

# Cấu trúc dữ liệu và Giải thuật

Các giải thuật tìm kiếm

# Các nội dung chính

1. Giới thiệu
2. Các giải thuật tìm kiếm phần tử
3. Các giải thuật tìm kiếm chuỗi con

# 1. Giới thiệu

- Bài học này sẽ trình bày một số giải thuật tìm kiếm cho hai bài toán tìm kiếm cơ bản:
  - Thứ nhất, là bài toán tìm một phần tử trong một dãy phần tử cho trước theo một khoá tìm kiếm
  - Thứ hai, là tìm sự xuất hiện của một chuỗi con trong một chuỗi cho trước

# 1. Giới thiệu

- Với bài toán thứ nhất, có hai chiến lược tìm kiếm là tìm kiếm bằng cách so sánh hay tìm kiếm trực tiếp dựa vào giá trị khoá cần tìm
- Với bài toán thứ hai cũng có nhiều giải thuật khác nhau, từ giải thuật tìm kiếm đơn giản (còn gọi là tìm kiếm thô), cho đến các giải thuật khá phức tạp như của Knuth-Morris-Pratt và của Boyer-Moore

## 2. Các giải thuật tìm kiếm phần tử

- Đặt bài toán:
  - Để đơn giản cho việc trình bày ý tưởng các giải thuật, ta sẽ chọn bài toán ở dạng đơn giản nhất như sau: Cho một dãy  $N$  số  $A = (a_0, a_1, \dots, a_{N-1})$  và giá trị cần tìm  $K$  (khóa tìm kiếm). Yêu cầu tìm vị trí một phần tử có giá trị bằng  $K$ .
- Có 2 chiến lược tìm kiếm:
  - Tìm kiếm bằng cách so sánh:
  - Tìm kiếm dựa trực tiếp vào giá trị khóa:

## 2.1. Tìm kiếm bằng so sánh

- Ý tưởng chung: từ khóa tìm kiếm K, ta chưa biết được vị trí của phần tử cần tìm, nên tiến hành so sánh K với lần lượt các phần tử trong dãy cần tìm cho đến khi ra kết quả (hoặc tìm thấy hoặc không tìm thấy)
- Có 2 loại giải thuật tìm kiếm theo cách này:
  - Tìm kiếm tuần tự (Sequential Search)
  - Tìm kiếm nhị phân (Binary Search)

# Tìm kiếm tuần tự

- Ý tưởng giải thuật:
  - Để tìm phần tử bằng  $K$  trong dãy  $N$  số  $A = (a_0, a_1, \dots, a_{N-1})$ , tiến hành so sánh  $K$  với lần lượt các phần tử trong dãy, cho đến khi:
    - Hoặc tìm thấy phần tử  $a_i = K$ , thì trả về vị trí  $i$  cần tìm
    - Hoặc đã so sánh với toàn bộ các phần tử của dãy nhưng vẫn không thấy, thì trả về kết quả không tìm thấy.

# Tìm kiếm tuần tự

- Cài đặt hàm

```
int SequentialSearch(int A[], int N, int K)
{
    int i=0;
    while (i<N && A[i] != K) i++;
    if (i<N) return i;    //Tìm thấy
    return -1;            //Không tìm thấy
}
```



# Tìm kiếm nhị phân

- Ý tưởng giải thuật:
  - Để tìm phần tử bằng  $K$  trong dãy  $N$  số  $A = (a_0, a_1, \dots, a_{N-1})$ , thì giải thuật này có một yêu cầu là dãy  $A$  đã được sắp xếp, giả sử là theo chiều tăng dần. Các bước của giải thuật đệ quy này như sau:
    - So sánh  $K$  với phần tử  $a_m$  ở giữa dãy ( $m=N/2$ ). Có 3 khả năng xảy ra:
      - Nếu  $K = a_m$  thì trả về vị trí tìm thấy  $m$
      - Nếu  $K < a_m$  thì tìm  $K$  trong dãy  $(a_0, a_1, \dots, a_{m-1})$
      - Trái lại, thì tìm  $K$  trong dãy  $(a_{m+1}, a_{m+2}, \dots, a_{N-1})$
    - Điểm dừng: khi tìm thấy hoặc khi dãy không còn phần tử nào thì trả về kết quả không tìm thấy.

# Tìm kiếm nhị phân

- Cài đặt hàm

```
int BSearch(int K, int A[], int b, int e) {
    if (b>e) return -1; //Không tìm thấy
    int m= (b+e)/2;
    if (K==A[m]) return m; //Tìm thấy
    else
        if (K<A[m])
            return BSearch(K, A, int b, m-1);
        else
            return BSearch(K, A, m+1,e);
}

int BinarySearch(int K, int A[], int N) {
    return BSearch(K,A,0,N-1);
}
```

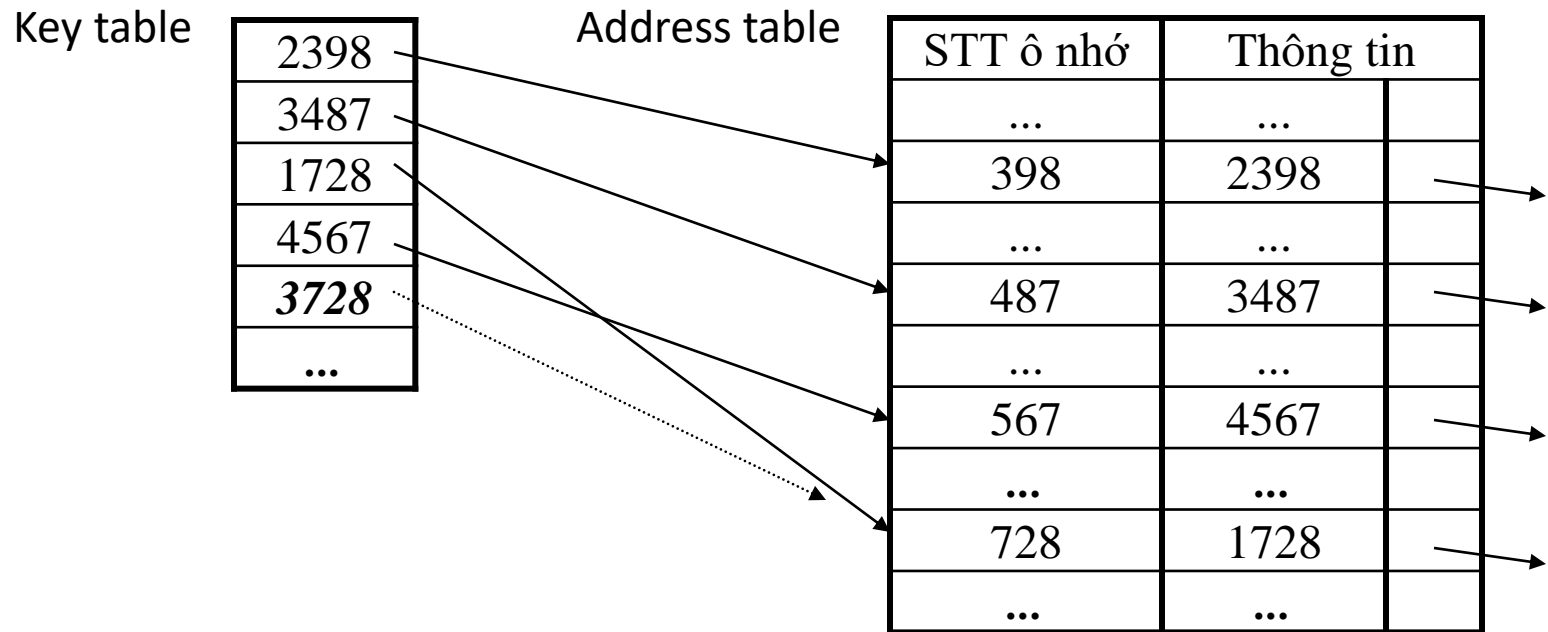
## 2.2. Các giải thuật tìm kiếm trực tiếp

- Đặt vấn đề
  - Nguyên tắc lưu trữ, tìm kiếm:
    - Không thông qua so sánh
    - Thông qua địa chỉ ô nhớ tính trực tiếp trên các giá trị khóa:
      - Địa chỉ thực =  $A_0$  + Địa chỉ tương đối
      - Quy ước:  $A_0 = 0$
  - Tìm kiếm dựa vào giá trị khóa
    - Xây dựng hàm địa chỉ (address function) - hàm băm (hash function)
      - $h(k): X \rightarrow Y$
      - X: các giá trị khóa k (lấy từ bảng khóa - key table)
      - Y: các giá trị địa chỉ tương đối trong bảng địa chỉ (address table)-bảng băm (hash table):  $0..(m-1)$ , m là kích thước hay độ dài của bảng  $\Rightarrow 0 \leq h(k) < m$
    - Lưu trữ: ô nhớ địa chỉ  $A_0+h(k)$  lưu giá trị khóa k và bản ghi tương ứng
    - Tìm kiếm: cho k, máy sẽ tự động tính ra  $h(k)$  và vào địa chỉ  $A_0+h(k)$  lấy ra giá trị khóa k và bản ghi tương ứng

# Các giải thuật tìm kiếm trực tiếp

- Đặt vấn đề

- Ví dụ minh họa:  $m=1000$ ,  $h(k) = k \bmod m$



- Nhận xét

- Kích thước của bảng địa chỉ có giới hạn  $\Rightarrow$  hiện tượng đụng độ địa chỉ
  - Yêu cầu: xây dựng hàm địa chỉ cho các giá trị "*rải*" đều trên bảng và cần đưa ra được các biện pháp khắc phục đụng độ

# Các giải thuật tìm kiếm trực tiếp

- Đặt vấn đề
  - Các vấn đề cần giải quyết
    - Xây dựng hàm địa chỉ ("*hàm rải*") cho tốt thông qua các phương pháp toán học
      - Phương pháp chia
      - Phương pháp nhân
      - Phương pháp phân đoạn
    - Chuẩn bị các biện pháp khắc phục đụng độ (băm lại - rehashing)
      - Biện pháp địa chỉ mở
      - Biện pháp móc nối

# Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp chia
  - Phương pháp đơn giản và dễ sử dụng
    - $h(k) = k \bmod m$
  - Nhận xét:
    - VD:
      - $m = 2 \Rightarrow h(k)$  có 2 giá trị
      - $m = 1000 \Rightarrow h(k)$  chỉ phụ thuộc vào 3 chữ số cuối của  $k$
    - Số giá trị của  $h(k)$  phụ thuộc vào  $m$
    - Nếu  $m$  nguyên tố  $\Rightarrow$  rải tốt nhất
  - Yêu cầu: cần chọn  $m$  sao cho  $h(k)$  rải đều
  - Cải tiến:
    - $h(k) = k \bmod m^*$ , với  $m^*$  là số nguyên tố lớn nhất  $< m$

# Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp nhân
  - Nguyên tắc
    - Bước 1: lấy giá trị  $k^2$
    - Bước 2: xác định  $h(k)$  thông qua các chữ số liên tục ở giữa số  $k^2$
  - Ví dụ:
    - $m < 1000$
    - lấy  $k^2$  và chọn 3 chữ số ở giữa, không lấy 2 chữ số đầu, 2 chữ số cuối

k	$k^2$	$h(k)$
2398	5750404	504
3487	12159169	159 or 591
1728	2985984	859
4567	20857489	857 or 574
3728	13897984	897 or 979

# Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp phân đoạn (partitioning)
  - Nguyên tắc:
    - Áp dụng khi khóa có kích thước lớn
    - Chia khóa thành các đoạn có độ dài như nhau = độ dài địa chỉ
    - Phối hợp các đoạn
      - Ví dụ: cộng lại, chọn một vài vị trí và ghép lại
  - Các kỹ thuật phân đoạn
    - Tách (splitting)
      - Tách từ phải qua trái
      - Xếp thành hàng các đoạn
    - Gấp (folding)
      - Gấp theo các đường biên (giống gấp giấy)
      - Xếp thành hàng các đoạn



# Các giải thuật tìm kiếm trực tiếp

- Xây dựng hàm địa chỉ - Phương pháp phân đoạn (partitioning)

– Ví dụ:

- $k = 34289421$
- Độ dài địa chỉ: 3 ( $m < 1000$ )
- Tách: 421, 289, 034
- Gấp: gấp theo biên 9, 4 và biên 2, 8  $\Rightarrow$  124, 289, 430

	<b>Tách</b>
	421
	289
	034
	<hr/>
<b>h(k)</b>	744

	<b>Gấp</b>
	124
	289
	430
	<hr/>
<b>h(k)</b>	843

# Các giải thuật tìm kiếm trực tiếp

- Các biện pháp khắc phục đụng độ
  - Vấn đề:
    - Tập giá trị khóa  $k$  thường rất lớn trong khi bảng địa chỉ thường ngắn  $\Rightarrow$  các khóa có giá trị khác nhau lại có giá trị địa chỉ giống nhau  $\Rightarrow$  đụng độ
    - Đụng độ là khi ô nhớ có địa chỉ tính ra đã bị chiếm  $\Rightarrow$  cần tìm một ô nhớ khác để lưu các giá trị khóa đó
    - Vấn đề đụng độ có thể xảy ra liên tục, đây chuyện nếu không có chiến lược khắc phục
  - Các chiến lược khắc phục đụng độ (rehashing)
    - Biện pháp địa chỉ mở (open addressing)
      - Phương pháp thử tuyến tính (băm lại tuyến tính)
      - Phương pháp thử bình phương (băm lại bình phương)
    - Biện pháp móc nối (chaining)
    - Quy ước: bảng khóa gồm  $n$  khóa khác nhau, bảng địa chỉ kích thước  $m$

# Các biện pháp khắc phục đụng độ

- Biện pháp địa chỉ mở (open addressing)
  - Nguyên tắc lưu trữ & tìm kiếm nếu có đụng độ
    - Thử các địa chỉ theo một các nhất định để tìm vị trí mới
    - Nếu đi hết bảng địa chỉ mà không tìm ra thì quay lại từ đầu bảng
    - Nếu không tìm thấy chỗ trống => overflow (tràn bảng)
  - Ví dụ minh họa
    - Khóa:
      - 123, 321, 234, 432, 345, 543, 678...
    - $m = 7, m^* = 7$
    - Hàm địa chỉ:  $h(k) = k \bmod m^* = k \bmod 7$
    - Đụng độ:
      - $h(543) = h(123) = 4$
      - $h(678) = h(321) = 6$

k	h(k)
123	4
321	6
234	3
432	5
345	2
543	4
678	6

Địa chỉ	k
0	
1	
2	345
3	234
4	123
5	432
6	321

# Các biện pháp khắc phục độ

- Biện pháp địa chỉ mở - Phương pháp thử tuyến tính

(băm lại tuyến tính)

– Bài toán: tìm vị trí mới cho khóa k

– Ý tưởng

- Bắt đầu từ địa chỉ đựng độ:  $i = h(k)$
- Thử theo chiều tăng của bảng địa chỉ: g/s với chỉ số j bắt đầu từ  $i+1$
- Nếu đi hết bảng mà vẫn chưa tìm ra  $\Rightarrow$  quay lại đi từ đầu bảng
- Dừng lại tại ô nhớ trống đầu tiên hoặc đã đi hết bảng:  $|j - i| \bmod m^* = 0$
- Không tìm thấy  $\Rightarrow$  overflow

– Thủ tục

- Học sinh tự viết

k	h(k)	Địa chỉ	k
123	4	0	
321	6	1	
234	3	2	345
432	5	3	234
345	2	4	123
<b>543</b>	4	5	432
<b>678</b>	6	6	321

# Các biện pháp khắc phục độ

- Biện pháp địa chỉ mở - Phương pháp thử tuyến tính (băm lại tuyến tính)

- Nhận xét đánh giá

- Hiện tượng hội tụ (clustering) xung quanh các khóa không xảy ra độ do phép thử tuyến tính. Ví dụ độ:

- $h(543) = 4 \Rightarrow$  sang vị trí 0

- $h(678) = 6 \Rightarrow$  sang vị trí 1

- Đến gần khu vực "hội tụ"  $\Rightarrow$  tốc độ xử lý chậm

- Hướng giải quyết

- Tránh tình trạng tìm địa chỉ mở theo công thức:

- Thử lần  $i$ :  $h_i = (H + i) \bmod m$  ( $i = 0..m-1$ ),  $h_0 = H = h(k)$

- VD:  $h_0(543) = 4$ ,  $h_1(543) = 5$ ,  $h_2(543) = 6...$

- Công thức mới:  $h_i = (H + G(i)) \bmod m$

- Yêu cầu:  $G(i)$  có tính thử ngẫu nhiên  $\Rightarrow$

- khắc phục hiện tượng hội tụ. Khó tìm được  $G(i)$  tốt.

Địa chỉ	k
0	<b>543</b>
1	<b>678</b>
2	345
3	234
4	123
5	432
6	321

# Các biện pháp khắc phục độ

- Biện pháp địa chỉ mở - Phương pháp thử bình phương
  - Bài toán:
    - Tìm vị trí mới cho khóa 543, 678
    - Đưa ra công thức:  $h_i = (H + G(i)) \bmod m^*$ ,  $G(i)$  có tính ngẫu nhiên cao
  - Ý tưởng giải thuật: phương pháp thử bình phương
    - $G(i) = i^2$
  - Nhận xét
    - Tránh được hiện tượng hội tụ
    - $h_i(x) = h_j(x) \Leftrightarrow (i-j)(i+j) \bmod m^* = 0 \Rightarrow$  phương pháp chỉ thử với  $[m^*/2]$  giá trị băm lại khác nhau hay phương pháp chỉ tìm với  $[m^*/2]$  vị trí
    - Nhược điểm: không tìm thấy vị trí trong khi còn nhiều ô nhớ trống
    - Cần đưa ra  $G(i)$  càng ngẫu nhiên càng tốt nhưng phải đảm bảo xét hết các vị trí trong bảng.
      - VD: dùng hoán vị ngẫu nhiên của dãy  $(1, 2, 3 \dots m^* - 1)$

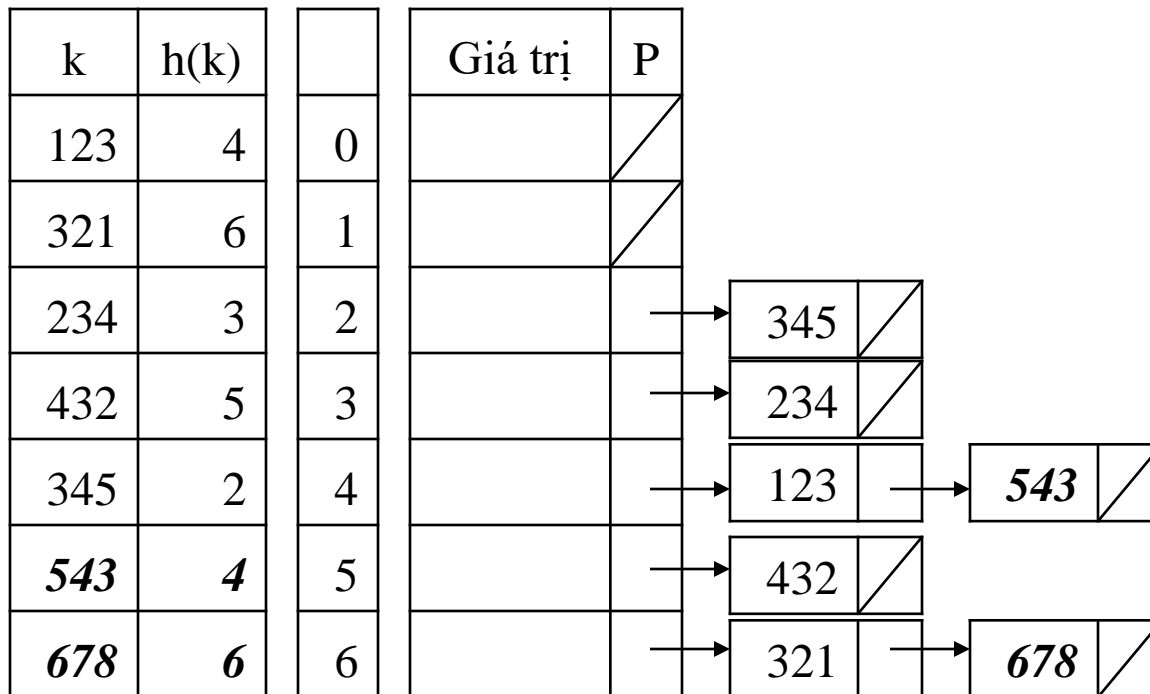
# Các biện pháp khắc phục đụng độ

- Biện pháp móc nối dây chuyền (chaining)
  - Nguyên tắc lưu trữ & tìm kiếm nếu có đụng độ
    - Dùng CTLT móc nối để lưu trữ và tìm kiếm DS giá trị địa chỉ đụng độ
  - Các kỹ thuật móc nối
    - Móc nối ngoài (external link): dùng thêm miền nhớ phụ ngoài bảng
    - Móc nối trong (internal link): dùng chính các ô nhớ trong bảng địa chỉ
  - Ví dụ minh họa
    - $m = 7, m^* = 7$
    - Hàm địa chỉ:
      - $h(k) = k \bmod m^* = k \bmod 7$

k	h(k)		Giá trị	P
123	4	0		
321	6	1		
234	3	2		→
432	5	3		→
345	2	4		→
<b>543</b>	<b>4</b>	5		→
<b>678</b>	<b>6</b>	6		→

# Các biện pháp khắc phục độ

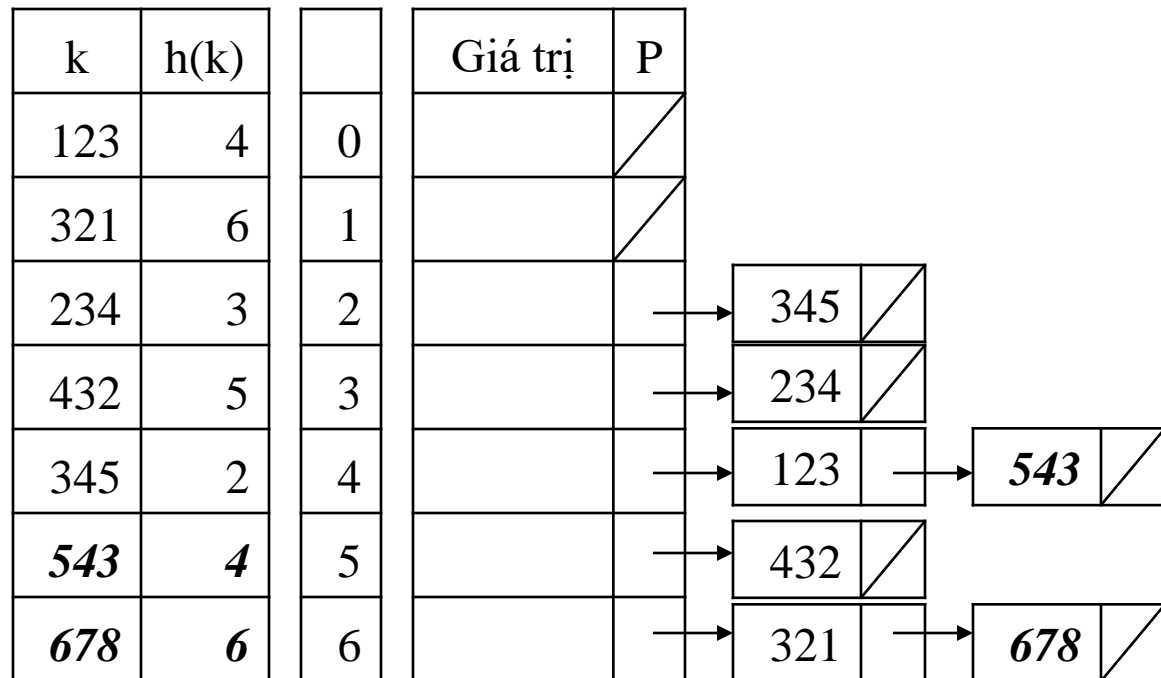
- Biện pháp móc nối dây chuyền (chaining)
  - Kỹ thuật móc nối ngoài (External Link)
    - Dùng CTLT móc nối để lưu trữ và tìm kiếm DS giá trị địa chỉ đựng độ
    - Dùng thêm miền nhớ ở ngoài bảng địa chỉ










# Các biện pháp khắc phục độ

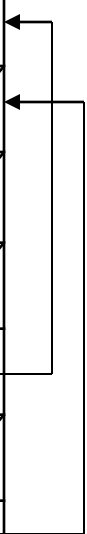
- Biện pháp móc nối dây chuyền (chaining)
  - Kỹ thuật móc nối ngoài (External Link)
    - Ý tưởng giải thuật
      - Dùng các nút ngoài để lưu các giá trị khóa
      - Nếu khóa đã có trong bảng => tìm thấy
      - Nếu không có => tạo ra nút mới lưu giá trị khóa
      - Nếu độ đầy => tạo ra nút mới lưu khóa đó rồi nối vào DS móc nối
  - Nhận xét
    - Tốn bộ nhớ



# Các biện pháp khắc phục độ

- Biện pháp móc nối dây chuyền (chaining)
  - Kỹ thuật móc nối trong (Internal Link)
    - Dùng CTLT móc nối để lưu trữ và tìm kiếm DS giá trị địa chỉ đựng độ
    - Chỉ dùng miền nhớ ở trong bảng địa chỉ để lưu các giá trị đựng độ
    - Chỉ dùng thêm vùng nhớ mới khi xảy ra hiện tượng tràn

k	h(k)		Giá trị	P
123	4	0	<b>543</b>	
321	6	1	<b>678</b>	
234	3	2	345	
432	5	3	234	
345	2	4	123	
<b>543</b>	<b>4</b>	5	432	
<b>678</b>	<b>6</b>	6	321	



# Các biện pháp khắc phục đụng độ

- Biện pháp móc nối dây chuyền (chaining)

- Kỹ thuật móc nối trong (Internal Link)

- Ý tưởng giải thuật

- Tính địa chỉ của khóa đưa vào  $h(k)$

- Nếu  $k_0$  tại đó (đụng độ):  $h(k) = h(k_0)$

- » Nếu tìm trong DS các khóa "đụng độ" đã chứa  $k \Rightarrow$  STOP

- » Tìm ô trống tiếp theo cho  $k \Rightarrow$  thêm vào DS móc nối

- Nếu  $k_0$  tại đó:  $h(k) \neq h(k_0)$ :

(  $k_0$  thuộc vào DS các khóa  
đụng độ được thêm vào trước đó,



ô nhớ đó thực sự không phải của  $k_0$ )

- » Tìm ô nhớ trống mới cho  $k_0$

- » Đưa  $k$  vào địa chỉ đó

- Nhận xét

- Trong trường hợp thứ 2, việc sao chép  $k_0$  không đơn thuần chỉ thực hiện với nội dung của  $k_0$  mà còn phải sửa lại địa chỉ của con trỏ trỏ đến nó  $\Rightarrow$  dùng **CTLT móc nối kép hoặc vòng**

k	h(k)		Giá trị	P
123	4	0	<b>543</b>	
321	6	1	<b>678</b>	
234	3	2	345	
432	5	3	234	
345	2	4	123	
<b>543</b>	<b>4</b>	5	432	
<b>678</b>	<b>6</b>	6	321	

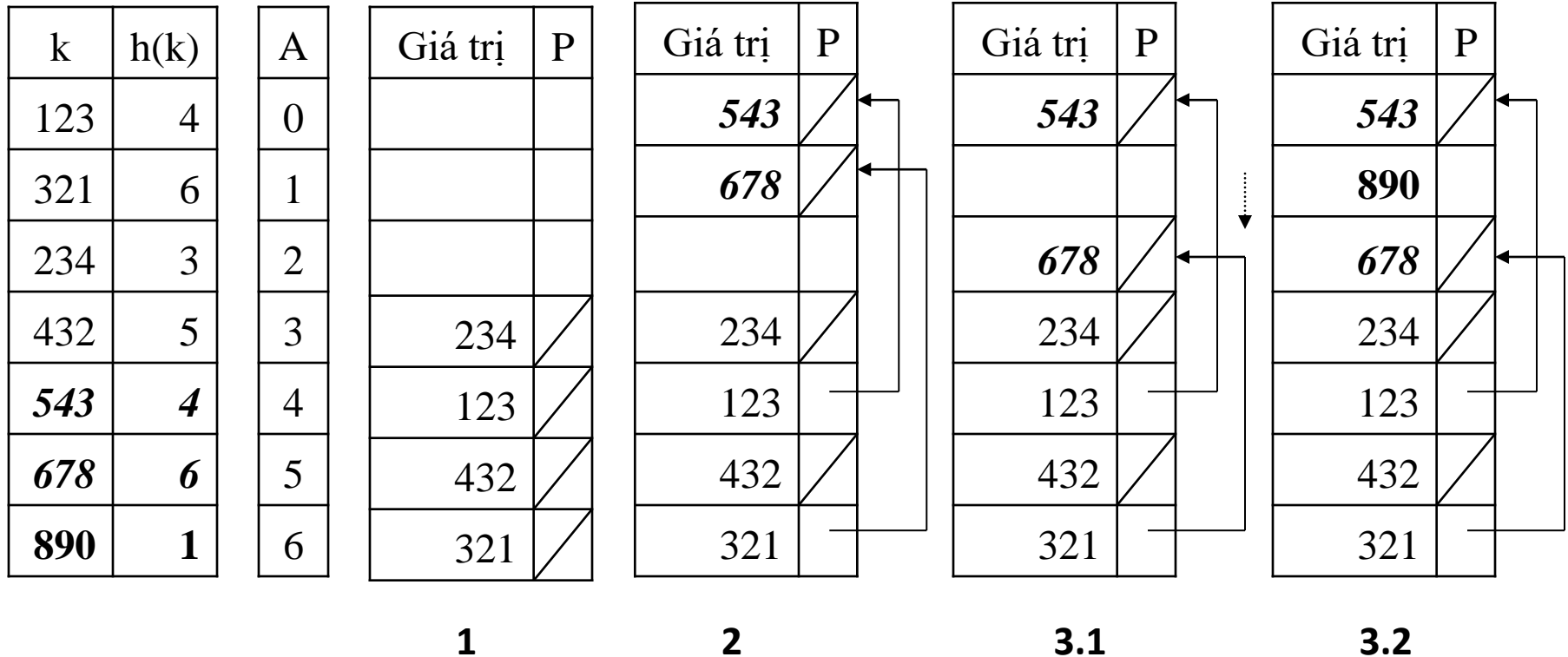
# Các biện pháp khắc phục độ

- Biện pháp móc nối dây chuyền (chaining)
  - Kỹ thuật móc nối trong (Internal Link)

- Minh họa:

- $m = 7, m^* = 7$

- $h(k) = k \bmod m^* = k \bmod 7$



# Các biện pháp khắc phục độ

- Phân tích & Đánh giá số bước thực hiện

- Nguyên tắc

- Giả sử các khoá xuất hiện đồng đều

- Cách ước lượng: lý thuyết xác suất

- $p_i$  : xác suất lần thử thứ  $i$  mà tìm thấy vị trí mới

- $E_{k+1}$  : số lần thử trung bình để tìm thấy vị trí mới cho khóa  $(k+1)$   
(giả sử trong bảng đã có  $k$  khóa)

$$E_{k+1} = \sum_{i=1}^{k+1} i p_i$$

- $m$ : kích thước bảng địa chỉ

- $n$ : số khóa hiện có trong bảng (  $n \leq m$  )

- $E$ : số lần thử trung bình để truy cập một khóa ngẫu nhiên trong bảng  
(tìm thấy vị trí trống mới cho nó hoặc tìm thấy khóa đó đã lưu trong bảng)

$$E = \sum_{k=1}^n \frac{1}{n} E_k$$

- $\alpha$  : hệ số tải (load factor)

$$\alpha = \frac{n}{m+1}$$

# Các biện pháp khắc phục độ

- Phân tích & Đánh giá số bước thực hiện

- Minh họa qua một số biện pháp

- $\alpha$  : hệ số tải (load factor)

$$\alpha = \frac{n}{m+1}$$

- Biện pháp địa chỉ mở

- a) Phép thử ngẫu nhiên:

$$E = -\frac{1}{\alpha} \ln(1 - \alpha)$$

- b) Phép thử tuyến tính:

$$E = \frac{1 - \alpha / 2}{1 - \alpha}$$

- Biện pháp móc nối

- c) Kỹ thuật móc nối ngoài (External Link):  $E = 1 + \alpha/2$

a)

$\alpha$	E
0,10	1,05
0,25	1,15
0,50	1,39
0,75	1,85
0,90	2,56
0,95	3,15

b)

$\alpha$	E
0,10	1,06
0,25	1,17
0,50	1,50
0,75	2,50
0,90	5,50
0,95	10,50

c)

$\alpha$	E
0,10	1,05
0,25	1,12
0,50	1,25
0,75	1,37
0,90	1,45
0,95	1,47

# 3. Tìm kiếm chuỗi con

- Giới thiệu bài toán
- Giải thuật tìm kiếm thô (brute-force)
- Giải thuật Knuth-Morris-Pratt (KMP)

# Giới thiệu bài toán

- Cho trước một văn bản  $V$  gồm  $n$  kí tự ( $V = v_0, v_1, \dots, v_{n-1}$ ) và một chuỗi con  $P$  (gọi là mẫu) gồm  $m$  kí tự ( $P = p_0, p_1, \dots, p_{m-1}$ ). Yêu cầu tìm vị trí xuất hiện đầu tiên của  $P$  trong  $V$ .
- Bài toán này có nhiều giải thuật. Giải thuật thô khá đơn giản nhưng có thời gian xử lý tồi nhất tỉ lệ với  $m \times n$ . Giải thuật KMP cần các thao tác tiền xử lý trên chuỗi mẫu nên khá phức tạp, nhưng có thời gian tốt hơn nhiều, chỉ tỉ lệ với  $m + n$ .



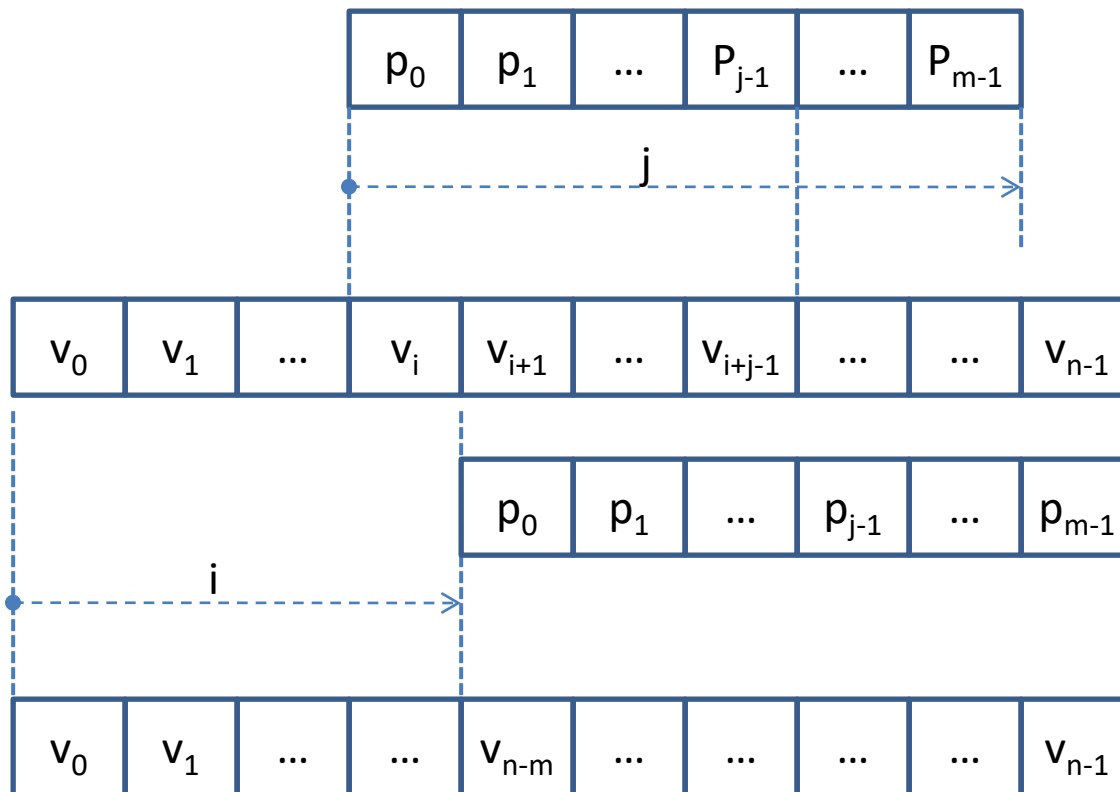
# Giải thuật tìm kiếm thô

- Ý tưởng giải thuật:
  - So sánh lần lượt các ký tự của mẫu  $P$  với các ký tự của văn bản  $V$  bắt đầu từ vị trí  $i$  ( $0 \leq i \leq n-m$ ) cho đến khi hoặc khớp tất cả các ký tự của  $P$  với các ký tự trong  $V$  thì  $i$  là vị trí cần tìm, hoặc so đến ký tự cuối cùng trong  $V$  vẫn không khớp thì kết luận tìm kiếm không thấy, hoặc gặp bất kỳ ký tự nào không khớp thì quay lại so sánh từ đầu của mẫu  $P$  với các ký tự của  $V$  bắt đầu từ vị trí  $i+1$ .

# Giải thuật tìm kiếm thô

$V = v_0, v_1, \dots, v_{n-1}$

$P = p_0, p_1, \dots, p_{m-1}$



```
i = j = 0;
do {
    while (j < m && p_j == v_{i+j}) {
        j++;
    }
    if (j < m && i < n - m) {
        i++;
        j = 0;
    }
} while (i <= n - m && j < m);

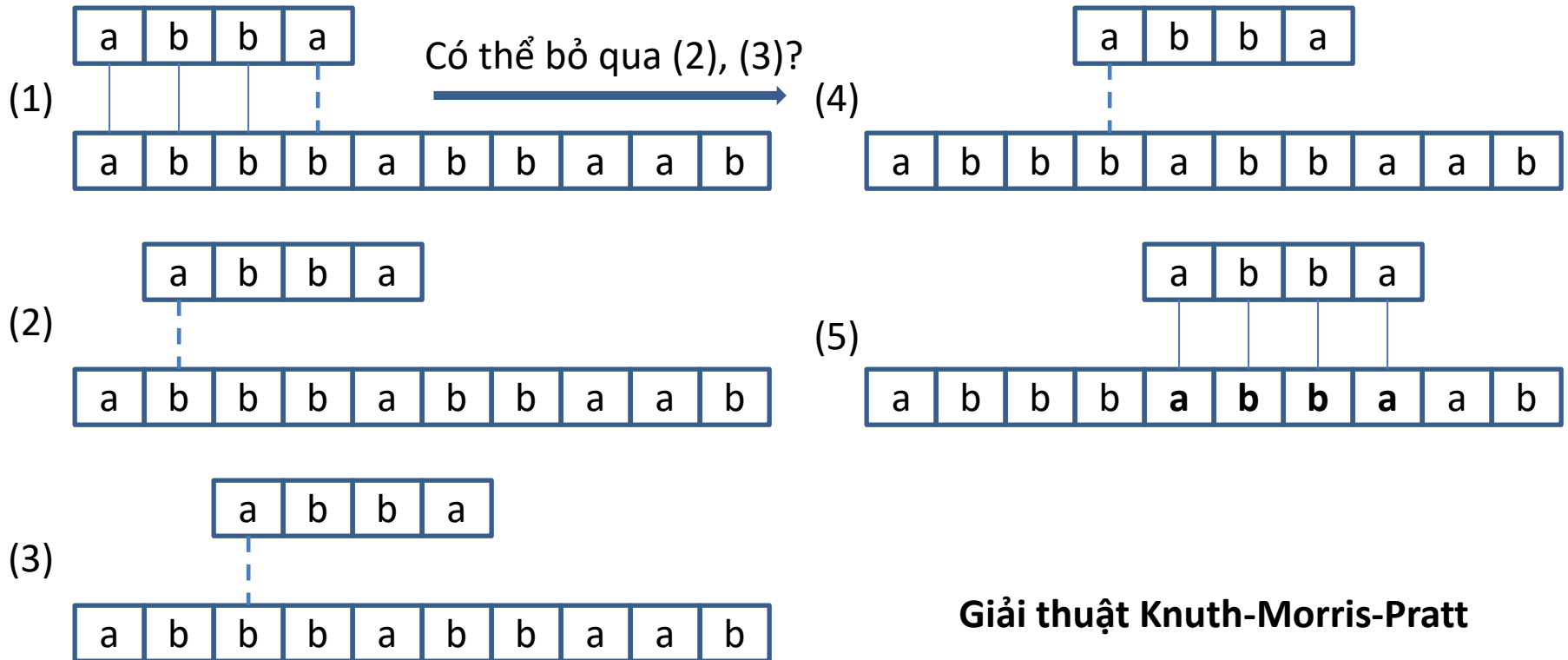
if (j == m) return i; //FOUND
else return -1; //NOT FOUND
```

# Giải thuật tìm kiếm thô – cài đặt

```
int BFSearch(char V[N], char P[M] ) {  
  /*Ham tra ve vi tri tim thay dau tien, tra  
  ve -1 neu khong tim thay*/  
  if (N<M) return -1;  
  int i, j;  
  i=j=0;  
  do {  
    while (j<M && V[i+j]==P[j]) {  
      j++;  
    }  
  }
```

```
    if (i<=N-M && j<M ) {  
      i++;  
      j=0;  
    }  
  } while (i<=N-M && j<M) ;  
  if (j==M) return i;  
  else return -1;  
} //end BFSearch
```

# Cải tiến giải thuật tìm kiếm thô

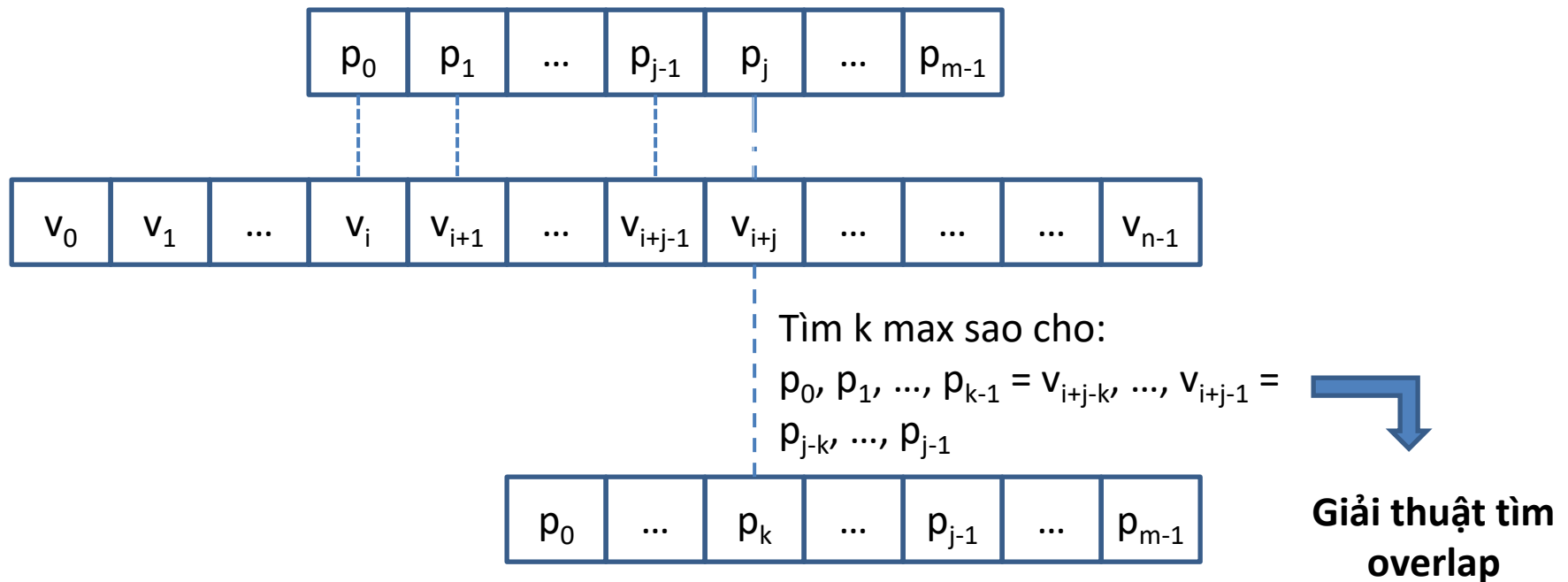


**Giải thuật Knuth-Morris-Pratt**

# Giải thuật Knuth-Morris-Pratt

- Ý tưởng của giải thuật:
  - Trong giải thuật này, khi ta đã so sánh bắt đầu từ vị trí kí tự thứ  $i$  trong văn bản và đến kí tự thứ  $j$  trong mẫu mà có sự không khớp với văn bản ( $j-1$  kí tự đầu tiên đã khớp), thì thay vì phải quay lại so sánh từ kí tự đầu tiên của mẫu với kí tự thứ  $i+1$  như trong giải thuật thô ở trên, ta thấy có thể tận dụng thông tin trong  $j-1$  kí tự đã khớp để bắt đầu việc so sánh từ một kí tự thứ  $k$  xác định trong mẫu ( $0 \leq k \leq M-1$ ) với kí tự hiện đang không khớp trong văn bản (không phải dịch lại vị trí  $i+1$ ).
  - Vị trí  $k$  cần tìm thoả mãn điều kiện:  $k$  là giá trị lớn nhất  $< j$  sao cho  $k-1$  kí tự đầu tiên trong mẫu trùng/khớp với  $k-1$  kí tự cuối cùng của  $j-1$  kí tự đầu tiên trong mẫu.

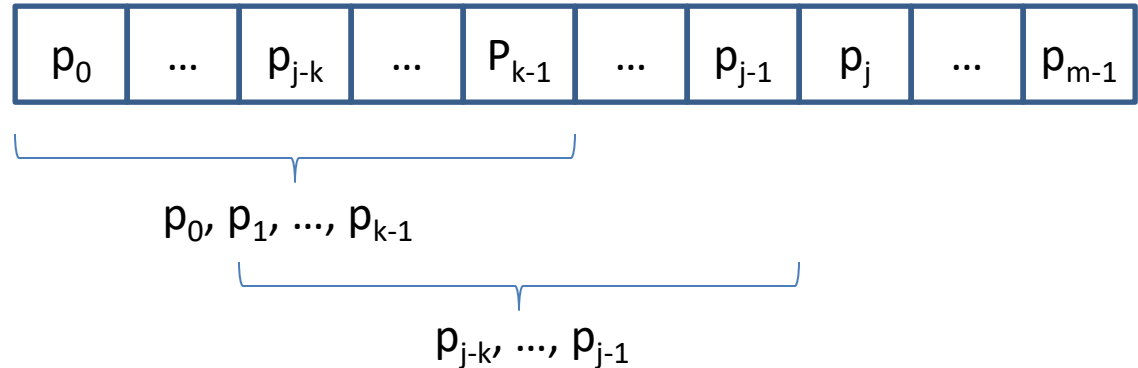
# Giải thuật Knuth-Morris-Pratt



# Giải thuật tìm overlap

Cho trước  $p[m]$ . Với  $j > 0$ , tìm  $k$   
 $\max (0 \leq k \leq j-1)$   
sao cho:

$p_0, p_1, \dots, p_{k-1} = p_{j-k}, \dots, p_{j-1}$



```
int Overlap(p[m], j) {  
    if (j < 2) return 0;  
    i = 1; // Tìm i = j-k  
    //Tìm vị trí đầu tiên j-k sao cho  $p_0 = p_{j-k}$   
    do {  
        while (i <= j-1 &&  $p_i \neq p_0$ ) i++;  
        if (i == j) return 0; //Không tìm thấy  
        else { //Tìm thấy, kiểm tra xem đây có phải là k cần tìm  
            k = i+1;  
            while (k <= j-1 &&  $p_k \neq p_{k-i}$ ) k++;  
            if (k == j) return j-i; //OK trả về giá trị cần tìm  
            else i++; //tiếp tục tìm vị trí j-k  
        }  
    } while (i <= j-1); // vẫn còn khả năng tìm i  
}
```

# Giải thuật tìm overlap

- Ví dụ: cho mẫu 10110011 ta sẽ có các giá trị của k như sau:

j	1	2	3	4	5	6	7
Phần còn lại	1	10	101	1011	10110	101100	1011001
k	0	0	1	1	2	0	1
Chuỗi khớp	Rỗng	Rỗng	1	1	10	Rỗng	1



# Giải thuật Knuth-Morris-Pratt

```
int KMPSearch(char V[N], char P[M] ) {  
    /*Ham tra ve vi tri tim thay dau tien, tra  
    ve -1 neu khong tim thay*/  
    if (N<M) return -1;  
    int i, j;  
    i=j=0;  
    do {  
        while (j<M && V[i+j]==P[j]) {  
            j++;  
        }  
    }
```

```
        if (i<=N-M && j<M ) {  
            j=Overlap(P, j);  
            if (j==0) i++;  
        }  
    } while (i<=N-M && j<M) ;  
    if (j==M) return i;  
    else return -1;  
} //end KMPSearch
```

# Bài tập

- Bài 1: Cho dãy  $N$  số nguyên ( $N \leq 1000$ ) không âm. Giả sử người ta muốn cài đặt phương pháp tìm kiếm trực tiếp cho dãy số trên. Yêu cầu:
  - Định nghĩa các cấu trúc dữ liệu cần thiết.
  - Viết thủ tục tìm kiếm theo phương pháp địa chỉ mở với phép thử tuyến tính
  - Viết thủ tục tìm kiếm theo phương pháp địa chỉ mở với phép thử bình phương
  - Viết thủ tục tìm kiếm theo phương pháp móc nối ngoài
  - Viết thủ tục tìm kiếm theo phương pháp móc nối trong