華中科技大學

数据结构

第4章　字符串/串(string)

主讲教师：祝建华

4.1 串的定义与操作

1.术语

(1)串：由零个或多个字符组成的有限序列。

n个字符C1,C2,...,Cn组成的串记作：

s='C1C2...Cn'　　n>=0

其中:s为串名，　C1C2...Cn为串值　　n为串长

例 PASCA语言:

s1='data1234'　　s2='123*abc'

C语言:

s1="data1234"　　s2="123*abc"

' A' 为字符　　"A"为字符串

%c为字符格式　　%s为字符串格式

(2)空串：不含字符的串/长度为零的串。

PASCA语言：　s=’’

C语言：　　s=″″

(3)空格串：仅含空格字符’　’的串。

例　s1=’Φ’　　s2=’ΦΦ’

　　s1=’　’　　s2=’　‘

(4)子串：串s中任意个连续的字符组成的子序列称为串s的子串。

主串---包含某个子串的串。

例　st=″ABC123A123CD″

　　s1=″ABC″　　s3=″123A″　　s4=″ABCA″

　　s2=″ABC12″　　s5=″ABCE″　　s6=″321CBA″

2.串的基本操作

(1)StrAssign(&T,chars)
将字符串常量chars赋值给T。

（2）StrCopy(&T,S)---S的串值复制到T中。
执行：StrCopy(T,S)；当S的串值为“data”,执行完后，T的串值也为“data”。

(3)StrEmpty(S)----判断字符串S是否为空，是则返回TRUE，不是则返回FALSE。

(4)StrLength(S)----返回字符串S中字符的个数。

(5)StrCompare(S,T)---比较S和T的串值大小
若S<T,返回负整数　　如："aBC"<"abc"
若S=T,返回0如：　　如："abc"="abc"
若S>T,返回正整数　　如："ABCD">"ABC "

(6) ClearString(&S)---将字符串S设置成空串。

(7) Concat(&T,S1，S2)----将S1和S2串值连接成的值赋给T。
设 S1的串值为："ABE123*DE123bcd"
S2的串值为："E123"
运算完后T的串值为："ABE123*DE123bcdE123 "，S1与S2不变。

(8)SubString(&sub,S,pos,len)----取子串操作。

其中：1≤pos≤StrLength(S) &&

0≤len≤StrLength(S)-pos+1

例：当S的串值为"datastructure 2013",pos=5，len=9。操作后sub的值为"structure"。

(9)Index(S,T,pos)----子串判断操作。

其中：1≤pos≤StrLength(S)

判断字符串T在主串S的的区间pos---StrLength(S)中是否连续出现，是则返回第一次出现的位置，否则返回0。

例：当S的串值为“Stack and Queue”,pos=10，T的串值为“ue”。操作后的返回值为12。

当pos=10，T的串值为“and”。操作后的返回值为0。

(10)Replace(&S,T,V)——置换操作

用V代替主串S中出现的所有与T相等的不重叠的子串

例1：设主串S的串值：“abc123abc*ABC”，T：“abc”，V：“OK”。

操作后S的串值：“OK123OK*ABC”。

例2：当S：“abcaaaaaABC”，T：“aa”，V：″aaOK″

操作后S的串值：“abcaaOKaaOKaABC“。

(11) StrInsert(&S,pos,T)----将T插入S中第pos个字符之前。

其中：1≤pos≤StrLength(S)+1

(12) StrDelete(&S,pos,len)---删除操作。

其中：　1≤pos≤StrLength(S)-len+1

删除S从第pos个字符开始，长度为len的子串。

(13)DestroyString(&S)---销毁串。

(14)其它操作应用：

例1：用Replace(s,t,v)实现删除

设　　s="ABC//123"

执行：Replace(s,"//","")

有：　　s="ABC123"

例2：用Replace(s,t,v)实现插入

设　　s="ABC123"

执行：Replace(s,"123","**123")

有：　　s="ABC**123"

4.2 串的存储表示和实现

4.2.1 串的定长顺序存储表示

给每个定义的串分配一个固定长度的存储区

```
#define   MAXSTRLEN    255    //用户可在255以内定义最大长度
typedef unsigned char SString[MAXSTRLEN+1]; //0号单元存放
                                            //串的长度
```

PASCAL:　下标为0的分量存放串的实际长度

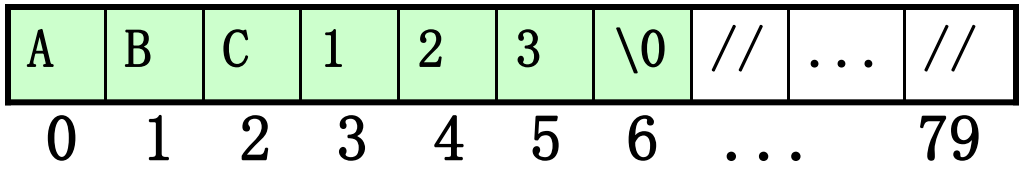
6	A	B	C	1	2	3	//	...	//

C:　在串值后加串结束标记‘\0’，串长为隐含值

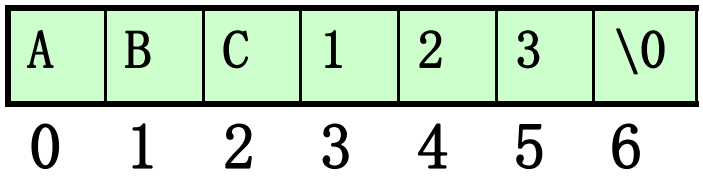
A	B	C	1	2	3	\0	//	...	//

1. 顺序存储----用一维字符数组表示一个串的值。

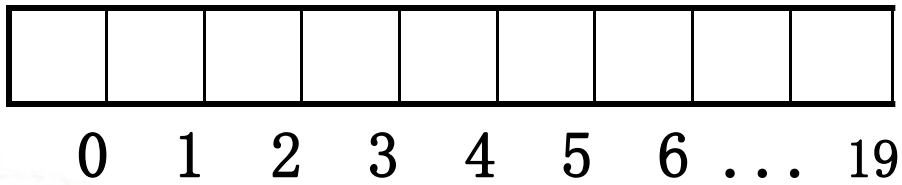
例1 char st1[80]=”ABC123”；

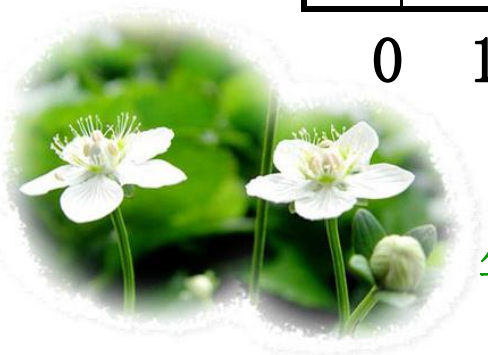


例2 char st2[]=”ABC123”；



例3 char st[20];





2. 串运算实现举例

例 联接运算：给定串s1,s2,将s1,s2联接为串t,记作：

Concat(t,s1,s2)

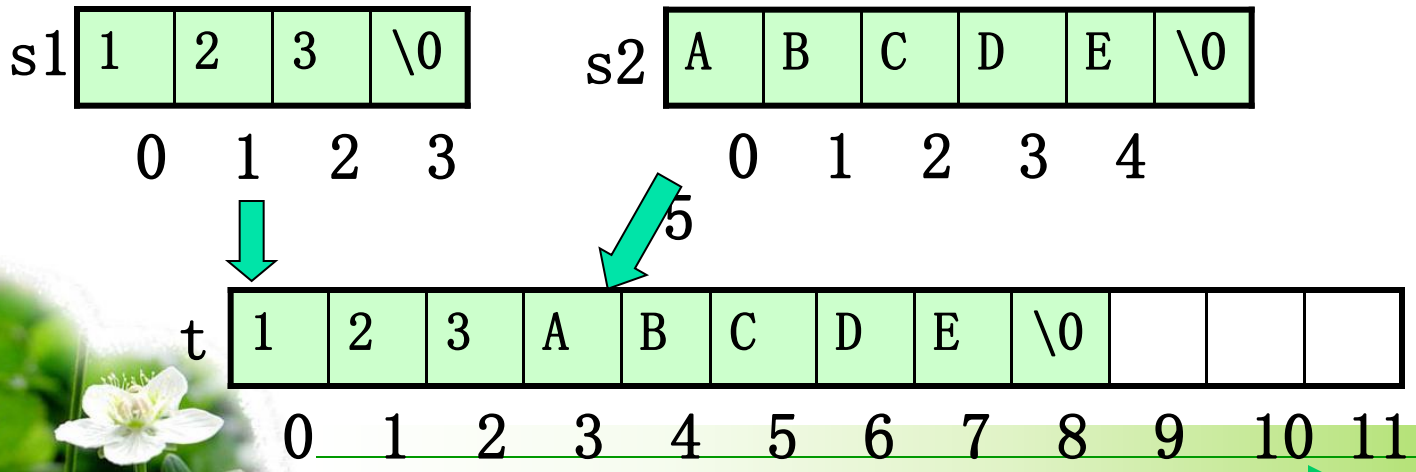
设 s1=”123”， s2=”ABCDE”

执行： Concat(t,s1,s2)；

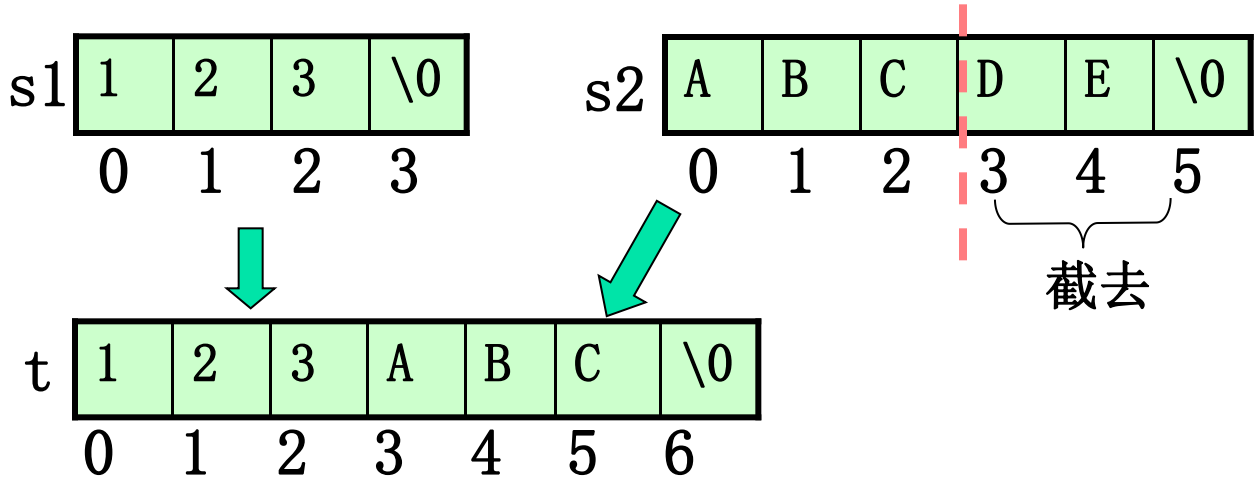
有： t=”123ABCDE”

实现Concat(t,s1,s2)的方法：

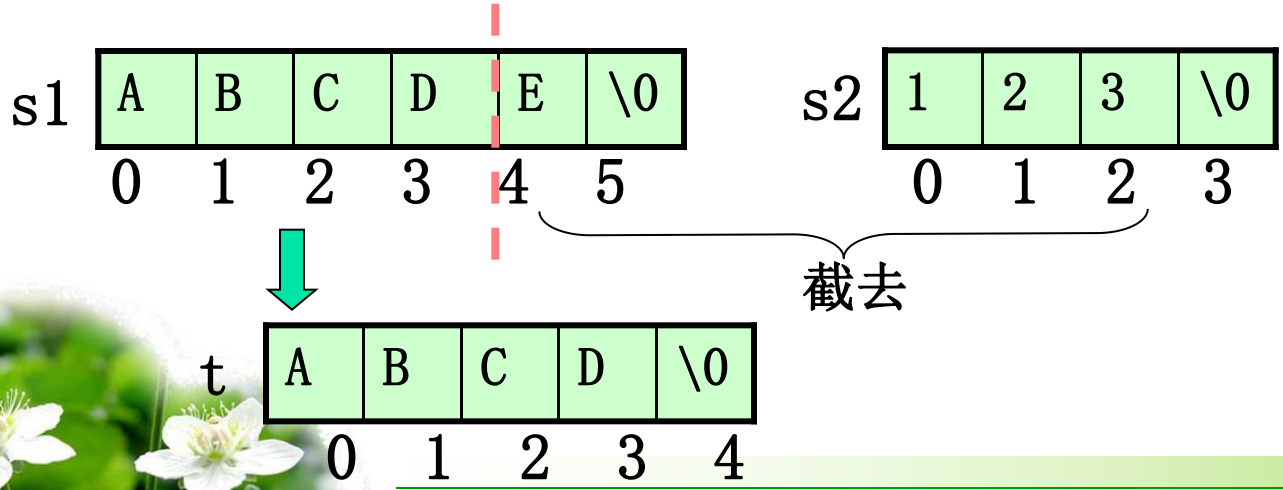
(1)s1的长度+s2的长度≤t的最大长度：



(2)s1的长度≤t的最大长度≤s1的长度+s2的长度：

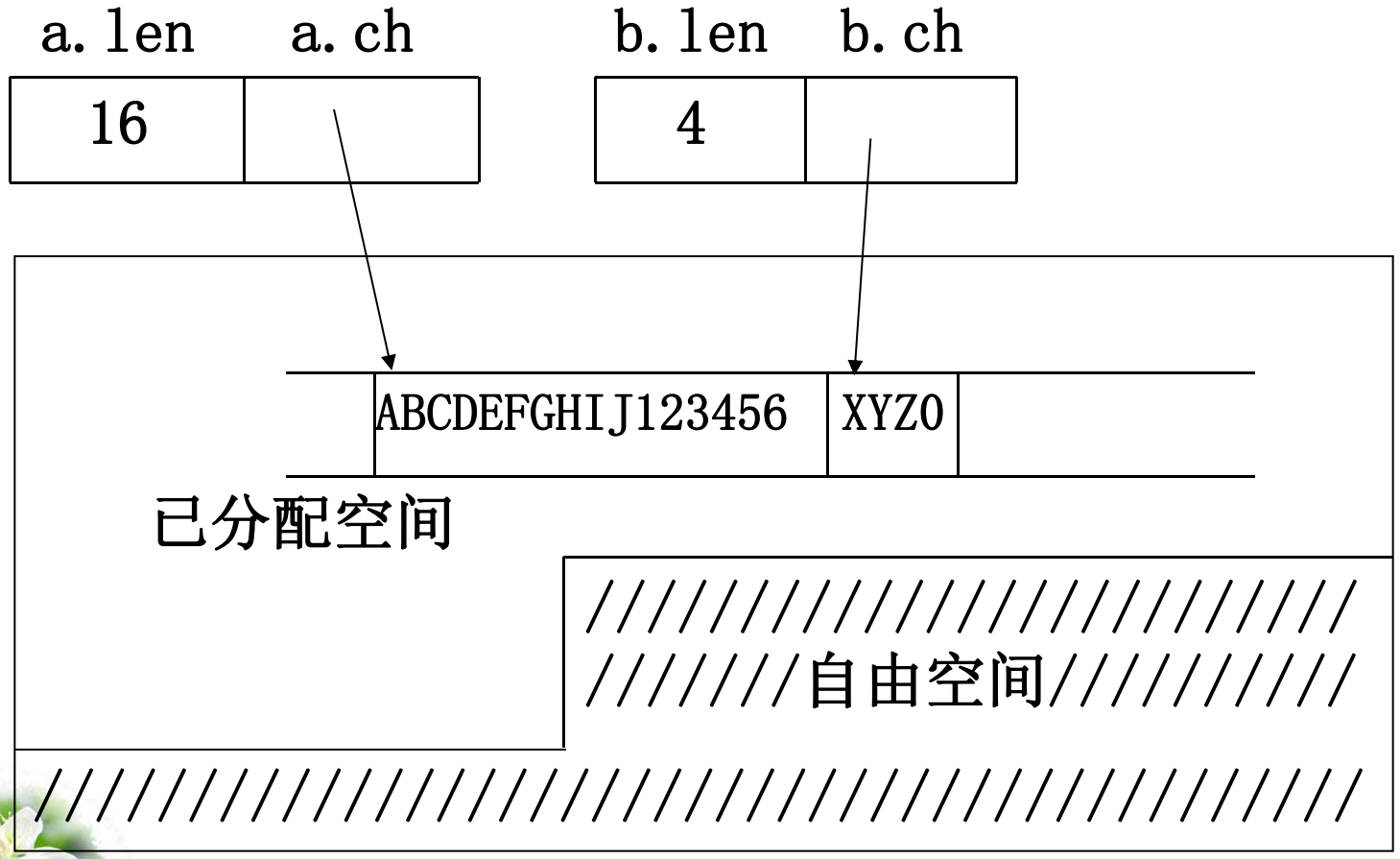


(3) t的最大长度<s1的长度：



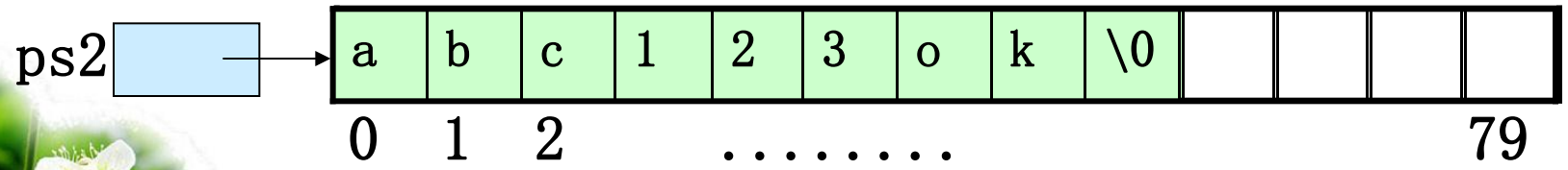
4.2.2 串的堆分配存储表示

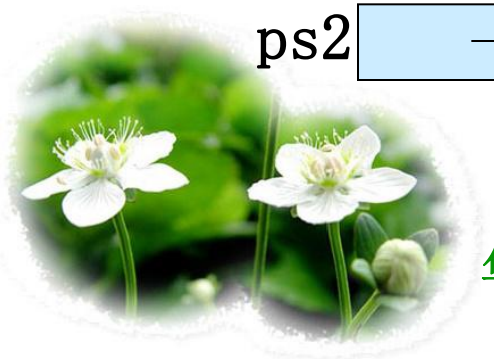
提供一个足够大的连续存储空间，存放字符串的值。



例1：C语言中利用动态分配使用系统堆。

```
{ char *ps1,*ps2; int len;
  scanf(”%d”,&len);                  //输入长度值
  ps1=(char *)malloc(len);           //ps1指向分配的存储空间
  gets(ps1);  puts(ps1);             //输入一个串，再输出
  ps2=(char *)malloc(80);            //ps2指向分配的存储空间
  strcpy(ps2,”abc123ok”) ;           //赋值，再输出
  puts(ps2);
  free(ps1); free(ps2);              //释放存储空间
 }
```

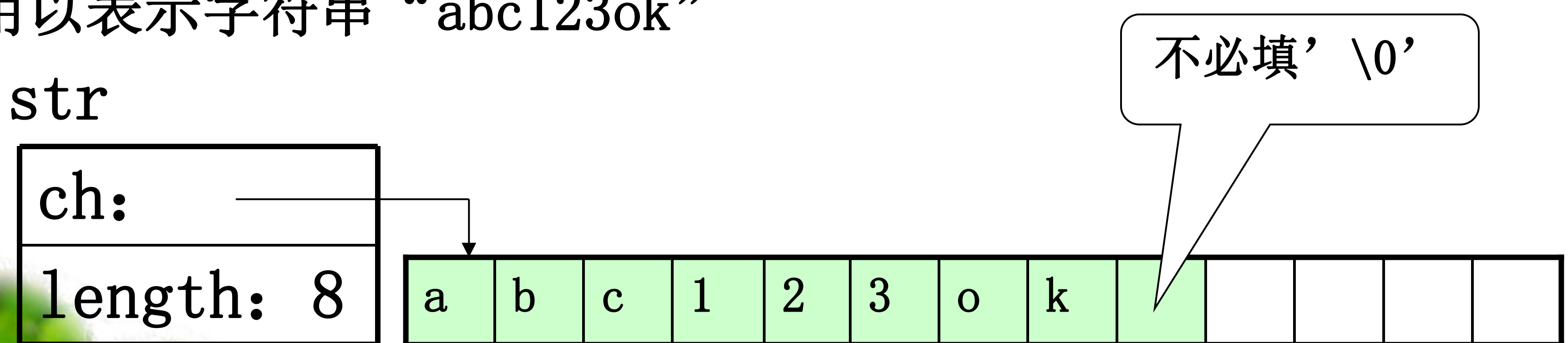




堆存储结构描述，定义将串长作为存储结构的一部分：

```
typedef  struct {
    char   *ch;      //若是非空串，则按串长
                     //分配存储区，否则ch为NULL
    int    length; //串长度
    }  HString;
HString  str;
```

用以表示字符串“abc123ok”



例2：字符串赋值操作

```
int StrAssign( HString *T,  char *chars)
{  char *c;  int i;
   if (T->ch) free(T->ch);              //释放T原有空间
   for(i=0,c=chars; *c; i++,++c); //求chars的串长i
   if (!i)
      {T->ch=NULL;T->length=0;}    //当chars为空串时
   else {
        if (!(T->ch=(char *) malloc(i*sizeof(char))))
            return OVERFLOW;
        T->length=i;
        for(; i>0; i--)                 //复制chars串值到串T
            T->ch[i-1]=chars[i-1];
         }
       return OK;
}
```

例3：输出字符串

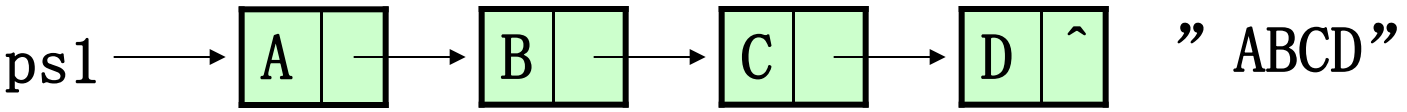
```
void StrPrint(HString T)
{
 int i;
 for(i=0;i<T.length;i++)
     putchar(T.ch[i]);
 }

void  main(void)
{
  HString str;
  str.ch=NULL;str.length=0;
  StrAssign(&str,” abcd123”);
  StrPrint(str);
}
```

4.2.3 串的单链表表示

例1 一个结点只放1个字符

```
struct node1
 { char data;                  //为一个字符
   struct node1 *next;         //为指针
  }*ps1;
```



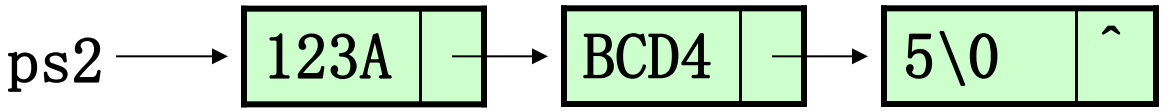
”ABCD”

存储密度=串值所占存储位/实际分配存储位

存储密度为 0.33

例2 一个结点放4个字符

```
struct node4
{ char data[4];          //为4个字符的串
  struct node4 *next;    //为指针
 }*ps2;
```

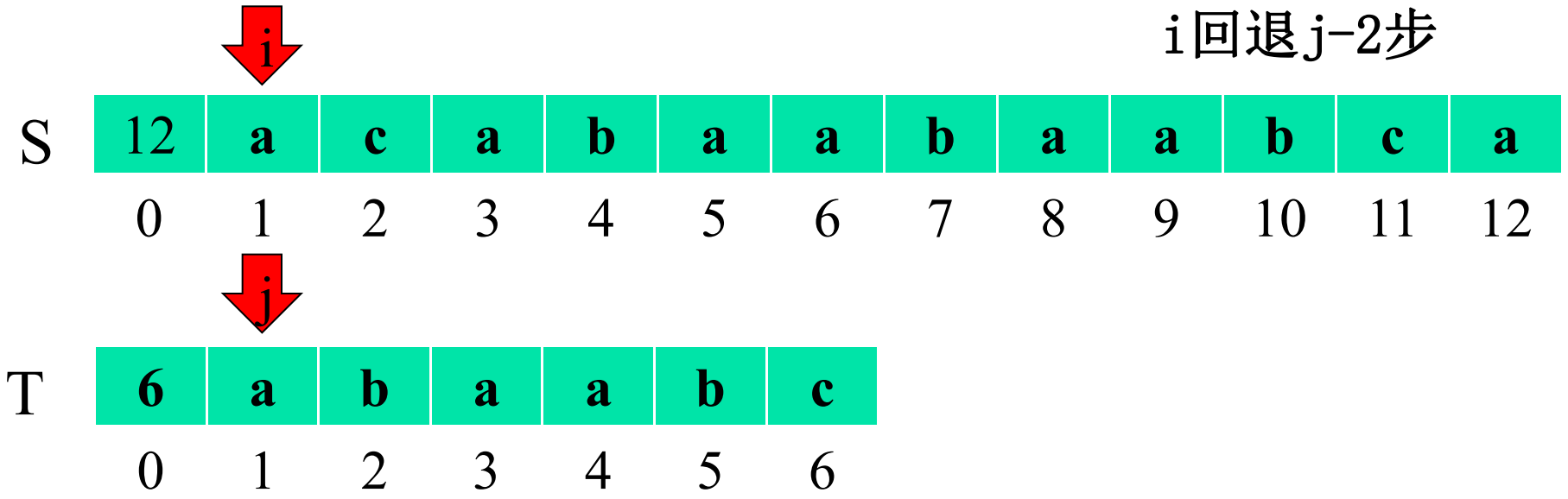


”123ABCD45”

存储密度为 0.67

4.3 串的匹配算法

4.3.1 求子串位置的定位函数index（S,T,pos）



4.3 串的匹配算法

4.3.1 求子串位置的定位函数index（S,T,pos）

匹配起点

得到匹配起点i-T[0]

S	12	a	c	a	b	a	a	b	a	a	b	c	a
	0	1	2	3	4	5	6	7	8	9	10	11	12

j

T	6	a	b	a	a	b	c
	0	1	2	3	4	5	6

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，不存在则0
   i=pos;j=1;
   while (i<=S[0] && j<=T[0])  {
      if (S[i]==T[j])  {++i;++j;}
      else {i=i-j+2;j=1;}                  //i-(j-1)+1 回到起点的后一字符
    }
   if (j>T[0]) return i-T[0];
   else          return 0;
}
```

4.3.2 改进算法（KMP算法）

当主串位置i和匹配串位置j失配时，$s_i \neq p_j$,有：

$$s_1 s_2 \cdots\cdots\cdots\cdots\cdots\cdots s_{i-k+1} \cdots s_{i-1} \; s_i \; s_{i+1} \cdots\cdots s_n$$

$$p_1 \cdots p_{k-1} p_k \cdots p_{j-k+1} \cdots p_{j-1} \; p_j \; p_{j+1}$$

当：$p_1 \cdots\cdots p_{k-1} = p_{j-k+1} \cdots p_{j-1}$时

可以不需要回溯i，将匹配串的第k个位置与主串位置i进行继续匹配。

定义：next（j）为匹配串位置j与主串位置i失配时，滑动匹配串，使其匹配串位置next(j)的字符与主串位置i的字符继续匹配。

$$next(j)=\begin{cases}0 & j=1\text{（第1个字符不相等，需要从主串位置i+1开始）}\\ max\{k|\ 1<k<j \text{ 满足} p_1\cdots p_{k-1}=p_{j-k+1}\cdots p_{j-1}\} & \\ 1 & \text{其它 (位置j前的2端没有相等串)}\end{cases}$$

S[i]

a	c	a	b	a	a	b	a	a	b	c	a

T[j]

a	b	a	a	b	c

	a	b	a	a	b	c
next	0	1	1	2	2	3

```
int KMP(Sstring S, Sstring T){
    for(i=1,j=1; i<=S[0]; i++)  {
        while (j>0 && S[i]!=T[j])  j=next[j];
        if ( j==0 || S[i]==T[j])   ++j;  //j=0表示T[0]和S[i]比较
                                         //实际上就是要T[1]和S[i+1]比较
         if (j>T[0])  return i-T[0];
        }
    return 0;  }
```

算法分析（摊销法）：

（1）、内层的while循环次数没有规则，内循环体执行一次都使j的值减小，但j不会小于0。

（2）、外层循环体中，只有++j能对j进行增加1，但总的对j加1的次数不会多于S[0](即 n 次)。

（3）、综合（1）（2）内循环体的执行的总次数小于n次，摊平执行一次。所以算法T(n)=O(n)

根据KMP算法实现的基本操作index

```
int  index(Sstring S, Sstring T, int pos)   {
   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;
}
```

next（j）的计算方法

next（1）=0

其它，由next(j)分析next(j+1)

当：k=next(j)，有

$p_1 \cdots p_{k-1} p_k \cdots\cdots p_{j-k+1} \cdots\cdots p_{j-1} p_j p_{j+1}$

((1))如果：$p_k=p_j$，next(j+1)=next(j)+1=k+1;

((2))否则：k_1=next(k),则有：

$p_1 \cdots p_{k1-1} p_{k1} \cdots p_{k-k1+1} \cdots p_{k-1} p_k \cdots p_{j-k+1} \cdots p_{j-k1+1} \cdots p_{j-1} p_j p_{j+1}$

((1))如果：$p_{k1}=p_j$，next(j+1)=next(k)+1=k_1+1;

((2))否则：k_2=next(k_1),则有…..

最终可能出现的只有2种可能：

((1))存在k_m：$p_{km}=p_j$，　　next(j+1)= k_m +1;

((2))否则：k_m=0, 则有：　next(j+1)= 1=k_m +1

```
void next(SString  T, int next[])
{  int  k, j;
   next[1]=0;  j=1;k=0;    \\ 对每个j，循环开始k存放的是next[j]的值
   while(j<T[0])              \\根据next[j] 计算next[j+1]
        if (k===0  || T[j]==T[k])
                {++j;++k; next[j]=k;}
        else    k=next[k];
}
```

类似KMP匹配算法，看循环中的k的变化，增加部分最多T[0]（即m）次，　k=next[k]最多执行m次，否则会使k值为负。

综合的T(m)=O(m)