	*1*	*2*	*3*	*4*	*5*	*6*	*7*
pattern	*a*	*b*	*a*	*b*	*a*	*c*	*a*
Pi[i]	*0*	*0*	*1*	*2*	*3*	*0*	*1*

Pi[4] = 2 means: *ab (1~2) = ab (3~4)*

Pi[4] = 2 means: *prefix = suffix*

KMP Algorithm

- Shift technique

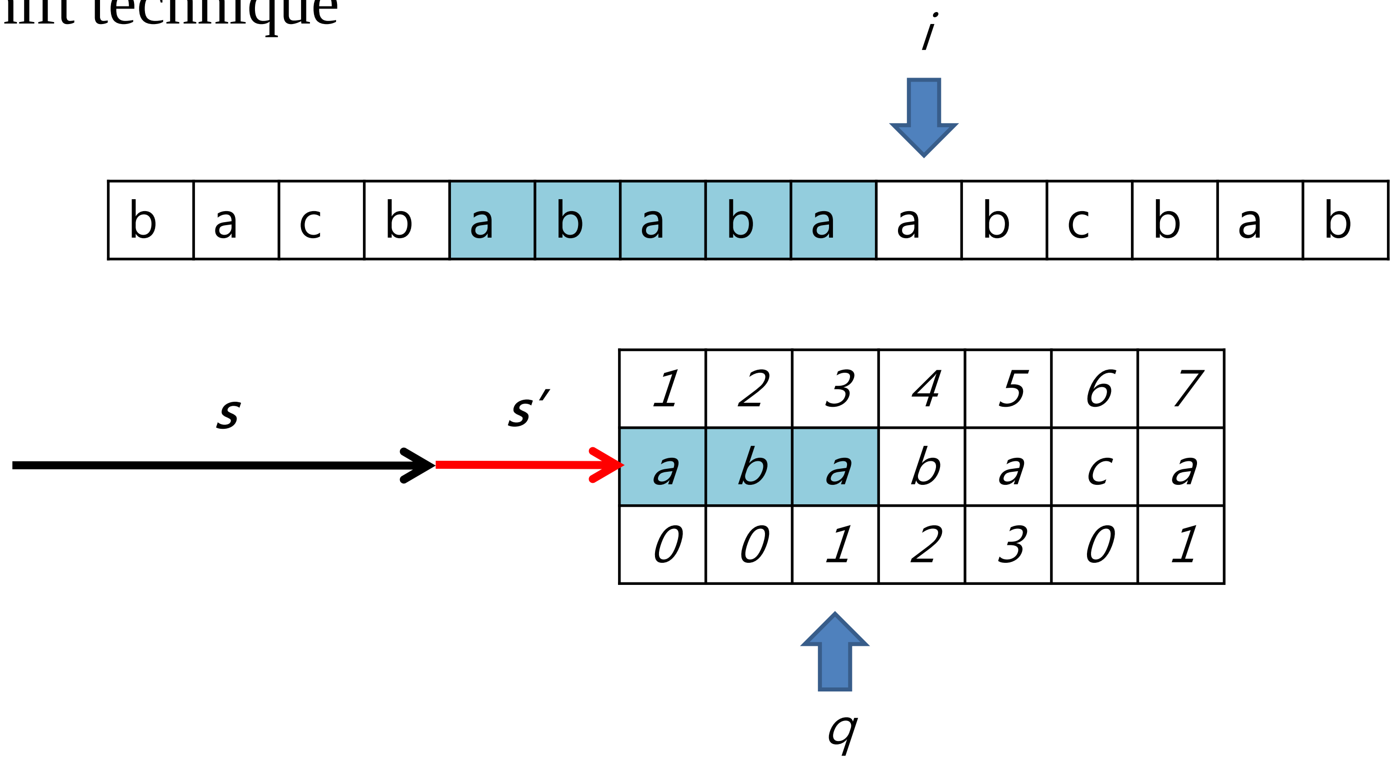


b	a	c	b	a	b	a	b	a	a	b	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

- Shift technique



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

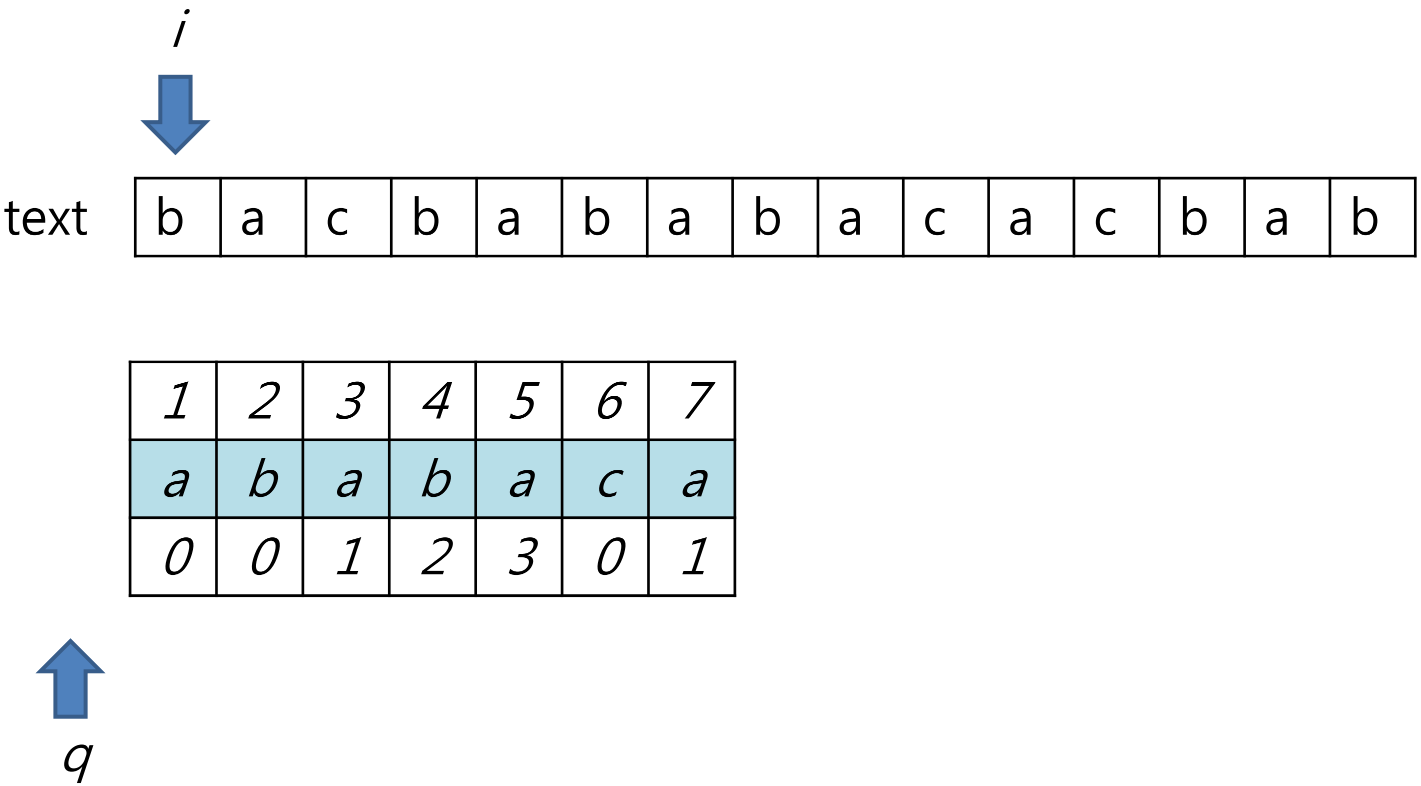
1	2	3	4	5	6	7
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

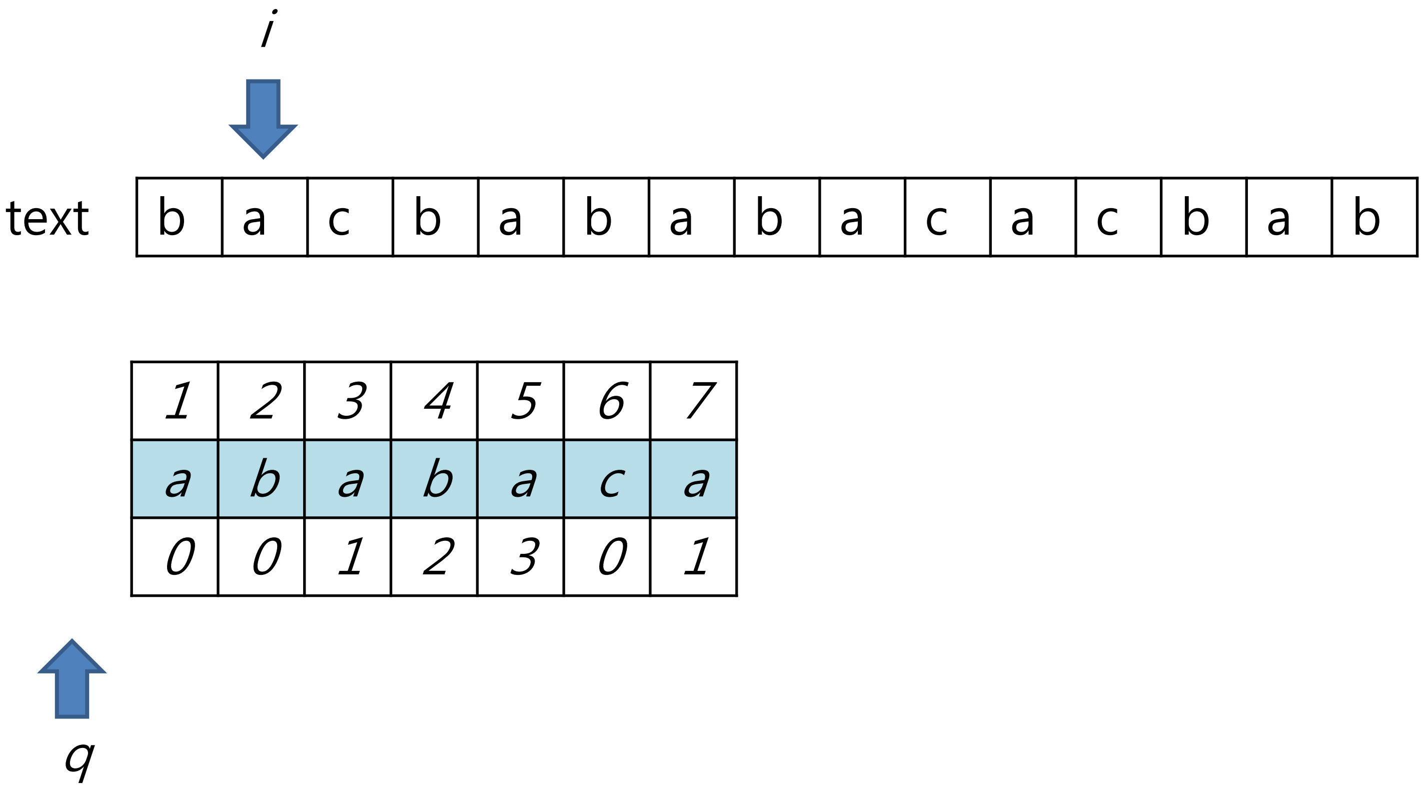


text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

Pattern[q+1] != text[i] → Tune the prefix function

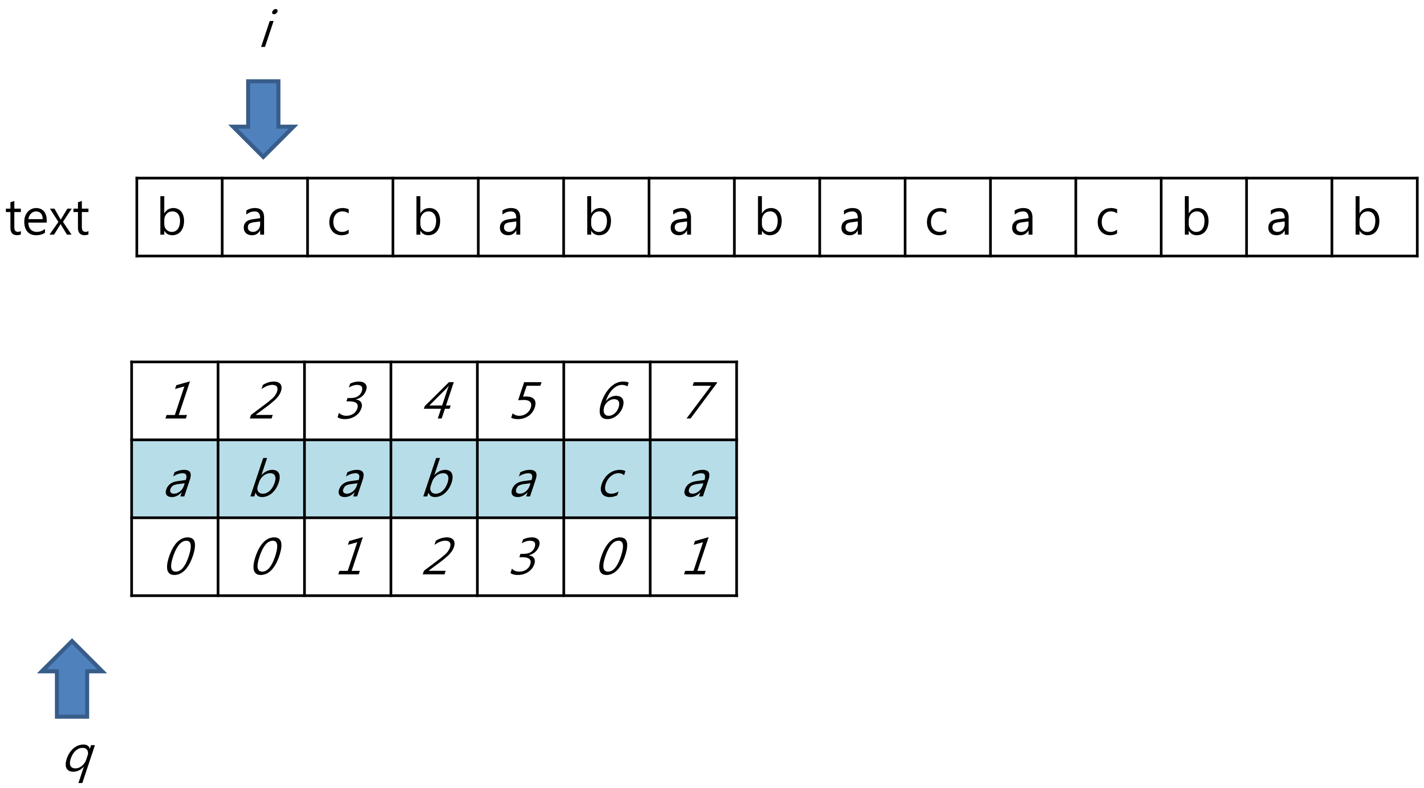
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

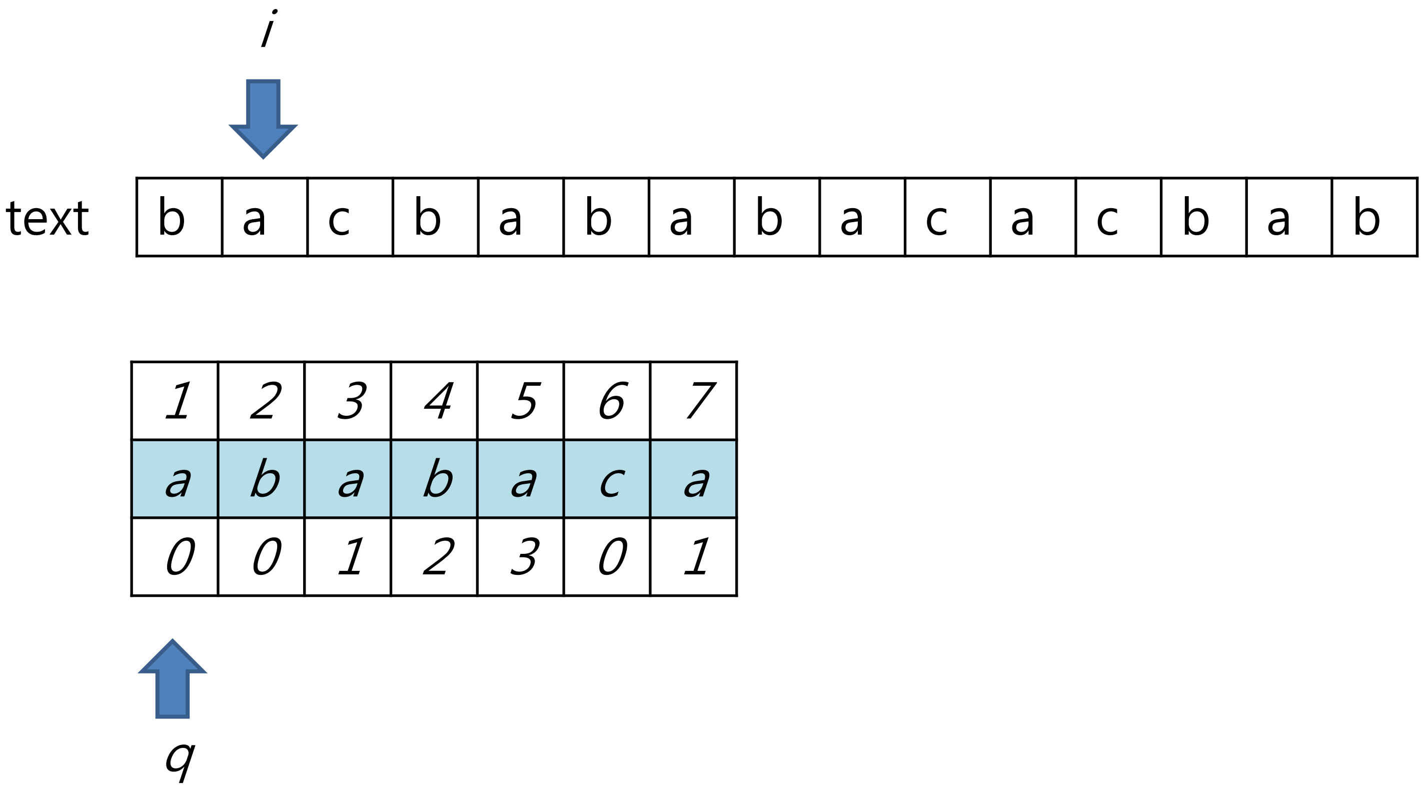


text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

Pattern[q+1] = text[i] → increase q

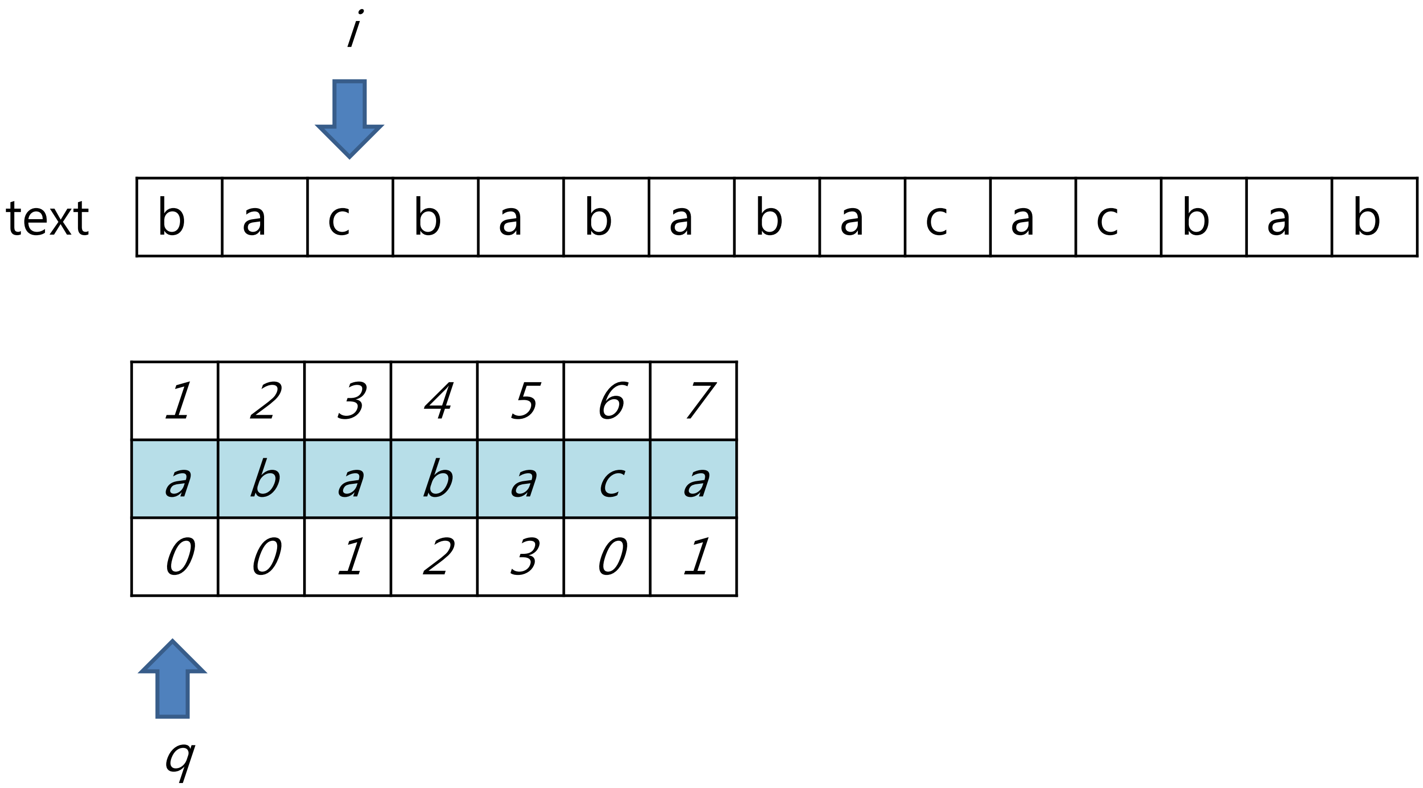
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

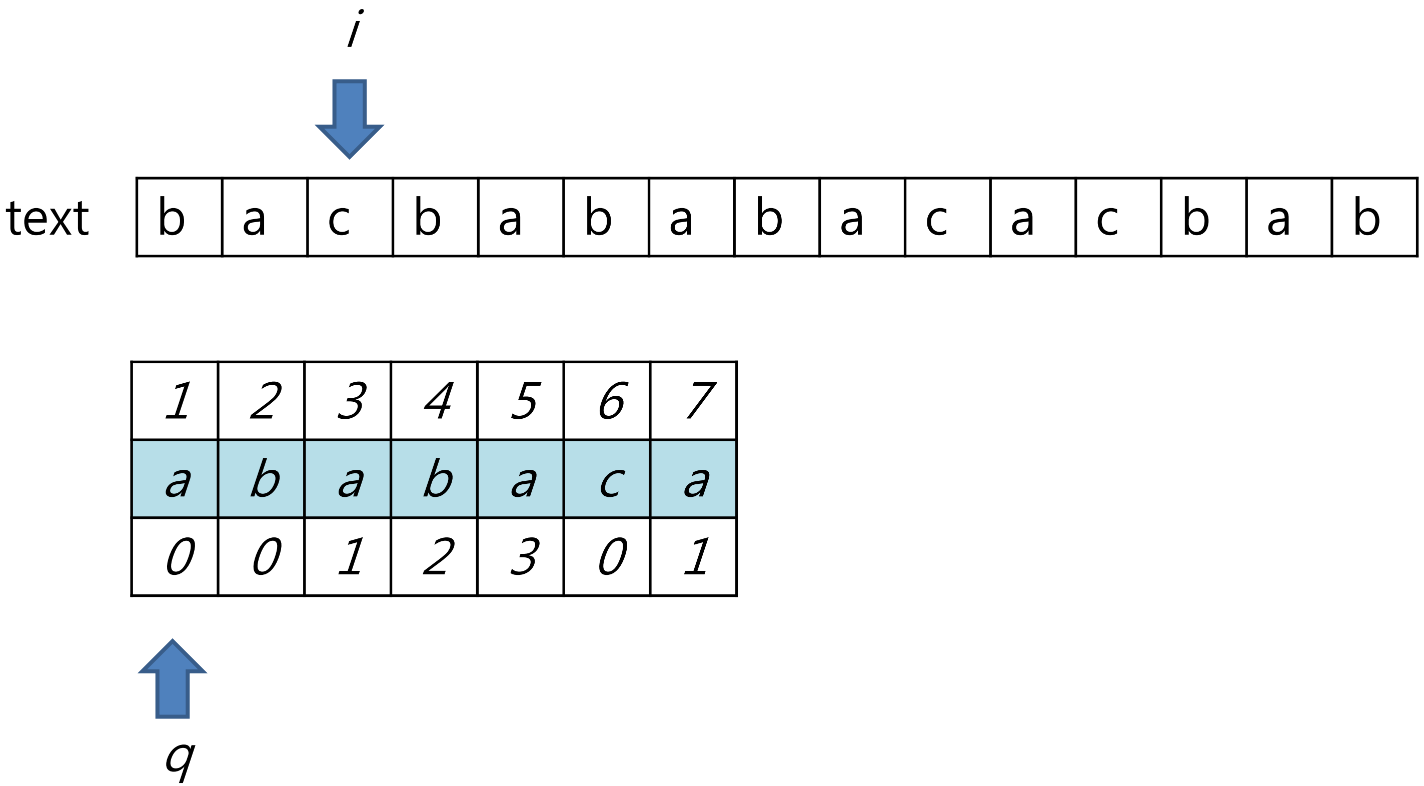
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

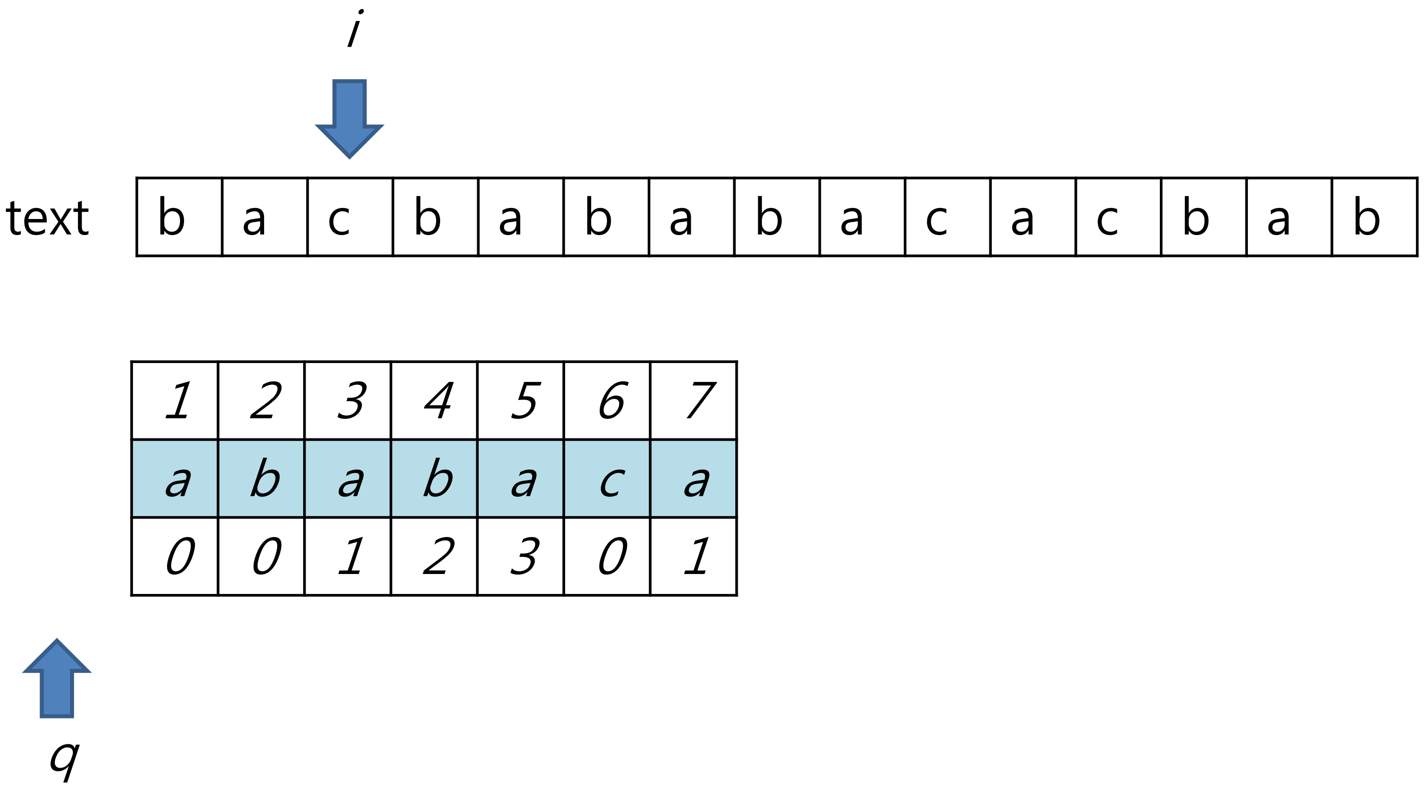


text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

Pattern[q+1] != text[i] → tune the prefix function

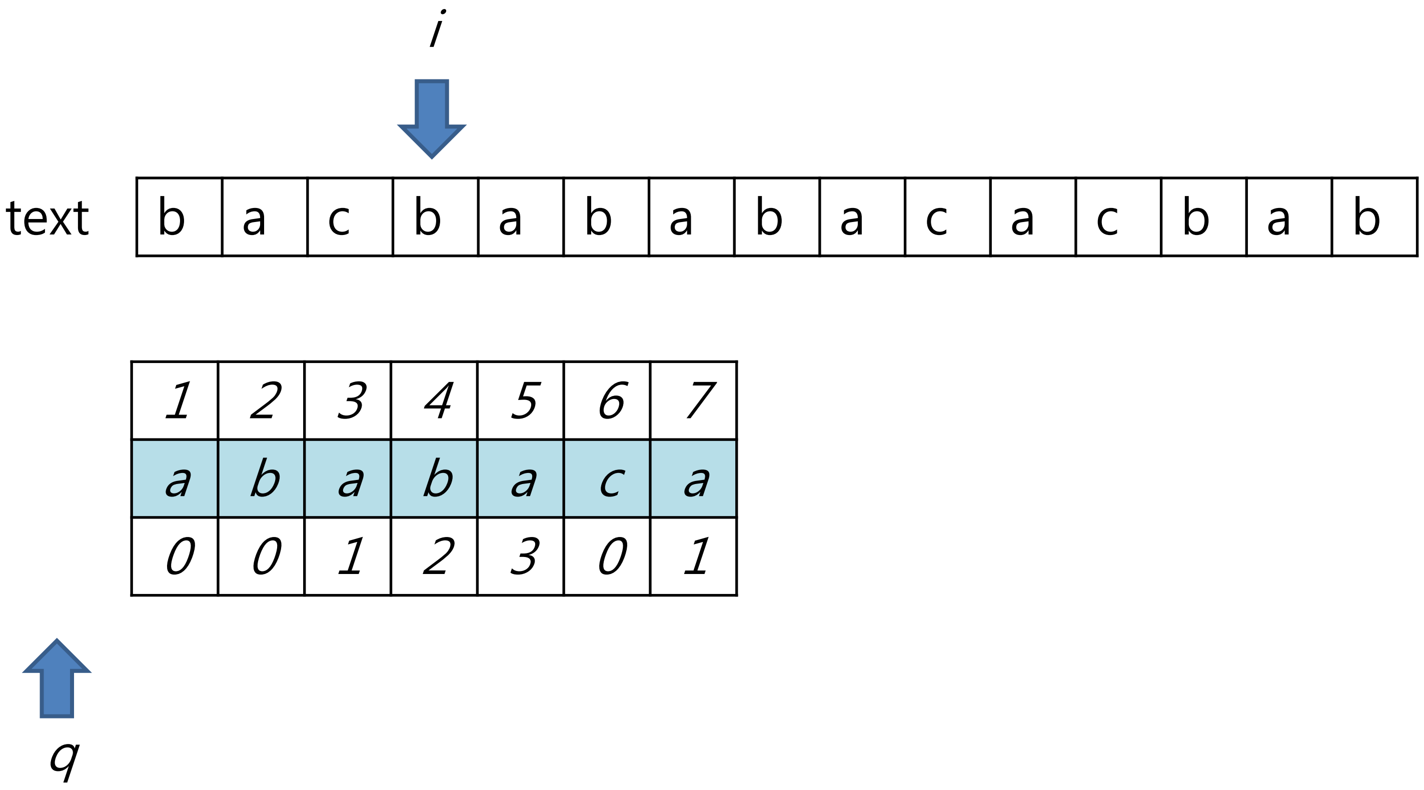
KMP Algorithm



text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

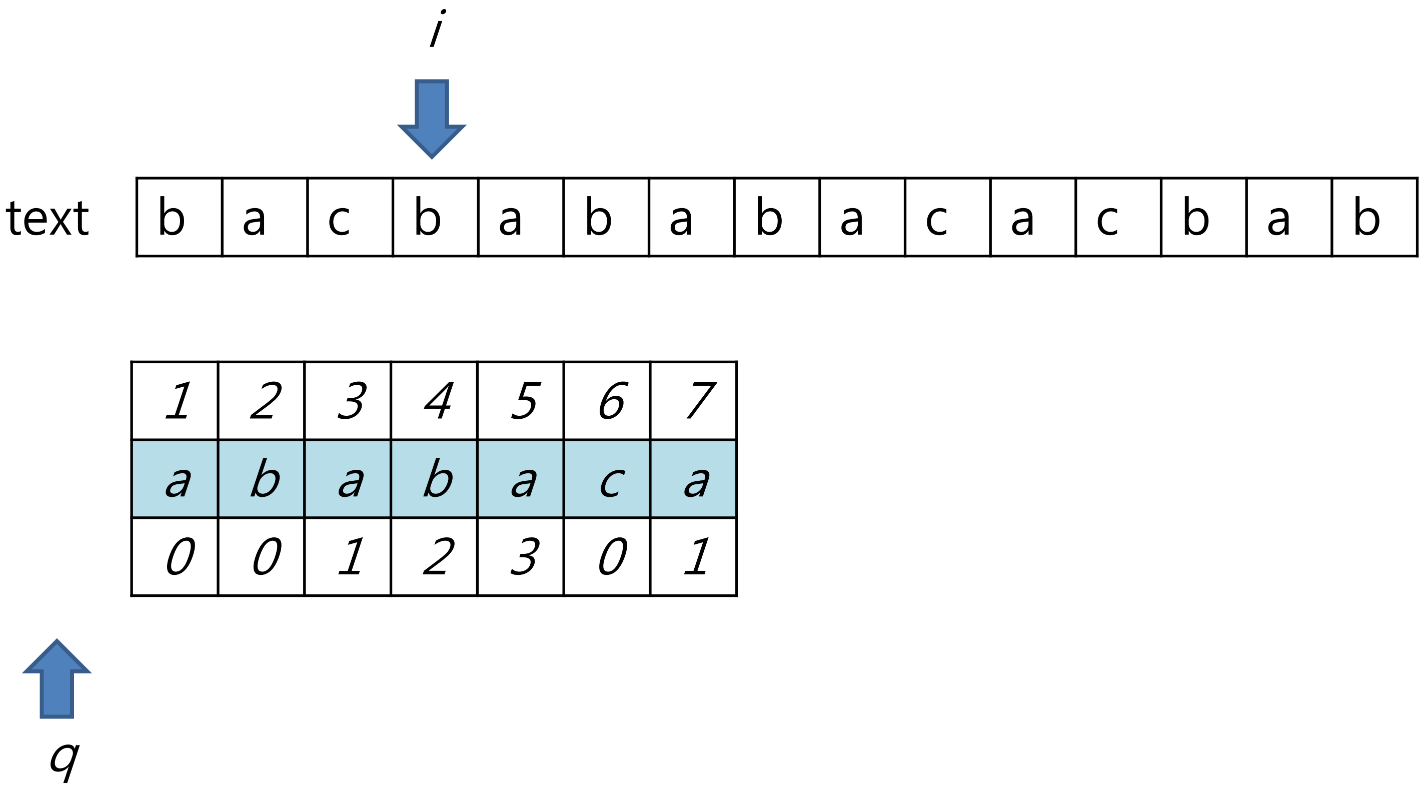
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

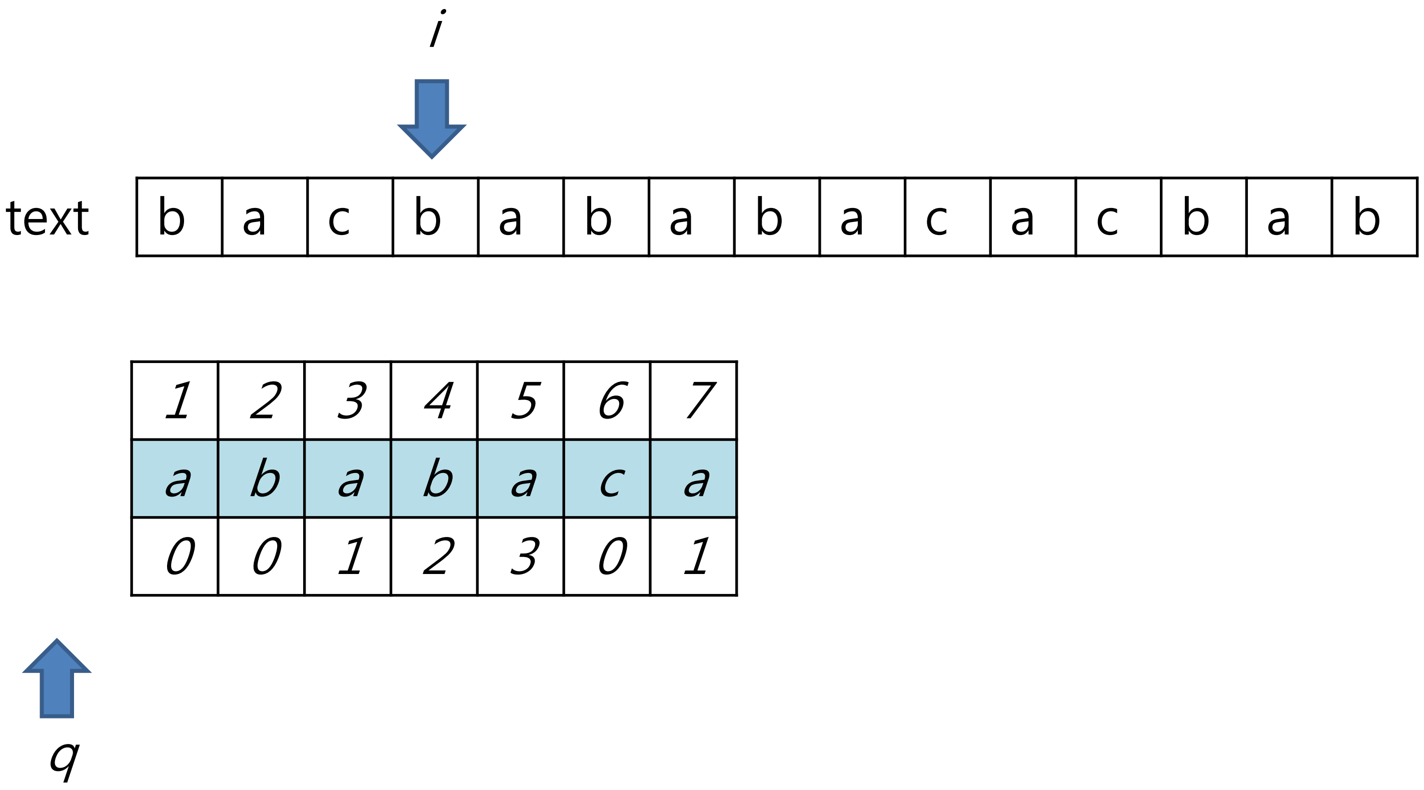


text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

Pattern[q+1] != text[i] → tune the prefix function

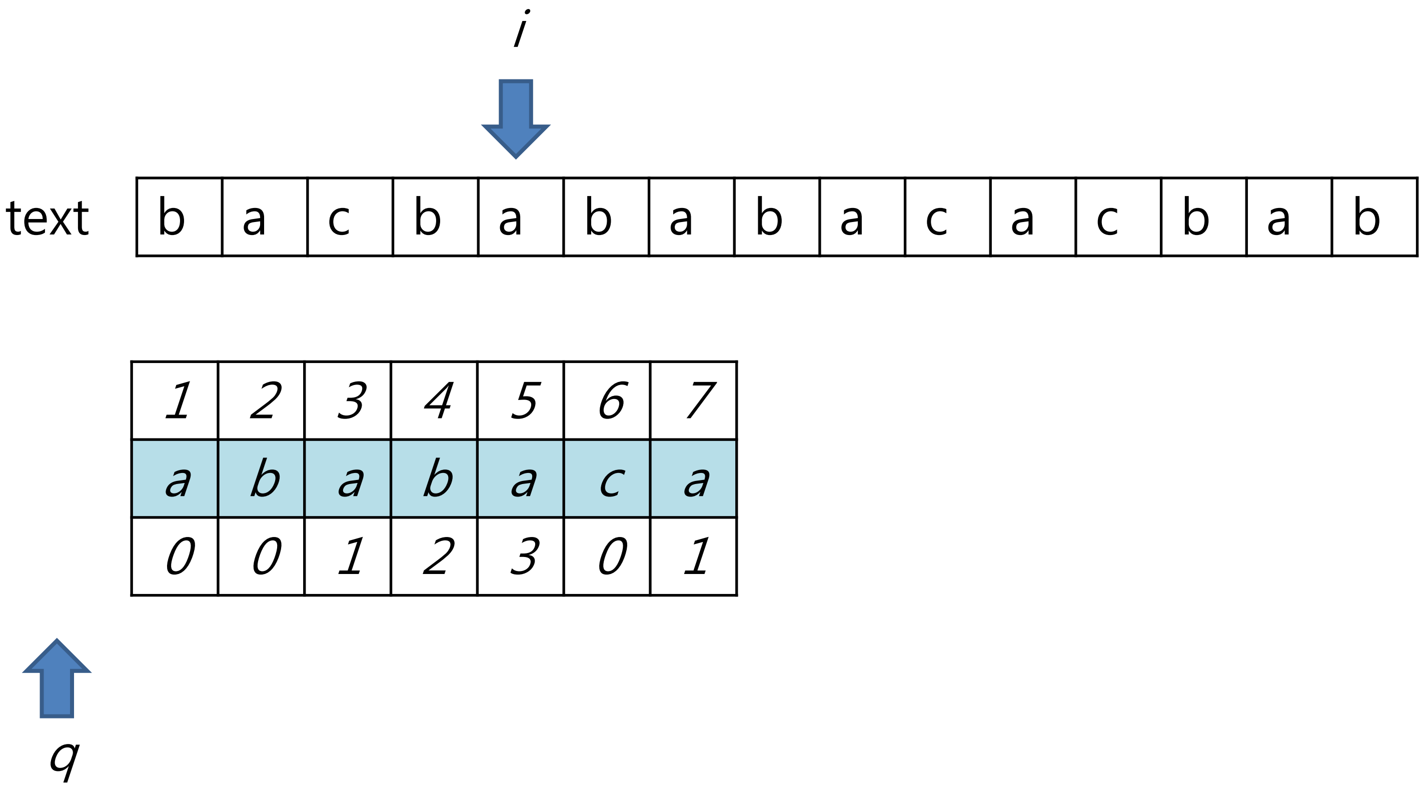
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

KMP Algorithm



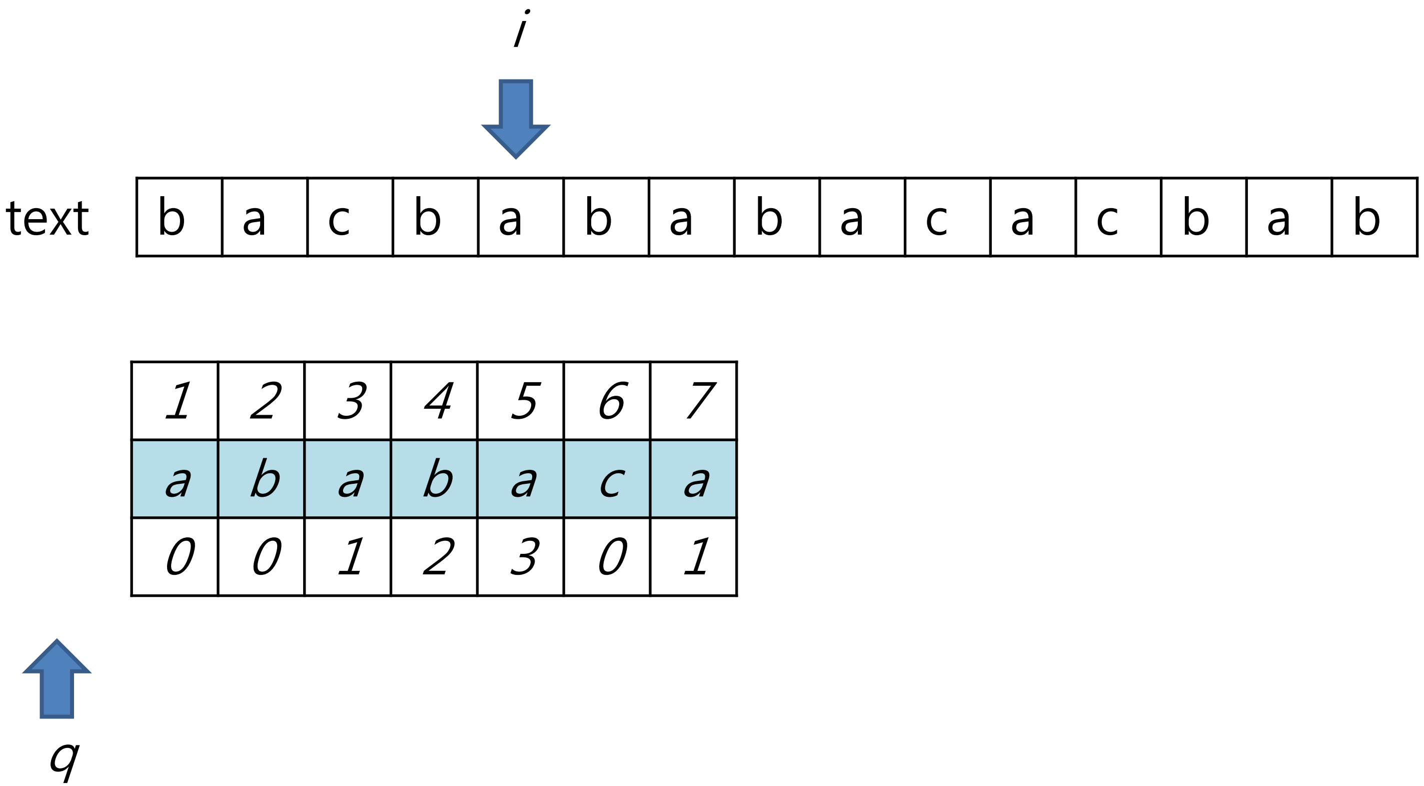
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

KMP Algorithm

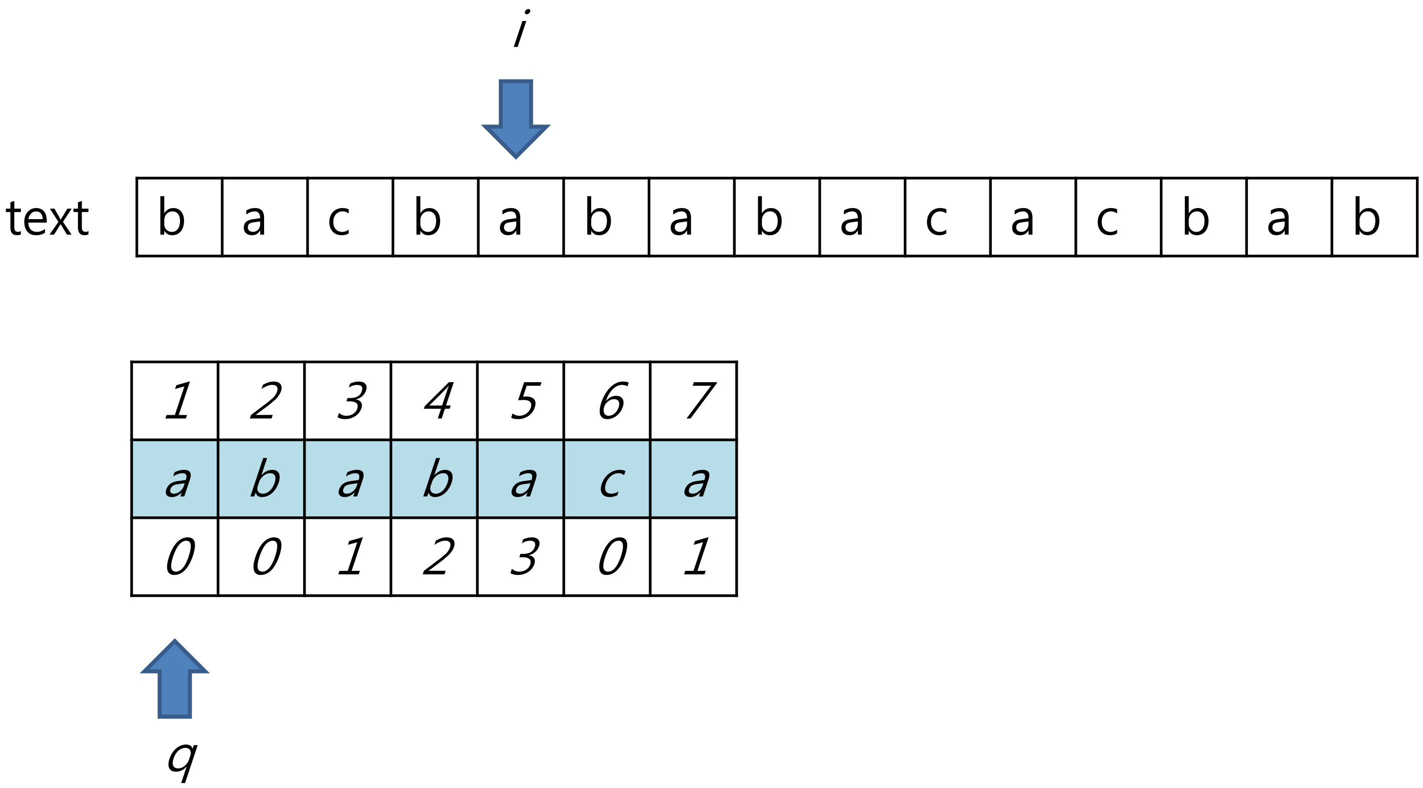


text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

Pattern[q+1] = text[i] → increase q

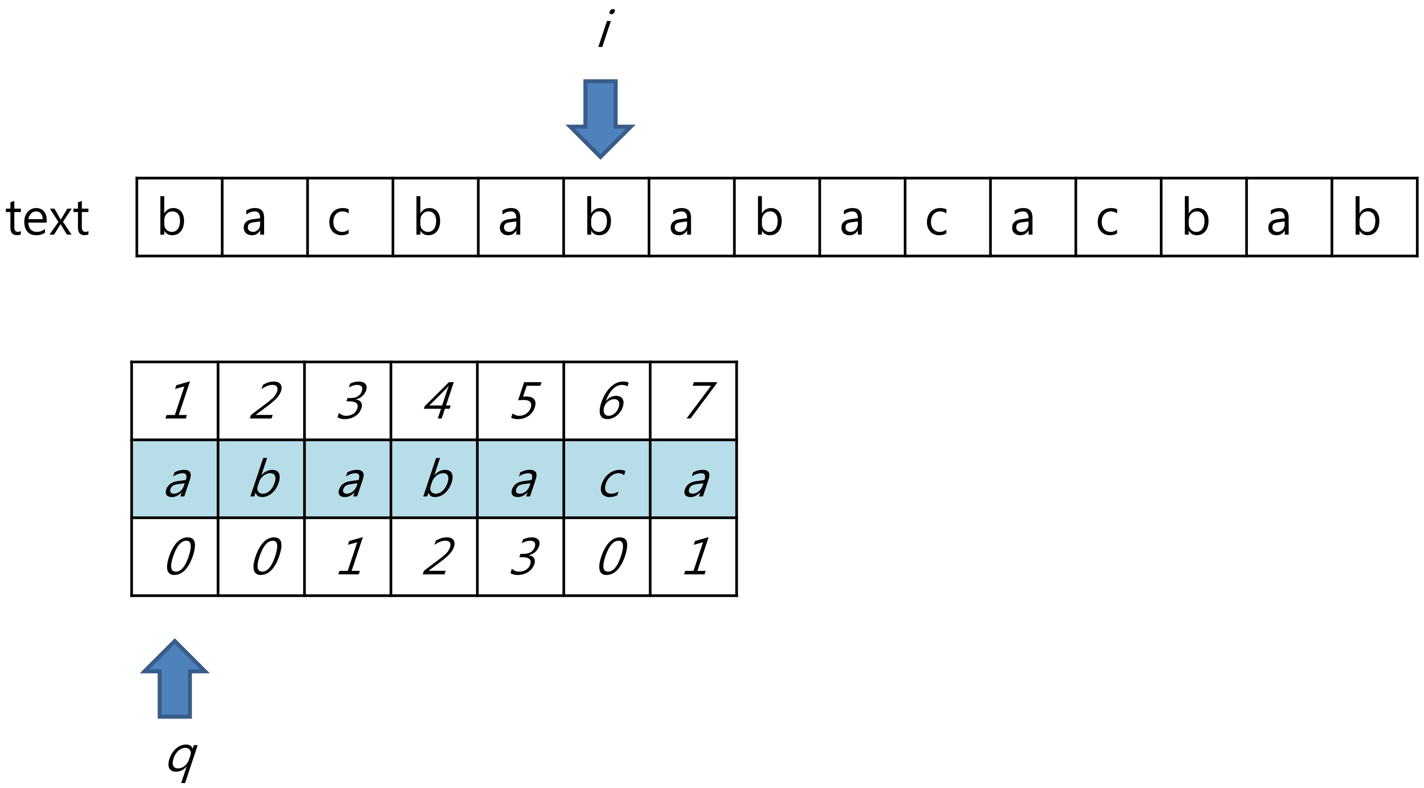
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm



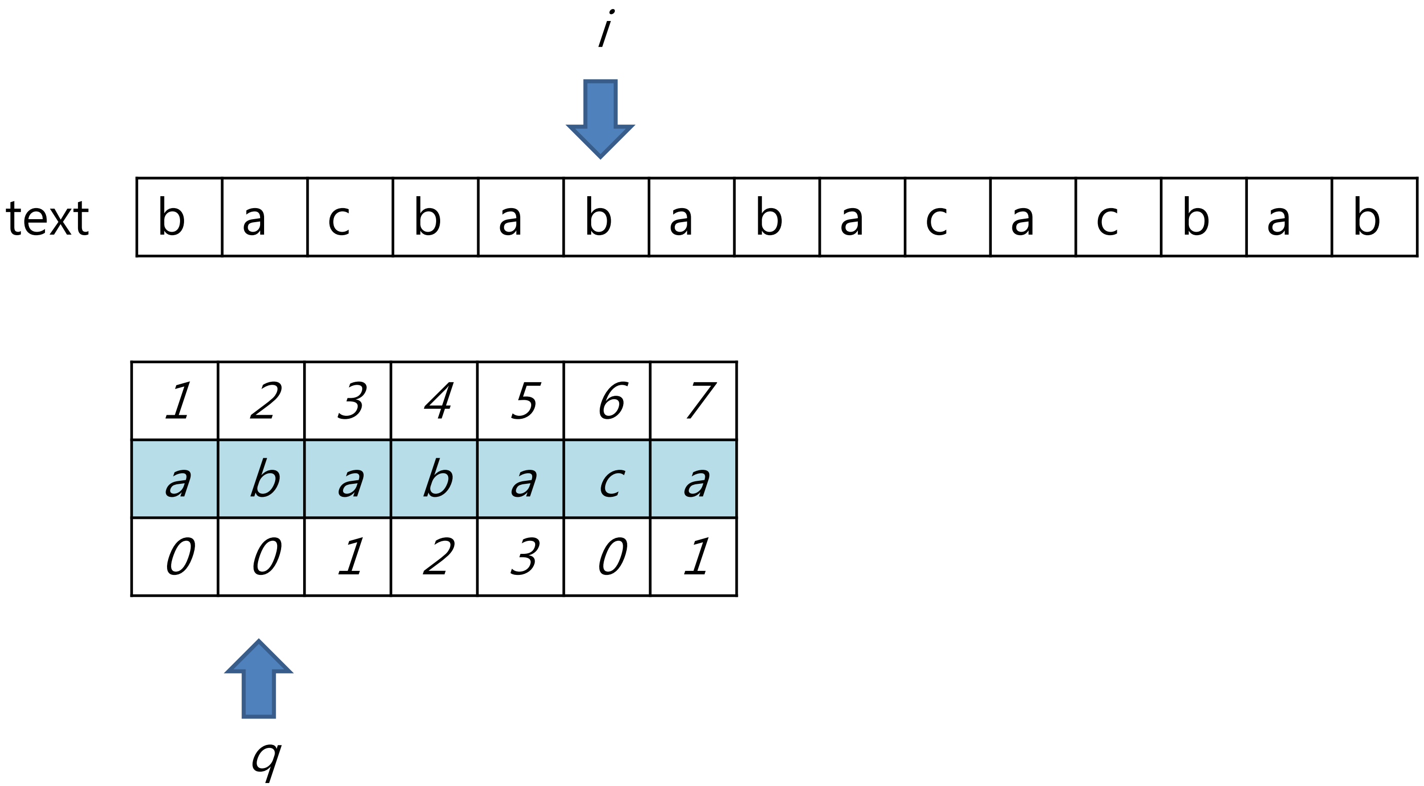
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

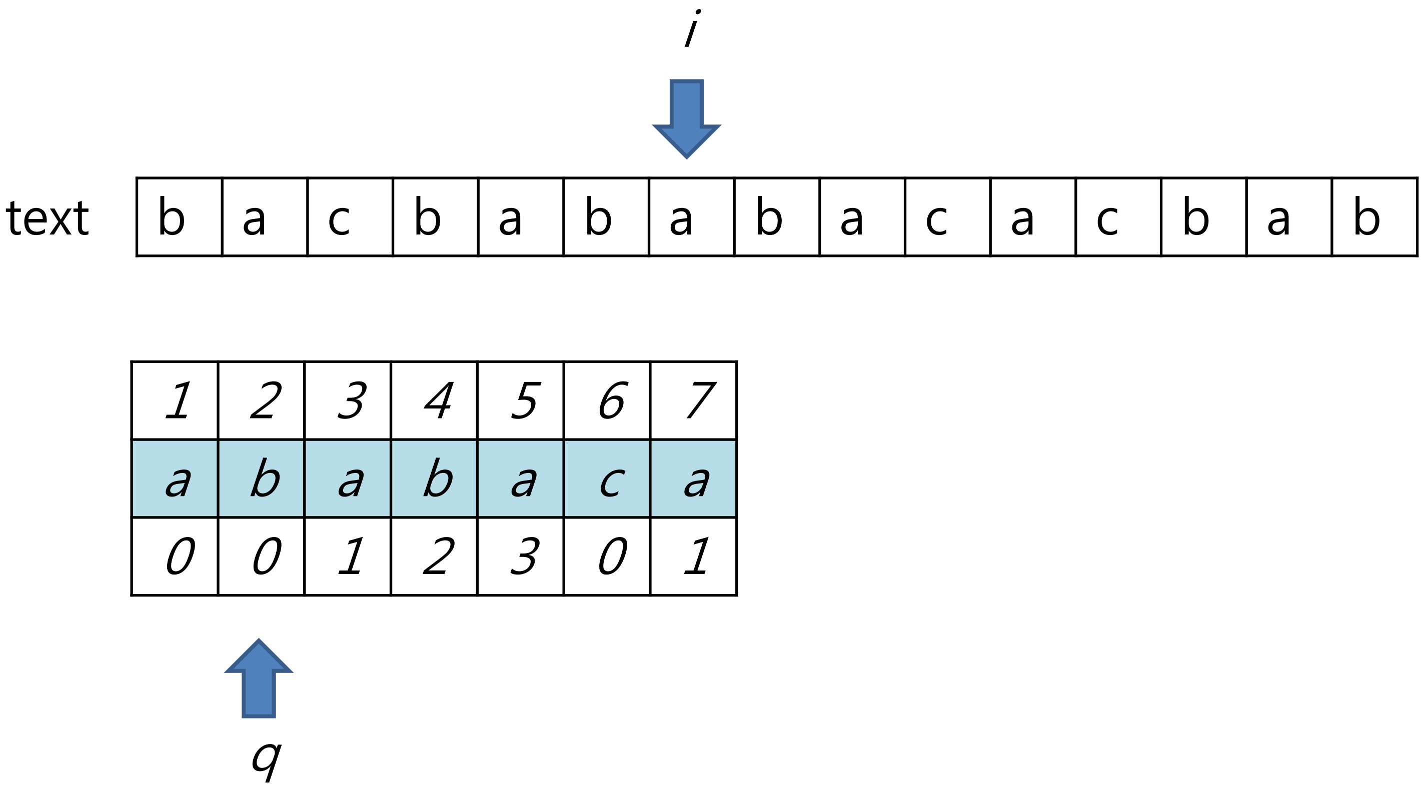
KMP Algorithm



text														
b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

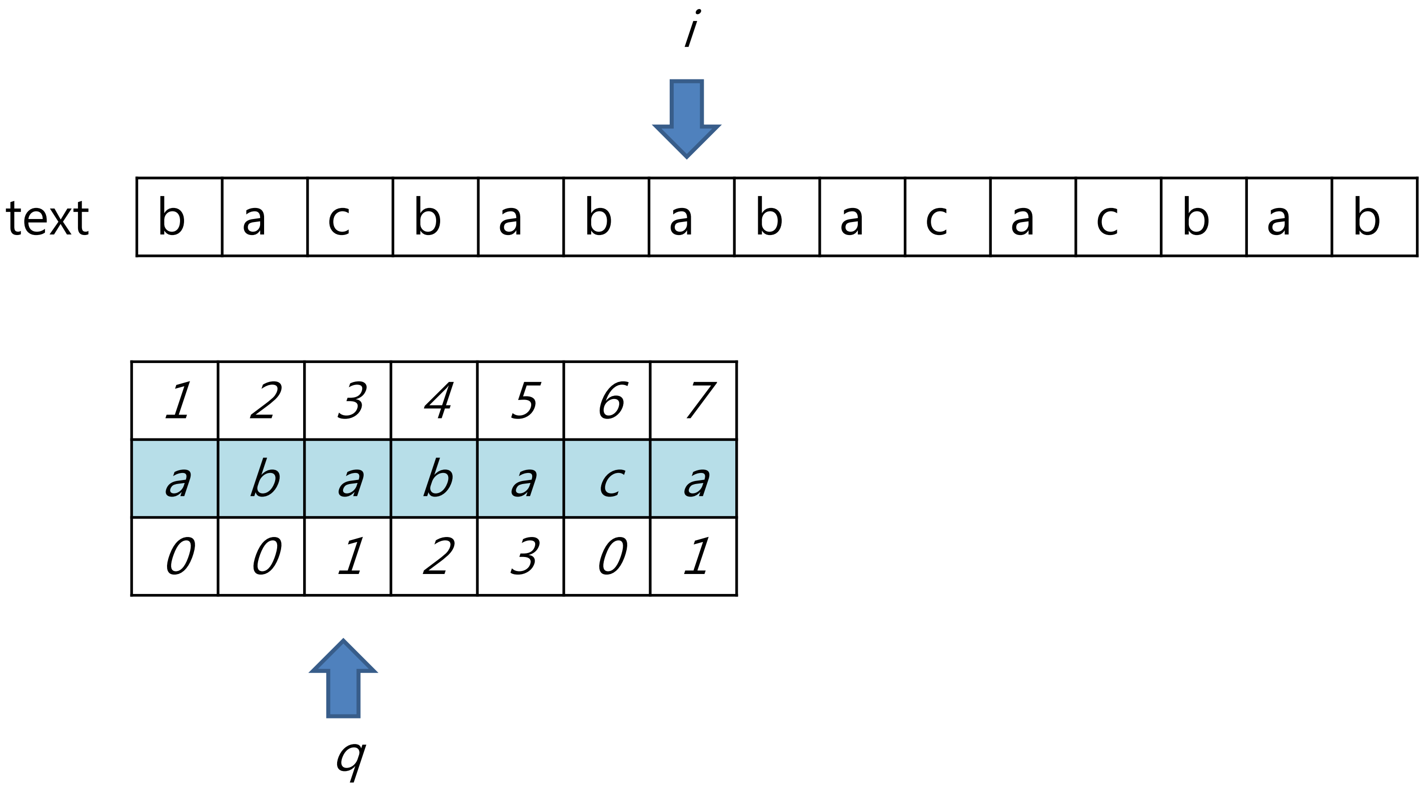
KMP Algorithm



text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

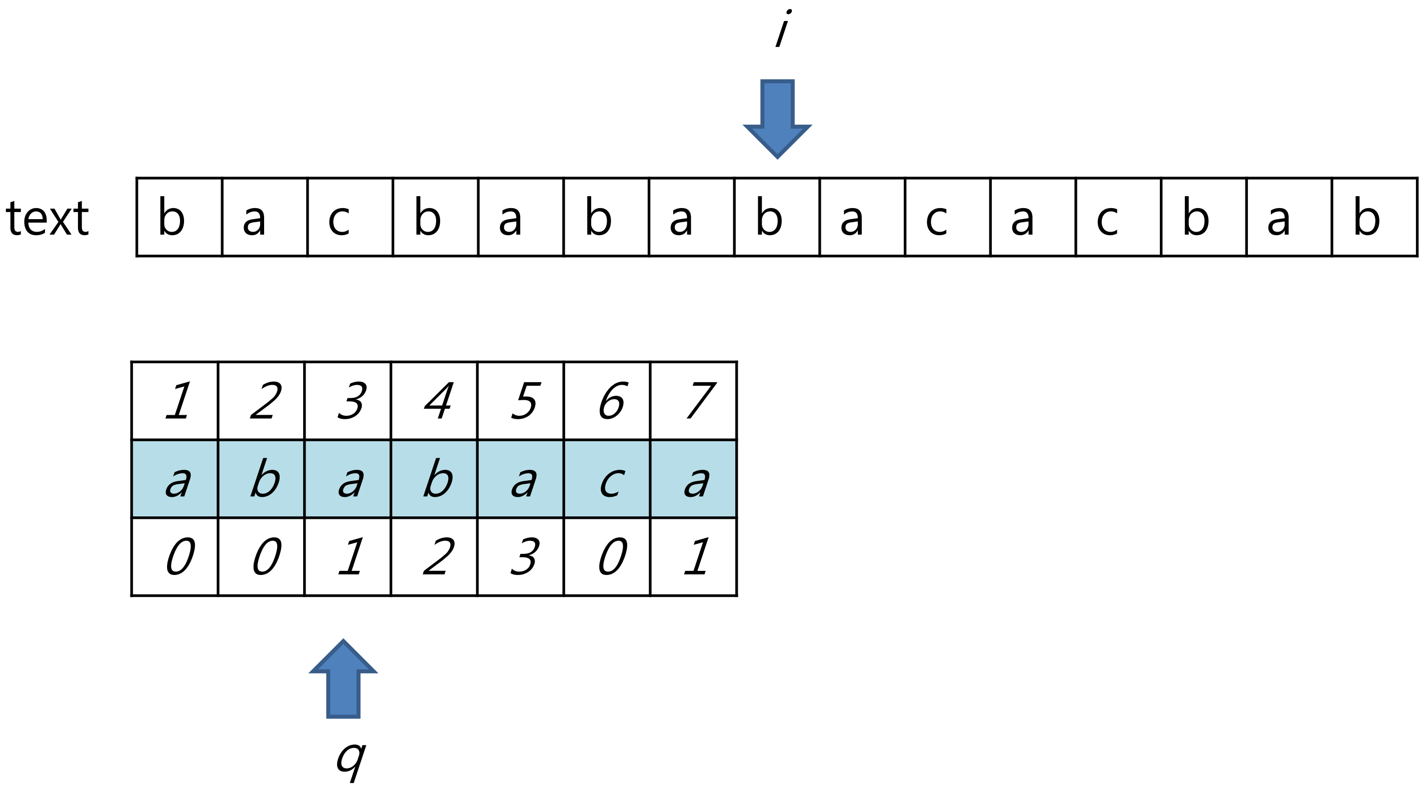
KMP Algorithm



text														
b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

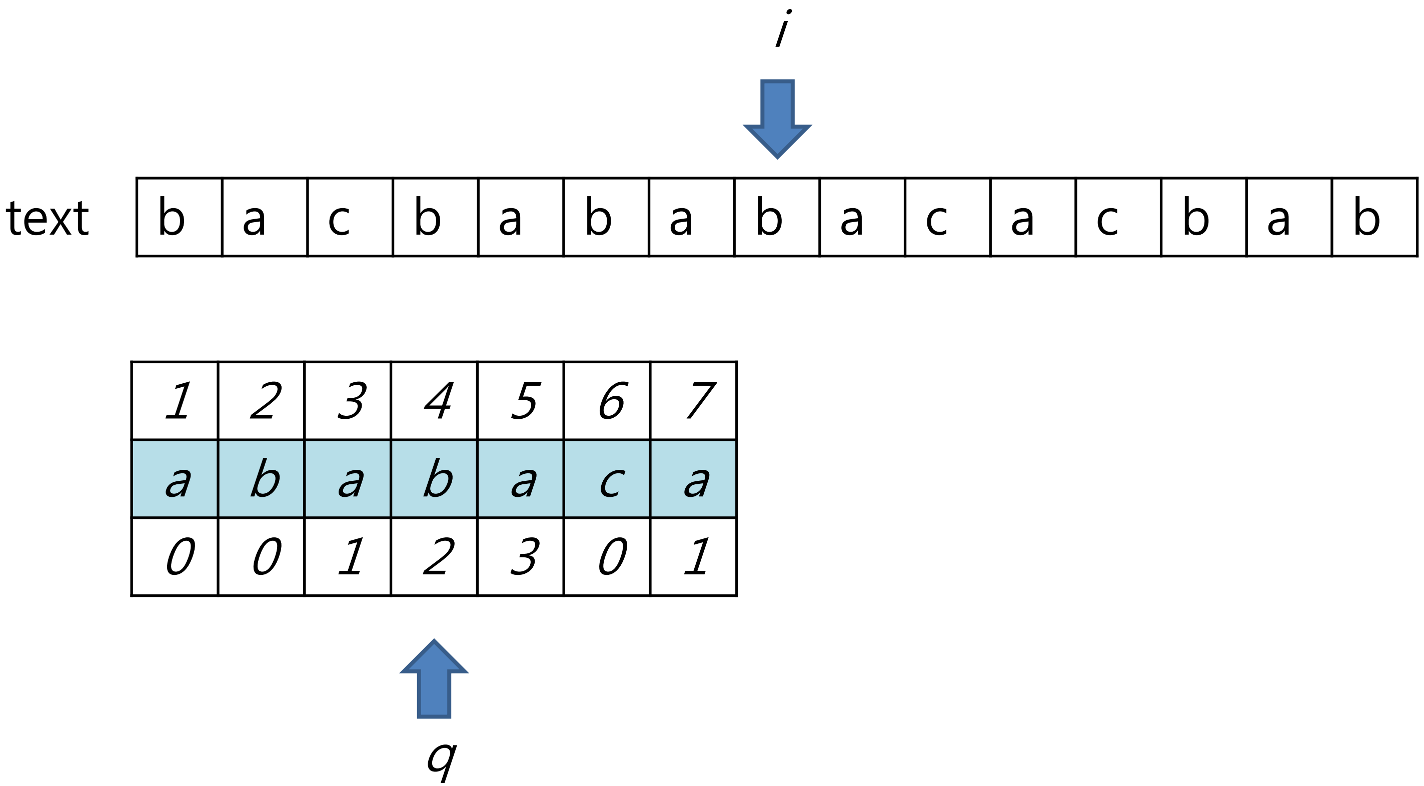
KMP Algorithm



text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm



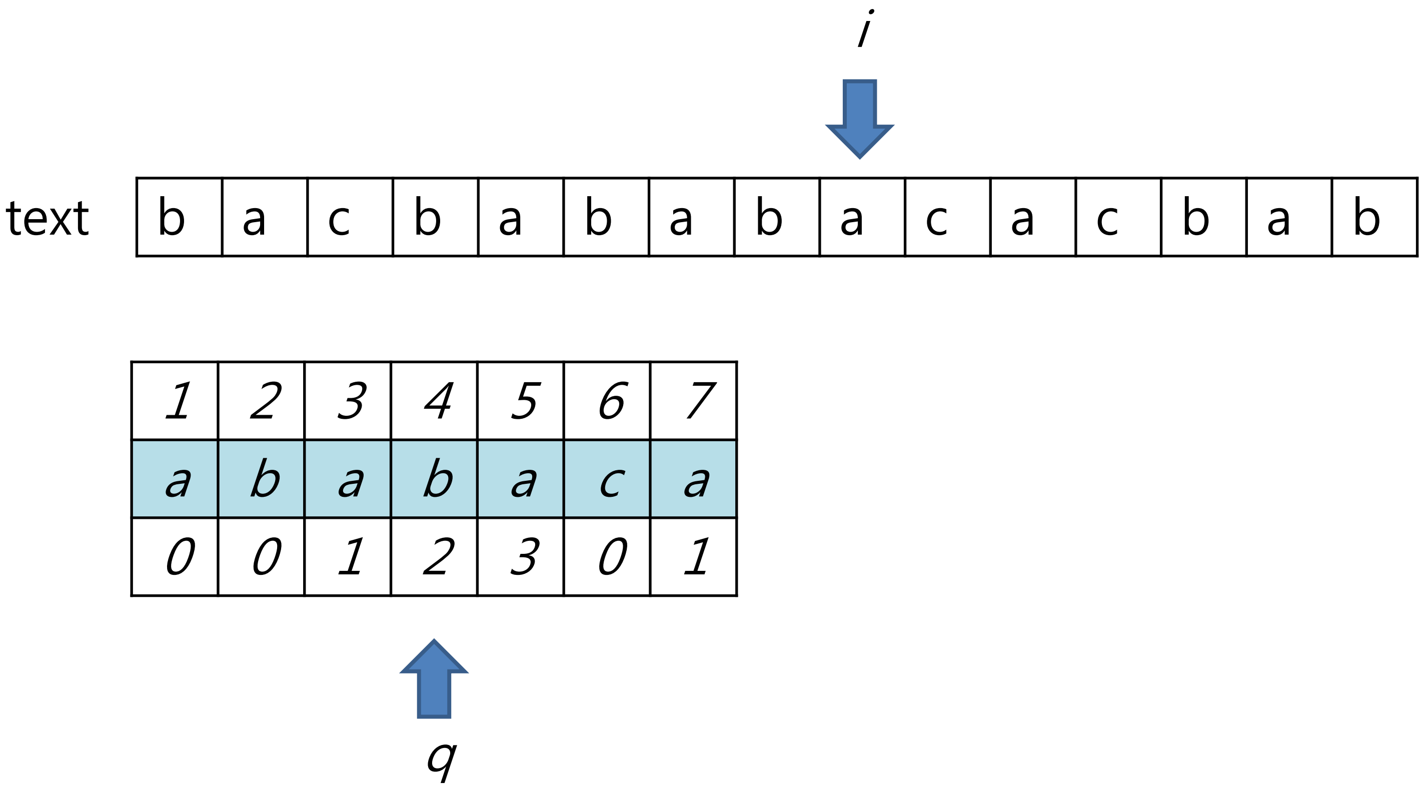
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

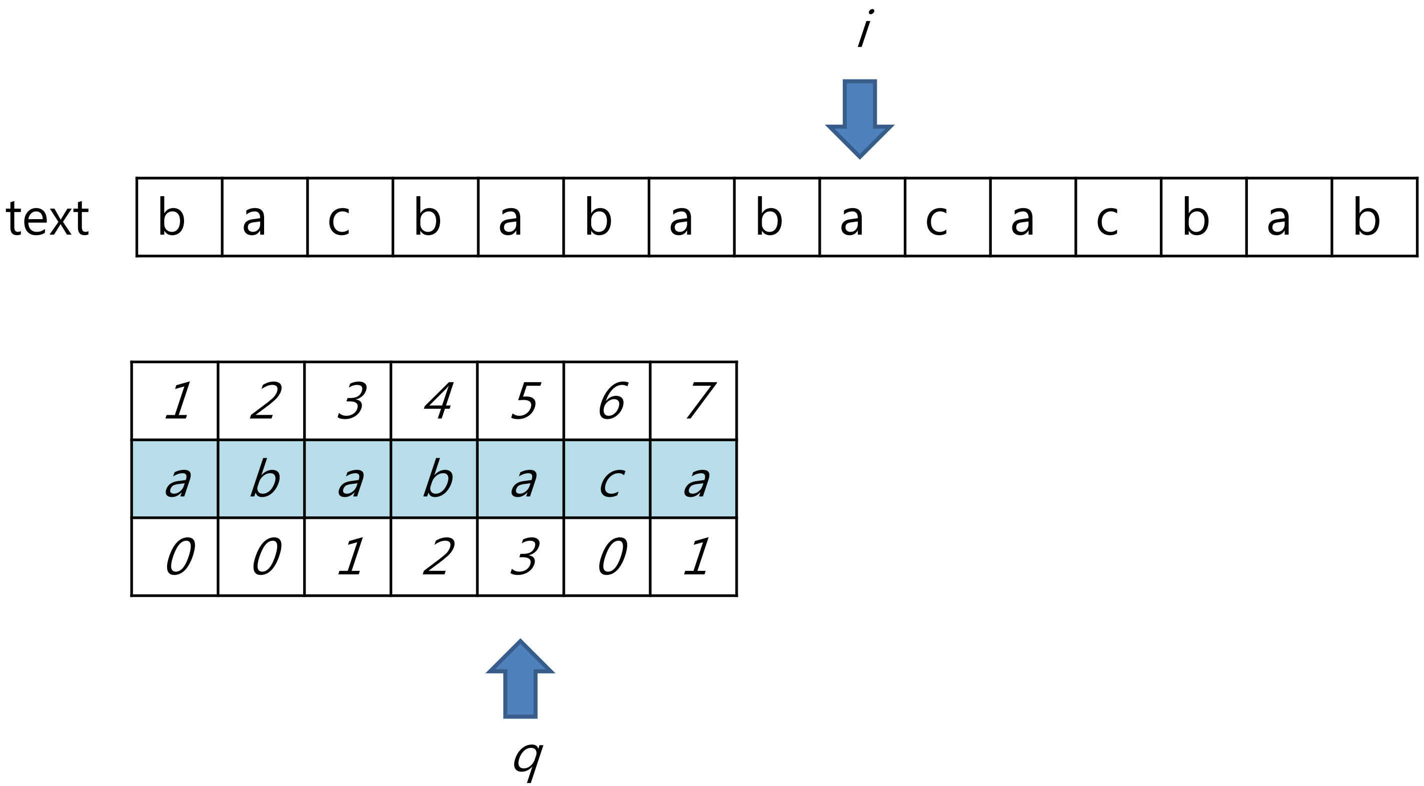
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

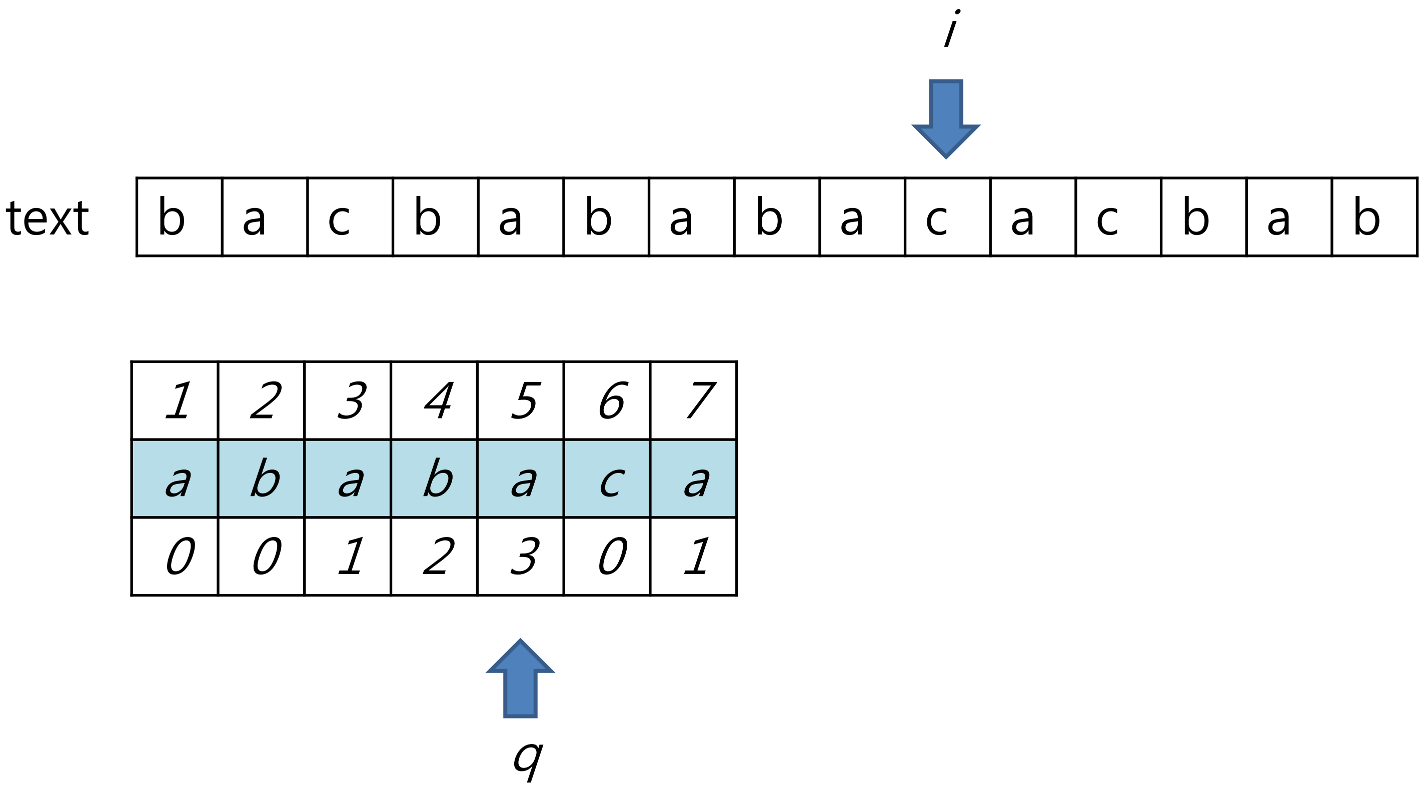
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm



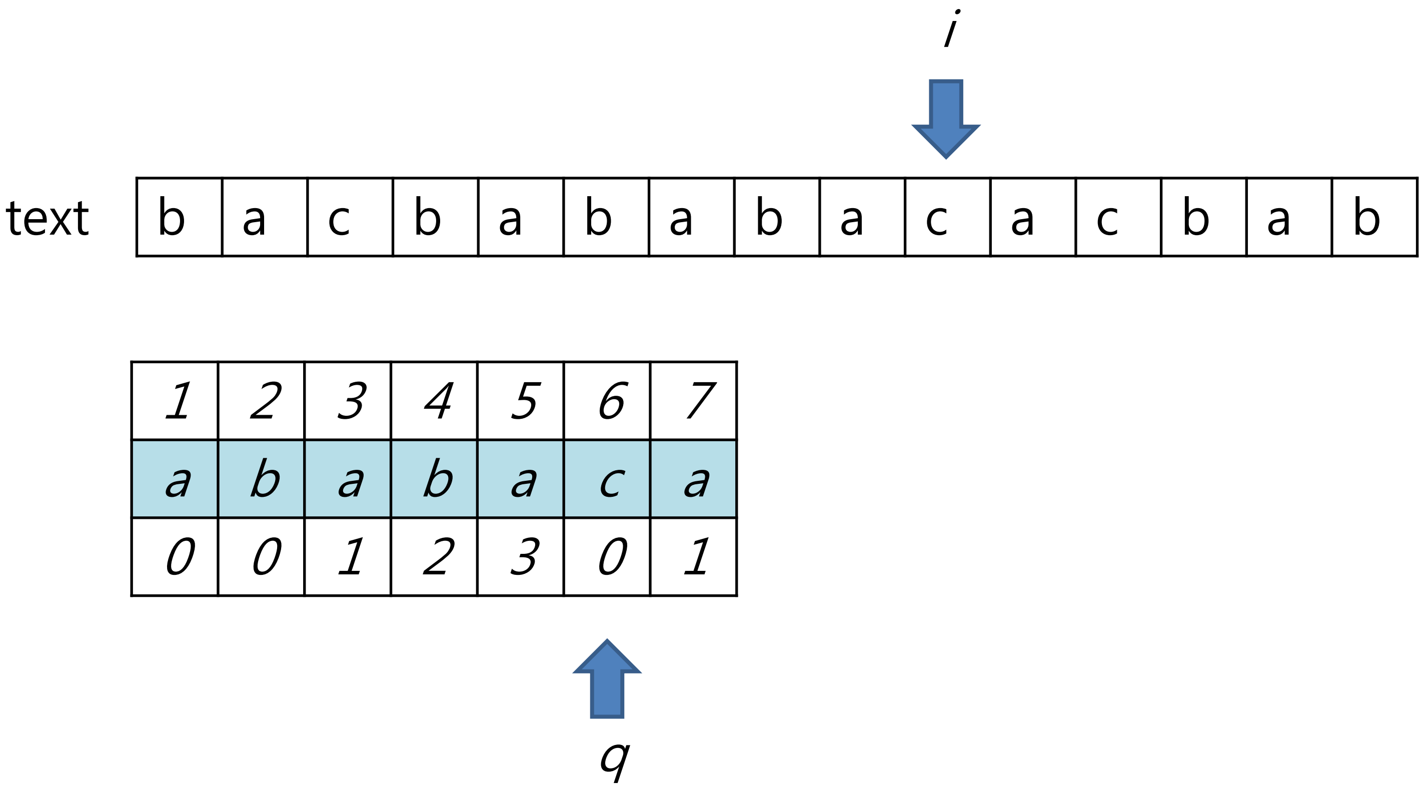
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

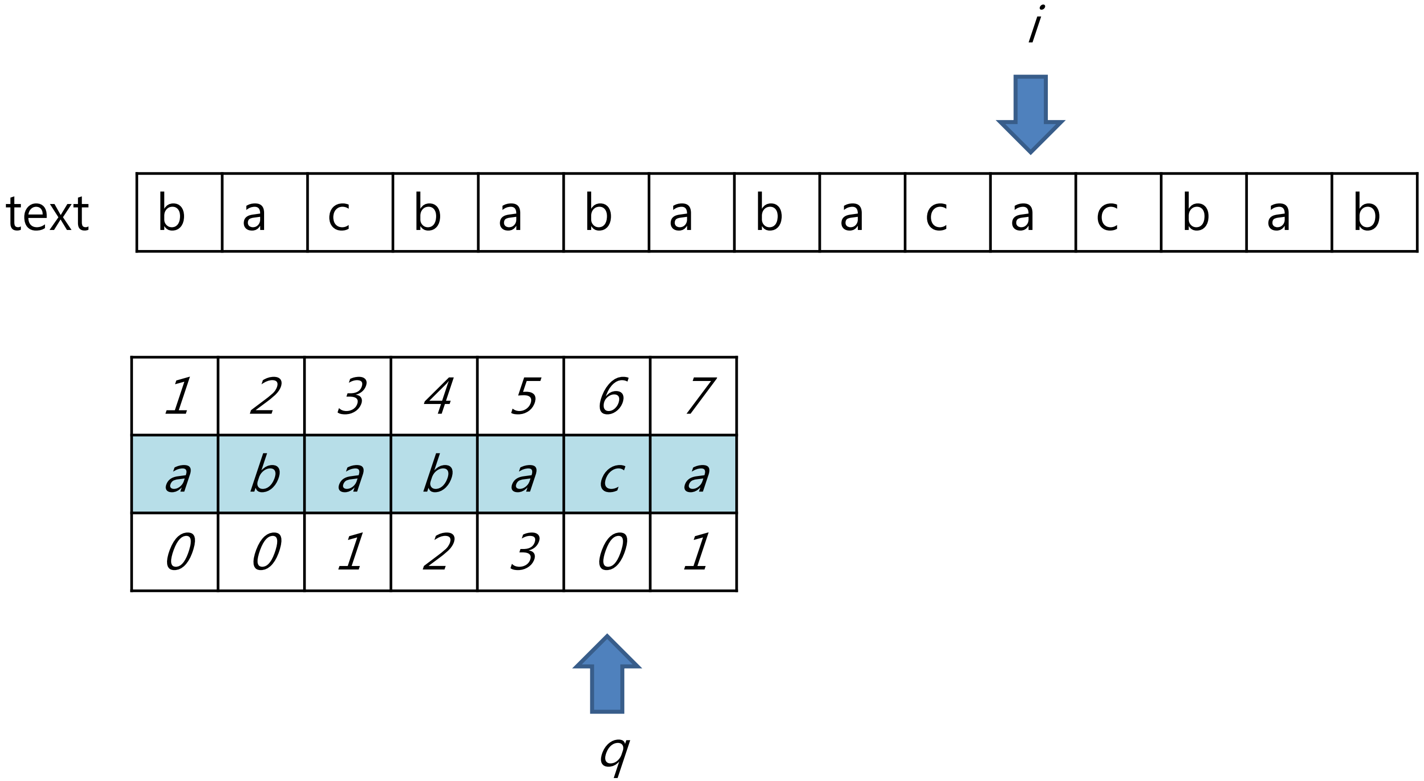
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

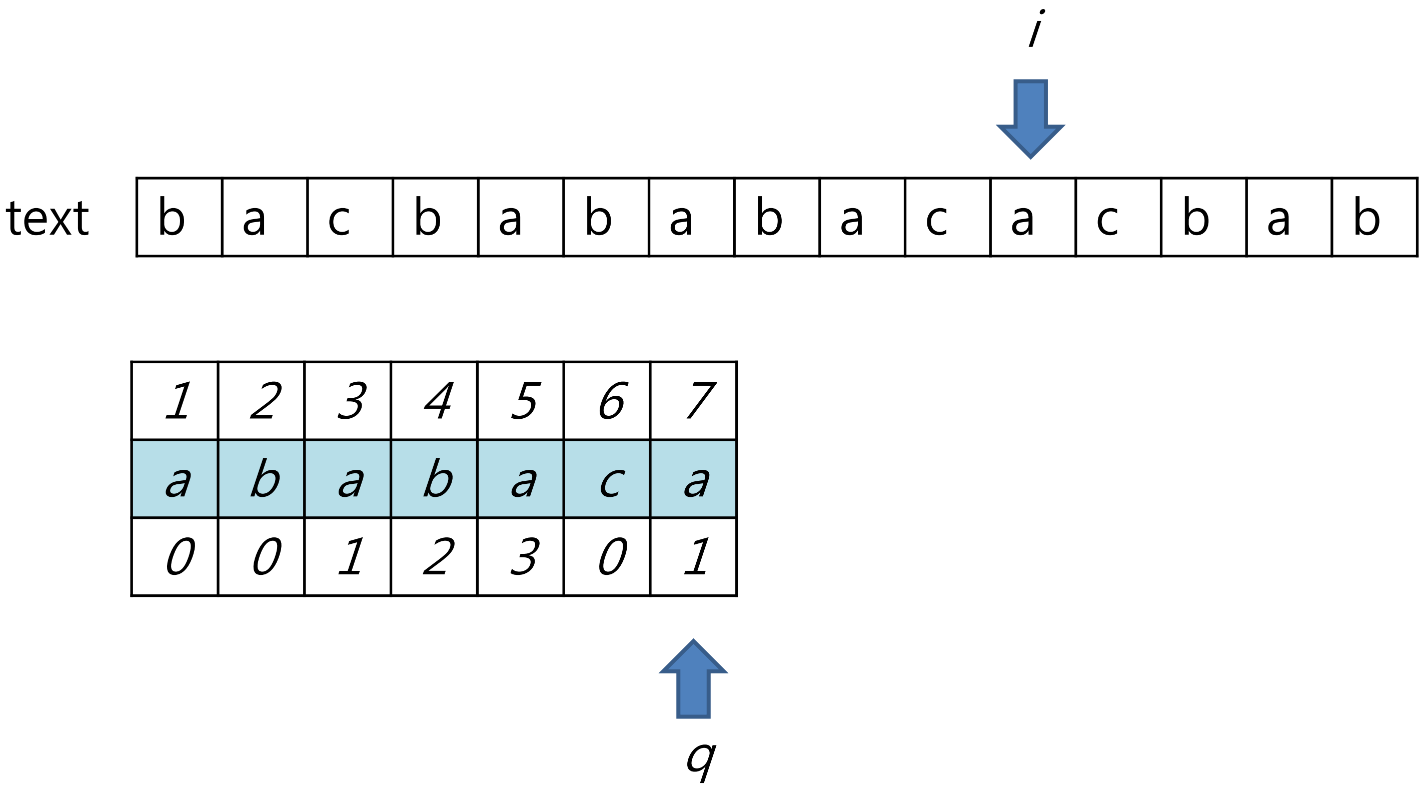
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

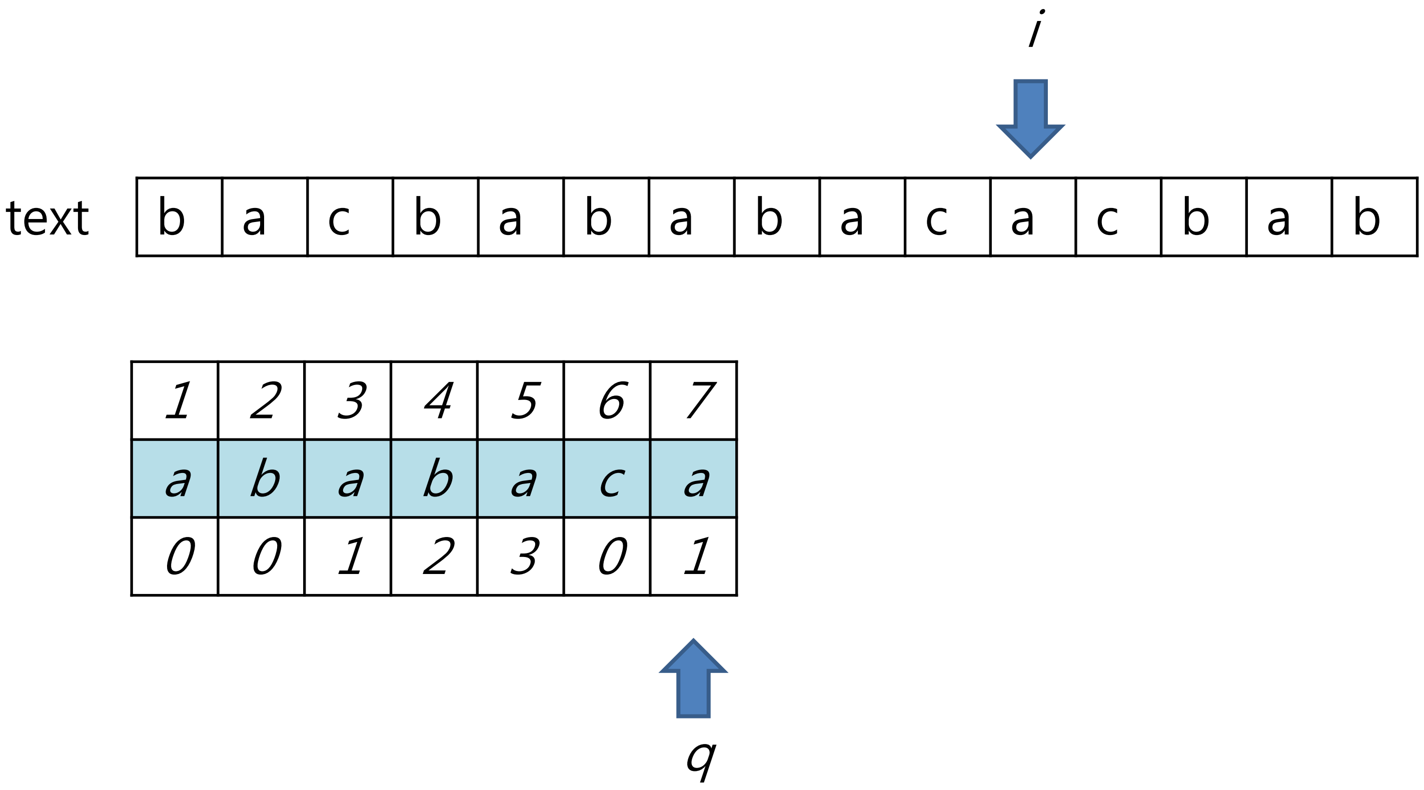
KMP Algorithm



text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

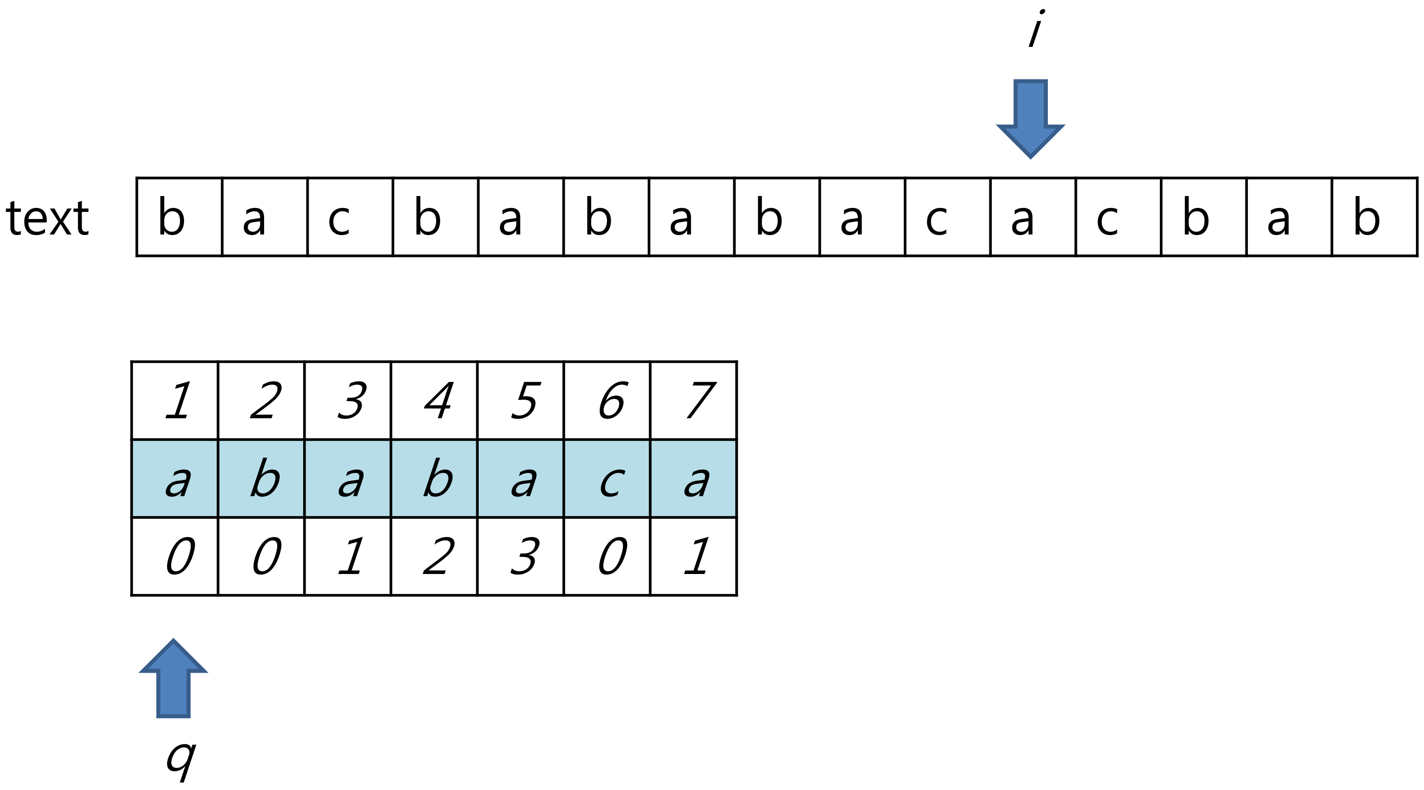


text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

Got a matching → tune the prefix function

KMP Algorithm



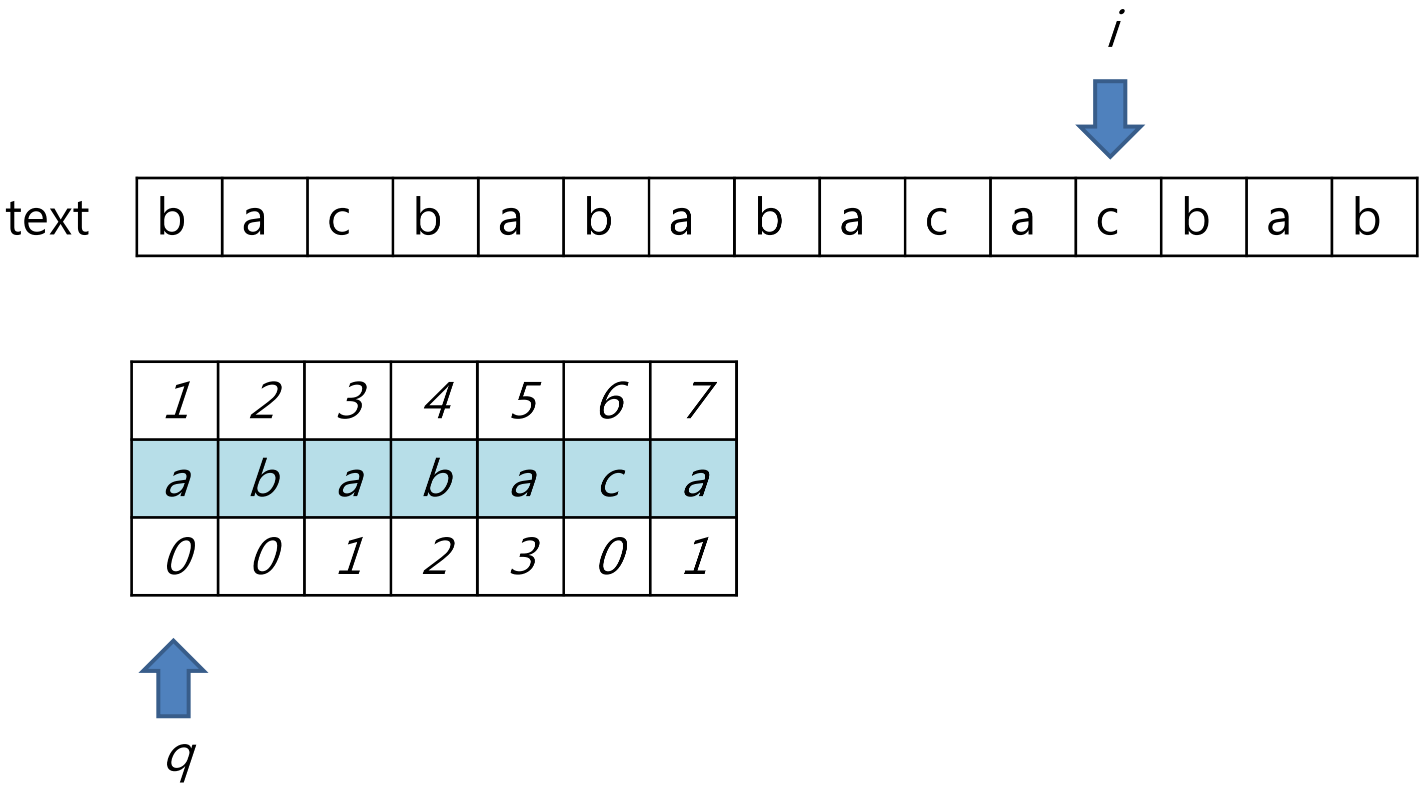
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

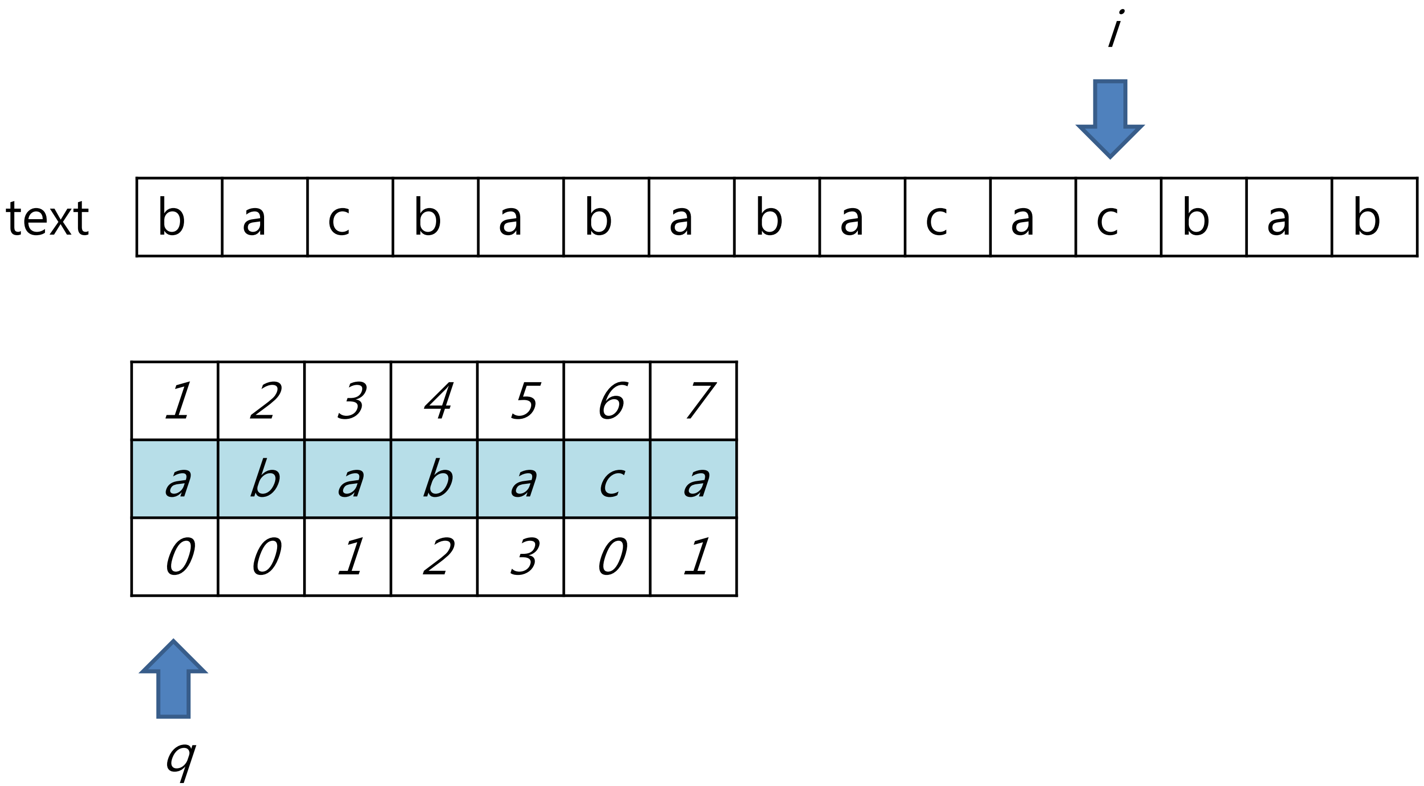
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

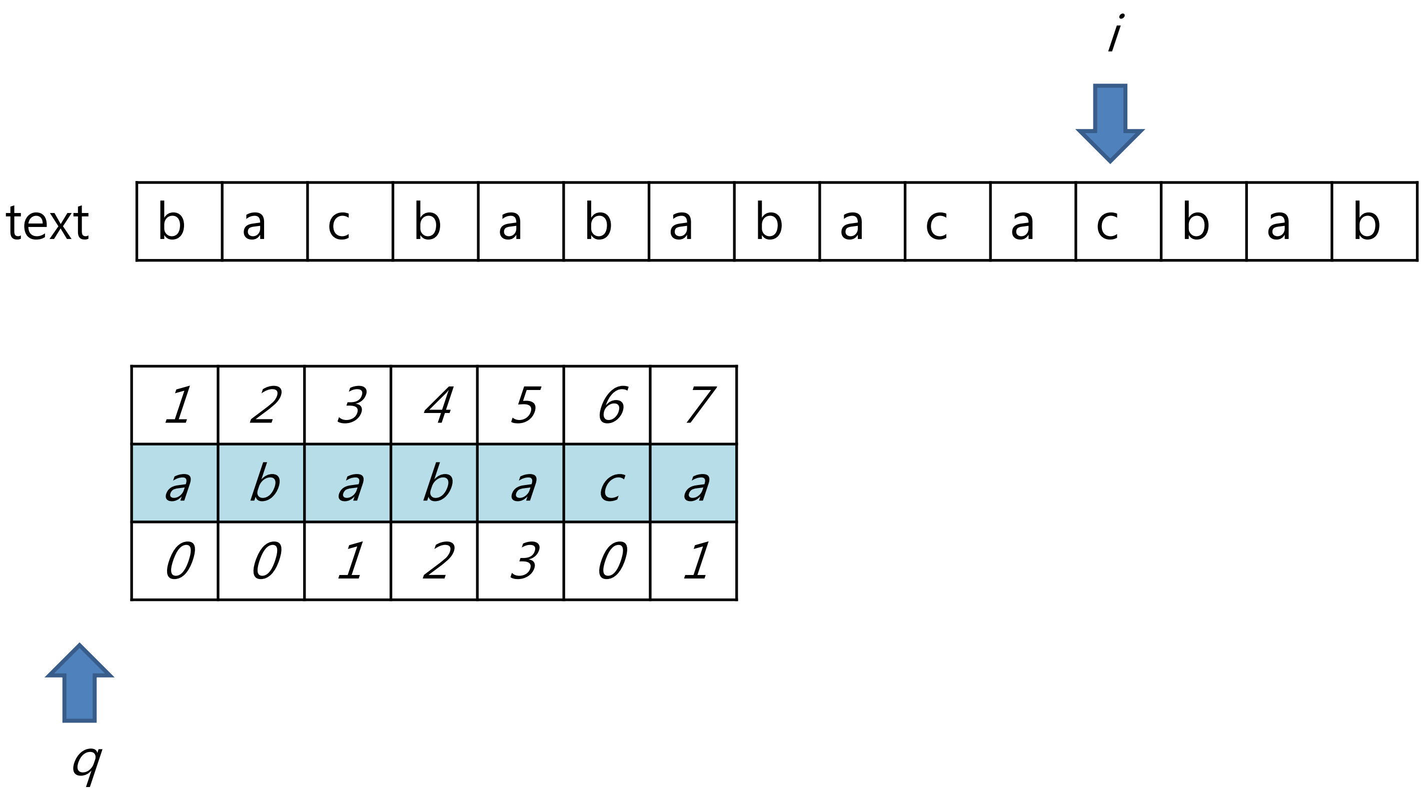


text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

Pattern[q+1] != text[i] → tune the prefix function

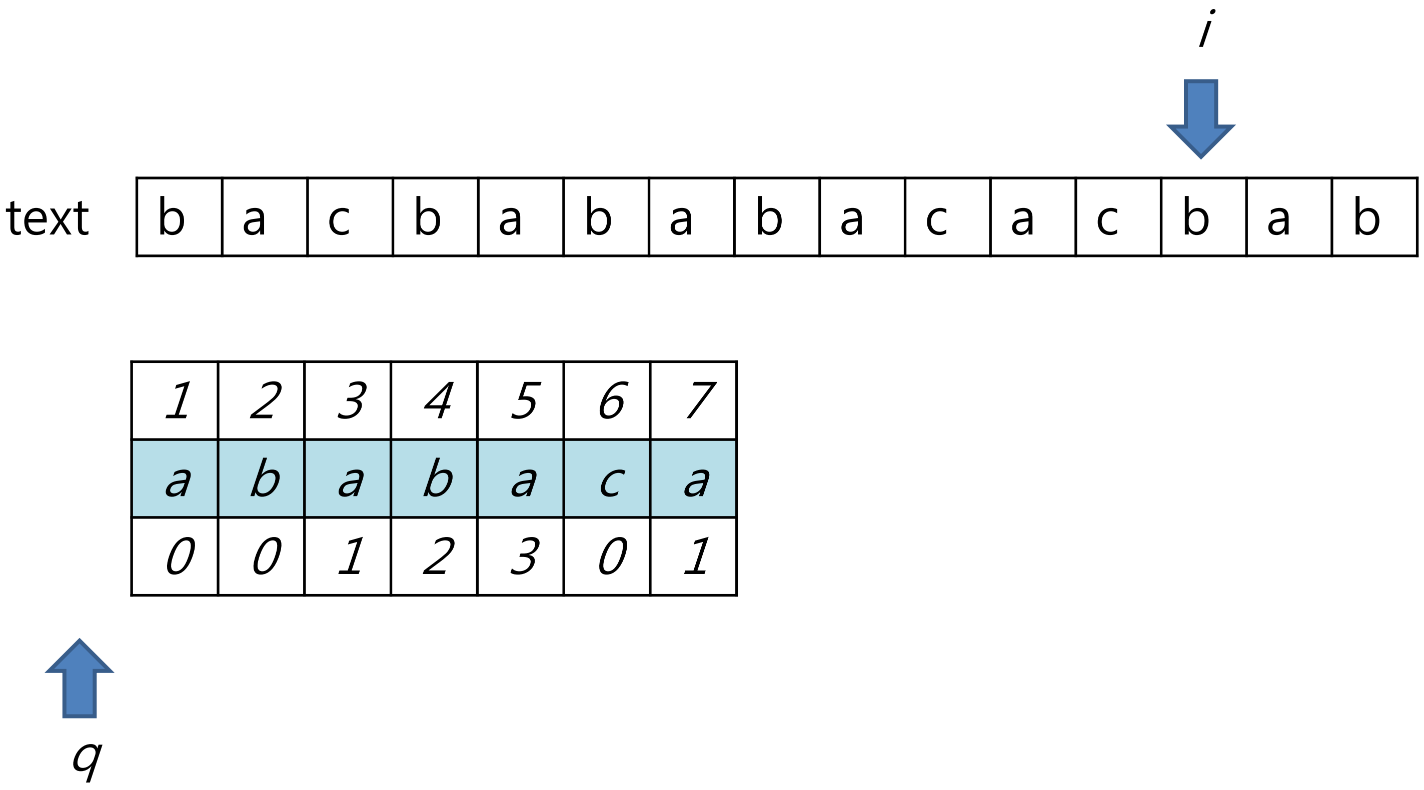
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm



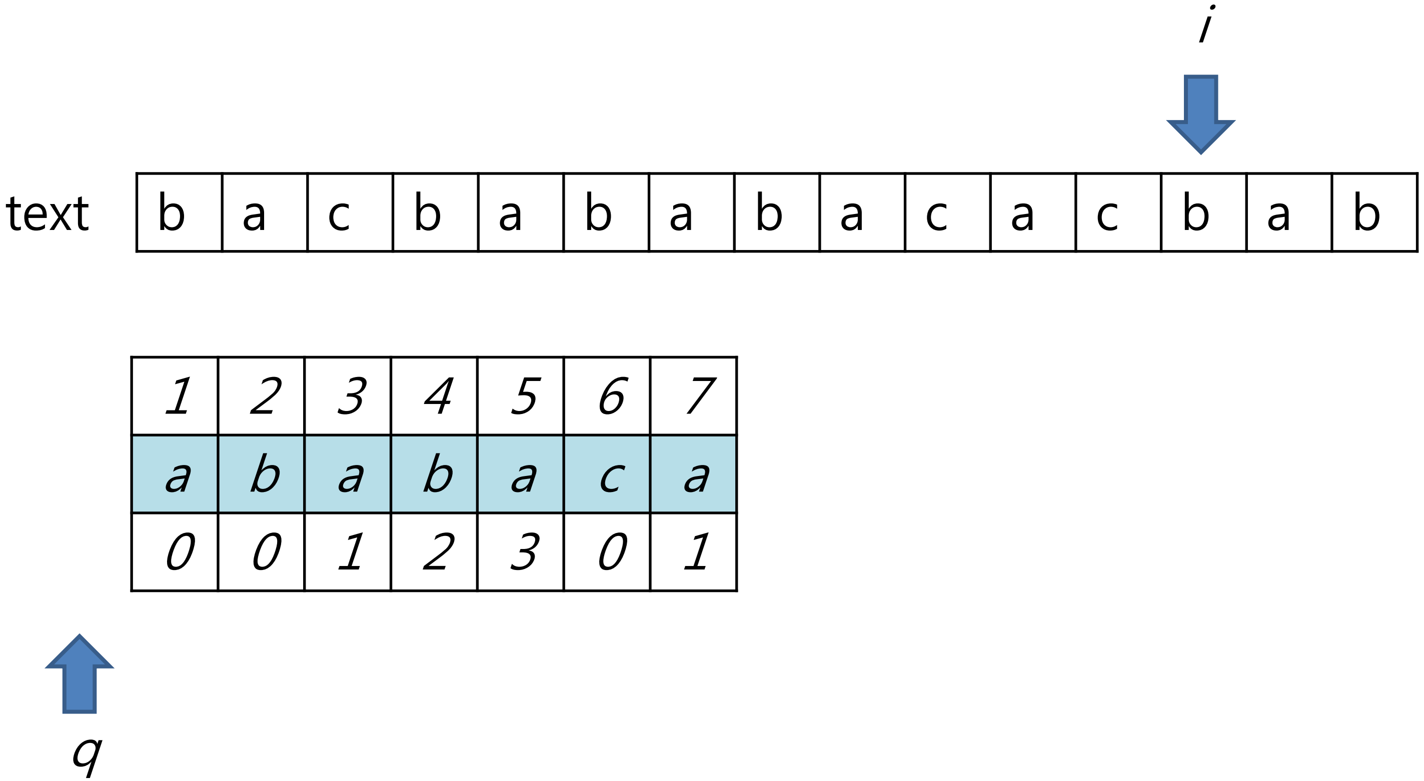
i

text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

q

KMP Algorithm

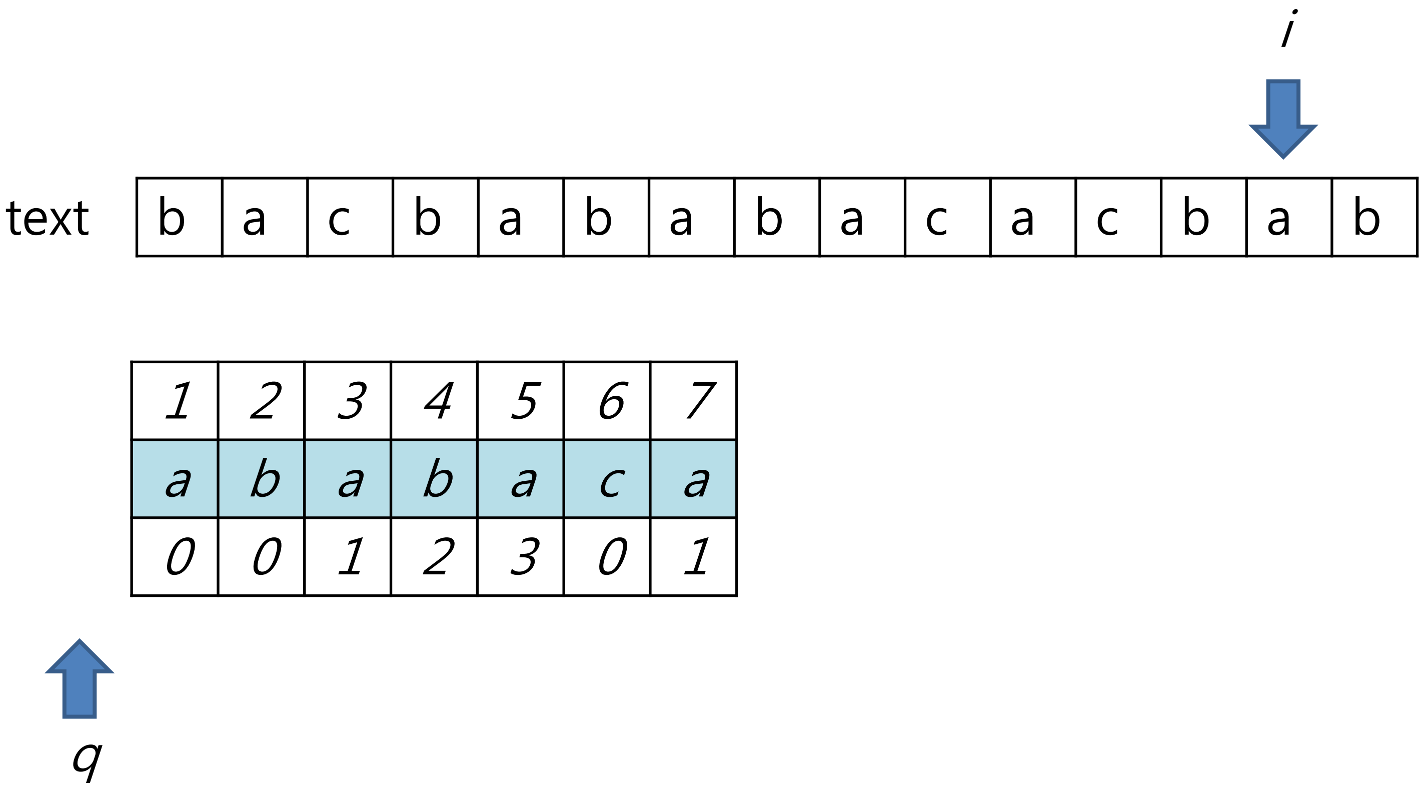


text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

Pattern[q+1] != text[i] → tune the prefix function

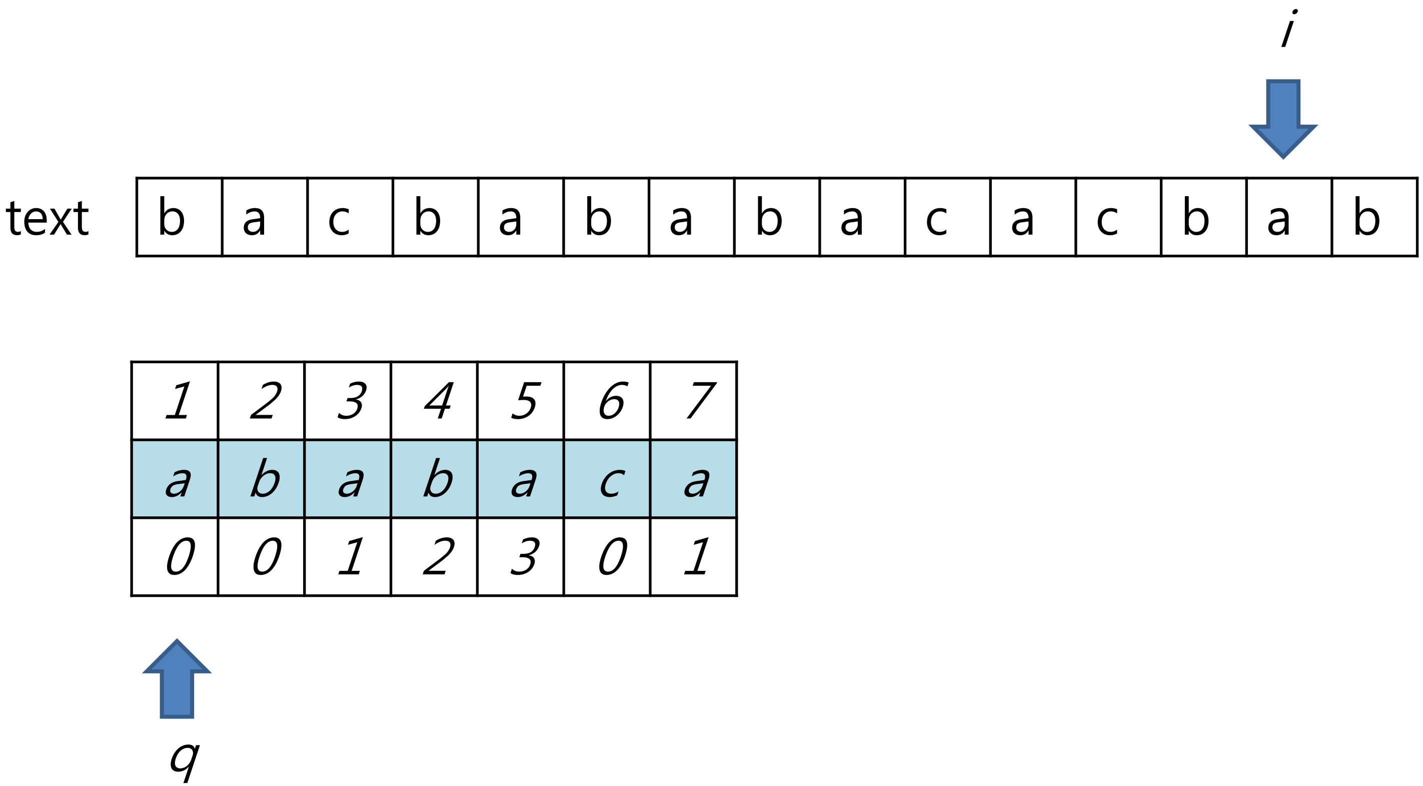
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

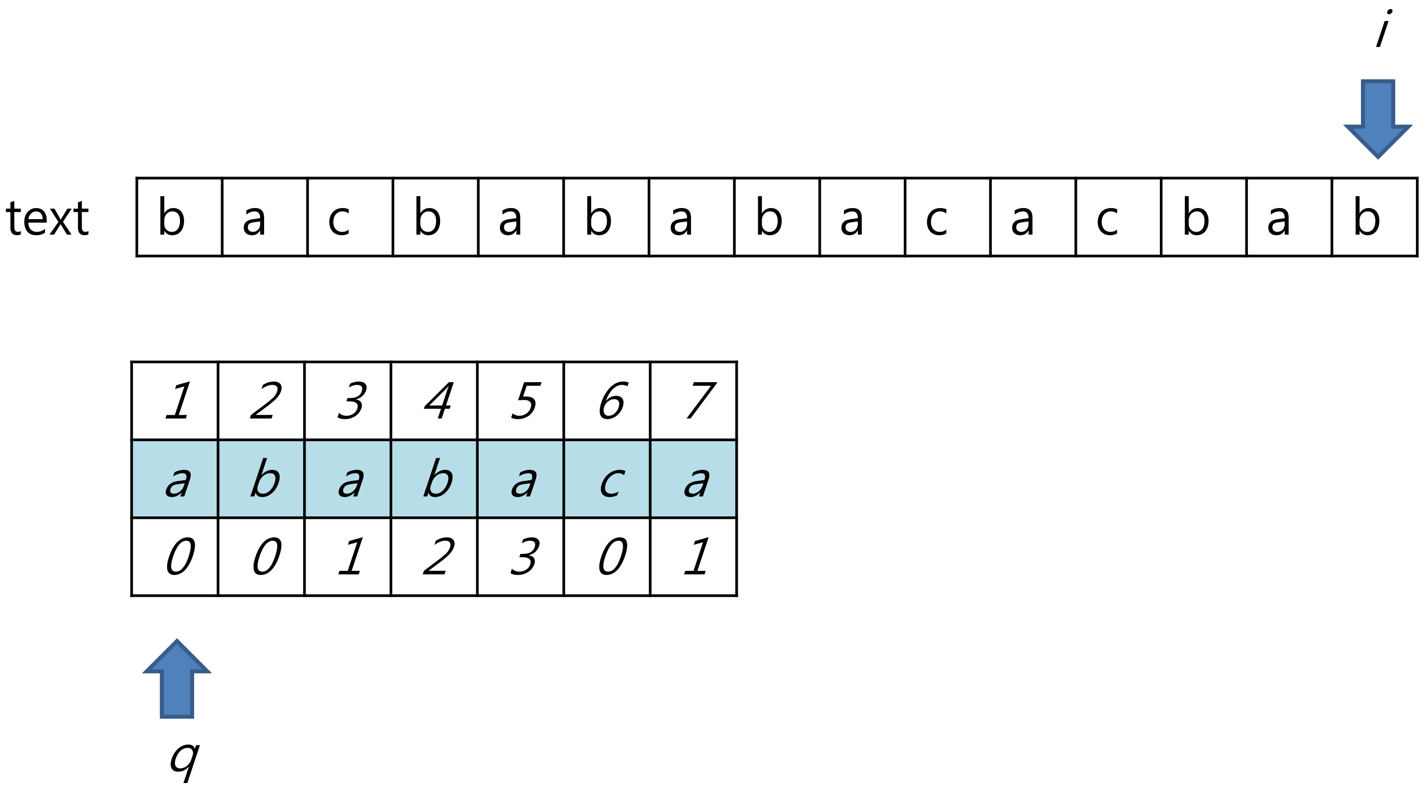
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	2	3	4	5	6	7
a	b	a	b	a	c	a
0	0	1	2	3	0	1

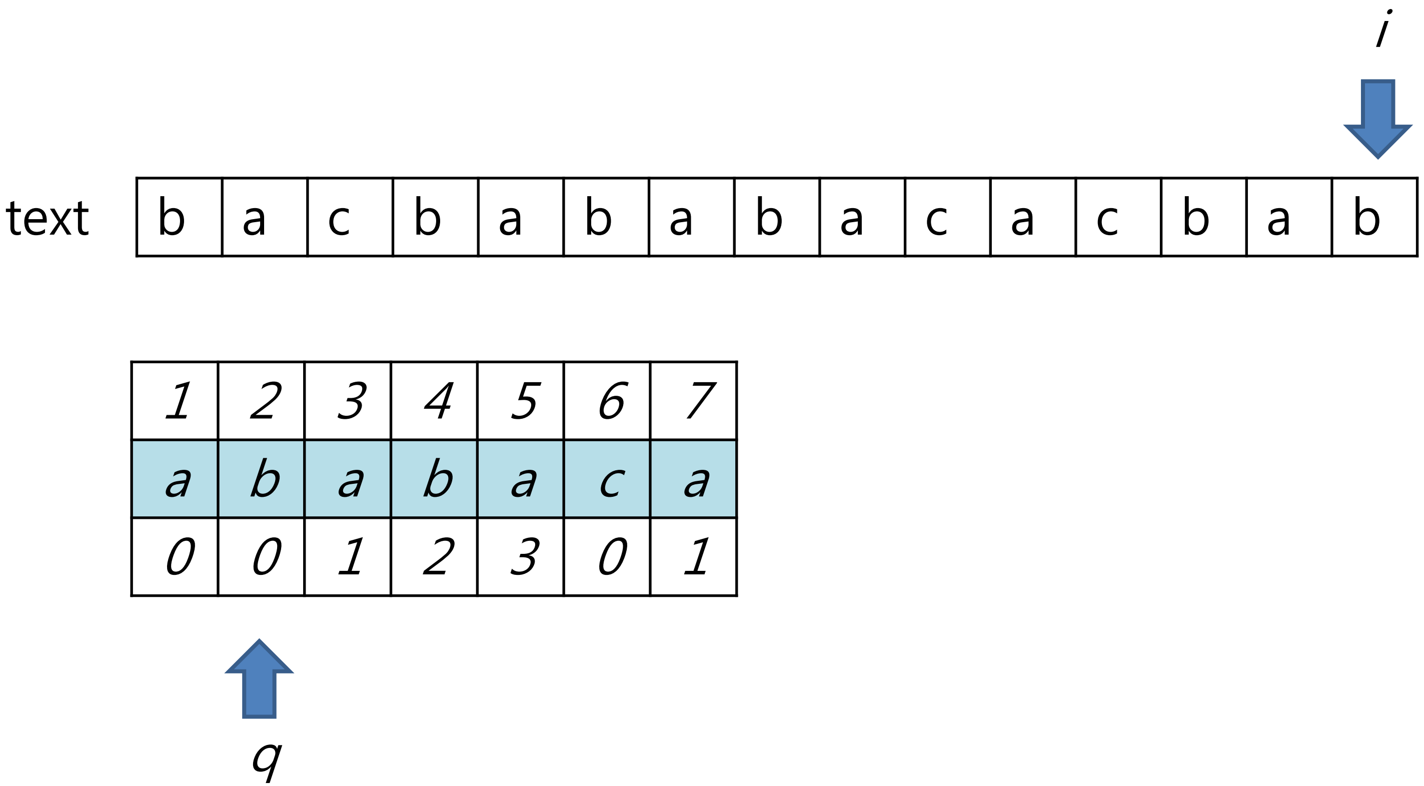
KMP Algorithm



text	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm



text															
	b	a	c	b	a	b	a	b	a	c	a	c	b	a	b

1	*2*	*3*	*4*	*5*	*6*	*7*
a	*b*	*a*	*b*	*a*	*c*	*a*
0	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

```
KMP-MATCHER($T$, $P$)
  $n \leftarrow length[T]$
  $m \leftarrow length[P]$
  $\pi \leftarrow$ COMPUTE-PREFIX-FUNCTION($P$)
  $q \leftarrow 0$                                ▹Number of characters matched.
  for $i \leftarrow 1$ to $n$                       ▹Scan the text from left to right.
      do while $q > 0$ and $P[q+1] \neq T[i]$
              do $q \leftarrow \pi[q]$            ▹Next character does not match.
         if $P[q+1] = T[i]$
            then $q \leftarrow q + 1$             ▹Next character matches.
         if $q = m$                               ▹Is all of $P$ matched?
            then print "Pattern occurs with shift" $i - m$
                 $q \leftarrow \pi[q]$            ▹Look for the next match.
```

KMP Algorithm

- How to compute the prefix function?

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*
pattern	*a*	*b*	*a*	*b*	*a*	*c*	*a*
Pi[i]	*0*	*0*	*1*	*2*	*3*	*0*	*1*

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	*a*	*b*	*a*	*b*	*a*	*c*	*a*
Pi[i]	*0*						

Pattern[0+1] = Pattern[2] ?

KMP Algorithm

- Example

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*
pattern	*a*	*b*	*a*	*b*	*a*	*c*	*a*
Pi[i]	*0*	*0*					

Pattern[0+1] = Pattern[3] ?

KMP Algorithm

- Example

i	*1*	*2*	*3*	*4*	*5*	*6*	*7*
pattern	*a*	*b*	*a*	*b*	*a*	*c*	*a*
Pi[i]	*0*	*0*	*1*				

Pattern[1+1] = Pattern[4] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2			

Pattern[2+1] = Pattern[5] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3		

Pattern[3+1] = Pattern[6] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3		

Pattern[3+1] = Pattern[6] ?

Pattern[1+1] = Pattern[6] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3		

Pattern[3+1] = Pattern[6] ?

Pattern[1+1] = Pattern[6] ?

Pattern[0+1] = Pattern[6] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3	0	

Pattern[3+1] = Pattern[6] ?

Pattern[1+1] = Pattern[6] ?

Pattern[0+1] = Pattern[6] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3	0	

Pattern[0+1] = Pattern[7] ?

KMP Algorithm

- Example

i	1	2	3	4	5	6	7
pattern	a	b	a	b	a	c	a
Pi[i]	0	0	1	2	3	0	1

Pattern[0+1] = Pattern[7] ?

KMP Algorithm

```
COMPUTE-PREFIX-FUNCTION($P$)
  $m \leftarrow length[P]$
  $\pi[1] \leftarrow 0$
  $k \leftarrow 0$
  for $q \leftarrow 2$ to $m$
       do while $k > 0$ and $P[k + 1] \neq P[q]$
              do $k \leftarrow \pi[k]$
          if $P[k + 1] = P[q]$
             then $k \leftarrow k + 1$
          $\pi[q] \leftarrow k$
  return $\pi$
```

Example

POJ 2406

POJ 3461

Homework

POJ-1961

POJ-2406

POJ-2752

POJ-2185

POJ-1200

POJ-3461

UVA -10298

UVA -11475