3.8 串的模式匹配算法（续）

DS/SS

改进算法---思路

如何理解“部分匹配”？

主串： a c a b a a c a a b c a c

模式串： a c a a b

KMP算法思路：

从主串S的第pos个字符起和模式的第一个字符比较之，若相等，继续逐个比较后续字符。当一趟匹配过程中出现字符比较不等时，不回溯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。

优点：

- 在匹配过程中，主串的跟踪指针不回溯
- 时间效率达到T(n) = O(n+m)

3.8 串的模式匹配算法（续）

DS/SS

改进算法---原理

设主串S=‘$s_1s_2\ldots s_i\ldots s_n$’，

模式串T= ‘$t_1t_2\ldots t_j\ldots t_m$’

在匹配过程中，当主串中第i个字符与模式串中第j个字符“失配”时（s_i不等于t_j），将模式串“向右滑动”，让模式串中第k（k<j）个字符与s_i对齐继续比较。这时有：

$$‘t_1t_2\ldots t_{k-1}’ = ‘s_{i-k+1}s_{i-k+2}\ldots s_{i-1}’ \quad \text{-----(1)}$$

而由部分匹配成功的结果可知：

$$‘t_1t_2\ldots t_{j-1}’ = ‘s_{i-j+1}s_{i-j+2}\ldots s_{i-1}’ \quad \text{-----(2)}$$

由(2)式可以推知：

$$‘t_{j-k+1}t_{j-k+2}\ldots t_{j-1}’ = ‘s_{i-k+1}s_{i-k+2}\ldots s_{i-1}’ \quad \text{-----(3)}$$

由(1)式与(3)式可以推知：

$$‘t_1t_2\ldots t_{k-1}’ = ‘t_{j-k+1}t_{j-k+2}\ldots t_{j-1}’ \quad \text{-----(4)}$$

3.8 串的模式匹配算法（续）

DS/SS

设主串S=‘$s_1s_2\ldots s_i\ldots s_n$’，

模式串T= ‘$t_1t_2\ldots t_j\ldots t_m$’

改进算法---Next函数定义

令next[j] = k，表示当模式串中第j个字符与主串中相应字符“失配”时，在模式中需重新和主串中该字符进行比较的字符的位置。根据其语义，定义如下：

$$Next[j] = \begin{cases} 0 & \text{当 } j = 1 \text{ 时} \quad \text{//相当于主串中i指针推进一个位置} \\ Max\{k \mid 1<k<j \text{ 且 } ‘t_1t_2\ldots t_{k-1}’ = ‘t_{j-k+1}t_{j-k+2}\ldots t_{j-1}’\} & \text{//保证得到第一个“配串”} \\ 1 & \text{其他情况} \end{cases}$$

Next函数值仅取决于模式串本身的结构而与相匹配的主串无关

j	1	2	3	4	5	6	7	8
模式串	a	b	a	a	b	c	a	c
next[j]								

j	1	2	3	4	5	6	7	8
模式串	a	b	a	a	b	c	a	c
next[j]	0	1	1	2	2	3	1	2

3.8 串的模式匹配算法（续）

DS/SS

改进算法---匹配过程

j	1	2	3	4	5	6	7	8
模式串	a	b	a	a	b	c	a	c
next[j]	0	1	1	2	2	3	1	2

第一趟　主串：a c a b a a b a a b c a c a a b c //i =2

　　模式串：a b //j =2, next[2]=1

第二趟　主串：a c a b a a b a a b c a c a a b c //i =2

　　模式串：a //j =1, next[1]=0

第三趟　主串：a c a b a a b a a b c a c a a b c //i =8

　　模式串：a b a a b c //j =6, next[6]=3

第四趟　主串：a c a b a a b a a b c a c a a b c //i =14

　　模式串：(a b) a a b c a c //j =9

3.8 串的模式匹配算法（续）

改进算法---KMP算法

```
int Index_KMP(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,函数值为0
    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];       else  return 0;
} // Index_KMP
```

算法时间复杂度： T(n) = O(n+m)

3.8 串的模式匹配算法（续）

DS/SS

递归法：

递归基础：　　next[1] = 0　　-----(1)

设next[j] = k，则有：　‘$t_1t_2\ldots t_{k-1}$’ = ‘$t_{j-k+1}t_{j-k+2}\ldots t_{j-1}$’　　-----(2)

则考察next[j+1]：

* 若$t_k = t_j$，有：　‘$t_1t_2\ldots t_{k-1}\ t_k$’ = ‘$t_{j-k+1}t_{j-k+2}\ldots t_j$’　　-----(3)

即 next[j+1] = k+1=next[j] + 1　　-----(4)

* 若t_k != t_j，表明：　‘$t_1t_2\ldots t_{k-1}\ t_k$’ != ‘$t_{j-k+1}t_{j-k+2}\ldots t_j$’，又是一个模式匹配问题。这时由于$t_k$!= t_j，设next[k] = k’，则又是两种情况：　$t_{k'}$= t_j以及$t_{k'}$!= t_j。对于$t_{k'}$= t_j，有：‘$t_1t_2\ldots t_{k'}$’ = ‘$t_{j-k'+1}t_{j-k'+2}\ldots t_{j-1}\ t_j$’　(1<k’<k<j　)　　-----(5)

即 next[j+1] = k’+1=next[k] + 1　　-----(6)

对于$t_{k'}$!= t_j，依次类推，直至t_j和模式串中某个字符匹配成功或者不存在任何k’满足(5)式，则next[j+1] = 1　　-----(7)

3.8 串的模式匹配算法（续）

DS/SS

j	1	2	3	4	5	6	7	8
模式串	a	b	a	a	b	c	a	c
next[j]	0	1	1	2	2	3	1	2

已知前6个字符的next函数值，依次求next[7]和next[8]

对于求next[7]，即j+1=7,j=6。已知next[6]=3,即k=3。

因为t_6 != t_3，而next[3]=1,即k’=1,考察而知，又有 t_6 != t_1，

所以，不存在k’满足匹配式，则next[j+1]=1，即next[7]=1

对于求next[8]，即j+1=8,j=7。已知next[7]=1,即k=1。

考察而知t_7 = t_1，

所以，next[8]=next[7+1]=next[7]+1=1+1=2

3.8 串的模式匹配算法（续）

改进算法---Next函数算法

```
void get_next(SString T, int &next[ ]) {
//求模式串T的next函数值并存入数组next
    j = 1;          next[1] = 0;        k = 0;     //初始化
    while (j <= T[0]) {                            //求出每个字符的next值
        if (k = = 0 || T[j] = = T[k]) { ++j;    ++k;    next[j] = k;}
        else  k =next[k];
    }
} // get_next
```

算法时间复杂度： O(m)

3.8 串的模式匹配算法（续）

改进算法---Next函数算法

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

对初值情况的处理

对$t_k = t_j$情况的处理

台化

符的next值

对t_k != t_j情况的处理

考察下一个字符

算法时间复杂度： O(m)

3.8 串的模式匹配算法（续）

改进算法---改进的Next函数

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

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

3.8 串的模式匹配算法（续）

DS/SS

```
void get_nextval(SString T, int &nextval[ ]) {
    j = 1;              nextval[1] = 0;     k= 0;
    while (j <= T[0]) {
          if (k = = 0 || T[j] = = T[k]) {
              ++j;  ++k;
               if (T[j] != T[k])   nextval[j] = k;
               else nextval[j] = nextval[k] ;  // 改进之处
          }
          else  k =nextval[k];
    }
} // get_nextval
```

以避免不必要的多一次比较

3.9 串的应用

串的应用---文本编辑和建立词索引表