

第四章、串

4.1. 串的定义和实现

4.1.1、串的定义

定义：

即字符串（String）是由零个或多个字符组成的有限序列。

基本术语：

- 子串：串中任意个连续的字符组成的子序列。Eg: 'iPhone', 'Pro M' 是串T 的子串
- 主串：包含子串的串。Eg: T 是子串'iPhone'的主串
- 字符在主串中的位置：字符在串中的序号。Eg: '1'在T中的位置是8(第一次出现)
- 子串在主串中的位置：子串的第一个字符在主串中的位置。Eg: '11 Pro'在 T 中的位置为8(注意：位序从1开始而不是从0开始)

串是一种特殊的线性表，数据元素之间呈线性关系

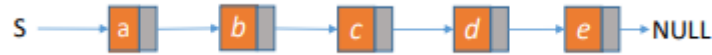
4.1.2、串的存储结构

顺序存储

```
1 #define MAXLEN 255          // 预定义最大串长为255
2 typedef struct{
3     char ch[MAXLEN];        // 每个分量存储一个字符
4     int length;              // 串的实际长度
5 }SString;                    // 静态数组实现
6
7 typedef struct {
8     char *ch;                // 按串长分配存储区
9     int length;
10 }HString;                     // 动态分配
11 HString S;
12 S.ch = (char *)malloc(MAXLEN * sizeof(char));
13 S.length = 0;
```

链式存储

```
1 typedef struct StringNode{
2     char ch;
3     struct StringNode * next;
4 }StringNode,* String;
```



```

1 typedef struct StringNode{
2     char ch[4];
3     struct StringNode * next;
4 }StringNode,* String;
  
```



基于顺序存储实现成操作

切一段子串:

```

1  /*
2     功能:用Sub返回主串S的第pos个字符起长度为len的子串
3     参数:主串S,返回的子串Sub,位置pos,长度len
4     返回值:
5  */
6  bool SubString(SString &Sub,SString S,int pos,int len) {
7      if(pos+len-1 > S.length)
8          return false;
9      for(int i = pos; i<pos+len; i++)
10         Sub.ch[i-pos+1] = S.ch[i];
11     Sub.length = len;
12     return true;
13 }
  
```

比较操作:

```

1  /*
2     功能:若S>T, 则返回值>0; 若S=T, 则返回值=0; 若S<T, 则返回值<0。
3     参数:字符串S,字符串T
4     返回值:
5  */
6  int StrCompare(SString S,SString T) {
7      for(int i = 0;i <= S.length && i <= T.length; i++){
8          if(S.chp[i] != T.ch[i])
9              return S.ch[i]-T.ch[i];
10     }
11     // 扫描通过的所有字符都相同, 则长度长的串更大
12     return S.length-T.length;
13 }
  
```

定位操作:

```

1  /*
  
```

```

2    功能:定位操作。若主串S中存在与子串T值相同的子串,则返回它在主串S
    中第一次出现的位置;否则返回值为0。
3    参数:字符串S,字符串T。
4    返回值:
5    */
6    int Index(SSString S,SSString T) {
7        int i = 0 ,n = StrLength(S), m=StrLeng(T);
8        SString sub;
9        while(i <= n-m+1) {
10            SubString(sub,S,i,m);
11            if(StrCompare(Sub,T)!=0) ++i;
12            else return i; // 返回子串在主串中的位置
13        }
14        return 0;
15    }

```

4.2. KMP 算法

朴树模式匹配算法

最坏时间复杂度: $O(nm)$

```

1    int StrCompare(SSString S,SSString T) {
2        for(int i = 0;i <= S.length && i <= T.length; i++){
3            if(S.ch[i] != T.ch[i])
4                return S.ch[i]-T.ch[i];
5        }
6        return S.length-T.length;
7    }

```

KMP 算法

```

1    int Index_KMP(SSString S,SSString T,int next[]) {
2        int i = 1,j = 1;
3        while (i<=S.length && j <= T.length) {
4            if(j == 0||S.ch[i]==T.ch[i]) {
5                ++i;
6                ++j;          // 继续比较后续字符
7            }else {
8                j = next[j];  // 模式串向右移动
9                // j=nextval[j]; KMP算法优化,当子串和模式串不匹配时
10            }
11        }
12        if(j > T.length)
13            return i-T.length; // 匹配成功
14        else

```

```
15         return 0;
16     }
```

next 数组的求法

nextval 数组的求法

next数组手算方法：当第j个字符匹配失败，由前 1~j-1 个字符组成的串记为S，
则： next[j]=S的最长相等前后缀长度+1

特别地，next[1]=0

nextval数组的求法：
先算出next数组
先令nextval[1]=0
for (int j=2; j<=T.length; j++) {
 if(T.ch[next[j]]==T.ch[j])
 nextval[j]=nextval[next[j]];
 else
 nextval[j]=next[j];
}

序号j	1	2	3	4	5	6
模式串	a	b	a	b	a	a
next[j]	0	1	1	2	3	4
nextval[j]	0	1	0	1	0	4