4.3 串的模式匹配算法

这是串的一种重要操作，很多软件，若有“**编辑**”菜单项的话，则其中必有“**查找**”子菜单项。

首先，回忆一下串匹配（查找）的定义:

INDEX (S, T, pos)

初始条件：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值相同的子串返回它在主串S中第pos个字符之后第一次出现的位置；否则函数 值为0。

4.3.1 求子串位置的定位函数 Index(S,T,pos)

```
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)。
   i = pos;   j = 1;
   while (i <= S[0] && j <= T[0]) {
     if (S[i] == T[j]) { ++i;  ++j; }   // 继续比较后继字符
     else { i = i-j+2;   j = 1; }     // 指针后退重新开始匹配
   }
  if (j > T[0])  return  i-T[0];
  else return 0;
} // Index
```

算法 4.5

在算法4.5中，分别利用计数指针 i 和 j 指示主串 S 和模式串 T 中当前正待比较的字符位置。

算法的基本思想是：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，则继续逐个比较后续字符；否则从主串的下一个字符起再重新和模式的字符比较之。依此类推，直至模式 T 中的每个字符依次和主串 S 中的一个连续的字符序列相等，则称**匹配成功**，函数值为和模式 T 中第一个字符相等的字符在主串 S 中的序号，否则称**匹配不成功**，函数值为零。

具体例子，讲义P80

4.3.2 模式匹配的一种改进算法

改进算法是D.E.Knuth与V.R.Pratt和J.H.morris同时发现的，因此称它为克努特-莫里斯-普拉特操作（简称为KMP算法）。该算法可以在O(m+n)的时间数量级上完成串的模式匹配操作。

其改进在于：每一趟匹配过程中出现字符比较不等时，不需回溯i的指针，而是利用已经得到的部分匹配结果将模式向右‘滑动’尽可能远的一段距离后，继续进行比较。

现讨论一般情况，假设主串为‘$s_1s_2...s_n$’,模式串为’ $p_1p_2...p_m$’,为了实现改进算法，需要解决下述问题：当匹配过程中产生‘失配’（即$s_i \neq p_j$）时，模式串‘向右滑动’可行的距离多远，即，当主串中第 i 个字符与模式中第 j 个字符‘失配’时，主串中第 i 个字符（i 指针不回溯）应与模式中哪个字符再比较？

假设此时应与模式中第 k（k<j）个字符继续比较，则模式中前 k-1个字符必须满足下列关系式(4-2)，且不可能存在k’>k满足下列关系式(4-2)

‘$p_1p_2...p_{k-1}$’ =’ $s_{i-k+1}s_{i-k+2}...s_{i-1}$’ (4-2)

而已经得到的‘部分匹配’的结果是

$$‘p_{j-k+1}p_{j-k+2}\cdots p_{j-1}’ = ‘s_{i-k+1}s_{i-k+2}\cdots s_{i-1}’ \qquad (4\text{-}3)$$

由式（4-2）和式（4-3）推的下列等式

$$‘p_1p_2\cdots p_{k-1}’ = ‘p_{j-k+1}p_{j-k+2}\cdots p_{j-1}’ \qquad (4\text{-}4)$$

反之，若模式串中存在满足式（4-4）的两个子串，则匹配过程中产生‘失配’（即$s_i \neq p_j$）时，仅需将模式‘向右滑动’至模式中第 k 个字符和主串中第 i 个字符对齐，此时，模式中头 k-1个字符的子串‘$p_1p_2\cdots p_{k-1}$’必定与主串中的第 i 个字符之前长度为 k-1的子串‘$s_{i-k+1}s_{i-k+2}\cdots s_{i-1}$’相等，由此，匹配仅需从模式中第 k 个字符与主串中的第 i 个字符比较起继续进行。

若令next[j]=k，则next[j]表明当模式中第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。由此可引出模式串的next函数的定义：

$$next[j] = \begin{cases} 0 & \text{当 } j = 1 \text{ 时} \\ Max\{k \mid 1 < k < j \text{ 且 } 'p_1 p_2 \cdots p_{k-1}' = 'p_{j-k+1} \cdots p_{j-1}'\} & \\ 1 & \text{其它情况} \end{cases}$$

例如：有模式串T= ‘abaabcac’ ，则其next[j]函数值如下所示：

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

```
int Index_KMP(SString S, SString T, int pos) {
    // 利用模式串T的next函数求T在主串S中的第pos个
    //字符之后的位置的KMP算法。其中，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; }  // 继续比较后继字符
    else  j = next[j];          // 模式串向右移动
    }
   if (j > T[0])  return  i-T[0];    // 匹配成功
   else return 0;
} // Index_KMP
```

算法 4.6

具体内容参考讲义P80-P84内容

求*next*函数值的过程是一个递推过程，分析如下:

已知：next[1] = 0；

假设：next[j] = k；又 T[j] = T[k]

则： next[j+1] = k+1

若： T[j] ≠ T[k]

则需往前回朔，检查 T[j] = T[？]

这实际上也是一个匹配的过程，

不同在于：主串和模式串是同一个串

```
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
```

算法 4.7

还有一种特殊情况需要考虑：例如：

S = '**aaabaaabaaabaaabaaab**'

T = '**aaaab**'

next[j]=01234

nextval[j]=00004

```
void get_nextval(SString &T, int &nextval[]) {
   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])  next[i] = j;
         else  nextval[i] = nextval[j];
      }
     else  j = nextval[j];
   }
} // get_nextval
```

算法 4.8