



# HUST

**ĐẠI HỌC BÁCH KHOA HÀ NỘI**  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.





ĐẠI HỌC  
BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY  
OF SCIENCE AND TECHNOLOGY

# THUẬT TOÁN ỨNG DỤNG

THUẬT TOÁN XỬ LÝ XÂU

ONE LOVE. ONE FUTURE.

- Thuật toán Boyer Moore
- Thuật toán Rabin Karp
- Thuật toán KMP

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $\text{Last}[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

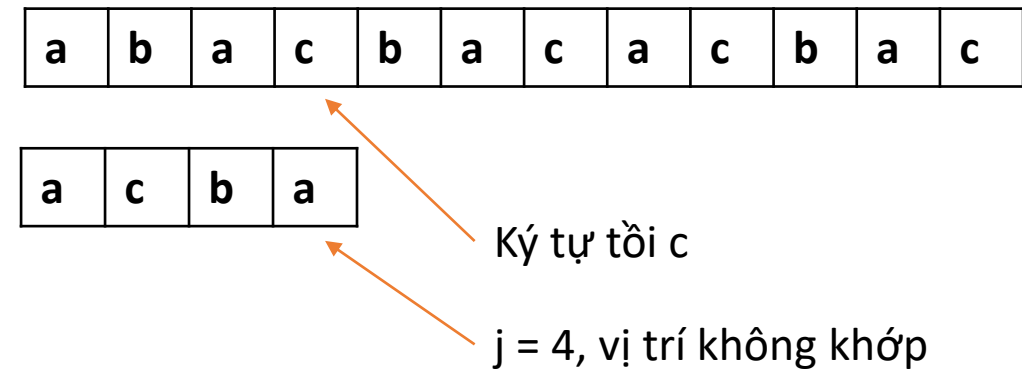
a	c	b	a
---	---	---	---

**$\text{Last}[a] = 4, \text{Last}[b] = 3,$   
 $\text{Last}[c] = 2$**



# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $Last[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự tại  $j$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - Last[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái



# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $Last[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự tại vị trí  $j$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - Last[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $Last[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự tại là  $x$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - Last[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $Last[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự tại là  $x$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - Last[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đôi sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $Last[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự tại là  $x$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - Last[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

- Bài toán tìm kiếm xâu mẫu: cho văn bản  $T$  là 1 chuỗi ký tự (độ dài  $N$ ) lấy từ 1 bảng cho trước và 1 xâu mẫu  $P$  (độ dài  $M$ ). Hãy tìm tất cả các vị trí xuất hiện của  $P$  trong  $T$
- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu  $P$ 
    - $\text{Last}[x]$ : vị trí bên phải nhất xuất hiện ký tự  $x$  trong  $P$
  - Khi tình trạng không khớp xảy ra với ký tự  $x$  (ký tự trên  $T$ ),  $P$  sẽ được trượt sang phải  $\max\{j - \text{Last}[x], 1\}$  vị trí trong đó  $j$  là chỉ số hiện tại (xảy ra không khớp) trên  $P$  khi so khớp ký tự từ phải qua trái

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN BOYER MOORE

```
computeLast(p){  
    for c = 0 to 255 do last[c] = 0;  
    k = p.length();  
    for i = k-1 downto i >= 0 do {  
        if last[p[i]] = 0 then last[p[i]] = i;  
    }  
}
```

```
boyerMoore(P, T){  
    computeLast(P);  
    s = 0;    cnt = 0;  
    N = T.length();    M = P.length();  
    while s <= N-M do {  
        j = M-1;  
        while j >= 0 && T[j+s] = P[j] do  
            j = j - 1;  
        if j == -1 then {  
            cnt++; s = s + 1;  
        }else{  
            k = last[T[j+s]];  
            s = s + (j - k > 1 ? j - k : 1);  
        }  
    }  
    return cnt;  
}
```

# THUẬT TOÁN RABIN KARP

- Thuật toán Rabin-Karp đổi các xâu cần so khớp sang số nguyên không âm
- Mỗi ký tự trong bảng chữ cái được biểu diễn bởi 1 số nguyên không âm nhỏ hơn  $d$
- Đổi xâu  $P[1..M]$  sang giá trị số nguyên dương

$$p = P[1]*d^{M-1} + P[2]*d^{M-2} + \dots + P[M]*d^0$$

- Đối sánh mẫu bằng cách so sánh 2 giá trị mã tương ứng
  - Nếu hai mã khác nhau thì hai xâu tương ứng là khác nhau
  - Nếu hai mã bằng nhau thì ta tiến hành so khớp từng ký tự
- Sử dụng lược đồ Horner để tăng tốc độ tính toán mã các xâu con trong T
- Với vị trí trượt  $s$ , đổi xâu con  $T[s+1 .. s+M]$  sang số:

$$T_s = T[s+1]*d^{M-1} + T[s+2]*d^{M-2} + \dots + T[s+M]*d^0$$

- Với vị trí trượt  $s+1$ ,  $T_{s+1}$  có thể được tính toán hiệu quả dựa vào  $T_s$  (đã được tính trước đó)

$$T_{s+1} = (T_s - T[s+1]*d^{M-1})*d + T[s+M+1]$$



# THUẬT TOÁN RABIN KARP

- Nhược điểm
  - Khi  $M$  lớn thì việc chuyển đổi xâu sang số mất thời gian đáng kể,
  - Có thể gây ra tràn số đối với kiểu dữ liệu cơ bản của ngôn ngữ lập trình
- Cách giải quyết: thực hiện phép chia cho  $Q$  và lấy giá trị số dư
  - Khi 2 số dư khác nhau có nghĩa 2 giá trị số khác nhau và 2 xâu tương ứng cũng khác nhau
  - Khi 2 số dư bằng nhau, tiến hành đối sánh từng ký tự như cách truyền thống

# THUẬT TOÁN RABIN KARP

```
hashCode(p){
    c = 0;
    for i = 0 to p.length()-1 do {
        c = c*256 + p[i];
        c = c%Q;
    }
    return c;
}

hashCode(s, start, end){
    c = 0;
    for i = start to end do {
        c = c*256 + s[i];
        c = c%Q;
    }
    return c;
}
```

```
rabinKarp(P, T){
    cnt = 0;  N = T.length();  M = P.length();
    e = dM-1;
    codeP = hashCode(P);  codeT = hashCode(T,0,M-1);
    for s = 0 to N-M do {
        if(codeP = codeT){
            ok = true;
            for j = 0 to M-1 do if P[j] != T[j + s] then {
                ok = false; break;
            }
            if ok then cnt++;
        }
        t = T[s]*e;  t = t %Q;      t = (codeT - t)%Q;
        codeT = (t*d + T[s+M])%Q;
    }
    return cnt;
}
```

# THUẬT TOÁN KMP

- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài n) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài m). Hãy tìm tất cả các vị trí xuất hiện của P trong T
- Thuật toán KMP
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: trái qua phải
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN KMP

- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài n) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài m). Hãy tìm tất cả các vị trí xuất hiện của P trong T
- Thuật toán KMP
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: trái qua phải
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt

a	b	a	c	b	a	c	a	c	b	a	c
---	---	---	---	---	---	---	---	---	---	---	---

a	c	b	a
---	---	---	---

# THUẬT TOÁN KMP

- Tiền xử lý:
  - $\pi[q]$ : độ dài của tiền tố dài nhất cũng đồng thời là hậu tố **ngắt** của xâu  $P[1..q]$

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Tiền xử lý:
  - $\pi[q]$ : độ dài của tiền tố dài nhất cũng đồng thời là hậu tố *ngắt* của xâu  $P[1..q]$

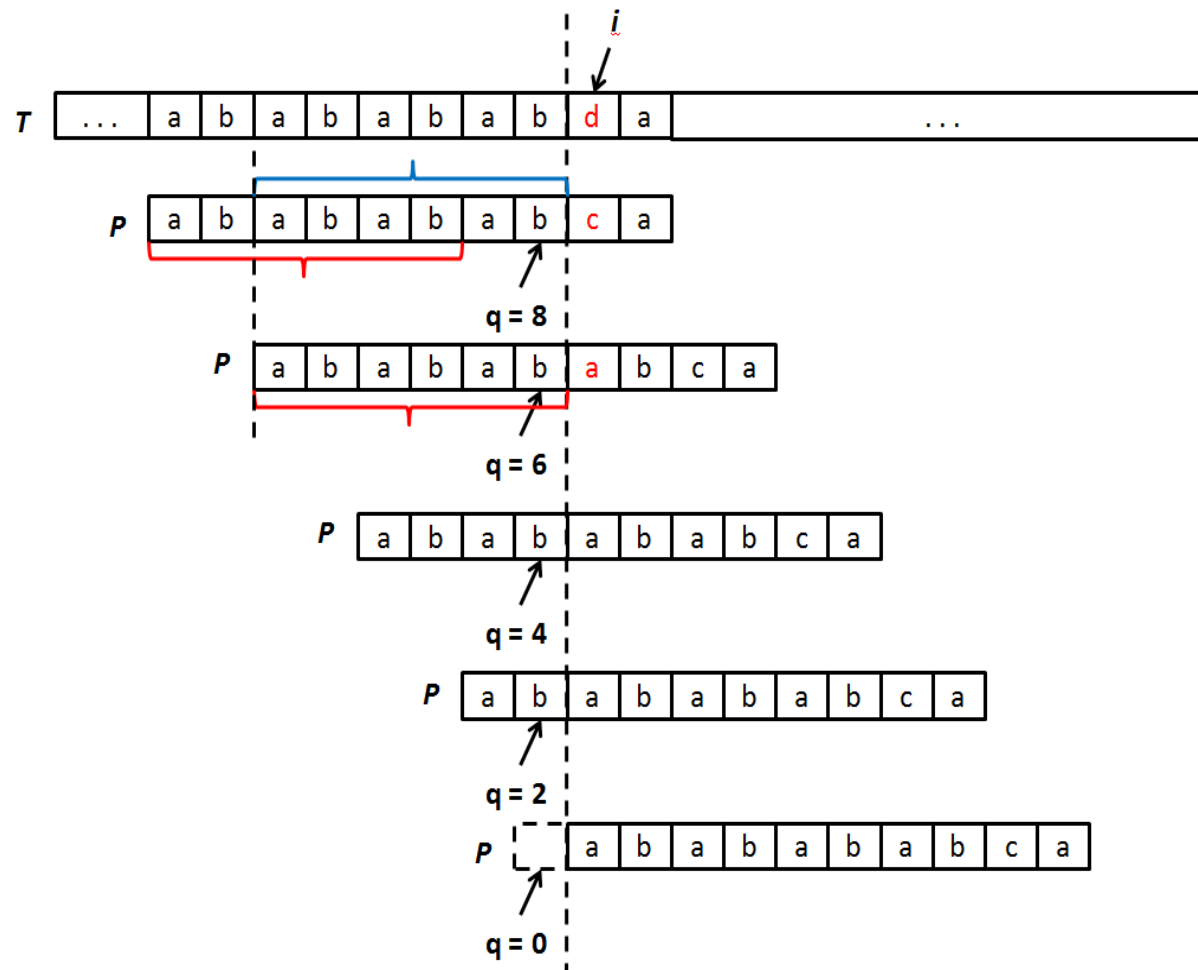
	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

```
computePI(P){
    pi[1] = 0;
    k = 0;
    for q = 2 → M do {
        while(k > 0 && P[k+1] != P[q])
            k = pi[k];
        if P[k+1] = P[q] then
            k = k + 1;
        pi[q] = k;
    }
}
```

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
    q = 0;  
    for i = 1..N do {  
        while q > 0 && P[q+1] != T[i]  
            q = pi[q];  
        if P[q+1] = T[i]  
            q = q + 1;  
        if(q = M){  
            output(i-M+1);  
            q = pi[q];  
        }  
    }  
}
```



# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
    q = 0;  
    for i = 1..N do {  
        while q > 0 && P[q+1] != T[i]  
            q = pi[q];  
        if P[q+1] = T[i]  
            q = q + 1;  
        if(q = M){  
            output(i-M+1);  
            q = pi[q];  
        }  
    }  
}
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
a	a	a	b	a	b	a	b	c	a	c	a	b	a	b	a	b	a	b	c	a

a	b	a	b	a	b	c	a
---	---	---	---	---	---	---	---

Khởi tạo:  $q = 0$

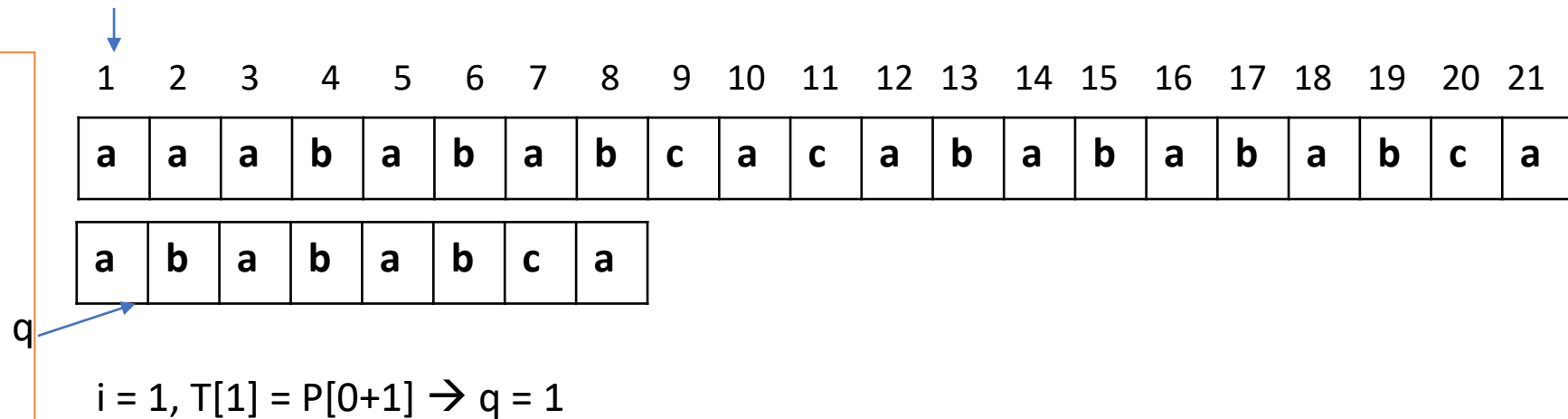
	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1



# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
    q = 0;  
    for i = 1..N do {  
        while q > 0 && P[q+1] != T[i]  
            q = pi[q];  
        if P[q+1] = T[i]  
            q = q + 1;  
        if(q = M){  
            output(i-M+1);  
            q = pi[q];  
        }  
    }  
}
```

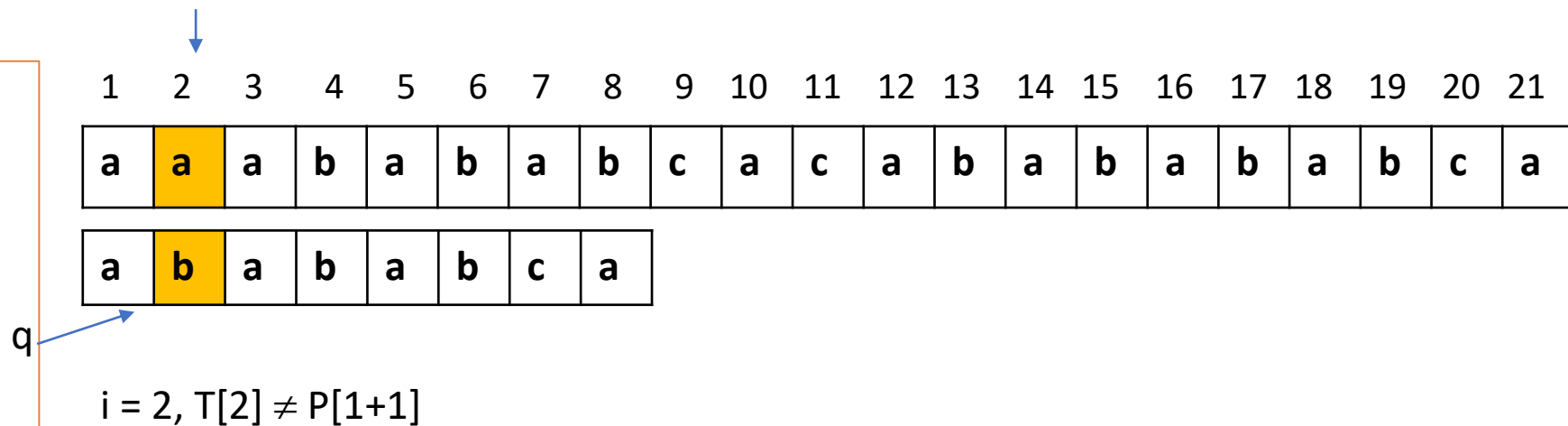


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

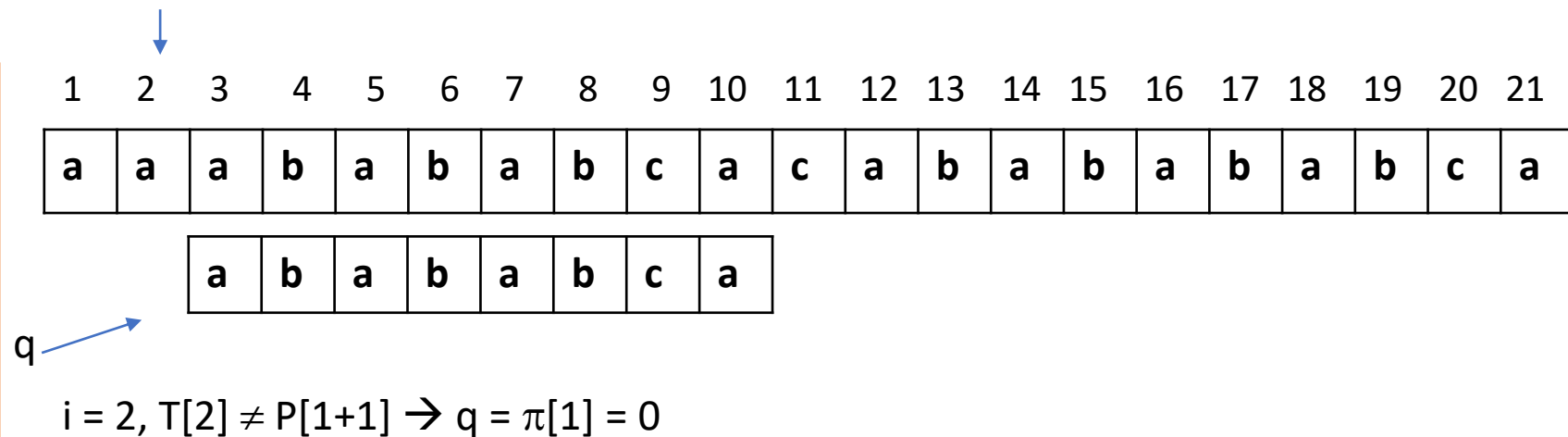


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

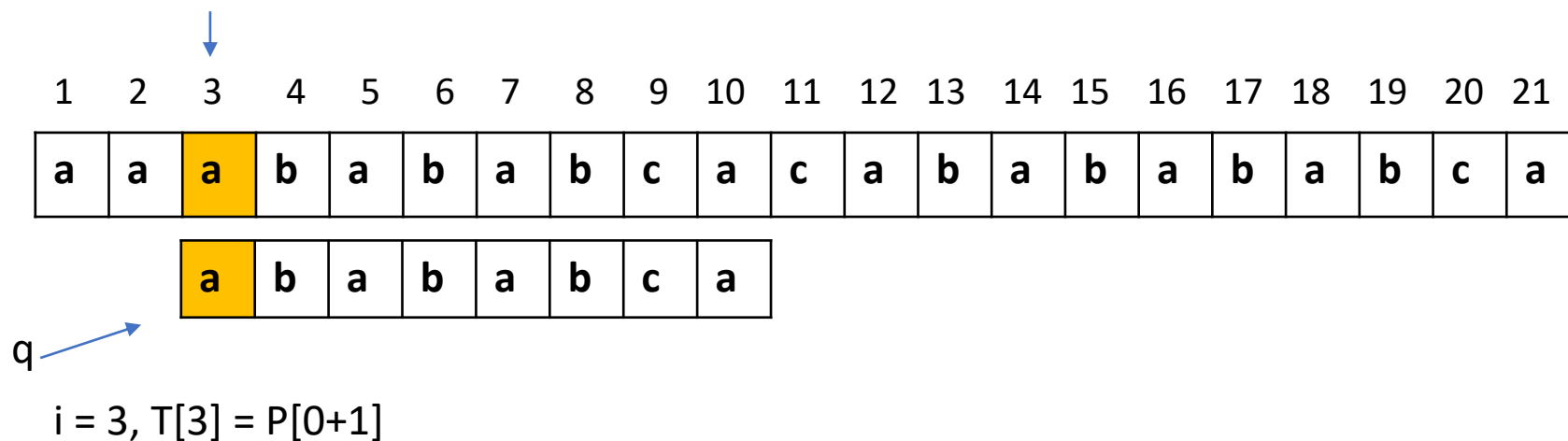


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

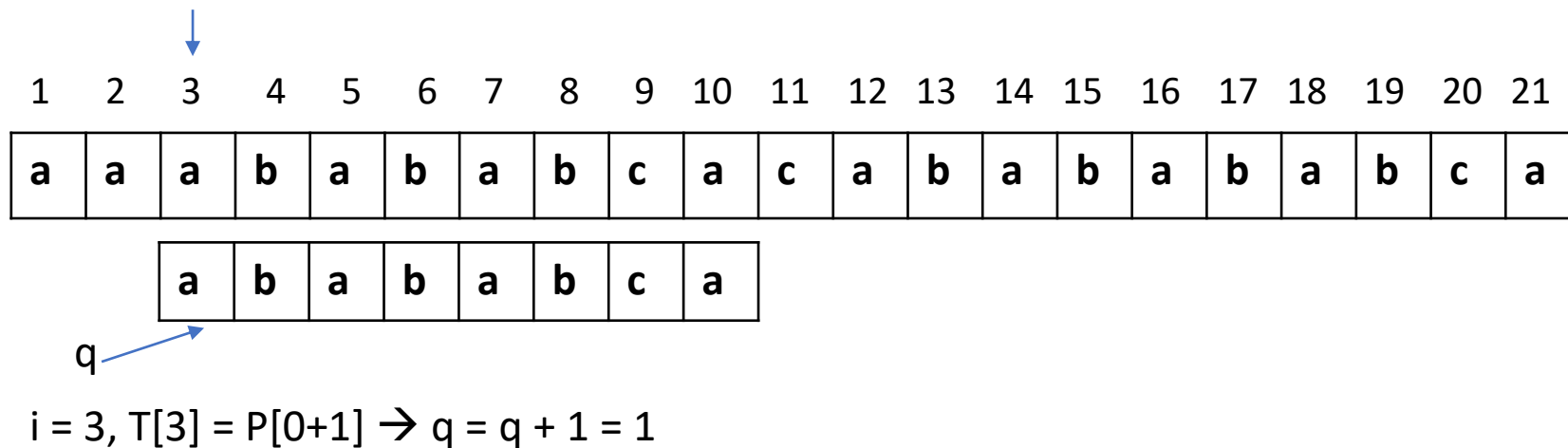


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

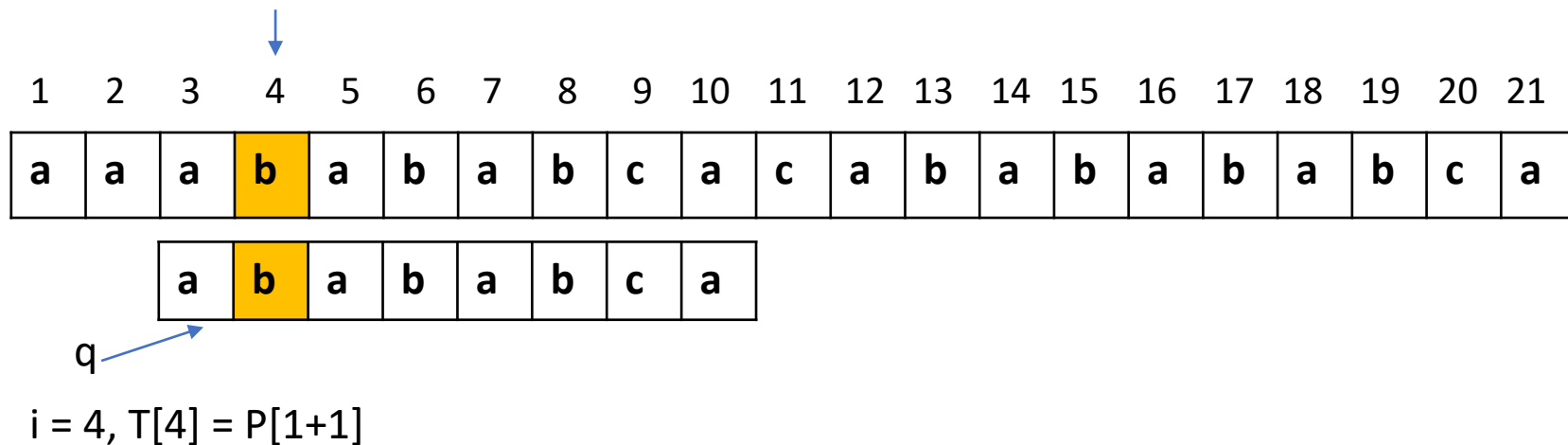


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

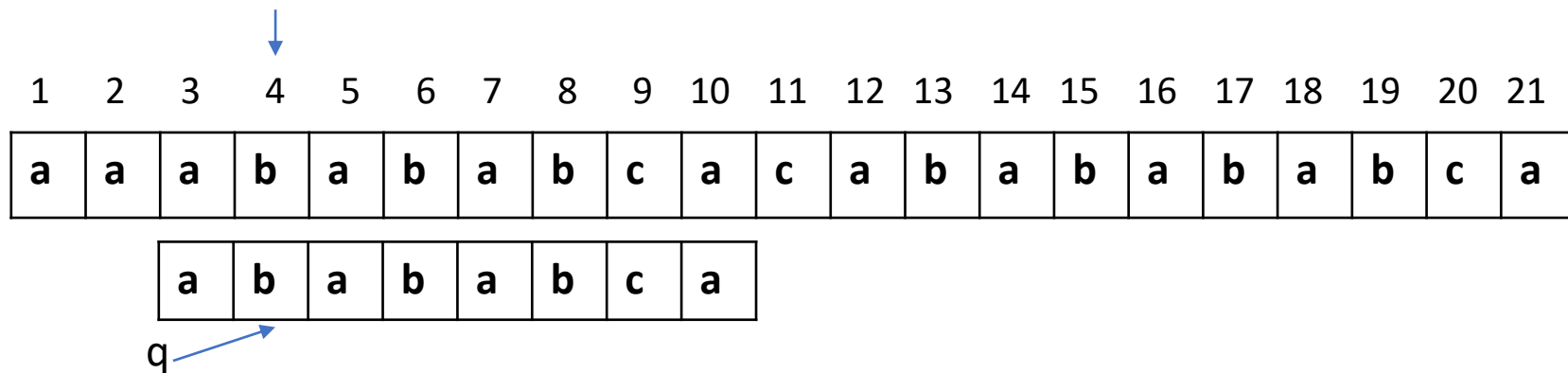


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

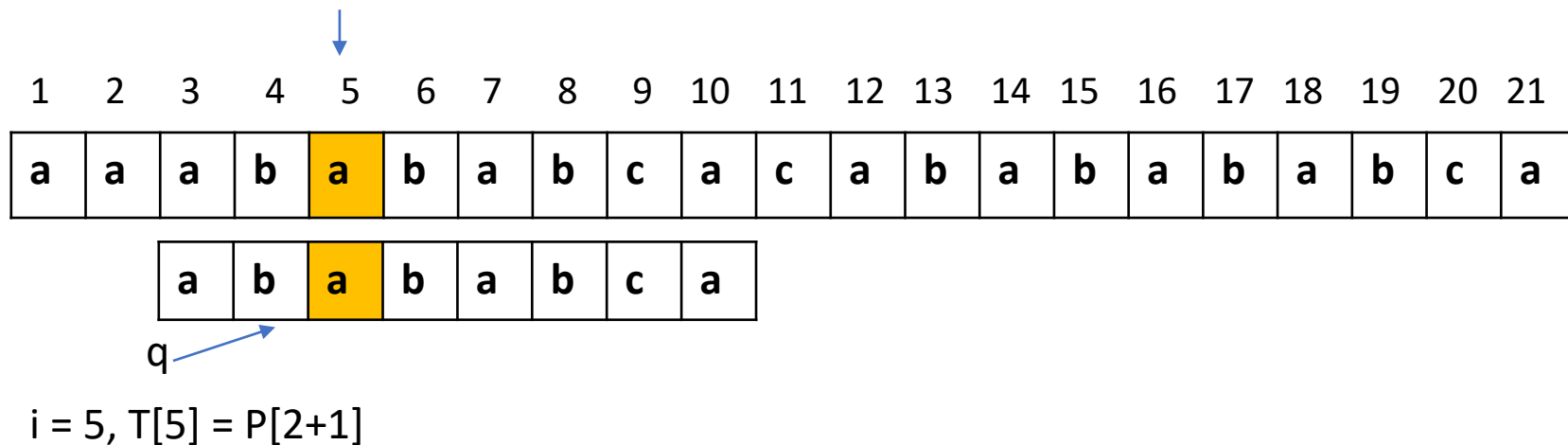


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



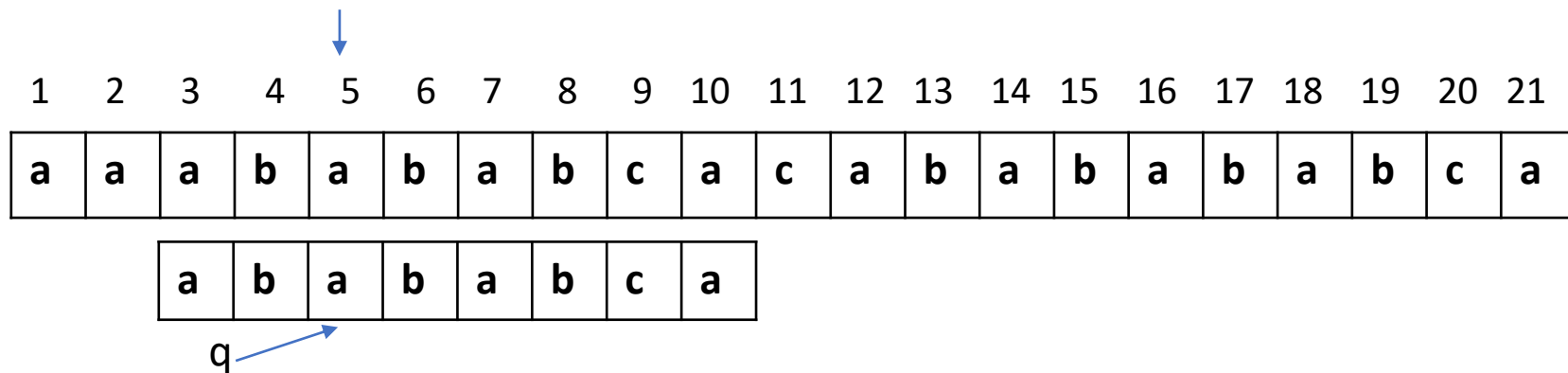
	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1



# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

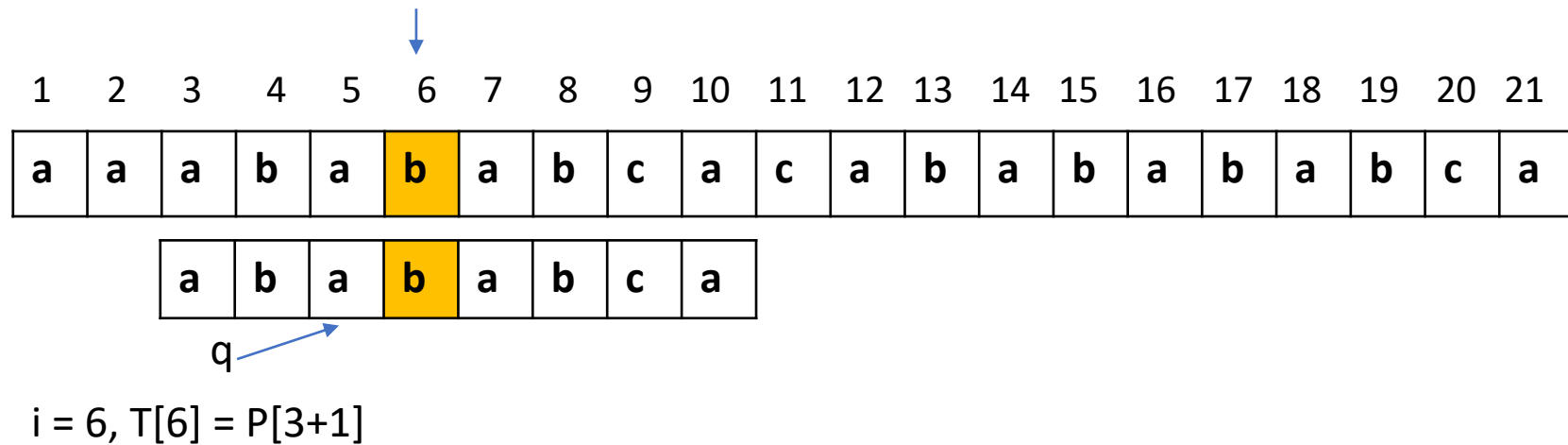


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

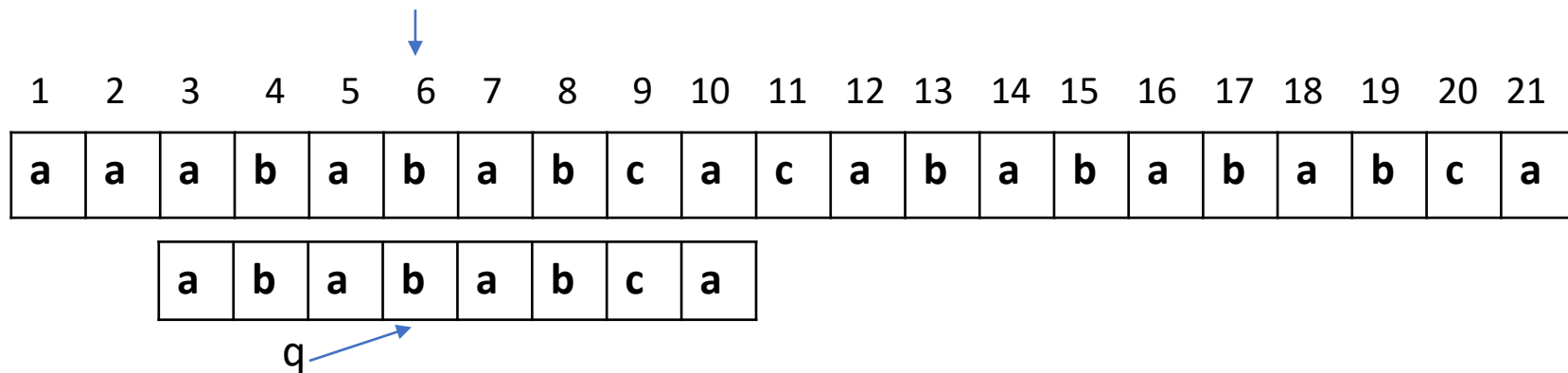


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

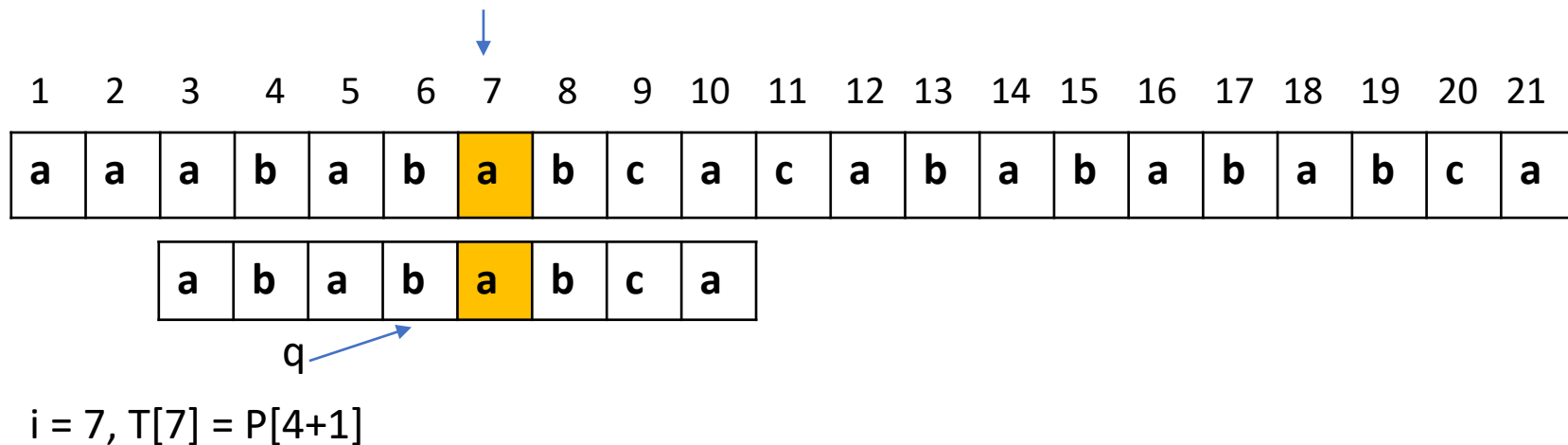


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

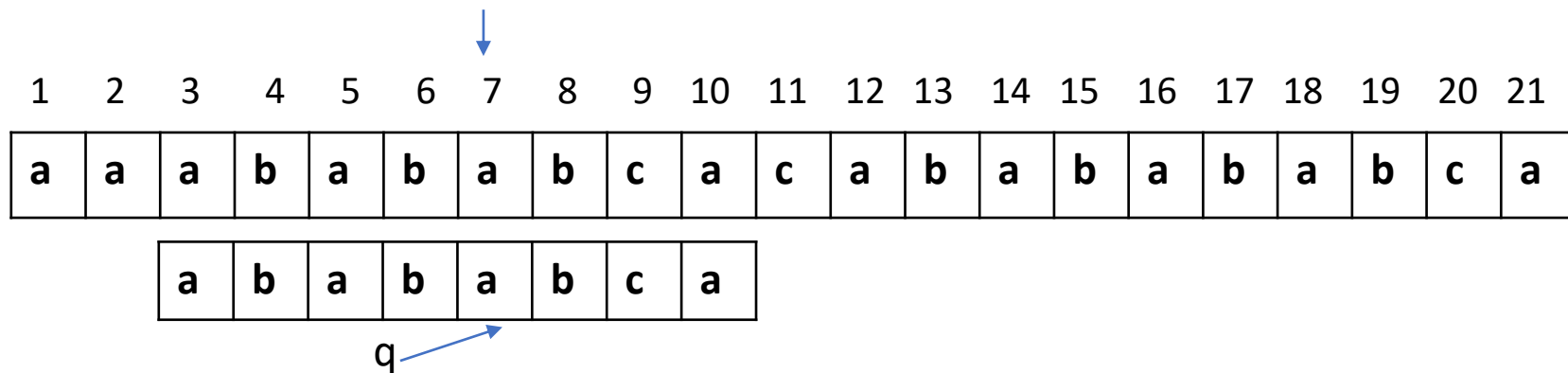


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



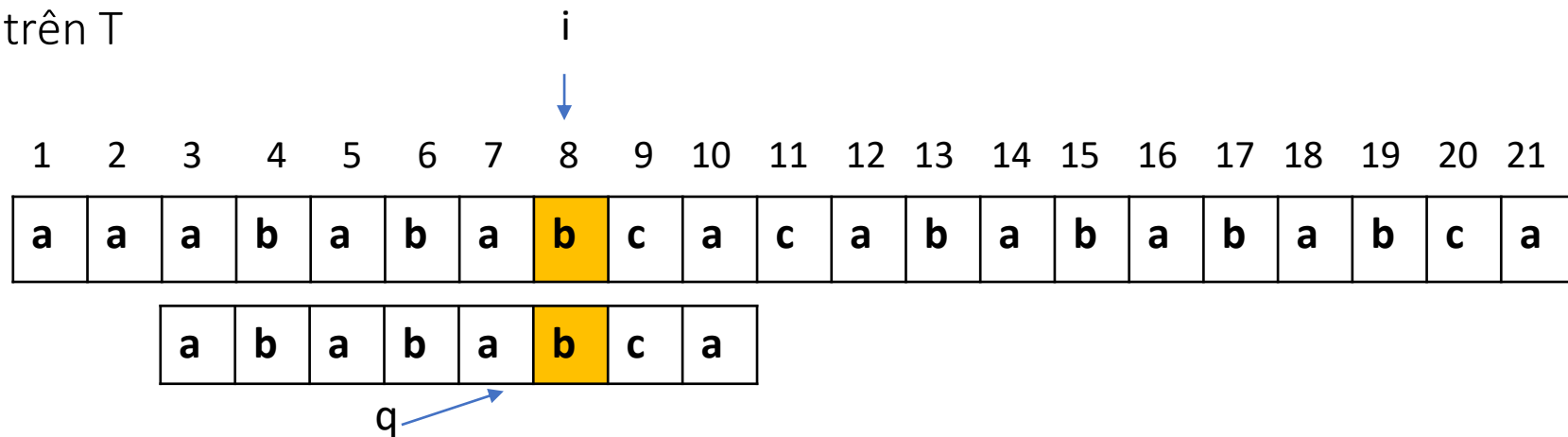
$i = 7, T[7] = P[4+1] \rightarrow q = q + 1 = 5$

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



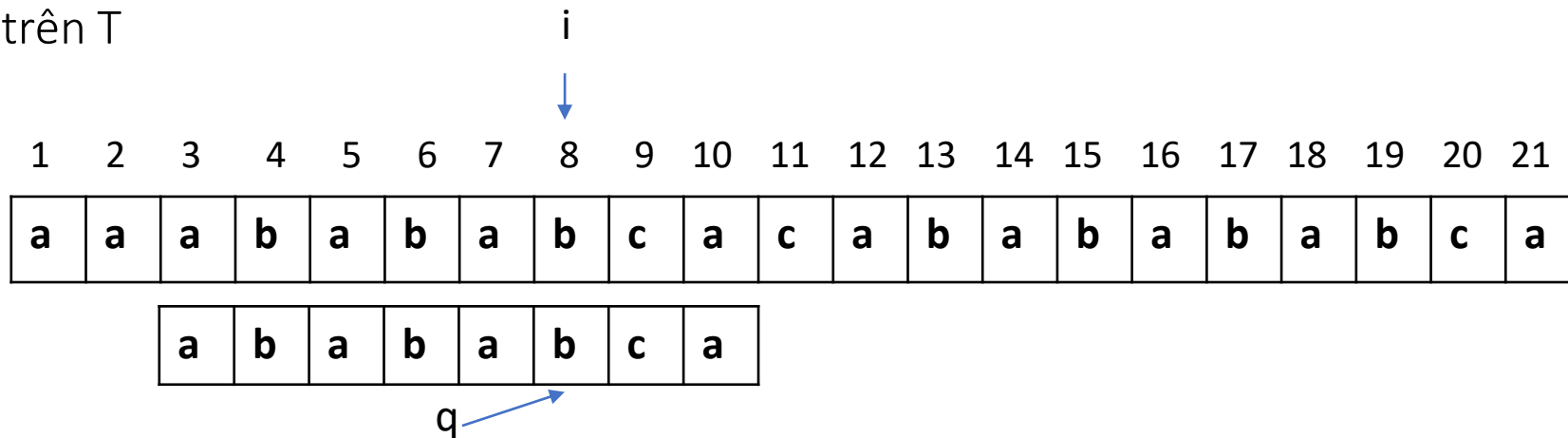
$i = 8, T[8] = P[5+1]$

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



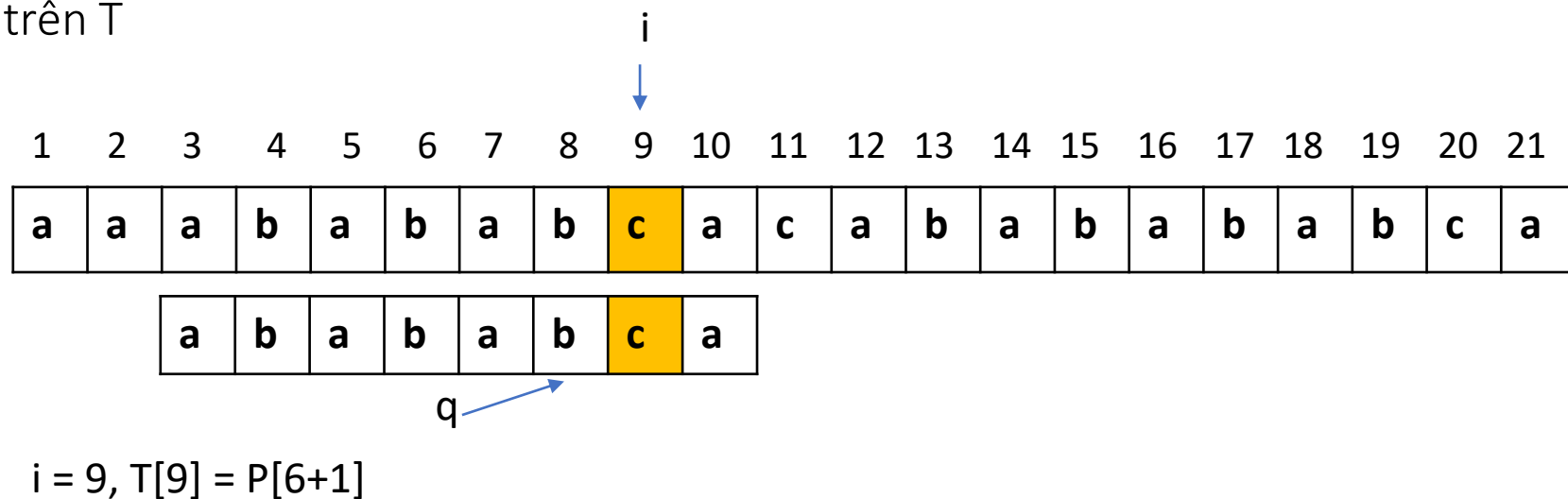
$i = 8, T[8] = P[5+1] \rightarrow q = q + 1 = 6$

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



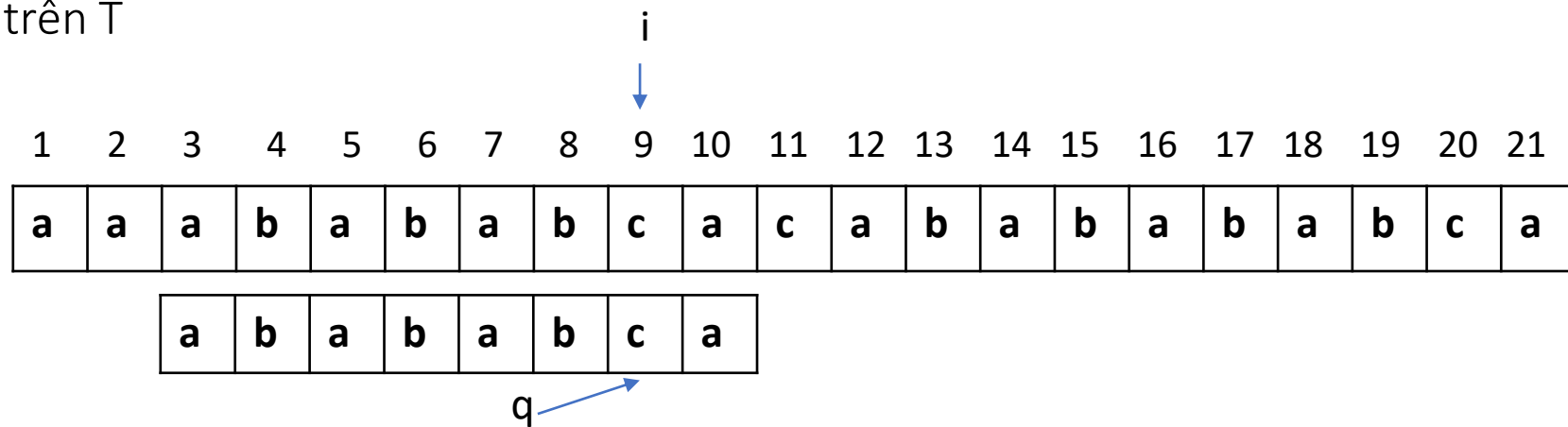
	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1



# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
    q = 0;  
    for i = 1..N do {  
        while q > 0 && P[q+1] != T[i]  
            q = pi[q];  
        if P[q+1] = T[i]  
            q = q + 1;  
        if(q = M){  
            output(i-M+1);  
            q = pi[q];  
        }  
    }  
}
```



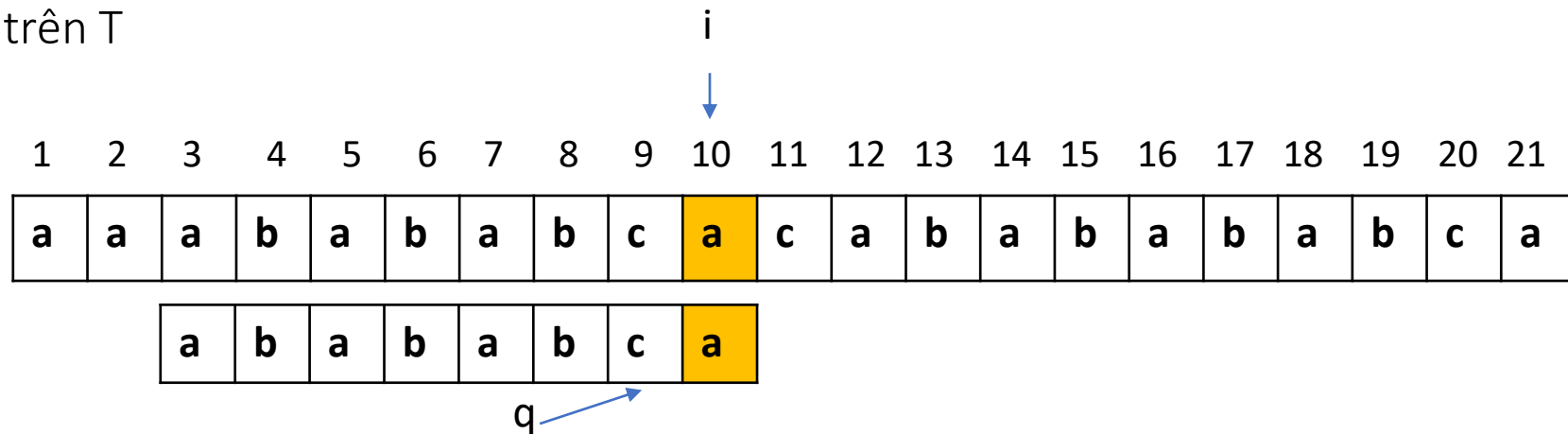
$i = 9, T[9] = P[6+1] \rightarrow q = q + 1 = 7$

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

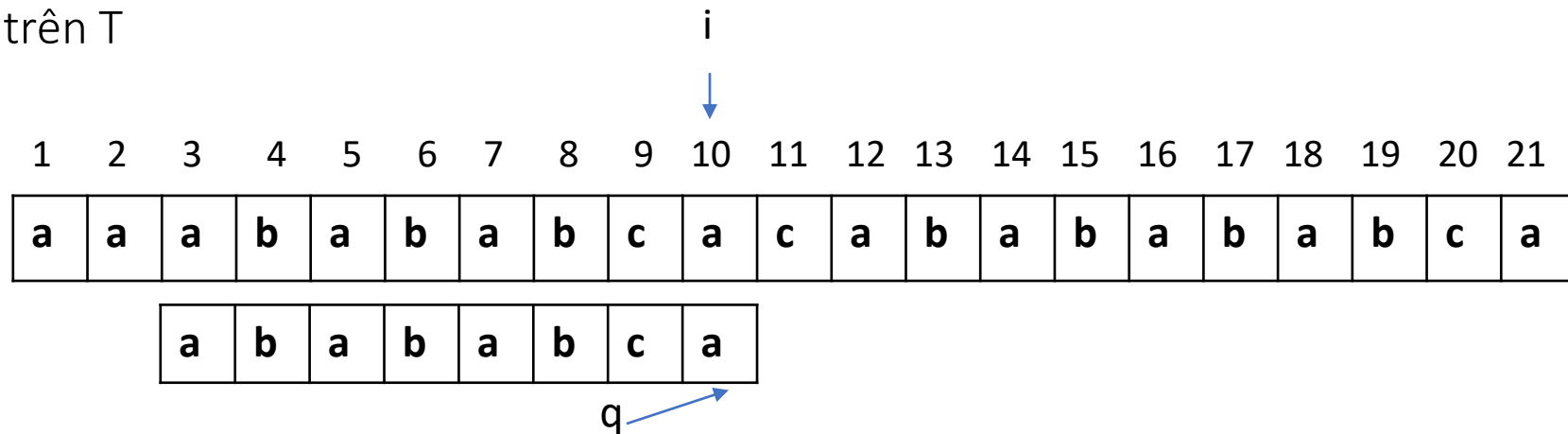


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



$i = 10, T[10] = P[7+1] \rightarrow q = q + 1 = 8$

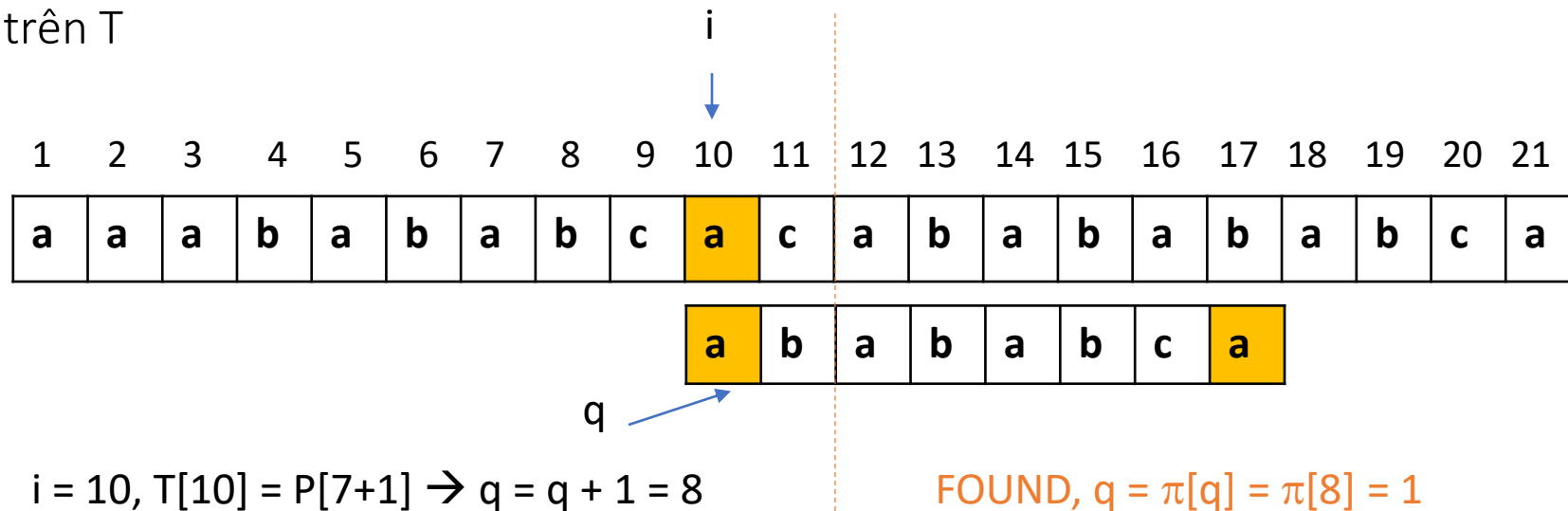
FOUND

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){
  q = 0;
  for i = 1..N do {
    while q > 0 && P[q+1] != T[i]
      q = pi[q];
    if P[q+1] = T[i]
      q = q + 1;
    if(q = M){
      output(i-M+1);
      q = pi[q];
    }
  }
}
```

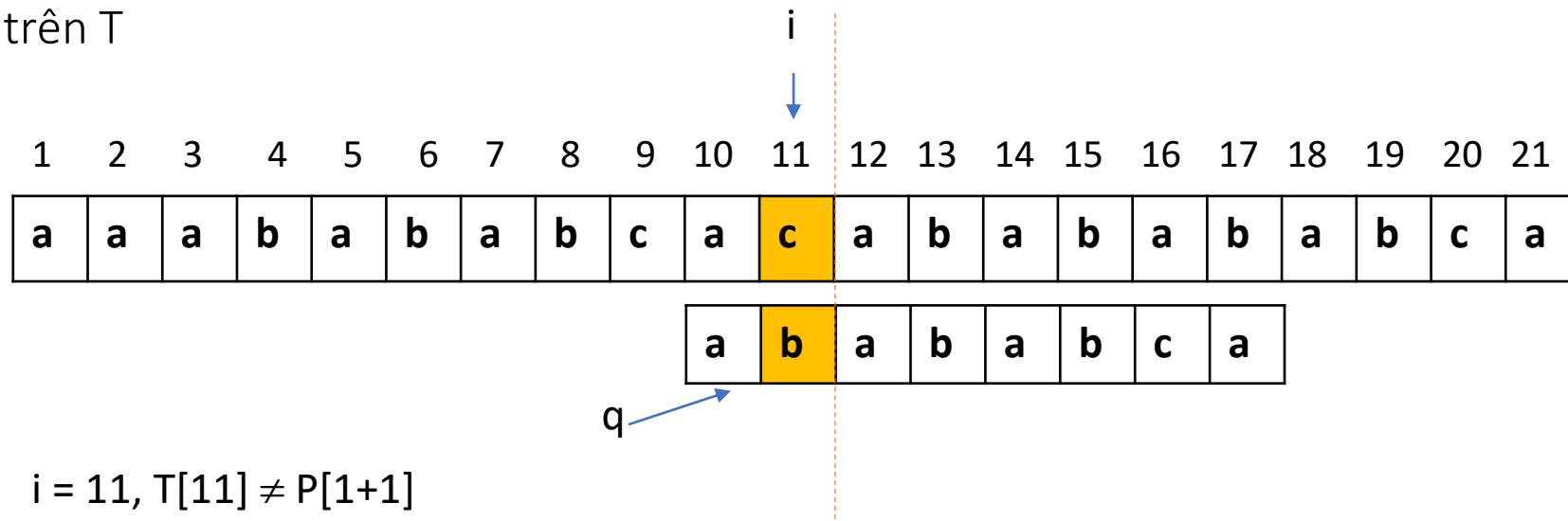


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt xâu mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```

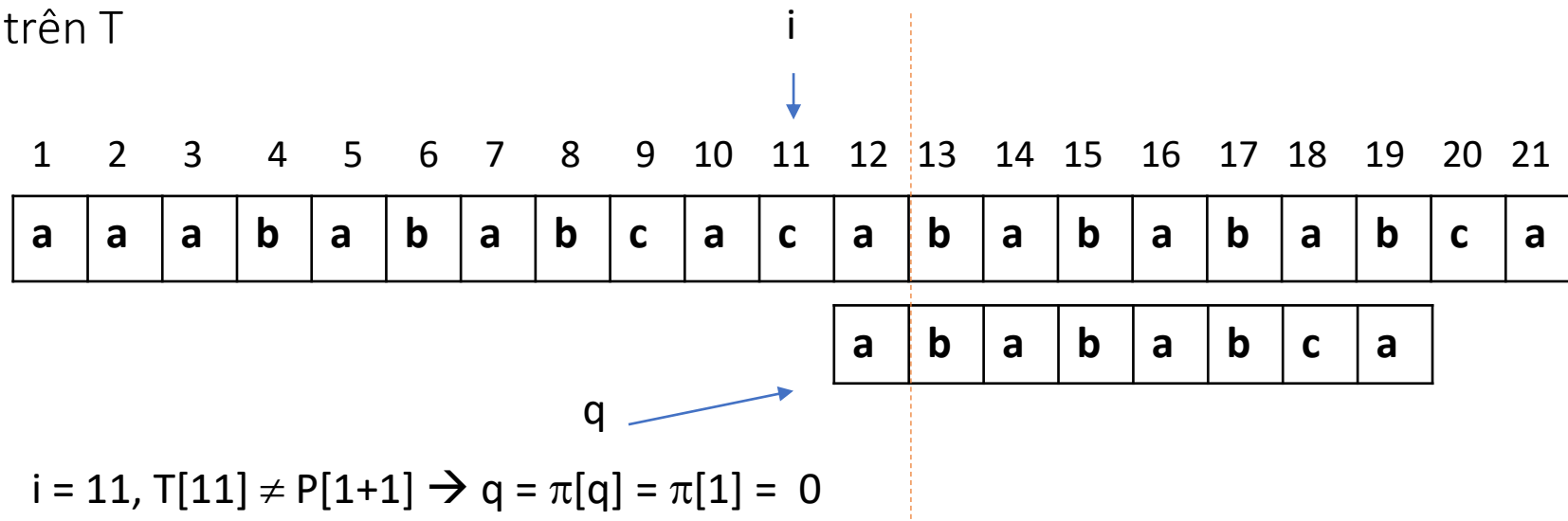


	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

- Trượt mẫu P từ trái qua phải trên T

```
kmp(P, T){  
  q = 0;  
  for i = 1..N do {  
    while q > 0 && P[q+1] != T[i]  
      q = pi[q];  
    if P[q+1] = T[i]  
      q = q + 1;  
    if(q = M){  
      output(i-M+1);  
      q = pi[q];  
    }  
  }  
}
```



	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
$\pi$	0	0	1	2	3	4	0	1

# THUẬT TOÁN KMP

```
computePi(p){
    pi[1] = 0;
    int k = 0;
    for q = 2 to p.length()-1 do {
        while(k > 0 && p[k+1] != p[q]) do
            k = pi[k];
        if (p[k+1] = p[q]) then k = k + 1;
        pi[q] = k;
    }
}
```

```
kmp(P, T){
    P = "-" + P; T = "-" + T;
    computePi(P);
    cnt = 0;
    N = T.length()-1;
    M = P.length()-1;
    q = 0;
    for i= 1 to N do {
        while(q > 0 and P[q+1] != T[i]) do
            q = pi[q];
        if(P[q+1] = T[i]) then
            q = q + 1;
        if(q = M) then {
            cnt += 1; q = pi[q];
        }
    }
    return cnt;
}
```

A large graphic on the left side of the slide. It features a dark blue background with a circular pattern of red dots of varying sizes, creating a sense of depth and movement. The word "HUST" is centered within this graphic in a white, bold, sans-serif font.

# HUST

# THANK YOU !