

第四章 串

- 1、串类型的定义**
- 2、串表示和实现**
- 3、串的模式匹配算法**
- 4、串操作应用举例**

3、串的模式匹配算法

1、串的定长顺序存储表示

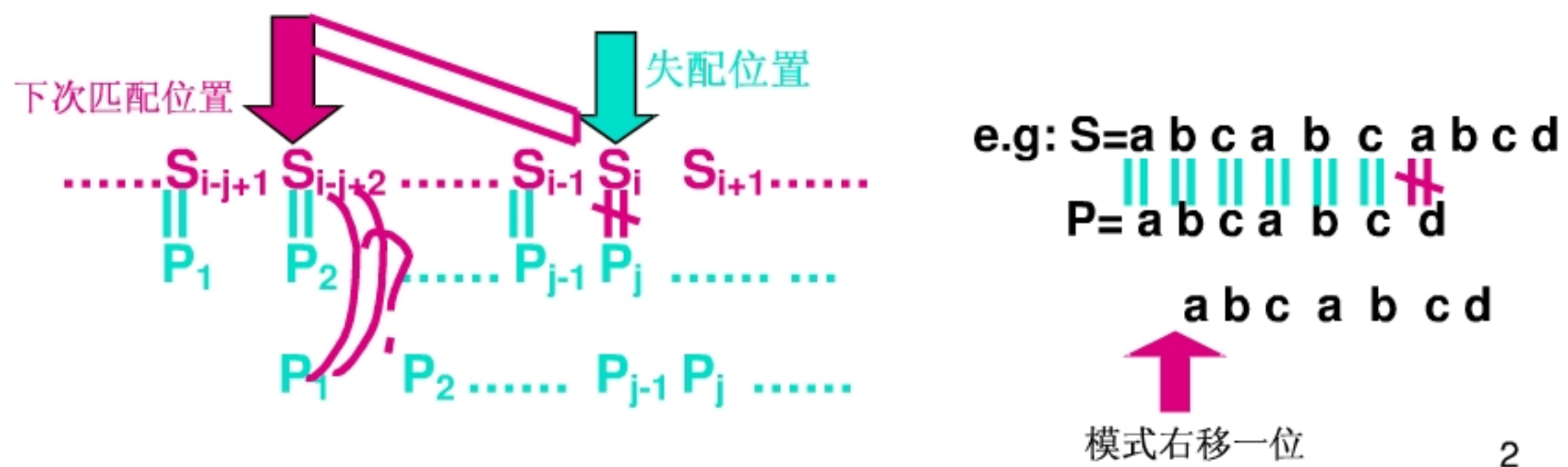
```
#define MAXSTRLEN 255 // 最大串长
```

```
typedef unsignedchar Sstring[ MAXSTRLEN + 1 ]; // 可用 0 号单元存放串长度
```

- 模式 **P**（样品、子串）：要寻找的字符串，存于 **T[1]** 至 **T[M]** 之中；
- 主串：在其中寻找模式的主字符串，存于 **S[1]** 至 **S[N]** 之中；
- 问题：在主串中寻找一个模式？如何做，更快？

采用最笨的办法，一旦发现出现字符不匹配，则整个模式相对于原来的位置右移一位。

如下图所示：



2

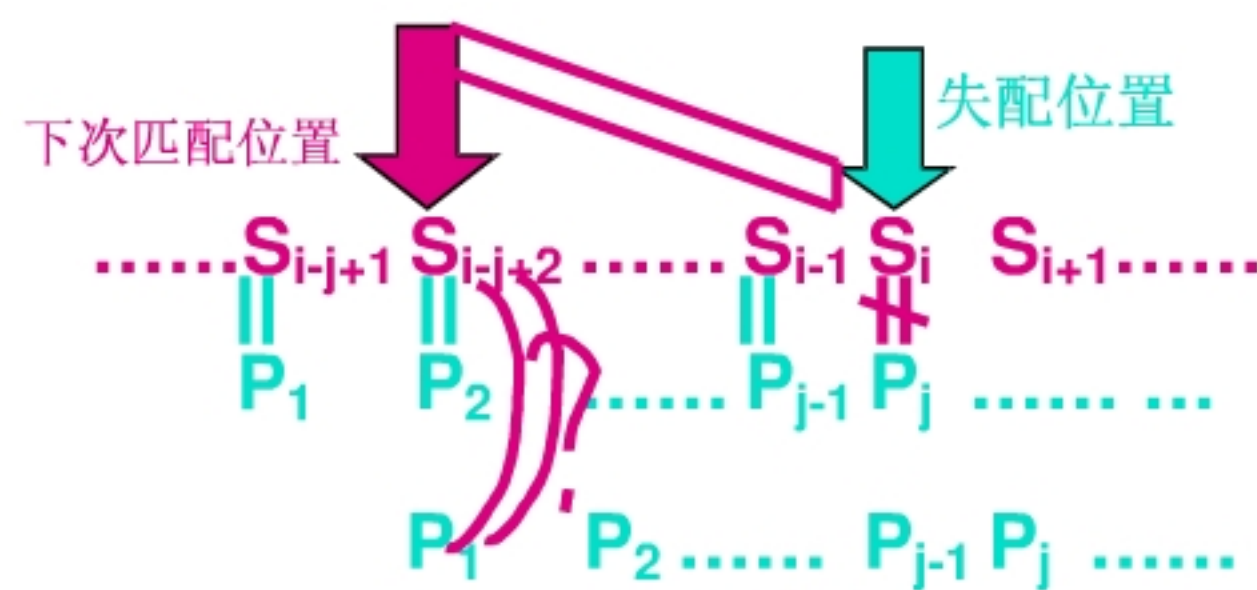
STRG

3、串的模式匹配算法

2、最原始的模式匹配程序：

```

int Index( SString S, SString T, int pos )
// 在主串S的第POS个字符之后，寻找模式T的匹配位置
{
    i = pos ; j = 1;
    while ( i <= S[0] && j <= T[0] )
    {
        if ( S[i] = T[j] ) { ++i ; ++j ; }
        else { i = i - j + 2; j = 1; }
    }
    if ( j > T[0] ) return i - T[0] // T在S中的匹配起始位置
    else return 0;
} // Index
    
```



3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- 起因：降低时间代价，从最坏情况下的 $O(n * m)$ 降低到 $O(n + m)$ ；
此处 n 是主串的字符个数、 m 是子串的字符个数。
- 历史：70 年 S.A.cook 从理论上证明可在 $O(n + m)$ 内完成，以后由以上三人给出实现的程序。

E.g: 说明最坏情况下时间复杂性的 $O(n * m)$ 的实例。



3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- 说明 KMP 算法的实例：

e.g: S = abcabcabcd

P = abcabcd

S = abcabcabcd

P = abcabcd

三次比较省去！

再右移一位，三次比较之后，再进行断点处的比较，比较上了！

S = abcabcabcd

P = abcabcd

右移一位，仍失配

本次比较省去！

S = abcabcabcd

P = abcabcd

又右移一位，仍失配

本次比较省去！

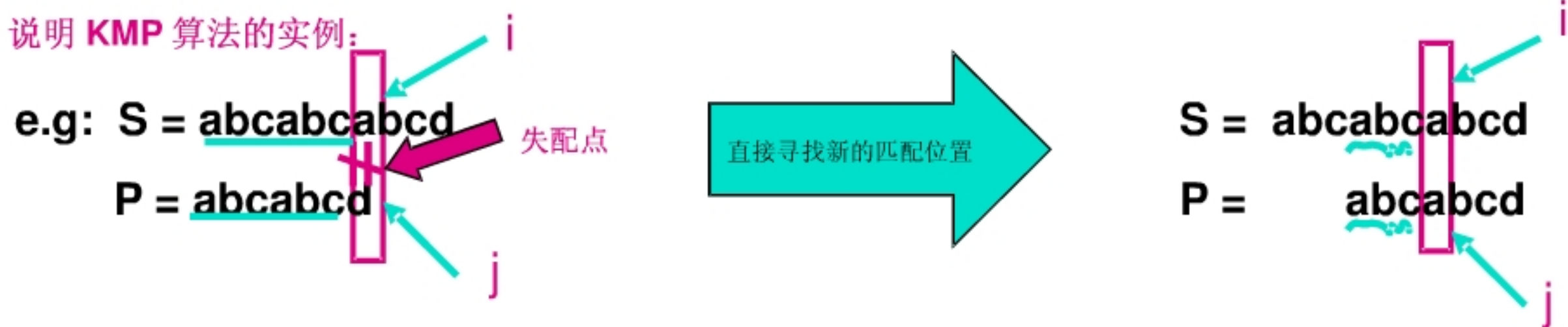


问题：能否省去上述五次比较，直接进行 S7 和 P4 之间的比较呢？

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- 说明 KMP 算法的实例：



省去上述五次比较，直接进行 S_i 和 P_j (即 S_7 同 P_4) 之间的比较的可能性。

- 分析：当 S_i 和 P_j 发生失配时， $S_{i-j+1} S_{i-j+2} \dots S_{i-1} = P_1 P_2 \dots P_{j-1}$



如果： $P_1 P_2 \dots P_{k-1} = P_{j-k+1} P_{j-k+2} \dots P_{j-1}$ ，可以直接比较

前缀（长度 $k-1$ ）= 以失配点的前一字符为结束位置的一串字符

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- 分析：多个前缀时，会出现什么问题呢？

e.g: $S = \text{aaaaabaab}$
 $P = \text{aaaab}$

前缀: $P_1P_2P_3P_4 = \text{aaaa}$

$P_1P_2P_3 = \text{aaa}$

$P_1P_2 = \text{aa}$

$P_1 = \text{a}$

- 1、如果，取前缀长度 $k-1 = j-1 (4)$ ，则 $S_i = P_k$ 即 P_j 进行比较，白做。故 前缀长度不可以为 $j-1$ 。
- 2、如果，取前缀长度 $k-1 = 1$ 或 2 ，即前缀分别为 $P_1=\text{a}$ 或 $P_1P_2=\text{aa}$ 则正确的位置会漏过去，不行。故：前缀长度太短也不行。

$S = \text{aaaaabaab}$
 $P = \text{aaaab}$

$S = \text{aaaaabaab}$
 $P = \text{aaaab}$

- 3、因此应选前缀长度 $k-1 < j-1$ （此例为 $k-1 = 3$ ）的最长的前缀。

3、串的模式匹配算法

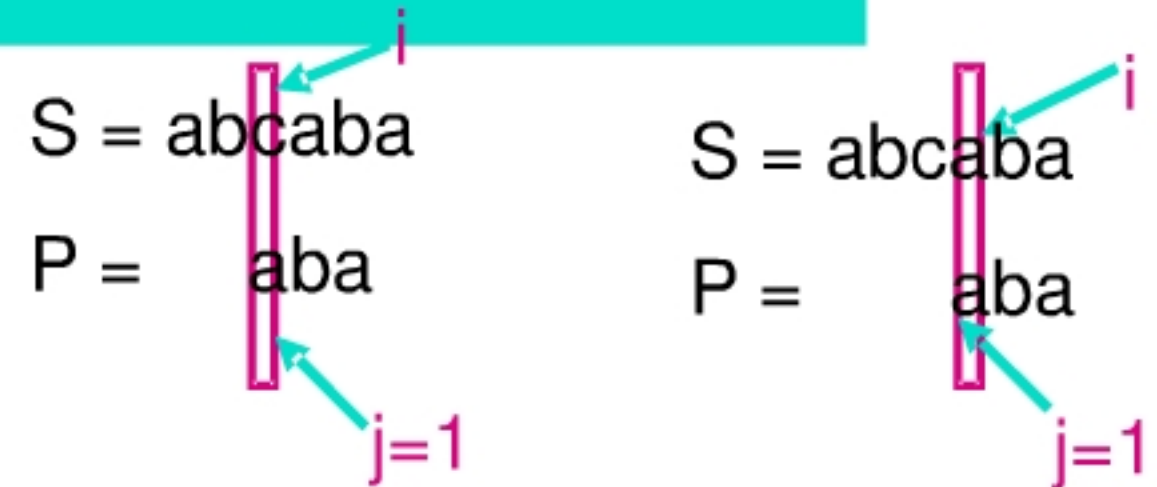
3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- **NEXT[j]** 的定义：当失配点发生在 **P_j** 处时，主串中的失配点将和模式中的哪一个字符进行比较。那一个字符的位置定义为 **NEXT[j]**。

$$\text{NEXT}[j] = \begin{cases} 0 & \text{如果 } j=1, \text{ 意味着失配点 } S_i \neq P_1, \text{ 下一次 } S_{i+1} \text{ 与 } P_1 \text{ 比较} \\ \text{MAX} \{ k \mid 1 < k < j \text{ 且 } P_1 \dots P_{k-1} = P_{j-k+1} \dots P_{j-1} \} & \\ 1 & \end{cases}$$

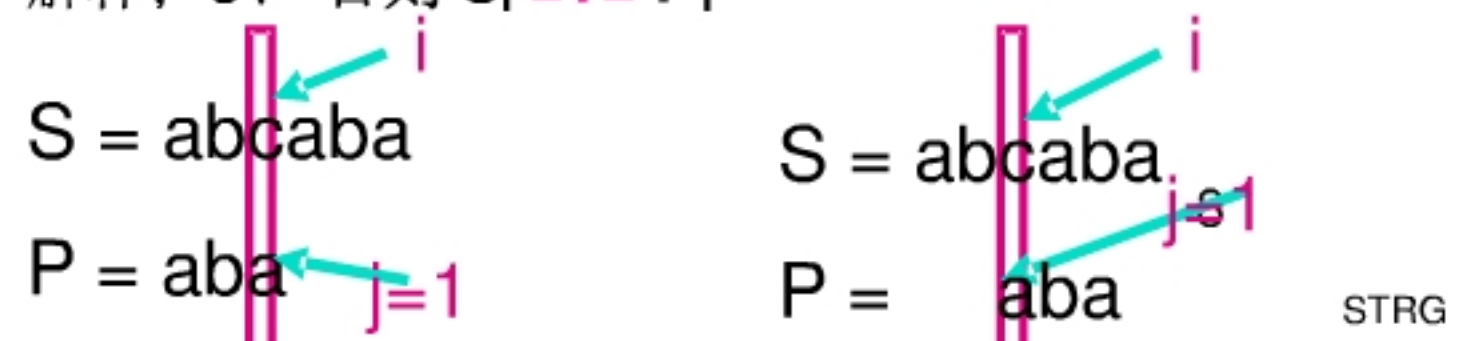


解释：1、



解释：2、 $k-1 < j-1$ 且 $k-1 \geq 1$ ，所以： $1 < k < j$

解释：3、否则 $S_i \neq P_1$



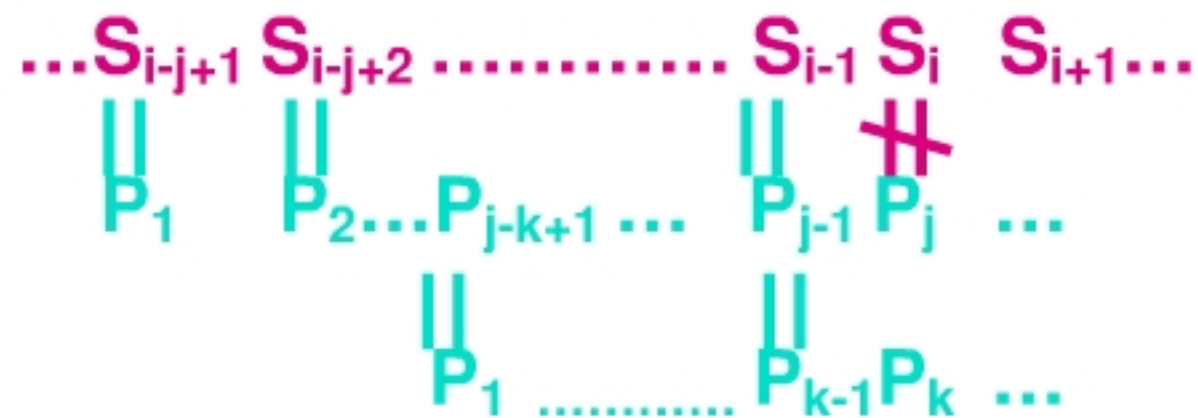
STRG

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

• 求 $NEXT[j]$ 的实例：

$$NEXT[j] = \begin{cases} 0 & \text{如果 } j=1, \text{ 意味着失配点 } S_i \neq P_1, \text{ 下一次 } S_{i+1} \stackrel{?}{=} P_1 \\ \text{MAX} \{ k \mid 1 < k < j \text{ 且 } P_1 \dots P_{k-1} = P_{j-k+1} \dots P_{j-1} \} & \\ 1 & \end{cases}$$



求 $NEXT[j]$:

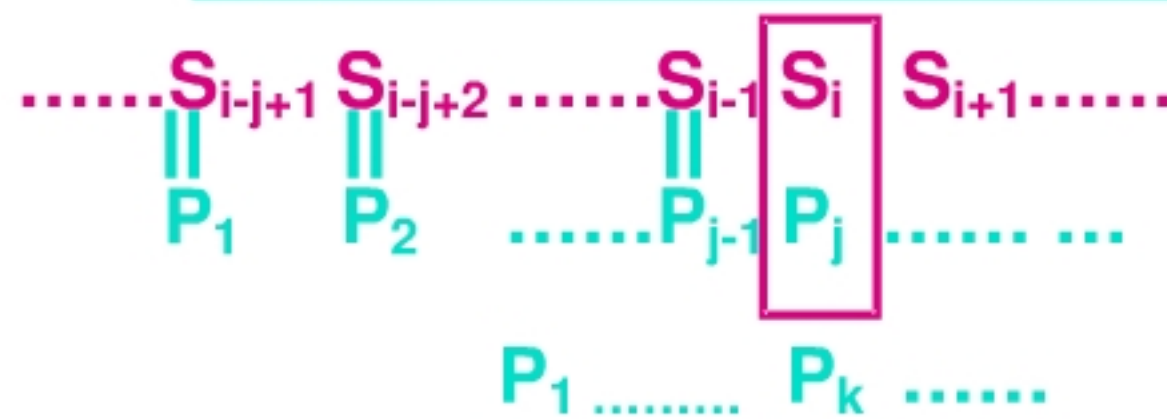
e.g:	j = 1234567	j = 12345678	j = 1234567
	abcbacd	abaabcac	aaaaaaa
NEXT[j]=	0111234	01122312	0123456
NEXTVAL[j]=	0110114	01021302	0000000

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- 利用 **NEXT[j]** 函数值寻找模式的程序：

```
int Index_KMP( SString S, SString T, int pos )
// 在主串S的第POS个字符之后，寻找模式T的匹配位置
// 已知 NEXT 函数值，T 非空，1<=pos<=Strlength(S)
{
    i = pos ; j = 1;
    while ( i <= S[0] && j <= T[0] )
    {
        if ( j == 0 || S[i] == T[j] ) { ++i; ++j; }
        else j = next[ j ];
    }
    if ( j > T[0] ) return i - T[0] // T在S中的匹配起始位置
    else return 0;
} // Index_KMP
```



3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法 (KMP 算法)

• NEXT[j] 函数值的求法:

1、由定义, $\text{next}[1] = 0$

2、若 $\text{next}[i] = j$, 求 $\text{next}[i+1] = ?$

由已知可得:

$$P_1 P_2 \dots P_{j-1} = P_{i-j+1} P_{i-j+2} \dots P_{i-1}$$

①、若 $P_j = P_i$; 则

$$P_1 P_2 \dots P_{j-1} P_j = P_{i-j+1} P_{i-j+2} \dots P_{i-1} P_i$$

所以, $\text{next}[i+1] = j + 1$

②、若 $P_j \neq P_i$; 不得认为

$\text{next}[i+1] = 1$; 参见下述例子:

$i=1$ i $i+1$
 $P = \text{abcabdabcabcwabcabdabcabdx}$

注意: i 是模式的指针。指针 j 为前缀个数 + 1

```
Void get_next ( SString T, int & next [ ] )
{ i = 1 ; next[1] = 0 ; j = 0 ;
  while ( i < T[0] )
  { if ( j == 0 || T[i] == T[j] ) { ++i ; ++j ; next[i] = j ; }
    else j = next[j] ;
  }
} // get_next
```

$\text{next}[i]=12$ 且 i $i+1$ 求 $\text{next}[i+1]$
 $P = \text{abcabdabcabcwabcabdabcabdx}$
 12?

虽然 $P_{12} \neq P_j$, 但 $P_1 P_2 \dots P_5 = P_{j-5} P_{j-4} \dots P_{j-1}$ 推出: $P_1 P_2 \dots P_5 P_6 = P_{i-5} P_{i-4} \dots P_{i-1} P_i$


由此可以推出: $\text{next}[i+1]=7$

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

- KMP 算法的时间复杂性： $O(n+m)$ n :主串长度， m :模式长度。

考察例子：



S=abcabcabcabcdef
P=abcabd

- 1、从 **S** 的指针观察，**i** 每右移一次，对应一次比较。因此，最多对应着 **n** 次比较。
- 2、从模式 **P** 进行考察，当失配时，**j = next[j]**。主串失配点 **S_i** 又将和 **P_j** 进行比较，对应着新增加的比较次数。**P** 相对原来的位置右移。右移位数最多 **n** 次，新增加的比较次数最多为 **n** 次。

所以，最多的比较次数最多为 **2n** 次，同理生成 **next[]** 函数值的代价也不会大于 **2m**。所以，总的代价 $< 2(n+m)$

3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

• KMP 算法的不用回溯的优点：

考察例子：

不回溯：
S=abcabcbcdabcbabd
P=abcabd

回溯：
S=abcabcbcdabcbabd
P=abcabd

不回溯：
S=abcabcbcdabcbabd
P=abcabd

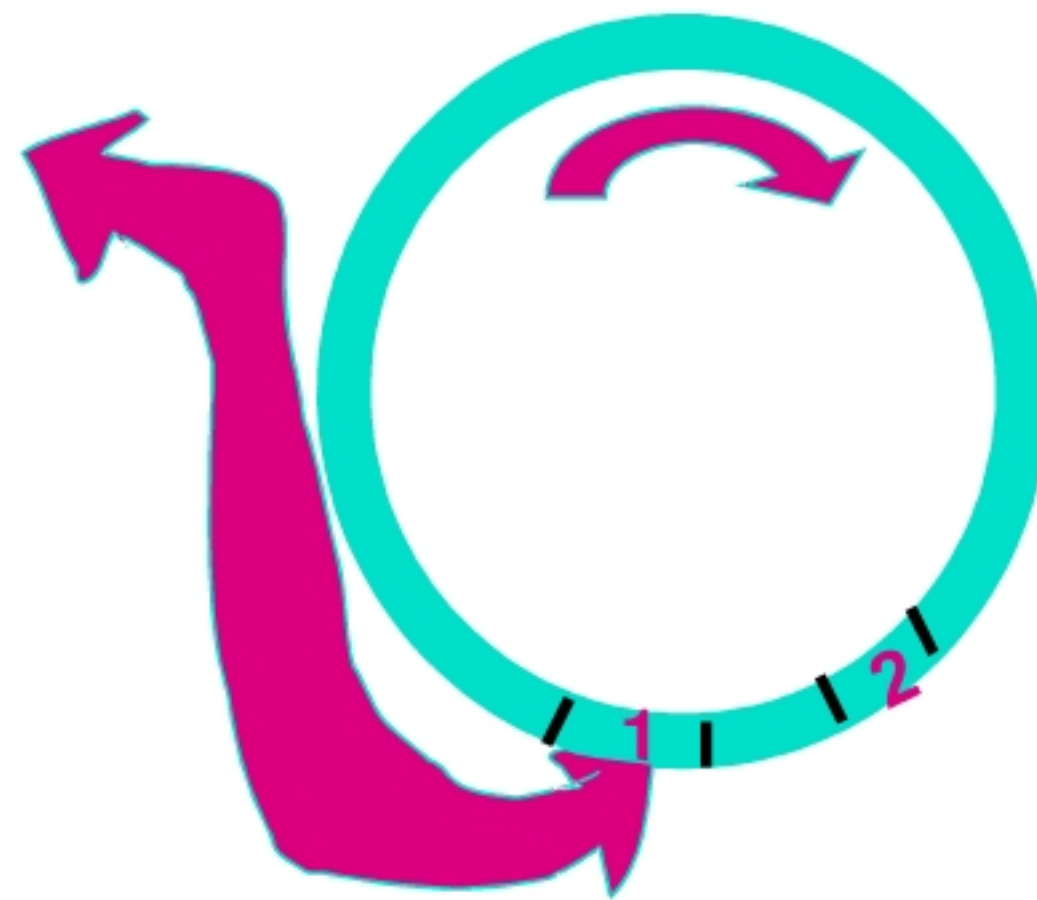
- 资料存于盘上
- 每个扇区 256 个字节
- 模式 257 个字节
- 模式的前 256 个字节匹配上了
- 下一个扇区调入内存，覆盖原来的 256 个字节
- 但和模式的第 257 个字符不同。

两种不同的处理方法，结果不同！！

存放主串的
每个扇区，
一次存放
256 个字节

存放模式以及模式的
next[] 值

内存



3、串的模式匹配算法

3、Knuth-Morris-Pratt 模式匹配算法（KMP 算法）

• 求 $NEXTAL[j]$:

$$NEXVAL[j] = \begin{cases} \text{MAX} \{ k \mid 1 < k < j \text{ 且 } P_1 \dots P_{k-1} = P_{j-k+1} \dots P_{j-1} \text{ AND } P_k \neq P_j \} \\ 1 & \text{上式不成立, 且 } P_j \neq P_1 \\ 0 & \text{上二式不成立 意味着 } P_j = P_1, \text{ 故: } S_i \neq P_1 \text{ 下一次 } S_{i+1} = ? = P_1 \end{cases}$$

解释:

$\dots S_{i-j+1} S_{i-j+2} \dots S_{i-1} S_i S_{i+1} \dots$
 $\quad \quad \quad \parallel \quad \parallel \quad \dots \quad \parallel \quad \parallel$
 $\quad \quad \quad P_1 \quad P_2 \dots P_{j-k+1} \dots \quad P_{j-1} P_j \dots$
 $\quad \quad \quad \parallel \quad \quad \quad \parallel \quad \parallel$
 $\quad \quad \quad P_1 \dots \dots \dots P_{k-1} P_k \dots$

如果 P_k 和 P_j 相等, 那么 S_i 不可能和 P_k 相等。所以, 这一步比较可以不予进行。但如 P_k 和 P_j 不相等, 则有可能 S_i 可能和 P_k 相等。比较不能省略。所以, 规定 $P_j \neq P_k$ 可以节约比较的次数, 加快匹配过程。

e.g: $P = \text{aaaab}$

NEXT 01234

NEXTVAL 00004

$S = \text{aaaxaaaaab}$
 $\quad \text{aaaab}$
 $\quad \quad \text{aaaab}$
 $\quad \quad \text{aaaab}$



14

STRG