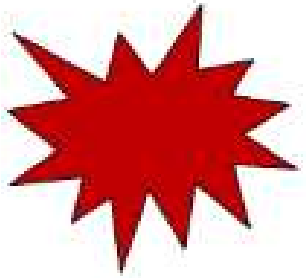
数据结构与算法

疫情面前，让我们一起努力！

1



从文献中可知，模式匹配是一种查找敏感信息的有效方法。

模式匹配问题，是串的主要应用。

串的模式匹配操作主要是Index (S, T, pos) 操作如何实现？？？

3

模式匹配：求子串在主串中位置的操作，即子串的定位操作：**Index（S,T,POS）。**

Index (S, T, pos) //求子串T在主串S中的位置（序号）

参数要求：串**S**和**T**存在，**T**是非空串，

1≤pos≤StrLength(S)。

操作结果：若主串 **S** 中存在和串 **T** 值相同的子串，则返回**T**在主串 **S** 中第**pos**个字符之后第一次出现的位置；否则函数值为0。

串的模式匹配

采用定长顺序结构为例介绍串的模式匹配算法

1、经典BF模式匹配算法

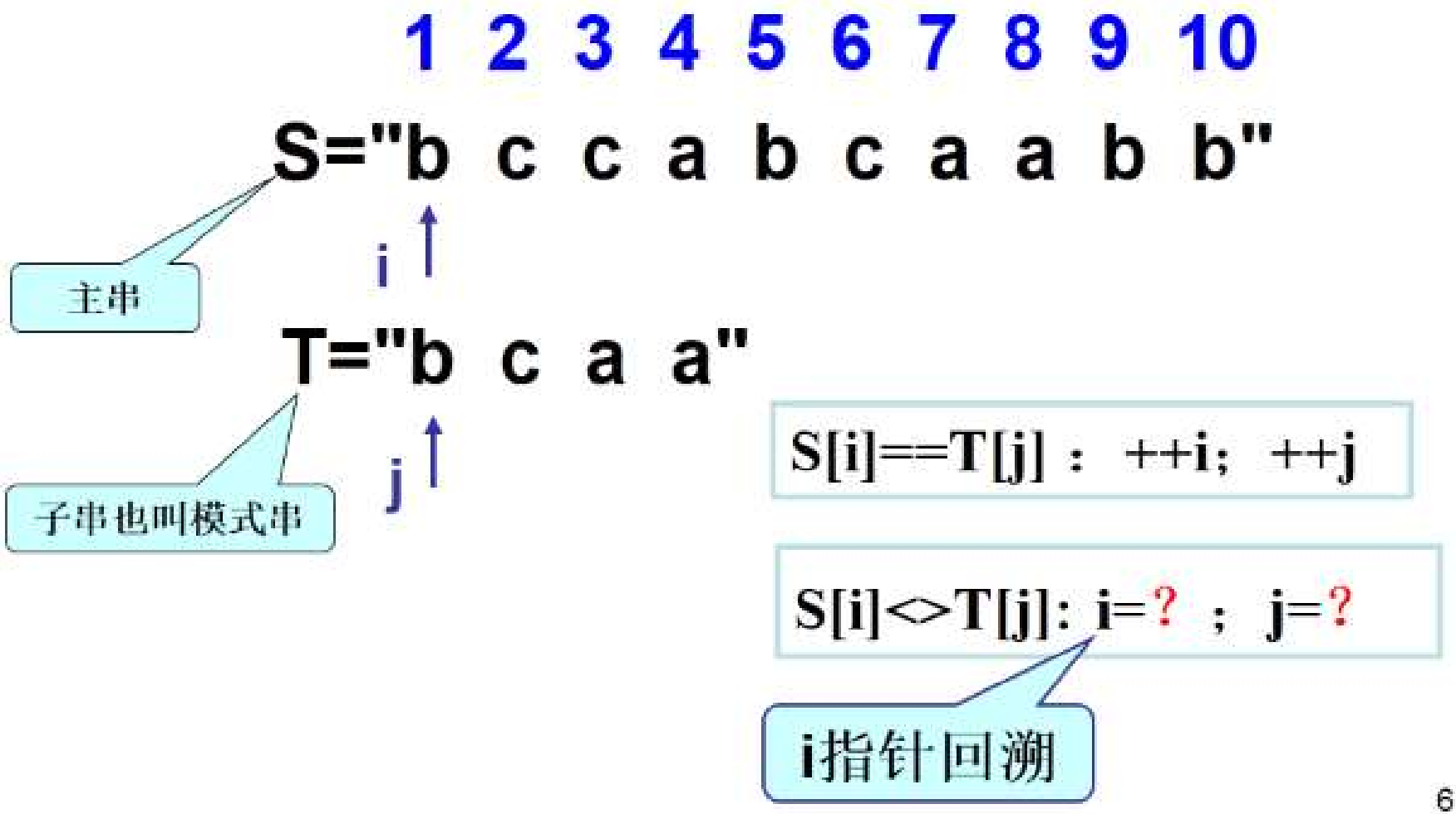
核心算法：

- 将主串的第pos个字符和模式串的第1个字符比较，
  若相等，继续逐个比较后续字符；
  若不等，从主串的下一字符起，重新与模式的第一个字符比较。

直到主串的一个连续子串字符序列与模式串相等，匹配成功，返回值为S中与T匹配的子串中第一个字符的序号。否则，匹配失败，返回值 0 。

5

串的模式匹配算法


1 2 3 4 5 6 7 8 9 10
S="b c c a b c a a b b"
i
主串
T="b c a a"
j
子串也叫模式串
S[i]==T[j]： ++i； ++j
S[i]<>T[j]: i=? ； j=?
i指针回溯

6

单选题 3分

当S[i]<>T[j]时，i，j指针如何变化？

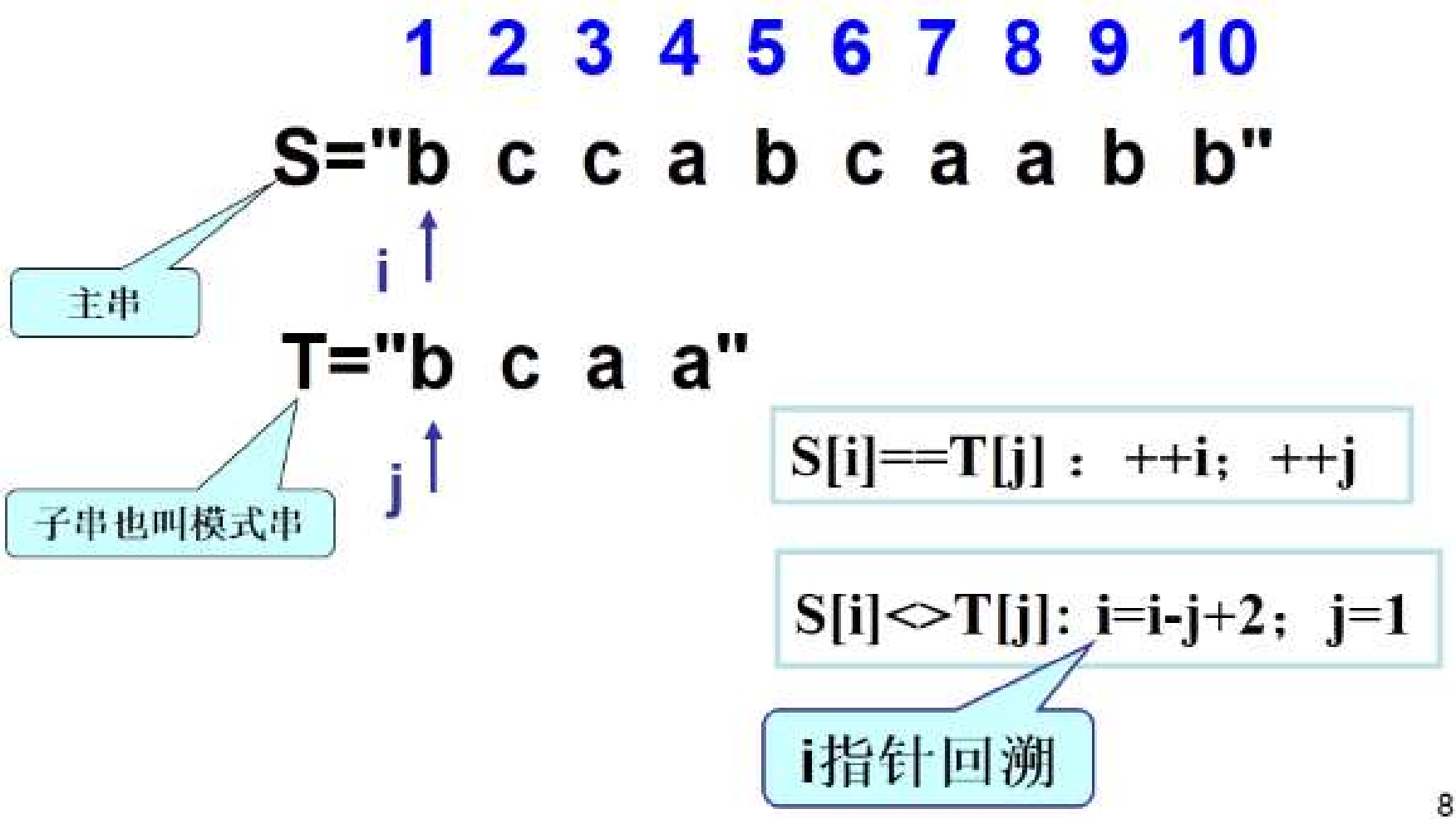
A i=1；j=1

B i=i-j+1；j=1

C i=i-j+2；j=1

D 以上都不对

串的模式匹配算法


1 2 3 4 5 6 7 8 9 10
S="b c c a b c a a b b"
i
主串
T="b c a a"
j
子串也叫模式串
S[i]==T[j]： ++i； ++j
S[i]<>T[j]: i=i-j+2； j=1
i指针回溯

8

串的模式匹配算法

```
int  Index(SString S, SString T, int pos) {
     // 返回子串T在主串S中第pos个字符之后的位置。若不存
       在，则函数值为0。
    // 其中，T非空，1≤pos≤StrLength(S)。
 (1)   i = pos;   j = 1;
 (2)  while (i <= S[0] && j <= T[0]) {
 (3)     if (S[i] == T[j]) { ++i;  ++j; }    // 继续比较后继字符
 (4)    else { i = i-j+2;   j = 1; }         // 指针后退重新开始匹配
   }
 (5) if (j > T[0])  return  i-T[0];//匹配成功，返回在主串的位置
 (6) else return 0;
} // Index
```

串的模式匹配算法

分析：BF匹配算法在最坏情况下的效率

例：S="aaaaaaaaaaaaaaab" , T="aaab" , pos=1

主串长度m=15，模式串长度n=4

最坏情况是：主串前面15-4个位置都部分匹配到子串的最后一位，即这11位比较了4次，最后4位也各比较了一次，还要加上4！

比较字符的次数为：11*4+4=48次

在最坏的情况下，需要比较字符的总次数为(n-m)*m+m次，算法的时间复杂度为

O（m*n）

串的模式匹配算法

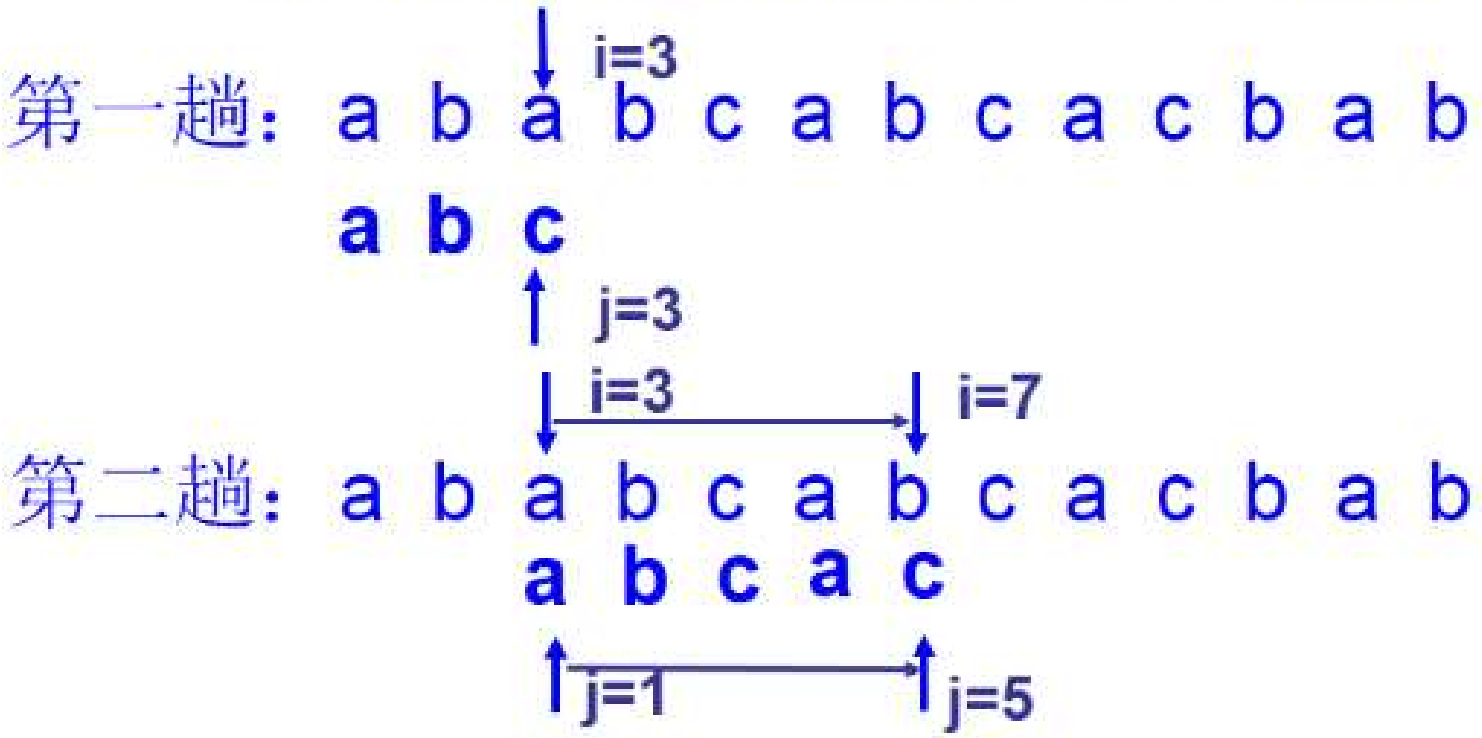
KMP算法的改进思想：

每当一趟匹配过程中出现字符比较不等时，不需回溯 i 指针，而是利用已经得到的“部分匹配”的结果将模式串向右“滑动”尽可能远的一段距离后，继续进行比较。

11

串的模式匹配算法

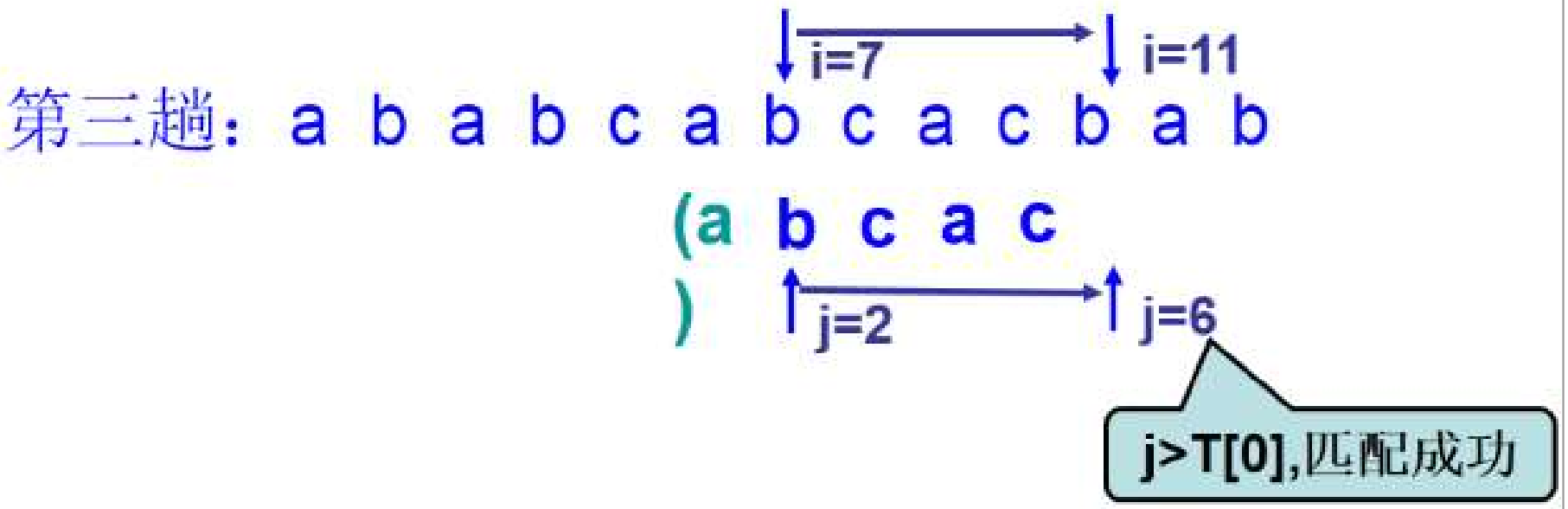
示例：S="ababcabcacbab"，T="abcac"



12

串的模式匹配算法

示例：S="ababcabcacbab"，T="abcac"



在整个匹配过程中，i指针没有回溯，时间复杂度为O（m+n）。

串的模式匹配算法-KMP算法

当 … 溯， … 个 字符 …

得到部分匹配结果："$S_{i-k+1}S_{i-k+2}\cdots S_{i-1}$"="$T_{j-k+1}T_{j-k+2}\cdots T_{j-1}$"

又得到满足的等式："$S_{i-k+1}S_{i-k+2}\cdots S_{i-1}$"="$T_1T_2\ldots T_{k-1}$"

原理分析 …

可得："$T_1T_2\ldots T_{k-1}$"="$T_{j-k+1}T_{j-k+2}\cdots T_{j-1}$"

主串 … 此时

主串的第i个字符应和模式串的第k（k<j）个字符再比较，则

$$S_1S_2\ldots S_{i-j+1}S_{i-j+2}\cdots S_{i-k+1}\cdots S_{i-j+k}S_{i-j+k+1}\cdots S_{i-2}\ S_{i-1}\ S_i\ S_{i+1}\cdots$$

$$\begin{array}{ccccccccc} \| & \| & \| & \| & \| & \| & \| & ? \\ T_1 & T_2 & T_{j-k+1} & T_k & T_{k+1} & T_{k-2} & T_{k-1} & T_j \end{array}$$

15

串的模式匹配算法-KMP算法

根据模式串推得的规律：　"$T_1 \cdots T_{k-1}$"="$T_{j-(k-1)} \cdots T_{j-1}$"
和已知的当前失配位置j，可以归纳出计算新起点 k 的表达式。

令 next[j]=k，表示当模式串第j个字符与主串中相应字符“失配”时，应用模式串中第k个字符重新和主串中字符进行比较。

定义：

模式串的*next*函数

$$\begin{cases} 0 & \text{当 } j=1 \text{ 时} \\ \max\{k \mid 1<k<j \text{ 且 } "T_1 \cdots T_{k-1}" = "T_{j-(k-1)} \cdots T_{j-1}"\} & \\ 1 & \text{其他情况} \end{cases}$$

串的模式匹配算法-KMP算法

例： 模式串 T： a a a a b

可能失配位 j： 1 2 3 4 5

next[j]： 0 1 2 3 4

$$next[j]=\begin{cases}0 & \text{当}j=1\text{时}\\ \max\{k \mid 1<k<j \text{ 且 } "T_1\cdots T_{k-1}"="T_{j-(k-1)}\cdots T_{j-1}"\} & \\ 1 & \text{其他情况}\end{cases}$$

j=4时，next[j]=3；属于k=2时情况，"T1"="T3" 成立，k可以取2

k可取2或3

next[j]= 1；属于“其他情况”；k=3时，"T1T2"="T2T3"成立，k可以取3

j=3时同理next[j]=2；j=5时，next[j]=4

因1<k<j,所以k可以取2

串的模式匹配算法-KMP算法

KMP算法如下　求子串T在主串S中第pos个字符之后的位置

```
:Int Index_KMP(SString S, SString T, int pos) {
  i=pos;     j=1;                    我们回忆一下简单模式匹配算法
  while ( i<=S[0] && j<=T[0] )
   { if (  j==0|| S[i] == T[j] ) {++i;   ++j;}   //不失配则继续比较后续字符
         else {  j=next[j];  } }//指针回溯重新开始下一次匹配
      if(j>T[0]) return i-T[0];  //子串结束，说明匹配成功
         else return 0;
}//Index_KMP
```

串的模式匹配算法-KMP算法

分析next[j]的计算过程：

已知：next[1] = 0；假设：next[j]=k；

表明模式串中满足"$T_1 \cdots T_{k-1}$"="$T_{j-k+1} \cdots T_{j-1}$" 推导next[j+1]=?

存在两种情况

（1）如果"T_k"="T_j"，则满足"$T_1 \cdots T_{k-1} T_k$"="$T_{j-k+1} \cdots T_{j-1} T_j$"

next[j+1]=k+1

（2）如果"T_k"≠"T_j" 这实际上有是串的模式匹配失配的问题，只不过主串和子串是同一个串而已。

相当于在位置k时失配了，那应该用位置为next[k]的字符去和T_j比较，即$T_{next[k]}$和T_j比较

又分为和上述相同的两种情况

串的模式匹配算法-KMP算法

（1）如果"$T_{next[k]}$"="T_j"，则next[j+1]=next[k]+1

（2）如果"$T_{next[k]}$"≠"T_j"，

相当于在位置next[k]时失配了，那应该用位置为next[next[k]]的字符去和T_j比较，即$T_{next[next[k]]}$和T_j比较

又分为和上述相同的两种情况,以此类推，直至T_j和模式串中的某个字符匹配成功，或者没有匹配的字符(没有部分匹配的结果)，只能从头匹配，则next[j+1]=1。

20

串的模式匹配算法-KMP算法

求next[j]函数值的算法如下：

```
void get_next(SString &T, int &next[] ) {
  // 求模式串T的next函数值并存入数组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
```

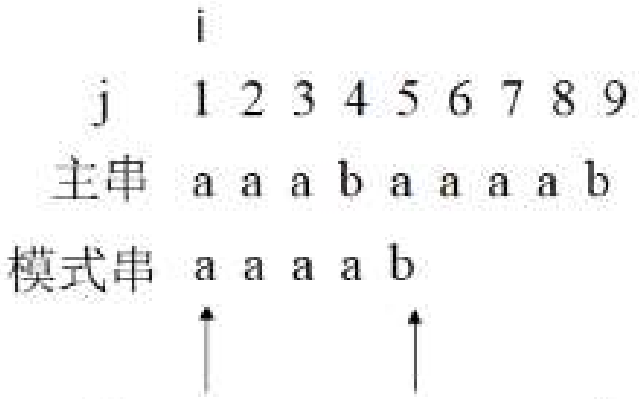
21

串的模式匹配算法-KMP算法

我们来看一个特殊例子　主串为"aaabaaaab"，模式串为"aaaab"

：

j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4



当i=4，j=4时，模式串和主串失配，然后对j=3、j=2、j=1均做了比较，但因为模式串的j=4和j=3、2、1位置上的字符是相等的，因此不需要再和主串的第4个字符进行比较，而应直接进行i=5和j=1的字符比较。

j=4，失配，next[4]=3

j=0时，i++，j++

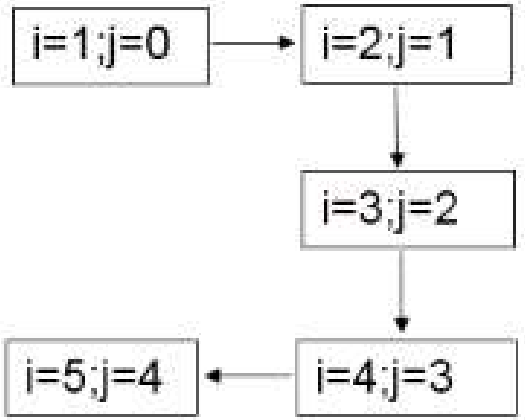
i=5，j=1

串的模式匹配算法-KMP算法

求next函数值的改进算法：

```
void get_nextval(SString &T, int &nextval[] ) {
  // 求模式串T的next函数值并存入数组next。
  i = 1;   nextval[1] = 0;   j = 0;
  while (i < T[0]) {
    if (j == 0 || T[i] == T[j])
      { ++i;  ++j;
        if (T[i]!=T[j]) nextval[i] = j;
        else  nextval[i]=nextval[j]; }
    else  j = nextval[j];
  }
} // get_next
```

j	1	2	3	4	5
模式串	a	a	a	a	b
next[j]	0	1	2	3	4
nextval[j]	0	0	0	0	4



23

前面我们学习了：

理解串的表示及实现

理解串的模式匹配

24