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.

NỘI DUNG

- 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 brute force: cho trượt từ đầu tới cuối rồi so khớp trong xâu mẫu.

=> độ phức tạp O(M*N)

==> Cần một thuật toán so khớp hiệu quả hơn

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

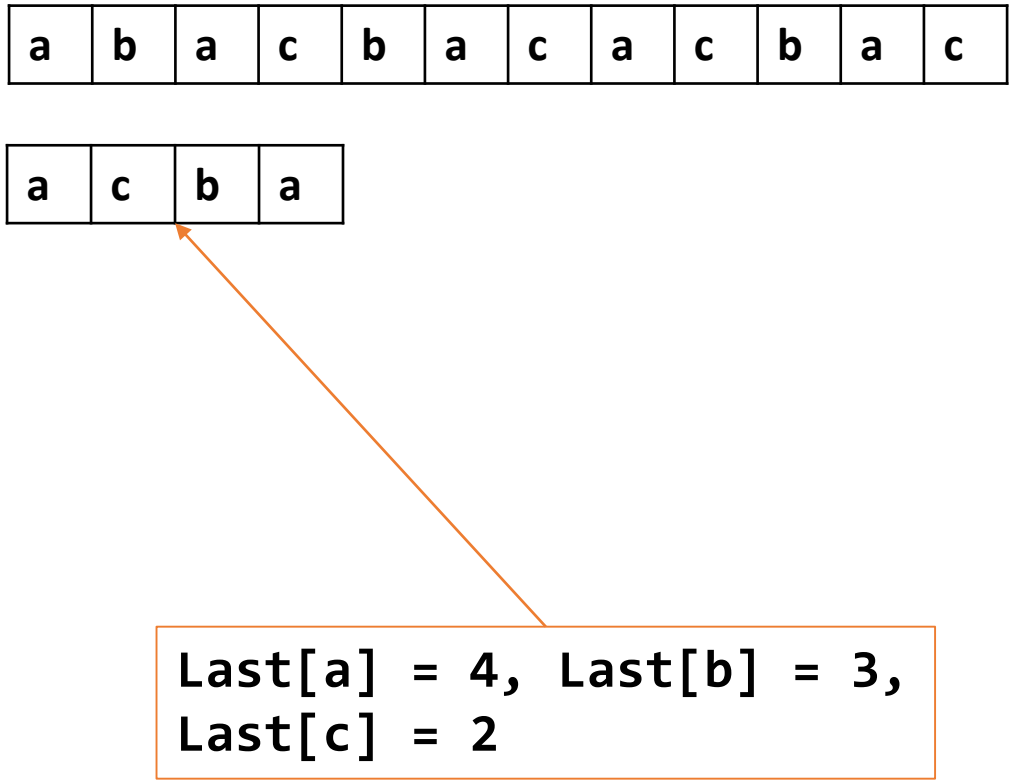
a	c	b	a

- Thuật toán Boyer Moore
  - 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

THUẬT TOÁN BOYER MOORE

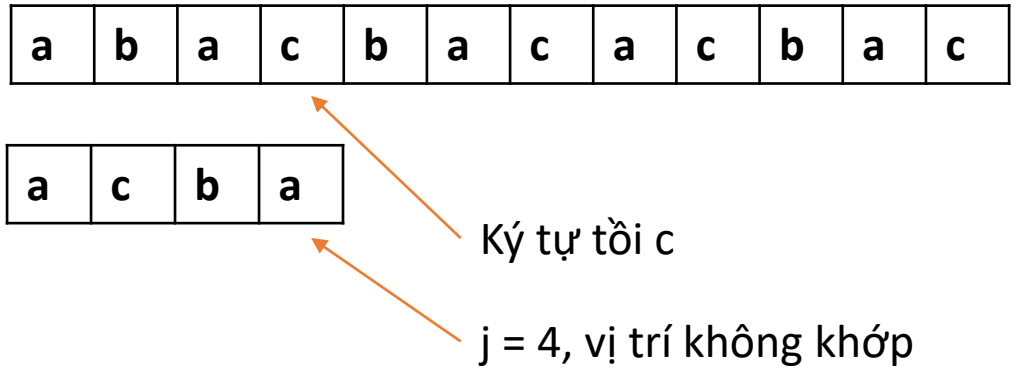
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T

- 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*



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 tồi là *x* (ký tự trên *T*), *P* sẽ được trượt sang phải

    max{*j* - Last[*x*], 1} vị trí ≥ 1

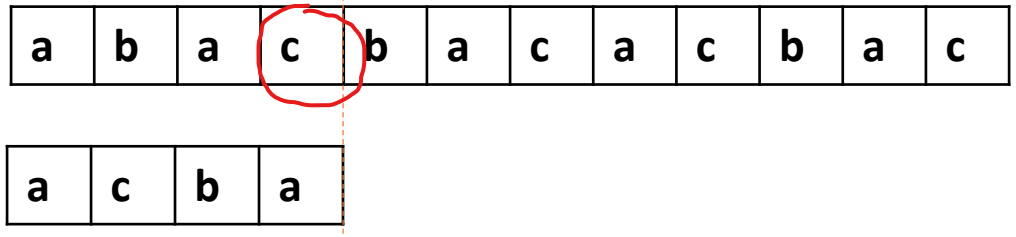
    đảm bảo việc so khớp vô ích không xảy ra và không bỏ qua trường hợp nào

    trong đó *j* là chỉ số hiện tại (xảy ra không khớp) trên *P* khi so khớp ký tự từ phải qua trái

    nếu match thì sau đấy chỉ trượt một ô thôi

THUẬT TOÁN BOYER MOORE

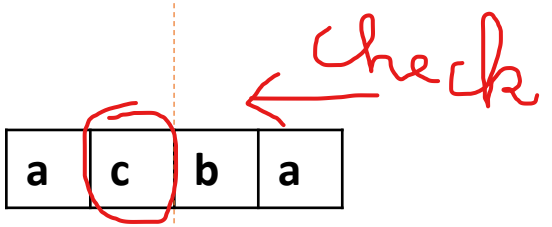
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T



- 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 tồi là *x* (ký tự trên *T*), *P* sẽ được trượt sang phải

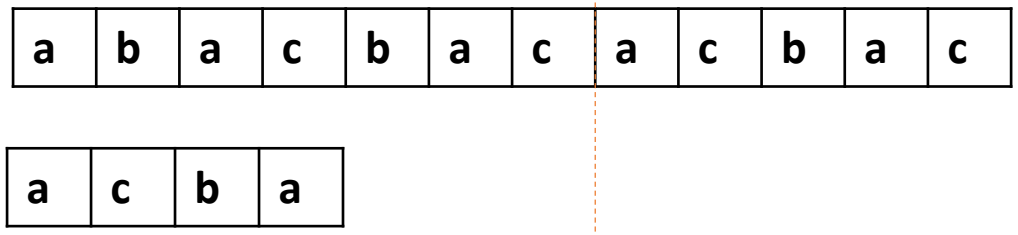
    max{*j* - Last[*x*], 1} vị trí

    trong đó *j* là chỉ số hiện tại (xảy ra không khớp) trên *P* khi so khớp ký tự từ phải qua trái



THUẬT TOÁN BOYER MOORE

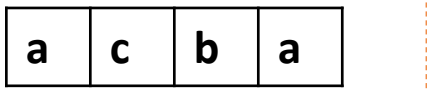
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T

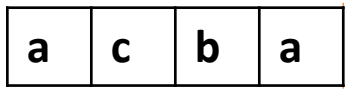


- Thuật toán Boyer Moore
  - Trượt xâu mẫu từ trái qua phải
  - Đối sánh: phải qua trái
  - Sử dụng thông tin tiền xử lý để bỏ qua càng nhiều ký tự càng tốt
  - Tiền xử lý xâu mẫu *P*
    - Last[*x*]: vị trí bên phải nhất xuất hiện ký tự *x* trong *P*
  - Khi tình trạng không khớp xảy ra với ký tự tồi là *x* (ký tự trên *T*), *P* sẽ được trượt sang phải

    max{*j* - Last[*x*], 1} vị trí

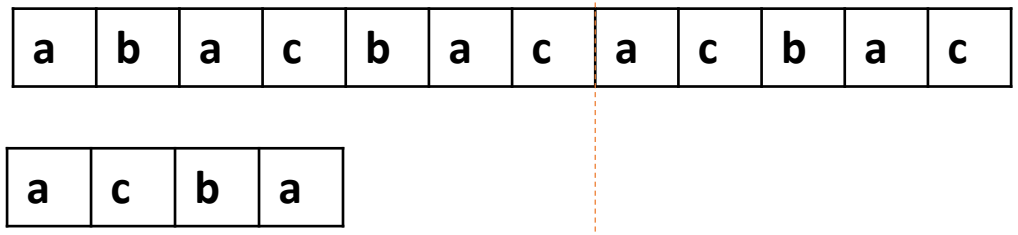
    trong đó *j* là chỉ số hiện tại (xảy ra không khớp) trên *P* khi so khớp ký tự từ phải qua trái





THUẬT TOÁN BOYER MOORE

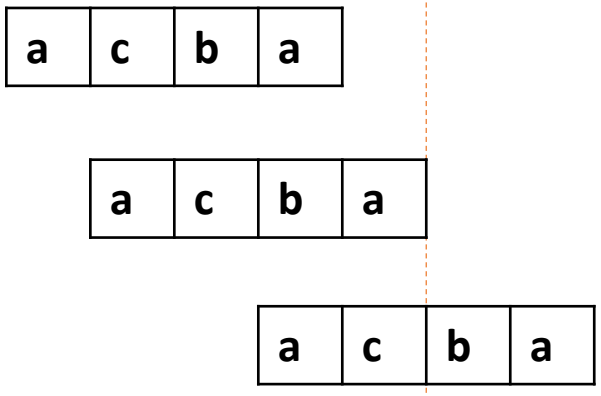
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T



- 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 tồi là *x* (ký tự trên *T*), *P* sẽ được trượt sang phải

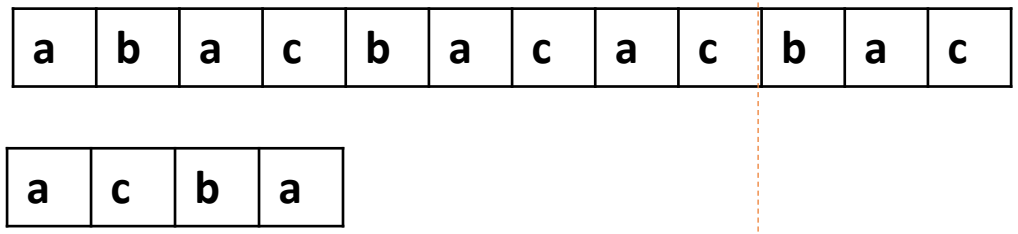
    max{*j* - Last[*x*], 1} vị trí

    trong đó *j* là chỉ số hiện tại (xảy ra không khớp) trên *P* khi so khớp ký tự từ phải qua trái



THUẬT TOÁN BOYER MOORE

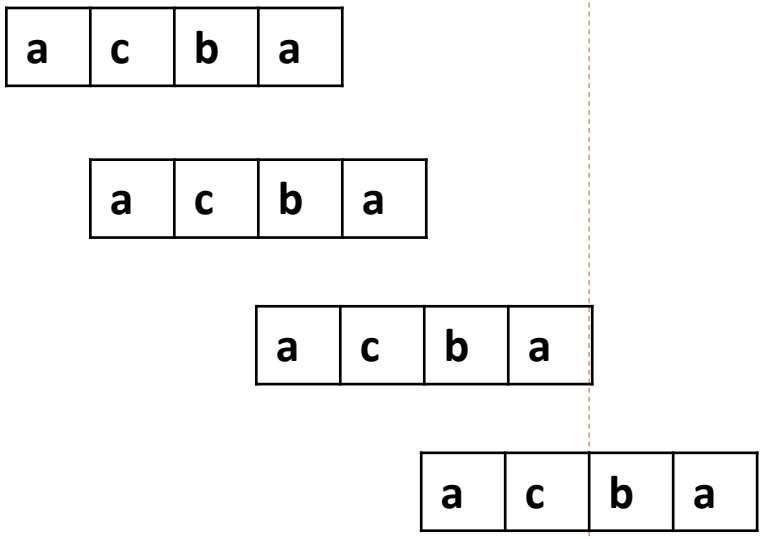
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T



- 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 tồi là *x* (ký tự trên *T*), *P* sẽ được trượt sang phải

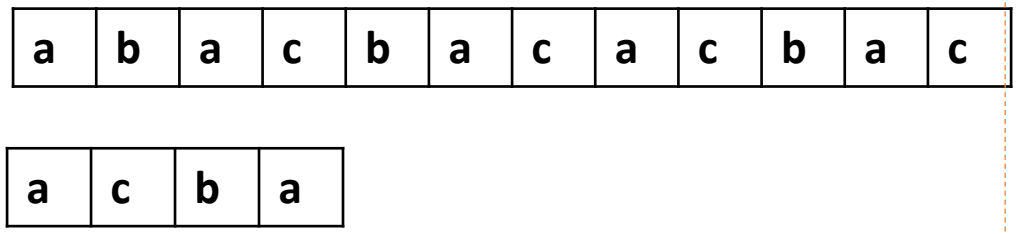
    max{*j* - Last[*x*], 1} vị trí

    trong đó *j* là chỉ số hiện tại (xảy ra không khớp) trên *P* khi so khớp ký tự từ phải qua trái



THUẬT TOÁN BOYER MOORE

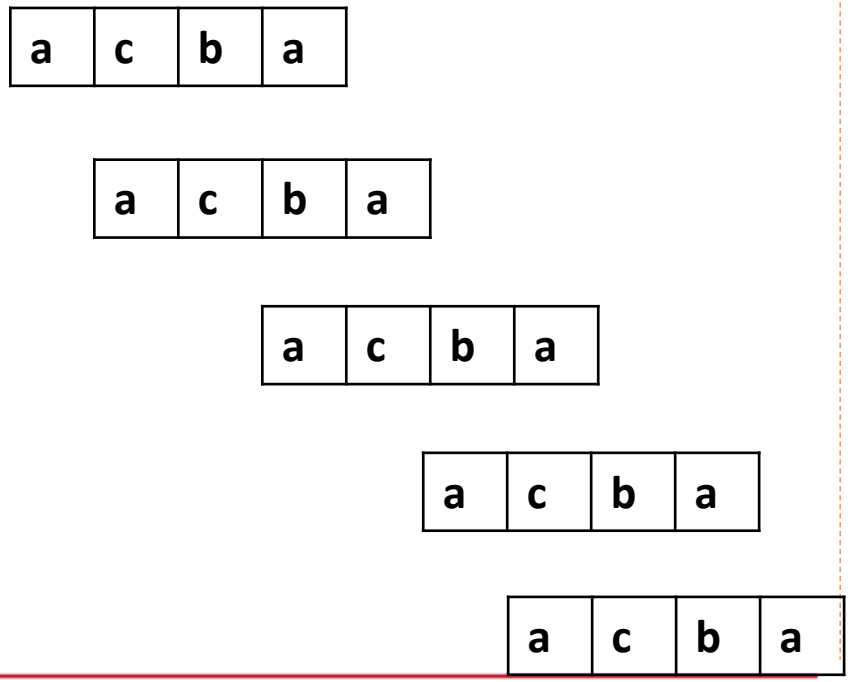
- Bài toán tìm kiếm xâu mẫu: cho văn bản T là 1 chuỗi ký tự (độ dài N) lấy từ 1 bảng cho trước và 1 xâu mẫu P (độ dài M). Hãy tìm tất cả các vị trí xuất hiện của P trong T



- 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 tồi 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



THUẬT TOÁN BOYER MOORE

```
computeLast(p){                    // Tiền xử lý
    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;
}
```

độ trượt

ký tự tồi nhất

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

hàm băm phổ biến của xâu

lấy d = 256

$$p = P[1]*d^{M-1} + P[2]*d^{M-2} + \ldots + P[M]*d^{0} \quad \text{mod Q}$$

thay vì so khớp lần lượt từng ký tự thì so khớp mã băm của chúng -> so khớp 2 số nguyên thôi.

vẫn cần kiểm tra lại để đảm bảo trường hợp bị xung đột.

- 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 tang 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:

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

- 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)

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

trick

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 = d^(M-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

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

a	c	b	a

- 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

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

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

a	c	b	a

- 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

THUẬT TOÁN KMP

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

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	0	0	1	2	3	4	0	1

tiền tố là xâu bắt đầu từ đầu tới một vị trí nào đó
hậu tố ngặt là xâu không bắt đầu từ ký tự đầu tiên

THUẬT TOÁN KMP

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

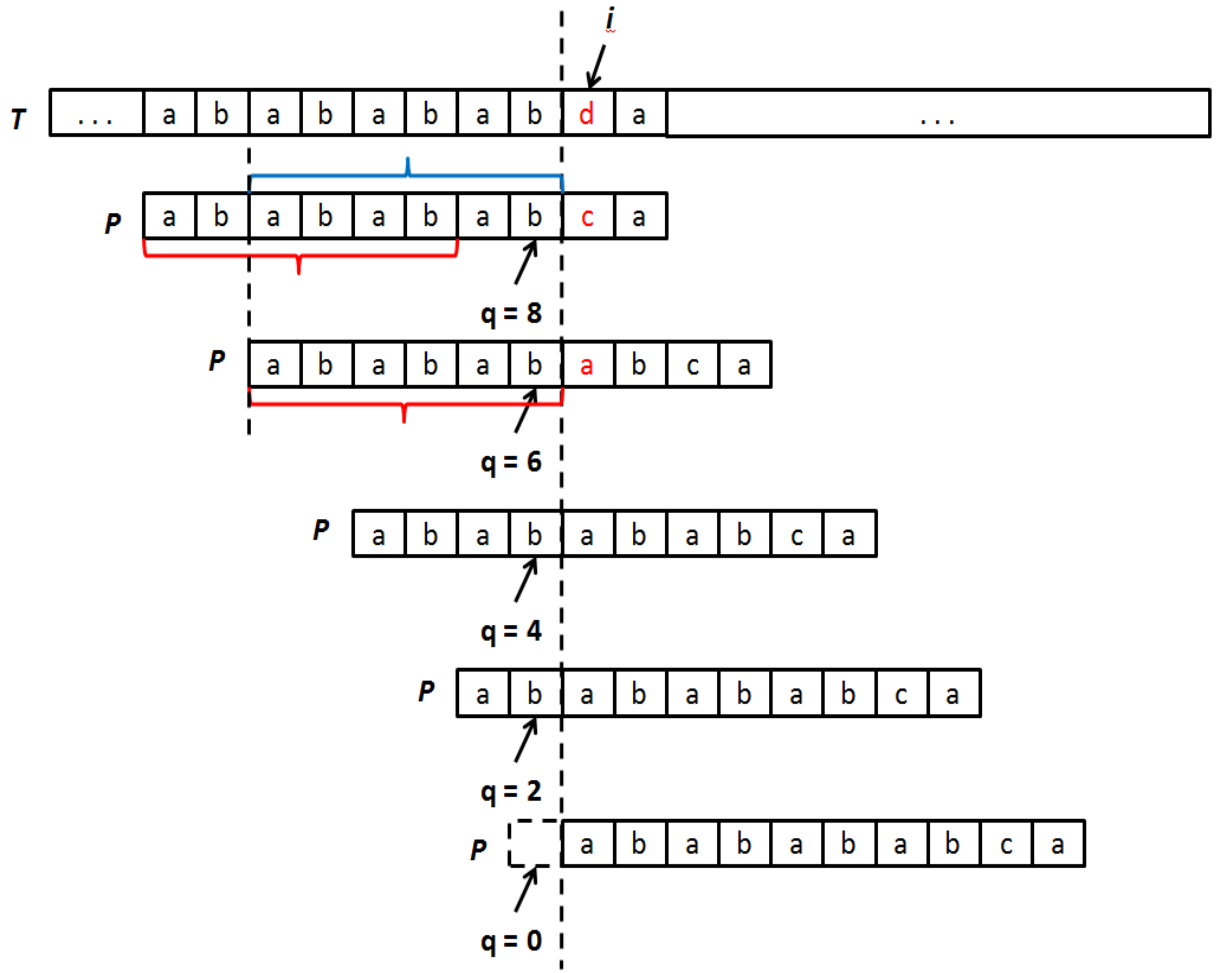
	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	0	0	1	2	3	4	0	1

```
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 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	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

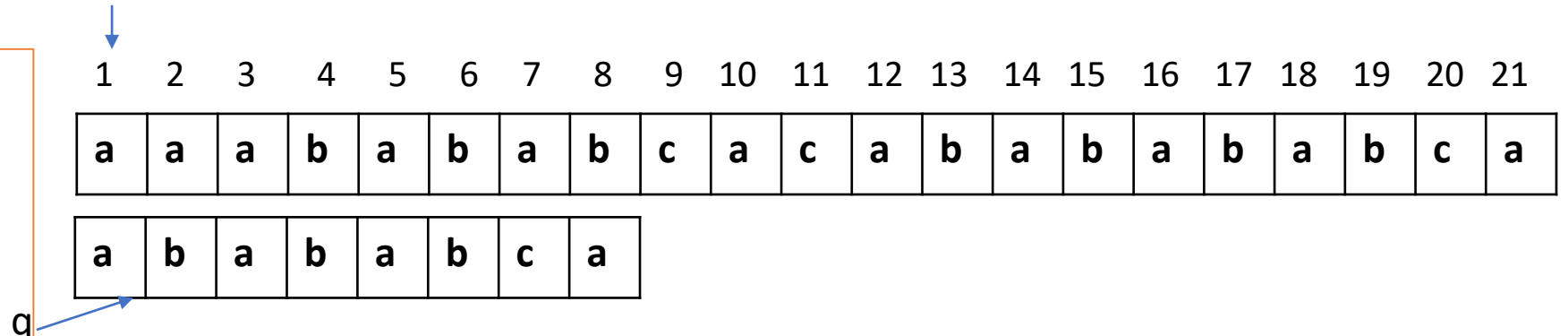
Khởi tạo: q = 0

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



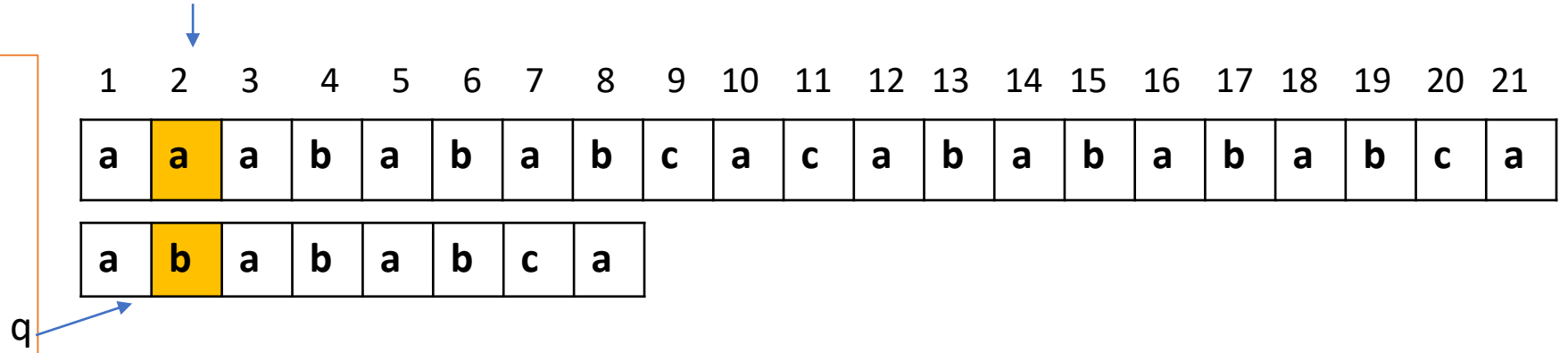
i = 1, T[1] = P[0+1] → q = 1

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



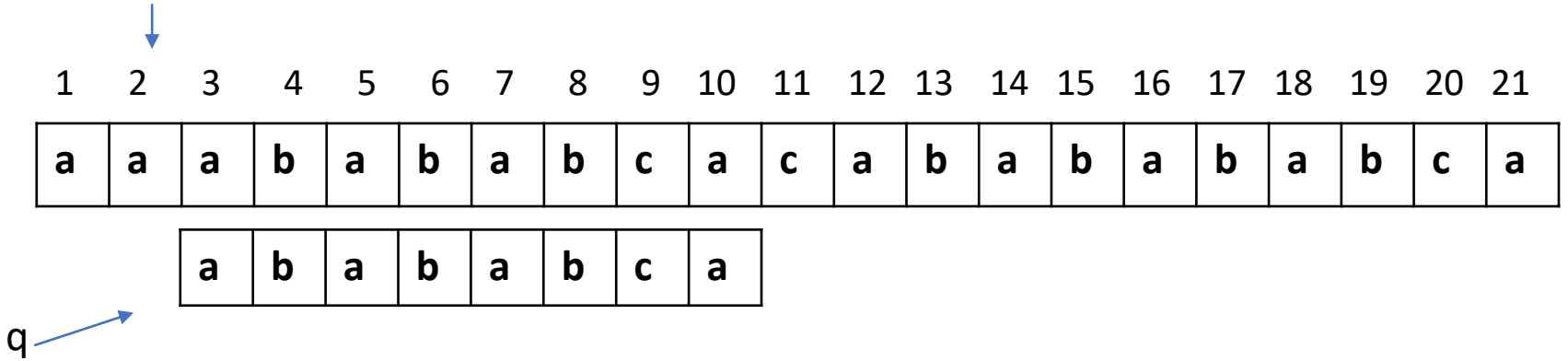
i = 2, T[2] ≠ P[1+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



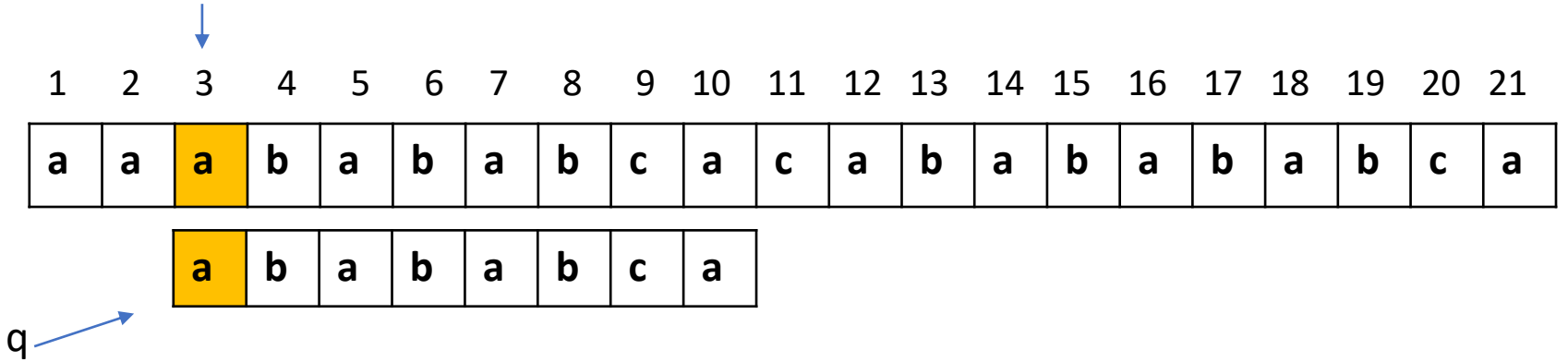
i = 2, T[2] ≠ P[1+1] → q = π[1] = 0

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



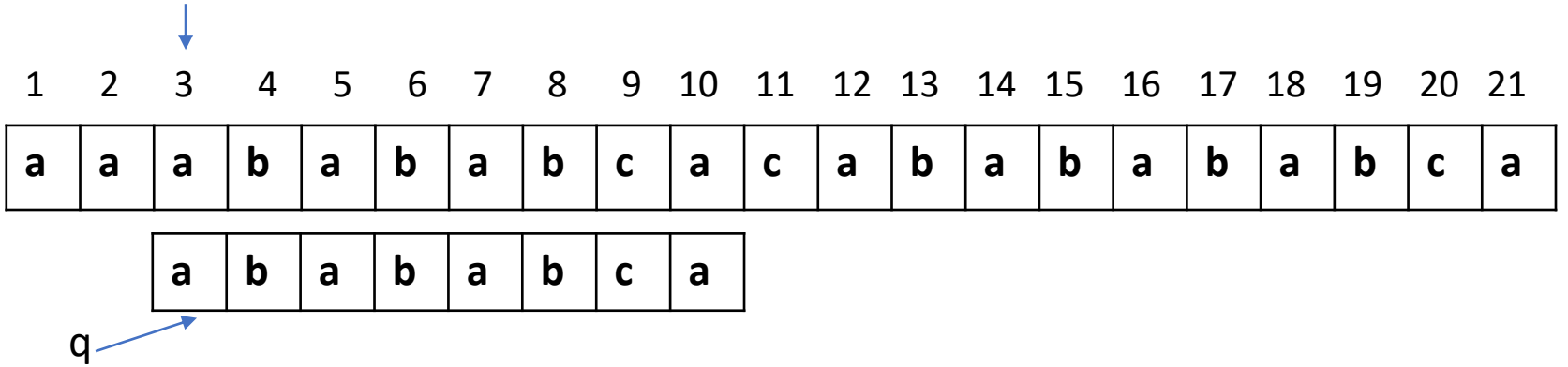
i = 3, T[3] = P[0+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



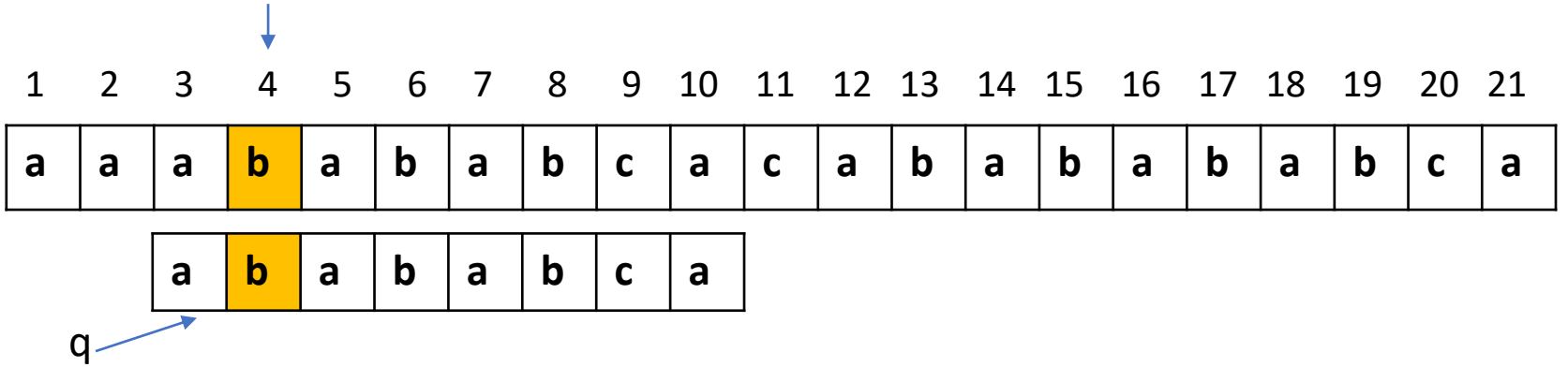
i = 3, T[3] = P[0+1] → q = q + 1 = 1

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



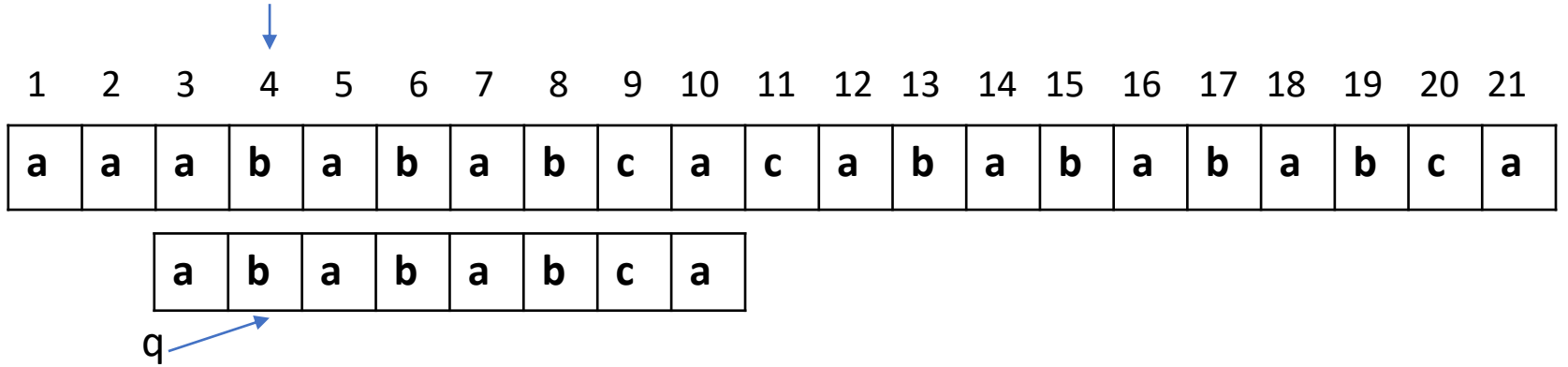
i = 4, T[4] = P[1+1]

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



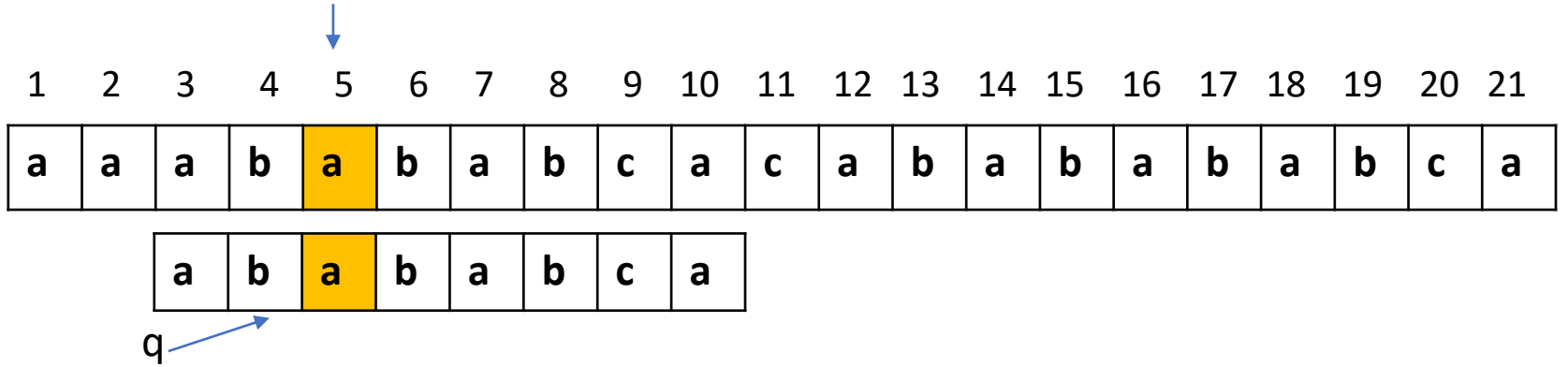
i = 4, T[4] = P[1+1] → q = q + 1 = 2

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



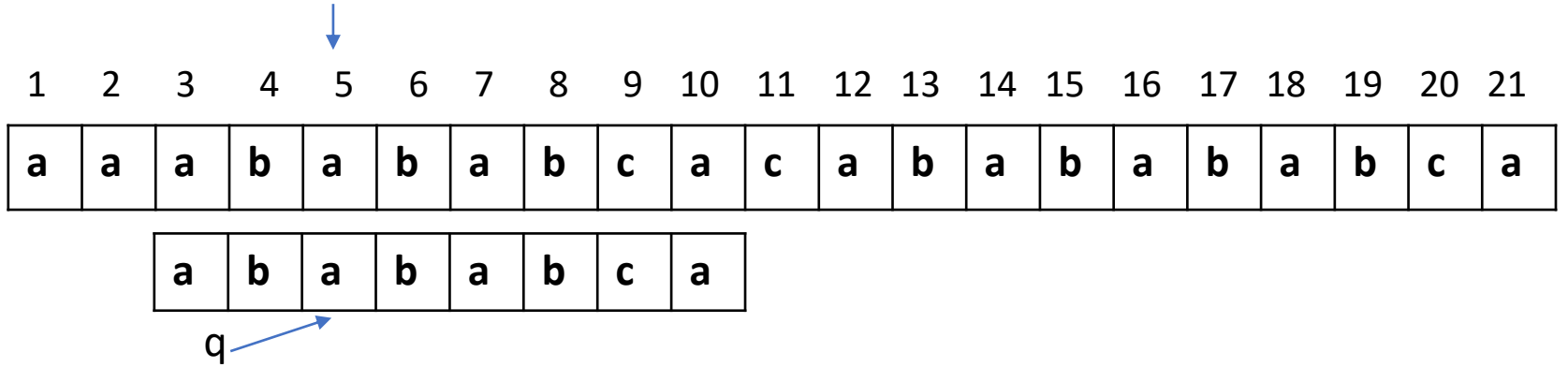
i = 5, T[5] = P[2+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



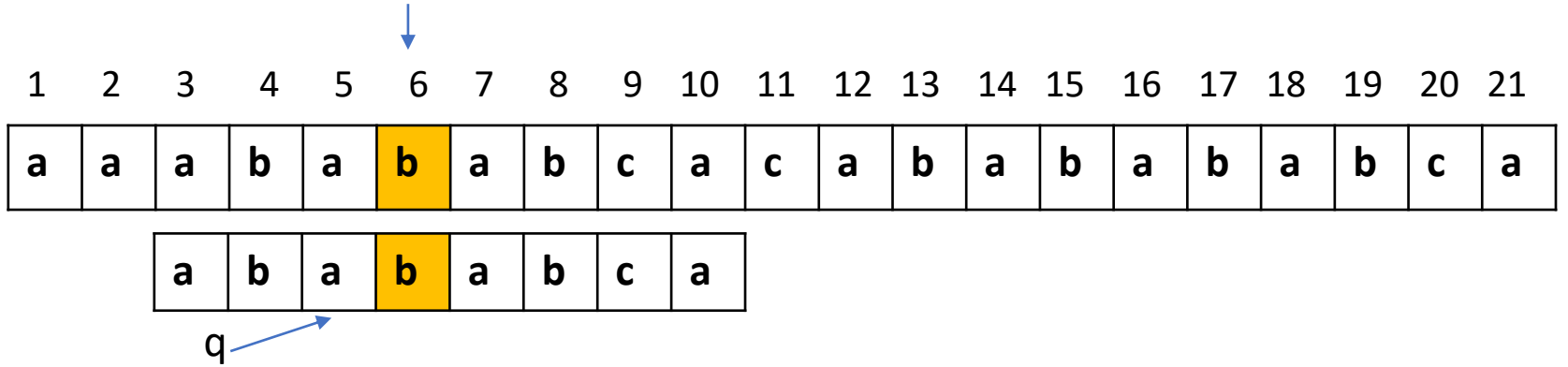
i = 5, T[5] = P[2+1] → q = q + 1 = 3

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



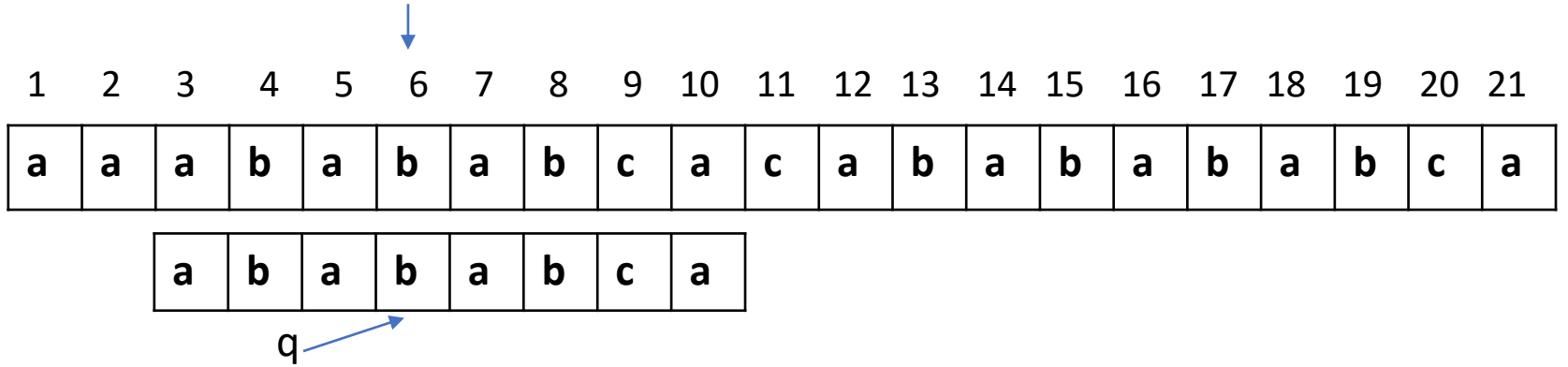
i = 6, T[6] = P[3+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



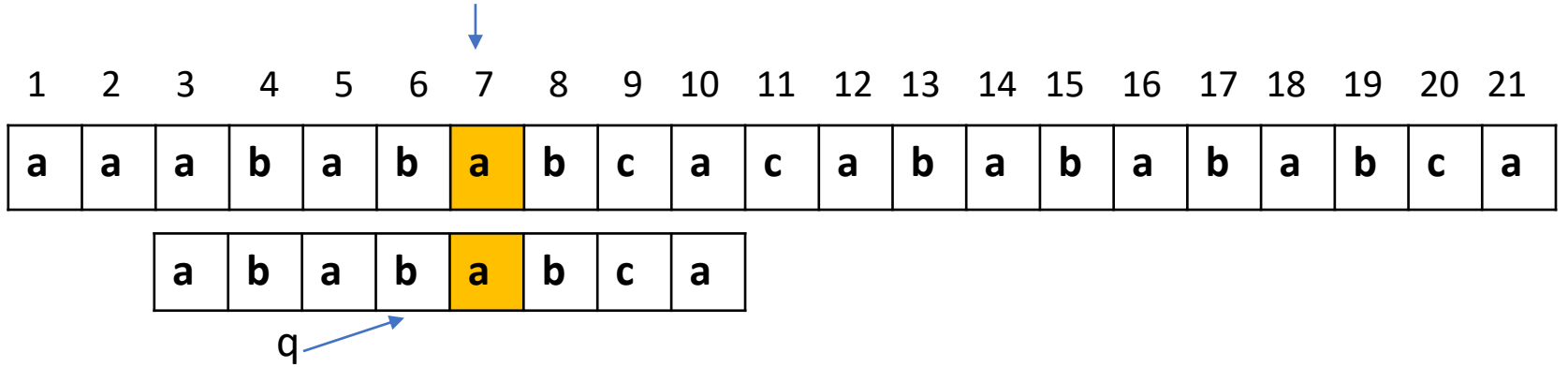
i = 6, T[6] = P[3+1] → q = q + 1 = 4

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



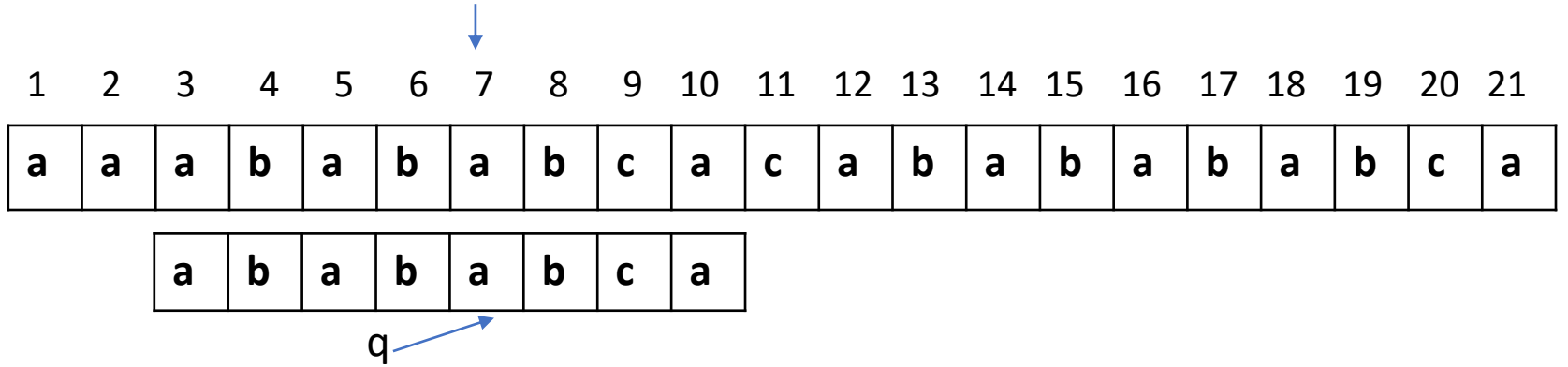
i = 7, T[7] = P[4+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



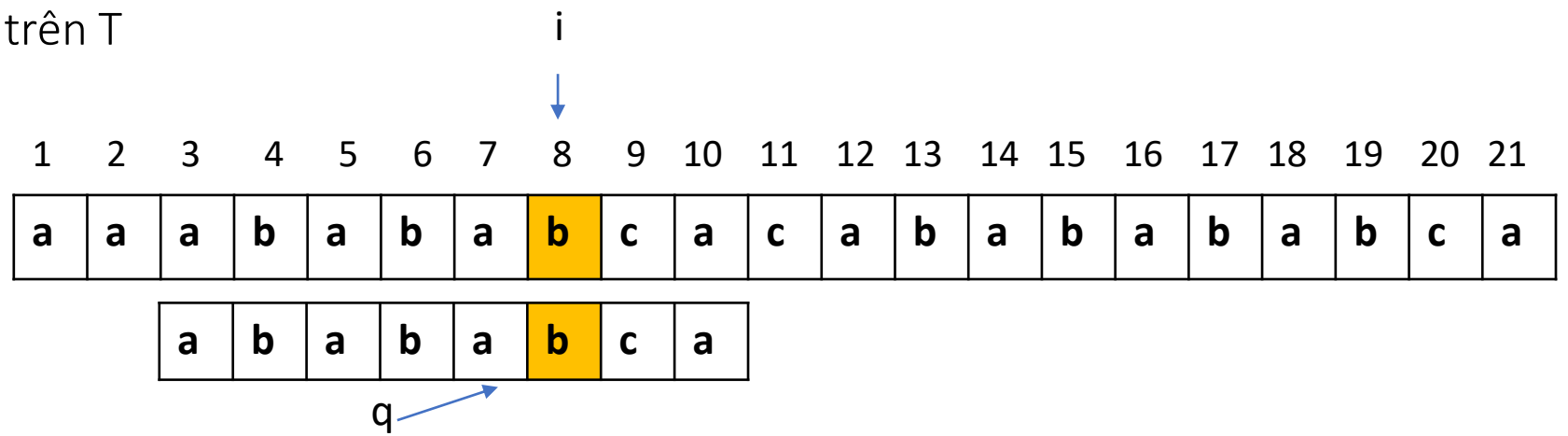
i = 7, T[7] = P[4+1] → q = q + 1 = 5

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



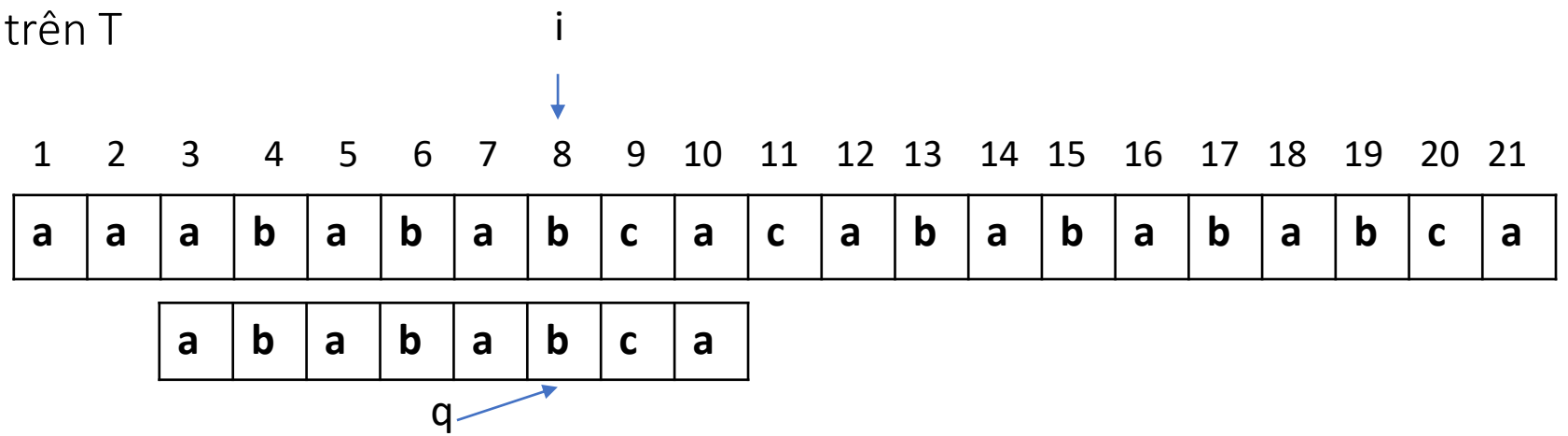
i = 8, T[8] = P[5+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



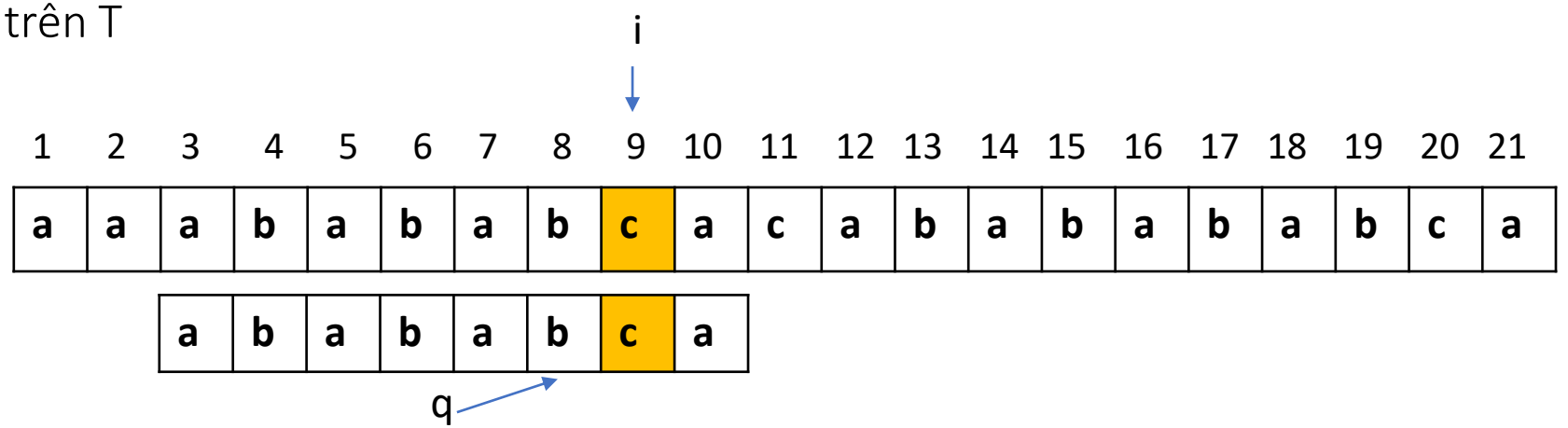
i = 8, T[8] = P[5+1] → q = q + 1 = 6

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



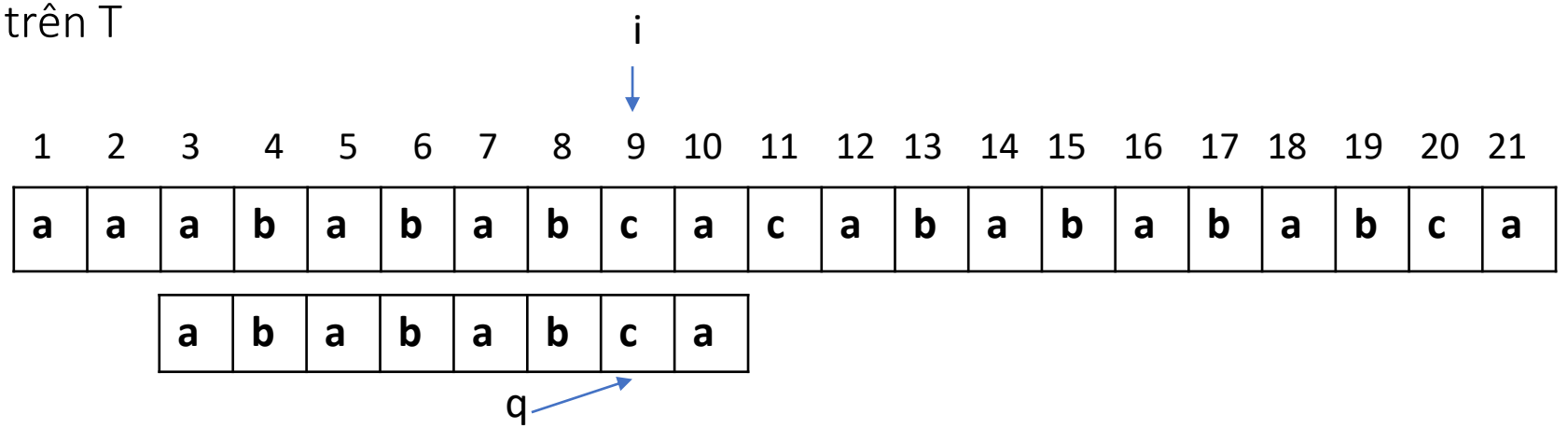
i = 9, T[9] = P[6+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



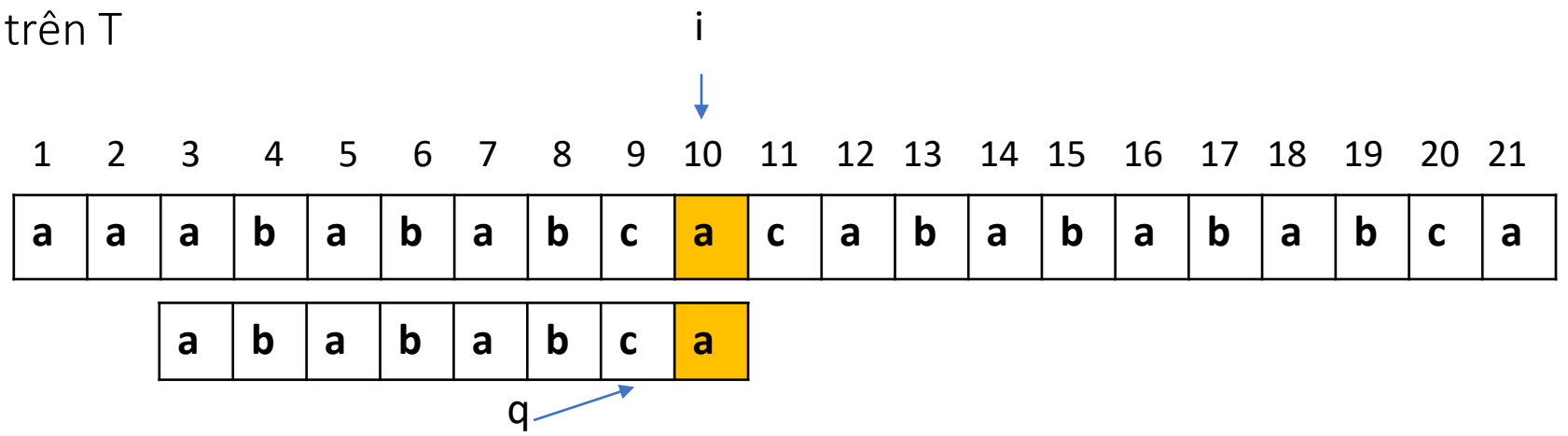
i = 9, T[9] = P[6+1] → q = q + 1 = 7

	1	2	3	4	5	6	7	8
P	**a**	**b**	**a**	**b**	**a**	**b**	**c**	**a**
π	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];
    }
  }
}
```



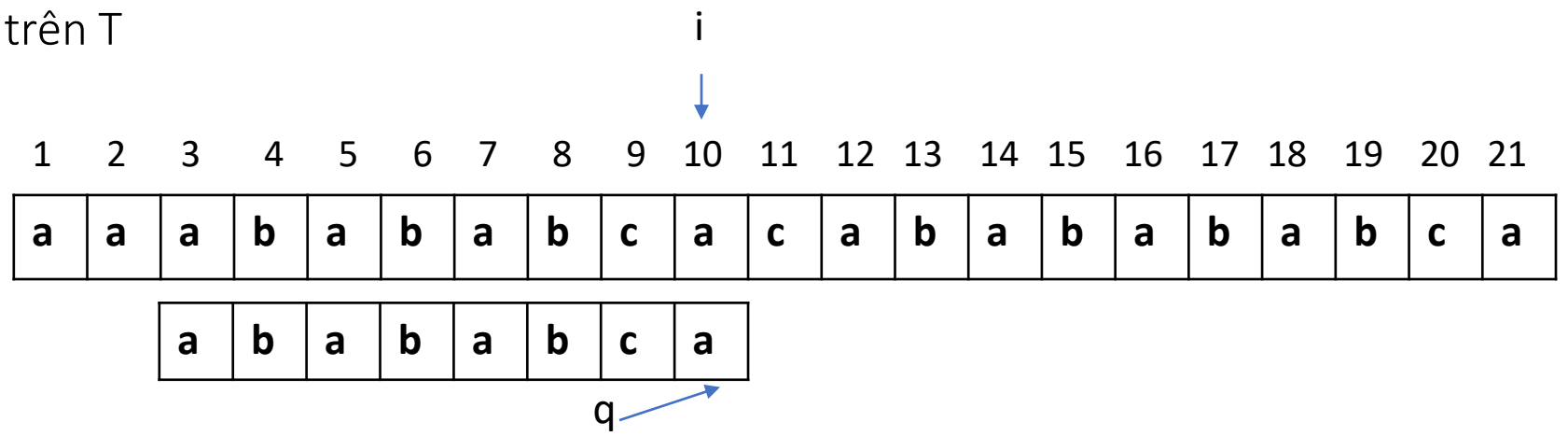
i = 10, T[10] = P[7+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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] → q = q + 1 = 8

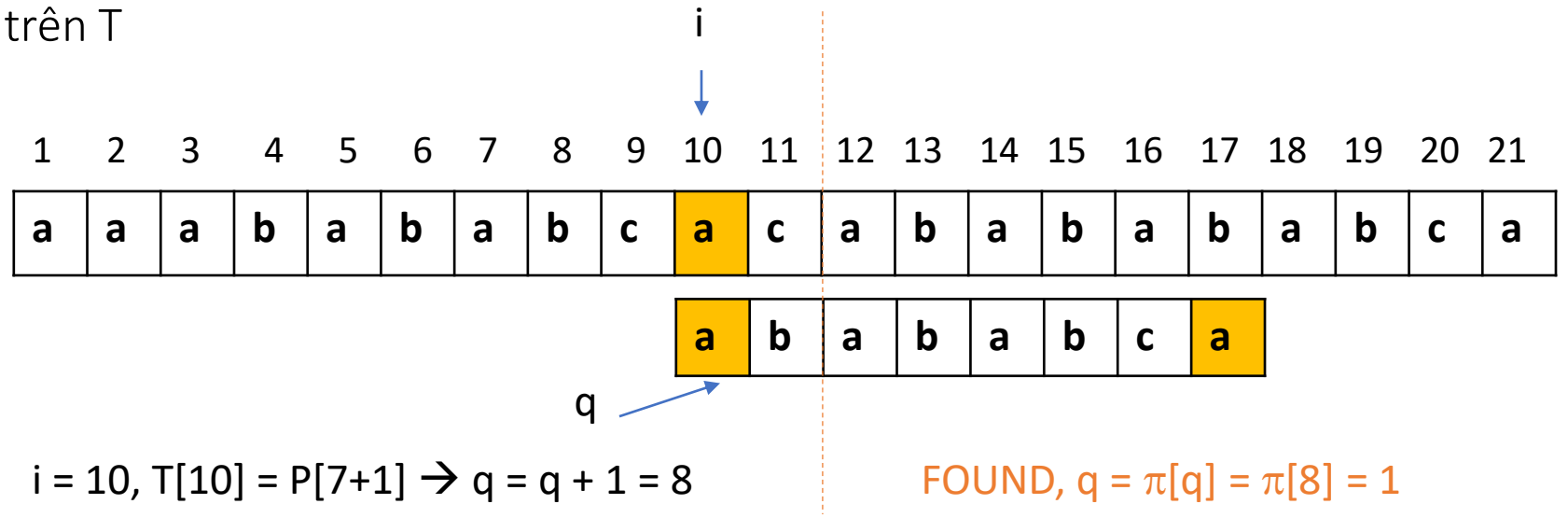
FOUND

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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] → q = q + 1 = 8

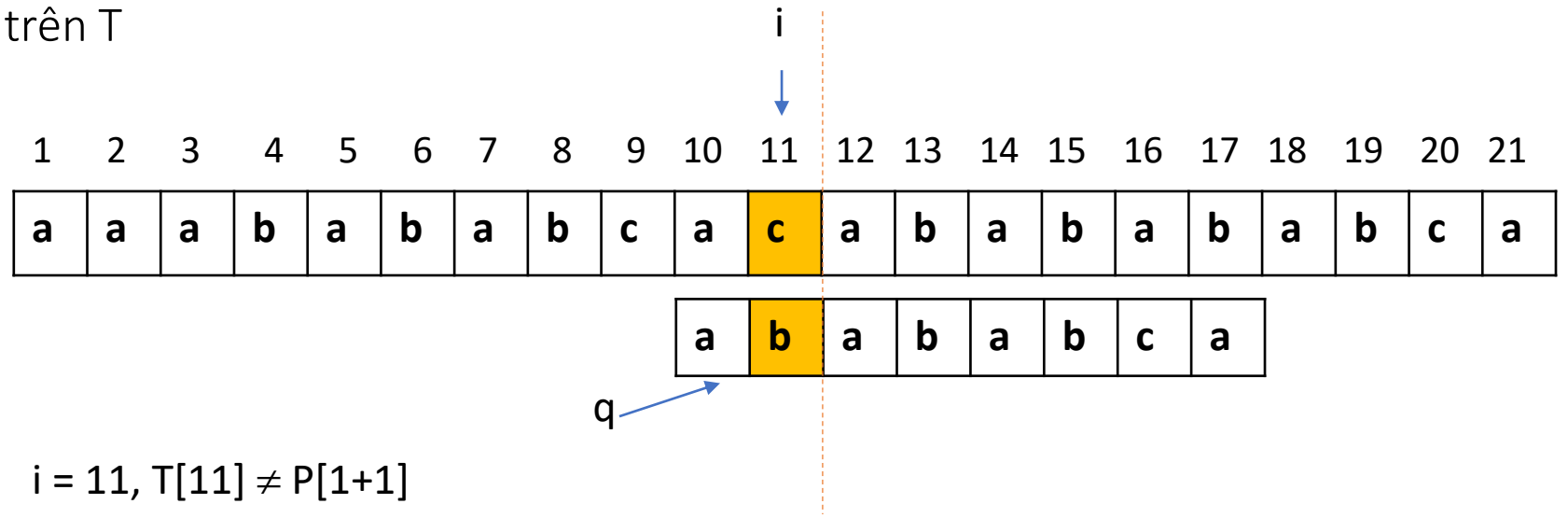
FOUND, q = π[q] = π[8] = 1

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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];
    }
  }
}
```



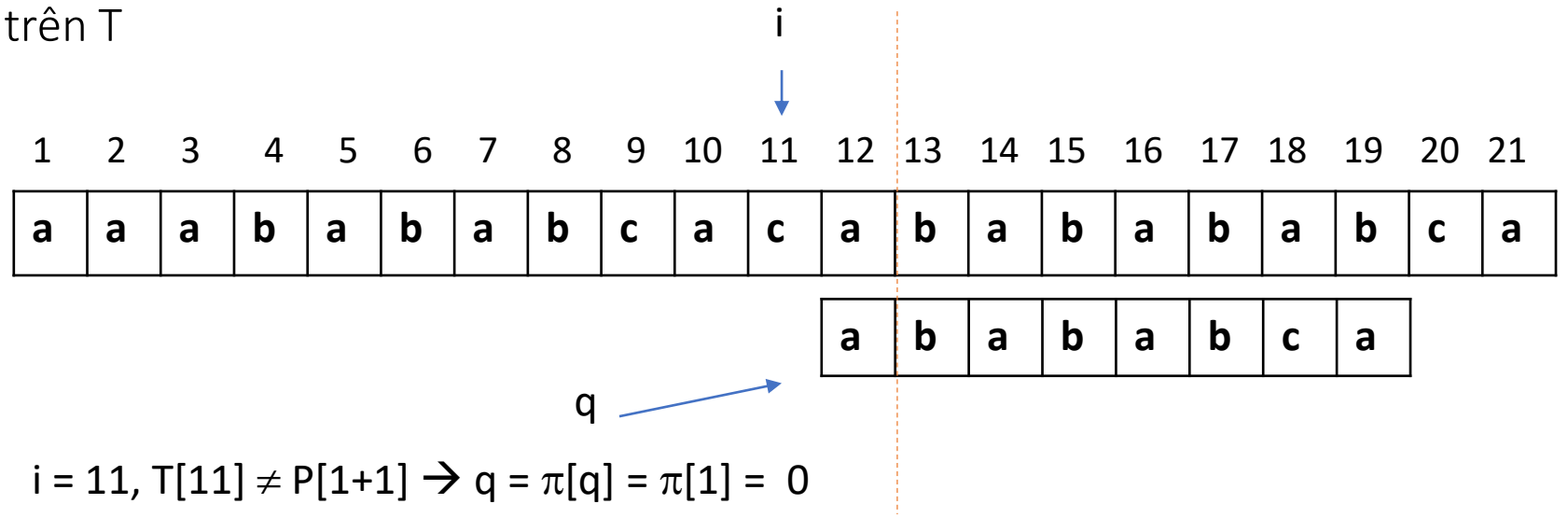
i = 11, T[11] ≠ P[1+1]

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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 = 11, T[11] ≠ P[1+1] → q = π[q] = π[1] = 0

	1	2	3	4	5	6	7	8
P	a	b	a	b	a	b	c	a
π	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;
}
```

THANK YOU !