3.6 串的逻辑结构

DS/SS

串的基本概念和术语

- 串（String）：由零个或多个字符组成的有限序列。
  S='$a_1a_2\ldots a_n$'
- 串名、串值
- 串长
- 空串、空格串
- 子串：串中任意连续的字符组成的子序列
- 主串：包含子串的串
- 字符在串中的位置、子串在串中的位置
- 串相等

串的操作特点：一般以子串整体为单位

3.6 串的逻辑结构（续）

串的抽象数据类型定义

ADT Sring {

数据对象：D={a_i | a_i属于CharacterSet, i = 1,2, … ,n, n>=0}

数据关系：R1={< a_{i-1}, a_i >| a_{i-1}, a_i属于D, i =2,3, …, n}

基本操作：

StrAssign(&T,chars)

StrCopy (&T,S)

StrEmpty (S)

StrLength(S)

SubString(&Sub,S,pos,len)

初始条件：S存在,

1<=pos<=StrLength(S),

0<=len<=StrLength(S)-pos+1;

操作结果：用Sub返回串S的第pos个字符起长度为len的子串

Index(S,T,pos) ---------- 查找

StrInsert(&S,pos,T) ---------- 插入

初始条件：S存在,1<=pos<=StrLength(S)+1

操作结果：在串S的第pos个字符之前插入串T

StrDelete(&S,pos,len) ---------- 删除

初始条件：S存在,1<=pos<=StrLength(S)-len+1

操作结果：从串S中删除第pos个字符起长度为len 的子串

}ADT String

3.7 串的存储结构

DS/SS

串的顺序存储（一）

```
//----------串的定长顺序存储表示-----------
#define MAXSTRLEN    255   //最大串长度
typedef unsigned char SString[MAXSTRLEN+1]
                  //0号单元存放串的长度
```

串顺序存储的特点：

- 连续存储单元静态分配
- 串操作中的原操作一般为“字符序列的复制”
- 截尾法处理串值序列的上越界

3.7 串的存储结构（续）

串的顺序存储（二）

```
Status SubString(SString &Sub,SString S, int pos, int len){
    if(pos <1 || pos>S[0] || len<0 || len> S[0] – pos +1) return ERROR;
    Sub[1..len] = S[pos..pos+len-1];
    Sub[0] =len;
    return  OK;
} // SubString
```

```
Status Concat(SString &T,SString S1, SString S2){
    …                                    //分三种情况
} // Concat
```

3.7 串的存储结构（续）

串的堆分配存储（一）

```
//----------串的堆存储表示-----------
typedef struct{
        char    *ch;      //按照串长动态分配存储区
        int     length;   //串长
}HString
```

串堆存储的特点：

- 连续存储单元动态分配
- 串操作中的原操作一般为“字符序列的复制”
- 一般不会越界

3.7 串的存储结构（续）

DS/SS

串的堆分配存储（二）

```
Status SubString(HString &Sub,HString S, int pos, int len){
    if(pos <1 || pos>S.length || len<0 || len> S.length – pos +1)
        return ERROR;
    if (Sub.ch) free (Sub.ch);          //释放旧空间
    if (!len) {Sub.ch = NULL;  Sub.length = 0;}          //空子串
        else{                                                    //完整子串
            Sub.ch = (char *) malloc (len*sizeof(char));
            Sub.ch[0..len-1] = S[pos-1..pos+len-2];
            Sub.length = len; }
    return  OK;  } // SubString
```

3.7 串的存储结构（续）

串的块链存储

```
#define CHUNKSIZE    80   //块大小
typedef  struct  Chunk{
          char       ch[CHUNKSIZE ]
          struct Chunk       *next
} Chunk;
```

```
typedef  struct {
          Chunk   *head,    *tail;
          int         curlen; //当前长度
} LString
```

串块链存储的特点：

- 可以动态分配块，但块内仍然顺序
- 存储密度小，因为有指针
- 串一般操作复杂，要访问到块内

3.8 串的模式匹配算法

DS/SS

一般算法

逻辑函数为 Index(S,T,pos)

S = ‘$a_1a_2…a_i…a_n$’　　T = ‘$t_1t_2…t_j…t_m$’

一般而言，m<<n

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，否则从主串的下一个字符起再重新和模式的字符比较之。

3.8 串的模式匹配算法（续）

DS/SS

```
int Index(SString S,SString T, int pos) { //返回子串T在主串S中第pos个
    i = pos;          j = 1;                    //字符之后的位置。若不存在，
    while (i<= s[0] && j<= T[0]) {              //函数值位0
          if (S[i] = = T[j]) { ++i;       ++j; }
          else {i = i-j+2;     j =1; }
    }
    if (j > T[0])   return   i – T[0];
    else  return 0;
} // Index
```

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

模式串T= ‘abcac’

第一趟	主串：	a b a b c a b c a c b a b	//i =3
	模式串：	a b c	//j =3
第二趟	主串：	a b a b c a b c a c b a b	//i =2
	模式串：	a	/j =1
第三趟	主串：	a b a b c a b c a c b a b	//i =7
	模式串：	a b c a c	//j =5
第四趟	主串：	a b a b c a b c a c b a b	//i =4
	模式串：	a	//j =1
第五趟	主串：	a b a b c a b c a c b a b	//i =5
	模式串：	a	//j =1
第六趟	主串：	a b a b c a b c a c b a b	//i =11
	模式串：	a b c a c	//j =6