字符串 (String)

字符串是 n (≥ 0) 个字符的有限序列，

记作　S：‘$c_1c_2c_3\ldots c_n$’

其中，S 是串名字

‘$c_1c_2c_3\ldots c_n$’是串值

c_i 是串中字符

n 是串的长度。

例如，S = ‘Tsinghua University’

字符串与线性表的区别与联系？

一些概念

- 字符串在c数组中的表示和初始化
  - 通过字符初始化
  - 通过字符串初始化
- 空串：含零个字符的串
- 空格串
- 字符串中的数据元素是字符
- 字符串的长度：所含字符的个数

一些概念

- 两个字符串相等：充分必要条件是两串的长度相等且两串中对应的字符也相等
- 子串：一个字符串中任意个连续的字符组成的子序列称为该串的子串
  - 子串个数计算
- 主串：包含子串的串

串：基本概念

- **两字符串相等**
  - ✓ S[0,n) = T[0,m) 意味着m=n且S[i]=T[i]
- **子串**: S.substr(i,k)= “$a_i\, a_{i+1} \ldots a_{i+k-1}$” =S[i, i+k),
  - ✓ S[i]起的连续k个字符

- **前缀**: S.prefix(k) = S.substr(0,k)=S[0, k)
  - ✓ S中最靠前的k个字符

- **后缀**: S.suffix(k) = S.substr(n-k,k)= S[n-k, n)
  - ✓ S中最靠后的k个字符

- **联系**: S. substr(i,k) = S.prefix(i+k).suffix(k)= S[i, i+k)
  - ✓ 空串是任何串的子串、前缀、后缀，任何串也是自身的子串、前缀、后缀

字符串抽象数据类型定义

ADT String {

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

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

基本操作:

StrAssign(&T,chars)生成字符串T

StrCopy(&T,S)复制字符串S

StrEmpty(S)判断S是否为空串

StrCompare(S,T)比较字符串S和T

StrLength(S)求字符串S长度

ClearString(&S)清空字符串S

Concat(&T,S1,S2)连接字符串S1和S2

SubString(&Sub,S,pos,len)求S长度为len位置为pos的子串

Index(S,T,pos)求子串在主串中的位置

Replace(&S,T,V)在S中用子串V替换子串T

StrInsert(&S,pos,T)在S中插入子串T

StrDelete(&S,pos,len)在S中删除长度为len的子串

DestroyString(&S)销毁串S

}

求子串在主串中的位置

```
int Index ( String S, String T, int pos ) {
    if (pos > 0) {
        n=StrLength(S); m=StrLength(T); i=pos;
        while (i<=n-m+1) {
            SubString (sub, S, i, m); //提取子串
            if (StrCompare(sub,T)!=0) ++i;
            else return i;
        }//while
    }//if
    return 0;
}//Index
```

练习

含有n个不同字符的字符串的
非空子串个数?
子串个数?
非空真子串个数?

串的存储与表示

- 定长顺序存储表示
  - 第一个元素用于存储字符串长度
  - C语言中从下标0开始存储第一个元素，以’\0’来标识字符串结束
- 堆分配存储表示
- 块链存储表示

串的表示和实现

1.定长顺序存储表示

用一组地址连续的存储单元存储串值的字符序列。

```
#define MAXSTRLEN 255
typedef unsigned char
    SString[MAXSTRLEN+1];//0号单元存串长
```

(1)串连接

```
Status Concat(SString &T,SString S1,SString S2) {
//如果连接后的串过长则截断
    if (S1[0]+S2[0]<=MAXSTRLEN) {
        T[1..S1[0]]=S1[1..S1[0]];
        T[S1[0]+1..S1[0]+S2[0]]=S2[1..S2[0]];
        T[0]=S1[0]+S2[0]; uncut=TRUE;
    }
    else if (S1[0]<MAXSTRLEN) {
        T[1..S1[0]]=S1[1..S1[0]];
        T[S1[0]+1..MAXSTRLEN]=S2[1..MAXSTRLEN-S1[0]];
        T[0]=MAXSTRLEN; uncut=FALSE;
    }
    else {
        T[1..MAXSTRLEN]=S1[1..MAXSTRLEN];
        T[0]=MAXSTRLEN; uncut=FALSE;
    }
    return uncut;
}//Concat
```

(2)求子串

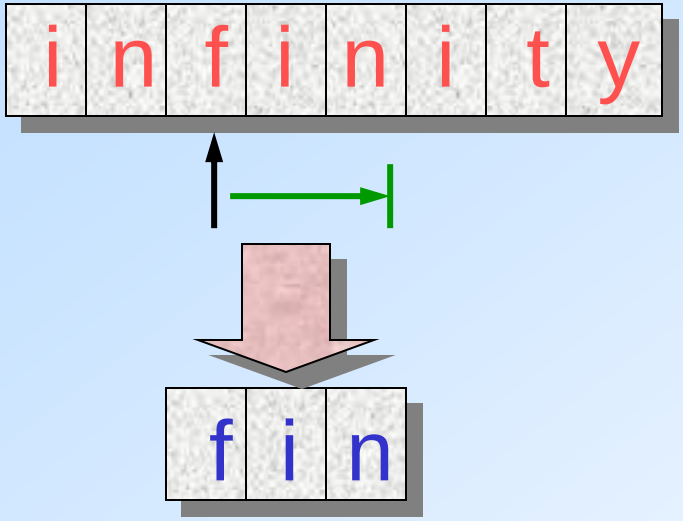
```
Status SubString (SString &Sub, SString S, int
    pos, int len){
    if (pos<1 || pos>S[0] || len<0 || len>S[0]-pos+1)
        return ERROR;
    Sub[1..len]=S[pos.. pos+len-1];
    Sub[0]=len; return OK;
}//SubString
```

见下页

使用顺序存储结构过程中可能出现串长度超过数组上限，经过截断的串已经不完整，克服这个问题可使用动态分配串值的存储空间。

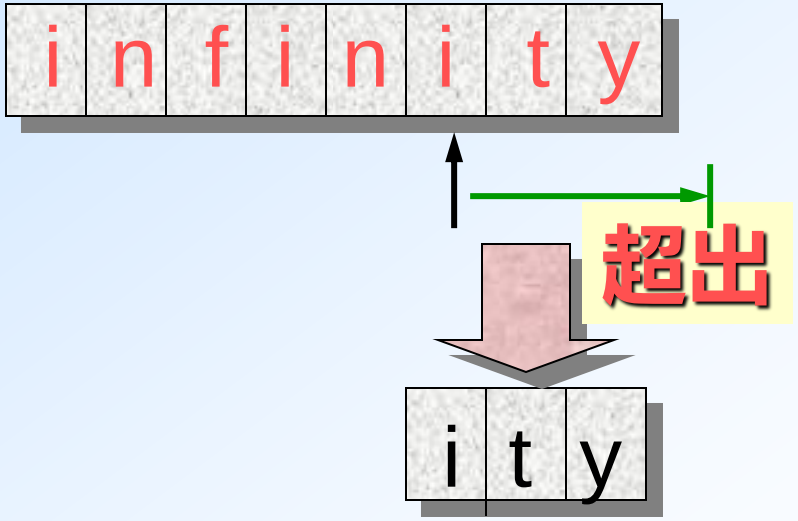
提取子串的算法示例

pos = 2, len = 3



pos+len −1
≤ length−1

pos = 5, len = 4



pos+len −1
≥ length
即len>length-pos+1

2 堆分配存储表示

堆是操作系统中为进程分配的自由存储空间，在C语言中用malloc()和free()来管理。

```
typedef struct {
        char *ch;
        int length;
}HString;
```

```
Status StrInsert(HString &S, int pos, HString T){
//在串S的第pos个字符之前插入串T，pos与实际存储位置差1
    if (pos<1||pos>S.length+1) return ERROR;
    if(T.length){
      if(!(S.ch=(char
    *)realloc(S.ch,(S.length+T.length)*sizeof(char))))
            exit(OVERFLOW);
      for (i=S.length-1;i>=pos-1;--i)
            S.ch[i+T.length]=S.ch[i];  //后移
      S.ch[pos-1..pos+T.length-2]=T.ch[0..T.length-1];
      S.length+=T.length;
    }
    return OK;
}//StrInsert
```

```
Status StrAssign(HString &T,char *chars){
//生成值为chars的串T
    if(T.ch) free(T.ch);
    for(i=0, c=chars;c;++i,++c);//求chars长度i
    if(!i) {T.ch=NULL; T.length=0;}  //空串
    else  {
          if(!(T.ch=(char *)malloc(i*sizeof(char))))
                exit (OVERFLOW);
          T.ch[0..i-1]=chars[0..i-1];
          T.length=i;
    }
    return OK;
}//StrAssign
```

```
int StrLength(HString S) {
//求串长度
    return S.length;
}//StrLength

int StrCompare(HString S, HString T){
//比较串S和T
    for (i=0;i<S.length && i<T.length;++i)
        if(S.ch[i]!=T.ch[i]) return S.ch[i]-T.ch[i];  //看字符大小
    return S.length-T.length;    //看长度大小
}//StrCompare

Status ClearString(HString &S){
//清空串S
    if(S.ch) {free(S.ch); S.ch=NULL;}
    S.length=0;
    return OK;
}//ClearString
```

```
Status Concat(HString &T,HString S1,HString S2)
    {
//连接串S1和S2
    if(T.ch) free(T.ch);  // 清空
    if(!(T.ch=(char   //分配空间
    *)malloc((S1.length+S2.length)*sizeof(char))))
          exit(OVERFLOW);
    T.ch[0..S1.length-1]=S1.ch[0..S1.length-1]; //赋值
    T.length=S1.length+S2.length;
    T.ch[S1.length..T.length-1]=S2.ch[0..S2.length-1];
    return OK;
}//Concat
```

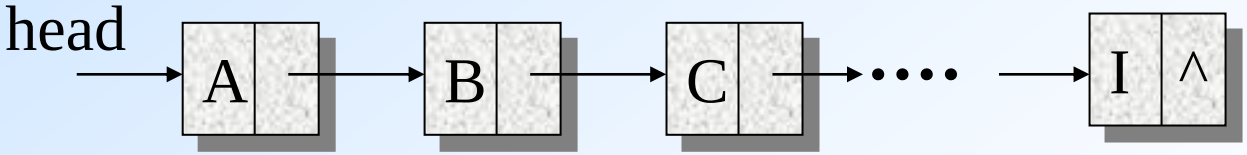
```
Status SubString(HString &Sub,HString S,int pos,int
    len){
//返回串S中从pos位置起长度为len的子串
    if (pos<1||pos>S.length||len<0||len>S.length-pos+1)
        return ERROR;
    if(Sub.ch) free(Sub.ch);  //清空
    if(!len){Sub.ch=NULL; Sub.length=0;}
    else{
        Sub.ch=(char *)malloc(len*sizeof(char));
        Sub.ch[0..len-1]=S[pos-1..pos+len-2];
        Sub.length=len;
    }
    return OK;
}//SubString
```

3.串的块链存储表示

用链表存储串，每个结点存储串的n个字符，当串长不是n的整数倍时最后一个结点剩余位置用空字符 # 补齐。

如串ABCDEFGHI当n=4时：

当n=1时：



串的块链存储表示

```
#define CHUNKSIZE 80
typedef struct Chunk{
    char ch[CHUNKSIZE];
    struct Chunk *next;
}Chunk;
typedef struct {
    Chunk *head, *tail;     //为指针便于连接操作
    int curlen;
}LString;
```

块链存储方式便于连接操作，但占用存储量大，操作复杂。

练习

实现字符串拷贝的函数 strcpy 为：

```
void strcpy(char *s , char *t)   /*copy t to s*/
{
 while (_______)
}
```

练习

实现字符串拷贝的函数 strcpy 为：

```
void strcpy(char *s , char *t)   /*copy t to
   s*/
{
 while (*s++=*t++！ =’\0’)
}
```

while(*s++=*t++); 在语义上等同于

```
while((*t) != 0){
      *s = *t;
      s++; t++;
}
```

练习

求采用顺序结构存储的串s和串t的一个最长公共子串

求采用顺序结构存储的串s和串t的一个最长公共子串

```
void  maxcomstr(orderstring *s,*t, int& index, int& length){
  int i,j,k,length1,con;
  index=0;length=0;i=1;
  while (i<=s.len){
      j=1;
      while(j<=t.len){
            if (s[i]= =t[j]) { //如果发现字符相等，求出子串长度
                        k=1;length1=1;con=1;
                         while(con){
                            if (1) i+k<=s.len && j+k<=t.len && s[i+k]==t[j+k]
                               {length1=length1+1;k=k+1; }
                            else (2) con=0          __;
                         }
                          if (length1>length) { index=i;  length=length1; } // 更新最长长度
                         (3)_ j+=k __;
            }
            else (4)   j++     ___;
       }
    (5) _i++_
  }
}
```

S=aaabcdef….

T=aaaabcdf

串的模式匹配

定义 在串中寻找子串（第一个字符）在串中的位置

词汇 在模式匹配中，子串称为模式，串称为目标。

示例 目标 T : “Beijing”

模式 P : “jin”

匹配结果 = 3

穷举的模式匹配过程

匹配模式：aba

第1趟

```
        i=1
T   a b b a b a
P   a b a
    j=1
```

第2趟

```
          i=2
T   a b b a b a
P     a b a
      j=1
```

第3趟

```
            i=3
T   a b b a b a
P       a b a
        j=1
```

第4趟

```
              i=4
T   a b b a b a
P         a b a
          j=1
```

√

匹配算法:

```
int Index(SString S, SString T, int pos){
//从串S中pos位置开始搜索模式T
    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从当前位置后移，j复位
    }
    if (j>T[0]) return i-T[0];
    else return 0;
}//Index
```

思考：该模式匹配算法的最少匹配次数和最多匹配次数？应如何改进？

O(n+m)

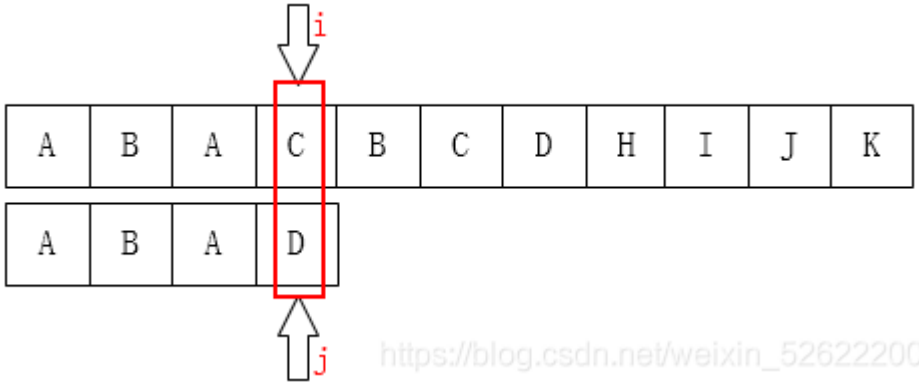
S= '21222310323321'
T= '1111'

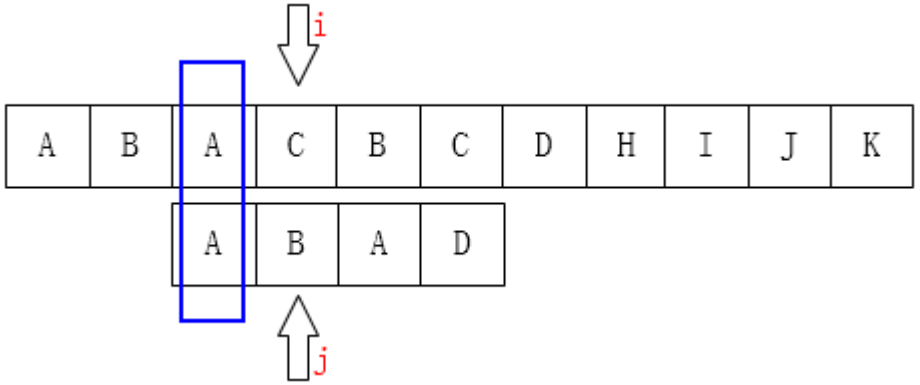
O(n*m)

S= '00000000000001'
T= '0001'

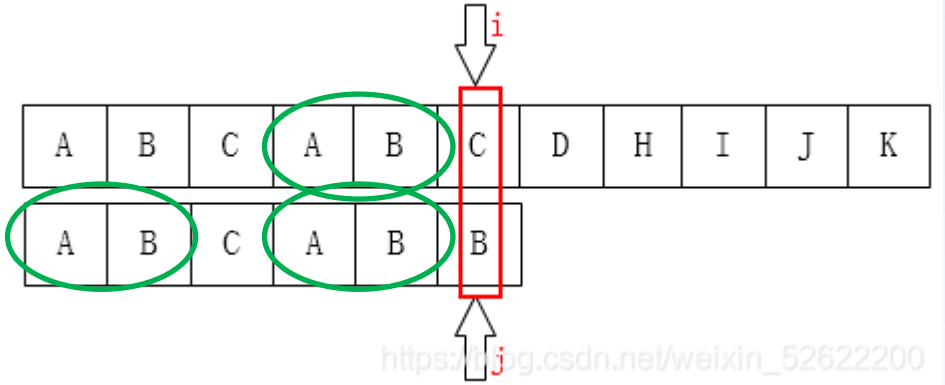
模式匹配的改进算法：KMP算法

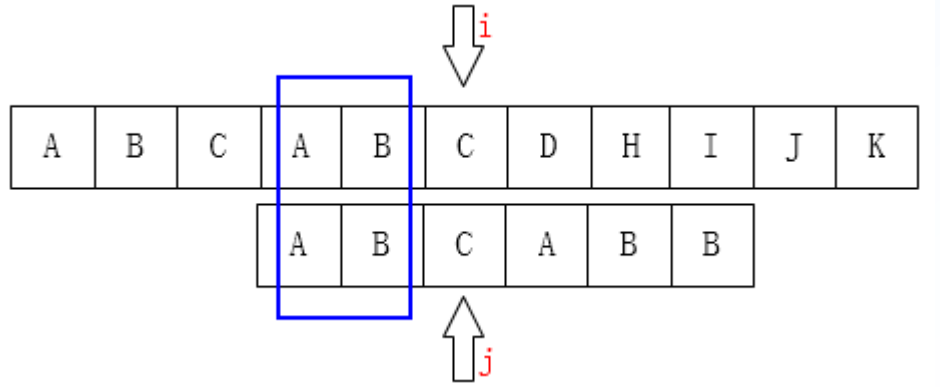
利用已经部分匹配这个有效信息，保持i指针不回溯，通过修改j指针，让模式串尽量地移动到有效的位置。





C和D不匹配了，我们要把j移动到哪？显然是第2位。因为前面有一个A相同

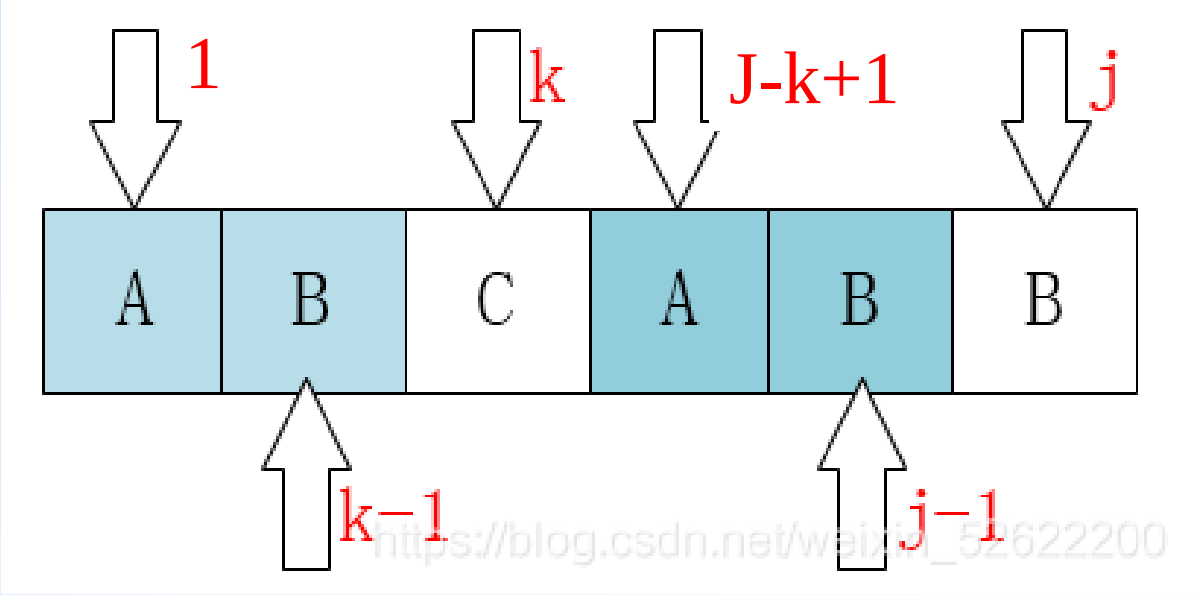




C和B不匹配了，可以把j指针移动到第3位，因为前面有两个字母是一样的

模式匹配的改进算法：KMP算法

当匹配失败时，j要移动的下一个位置k。存在着这样的性质：
最前面的k-1个字符和j之前的最后k-1个字符是一样的。



模式匹配的改进算法：KMP算法

匹配模式abcac

第1趟

i=3

T a b a b c a b c a c b a b

j=3

P a b c

第2趟

i=3 i=7

T a b a b c a b c a c b a b

j=5

P a b c a c

第3趟

i=7

T a b a b c a b c a c b a b

j=2

P a b c a c

√

第1趟　　T　a b a b c a b c a c b a b

P　a b c

在第一趟中i=3,j=3时比较失败，按穷举法i应变为初始值加1，即i=2;j每次重新变为1，再开始下一轮比较，但事实上i=2时已经作过比较不需要再比较一次，所以i值不变，只需j变为1再开始比较即可。

第2趟　　T　a b a b c a b c a c b a b

P　a b c a c

在第二趟中i=7,j=5时比较失败，按穷举法i应变为初始值加1，即i=4;j变为1，再开始下一轮比较，但事实上i=4,5,6时已经作过比较不需要再比较一次，所以i值不变，而j值也不需变为1再开始比较，因为字符c前的’a’在P的开始也有，而且也比较过了，j从2开始即可。

总之，利用已经比较完的结果，可以减少下次比较的次数，不需要移动指针i的位置，只需移动指针j的位置，而且按照模式中的包含关系，指针j也不需每次从1开始。

由模式串的包含关系找到j的移动位置：例

模式串	a	b	a	a	b	c	a	c
当前j	1	2	3	4	5	6	7	8
下一j	0	1	1	2	2	3	1	2

说明：next[j]=0指的是第一个字符比较就不成功，i指针需要后移，j指针变为1；其它值表示i指针不动，j指针移动的位置。

KMP算法：

```
int Index_KMP(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} //继续比较后续字符
        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
```

计算next函数值算法：

```
void get_next(SString T, int next[ ]) {
    i=1; next[1]=0; j=0;   //初始化
    while(i<T[0]) {  //枚举前缀，找最长的前缀a[0~j]（j<i）
        if (j==0 || T[i]==T[j]) {++i; ++j; next[i]=j; }
//找不到匹配的前后缀或找到了最长的前缀
        else j=next[j];  // j 回退，因为前面已经处理好了，
    所以可以顺便利用
    }
}//get_next
```

谢谢！

练习

含有n个不同字符的字符串的
非空子串个数

$$C(n + 1, 2) = n * (n + 1) / 2$$

子串（包括空串）为

$$n * (n + 1) / 2 + 1$$

非空真子子串（不包括空串和跟自己一样的子串）为

$$n * （n + 1） / 2 - 1$$