数据构造

主讲：王强

- 第2章　线性表
- 第3章　栈和队列
- 第4章　串、数组和广义表

线性构造

可表达为：（a_1，　a_2　，……，　　a_n）

补充：C语言中常用的串运算

调用原则库函数 #include<string.h>

串比较，strcmp(char s1,char s2)
串复制，strcpy(char to,char from)
串连接，strcat(char to,char from)
求串长，strlen(char s)
……

第4章　串、数组和广义表

教学内容

4.1　串

4.2　数组

4.3　广义表

教学目的

1. 了解串的存储措施，了解串的两种模式匹配算法，要点掌握**BF算法**。
2. 明确数组和广义表这两种数据构造的特点，掌握**数组地址计算措施**，了解几种特殊矩阵的压缩存储措施。
3. 掌握广义表的定义、性质及其**GetHead和GetTail的操作**。

4.1 串

串(String)-----零个或多种字符构成的有限序列

$$s = 'a_1 a_2 \mathrm{L}\ a_n'$$

串名

串值

串长 → n

空串 ← n=0

a=‘BEI’,
b=‘JING’
c=‘BEIJING’
d=‘BEI
JING’

子串

主串

字符位置

子串位置

串相等

空格串

串的抽象数据类型

ADT String {

数据对象： $D=\{a_i \mid a_i \in CharacterSet, i=1,2,\mathrm{L}\ ,n,\ \mathrm{n}\geq 0\}$

数据关系： $R_1=\{< a_{i-1},a_i > \mid a_{i-1},a_i \in D, i=1,2,\mathrm{L}\ ,n\}$

基本操作：

(1) StrAssign (&T,chars) //串赋值

(2) StrCompare (S,T) //串比较

(3) StrLength (S) //求串长

(4) Concat(&T,S1,S2) //串联

(5) SubString(&Sub,S,pos,len)　//求子串

(6) StrCopy(&T,S)　//串拷贝

(7) StrEmpty(S)　//串判空

(8) ClearString (&S)　//清空串

(9) Index(S,T,pos)　//子串的位置

(11) Replace(&S,T,V)　//串替代

(12) StrInsert(&S,pos,T)　//子串插入

(12) StrDelete(&S,pos,len)　//子串删除

(13) DestroyString(&S)　//串销毁

}ADT String

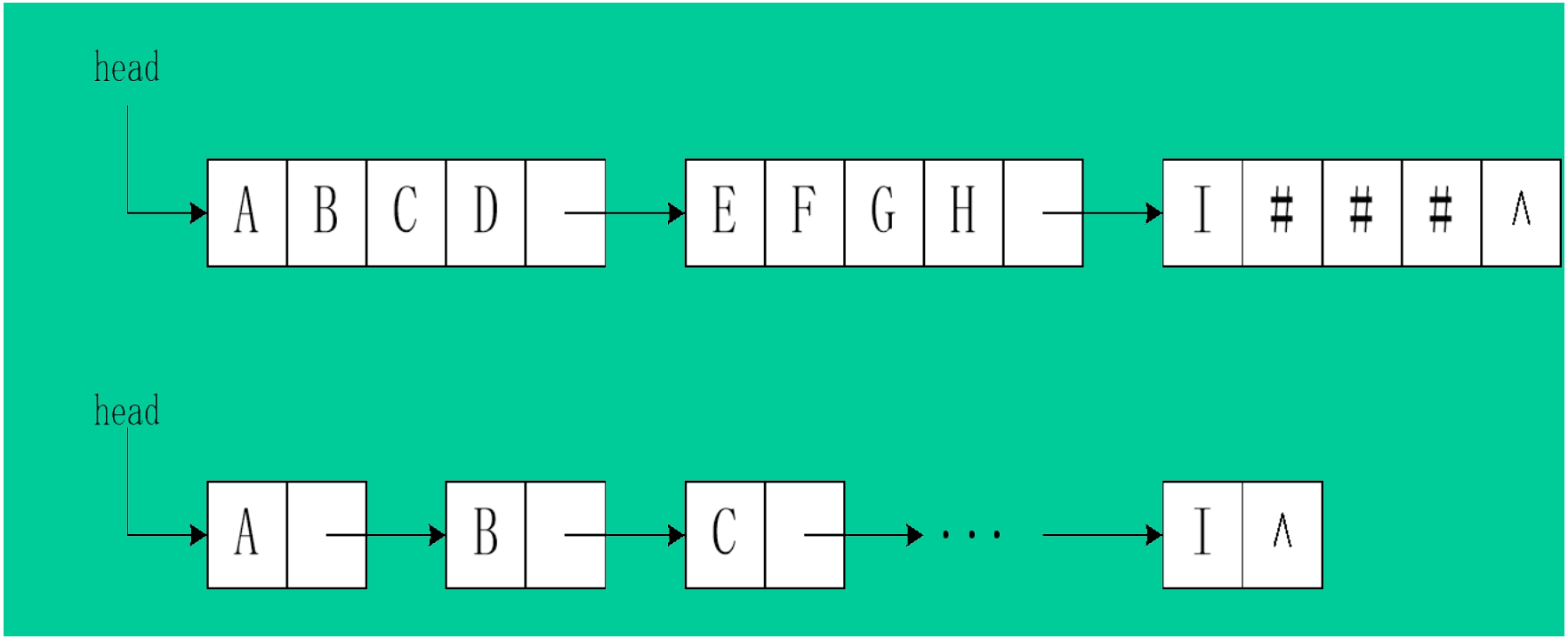
串的存储构造

- 顺序存储
- 链式存储

顺序存储表达

```
typedef struct{
    char *ch;          //若串非空,则按串长分配存储区,
                       //不然ch为NULL
    int  length;       //串长度
}HString;
```

链式存储表达



链式存储表达

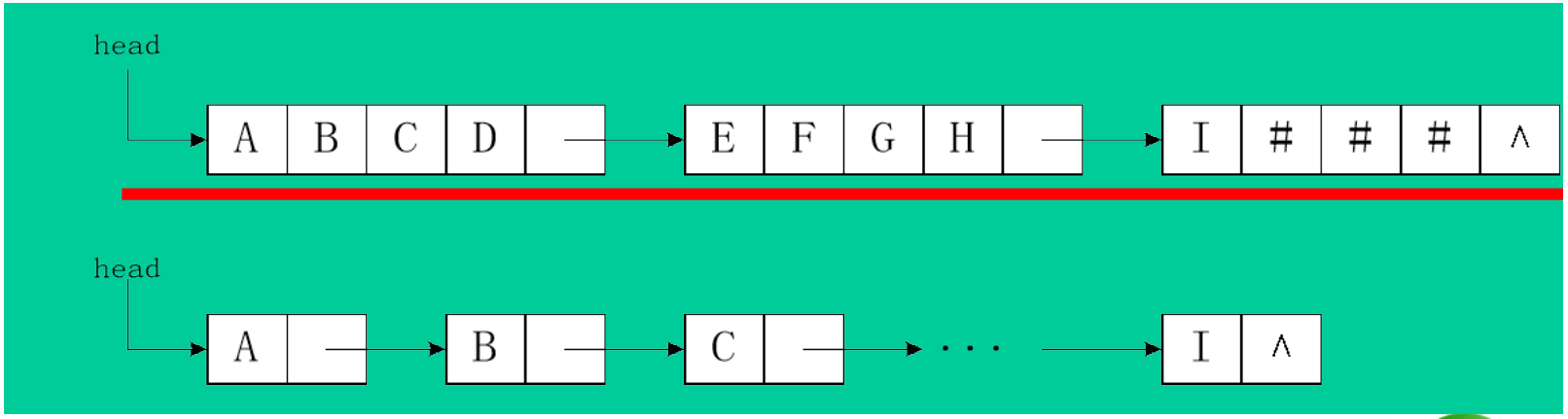
```
#define CHUNKSIZE 80      //可由顾客定义的块大小
typedef struct Chunk{
   char  ch[CHUNKSIZE];
   struct Chunk *next;
}Chunk;

typedef struct{
   Chunk *head,*tail;     //串的头指针和尾指针
   int curlen;            //串的目前长度
}LString;
```

链式存储表达

优点：操作以便
缺陷：存储密度较低

$$\text{存储密度} = \frac{\text{串值所占的存储位}}{\text{实际分配的存储位}}$$

可将多种字符存储在一种结点中，以克服其缺陷



串的模式匹配算法

算法目的：

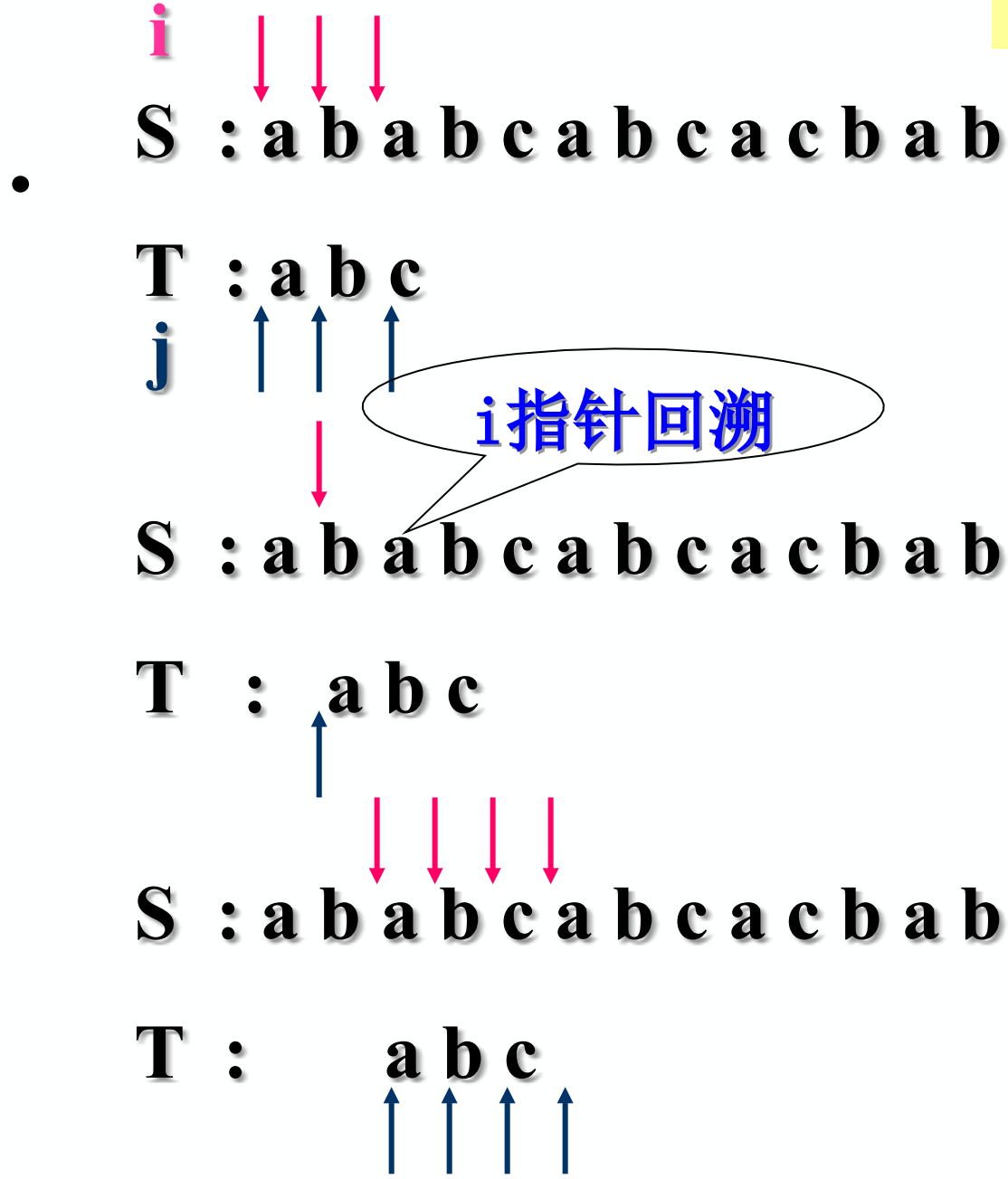
拟定主串中所含子串第一次出现的位置（定位）

即怎样实现教材P72 Index(S,T,pos)函数

算法种类：

- BF算法（又称古典的、经典的、朴素的、穷举的）
- KMP算法（特点：速度快）

BF算法设计思想


i
S :a b a b c a b c a c b a b
T :a b c
j
i指针回溯
S :a b a b c a b c a c b a b
T : a b c
S :a b a b c a b c a c b a b
T : a b c

BF算法设计思想

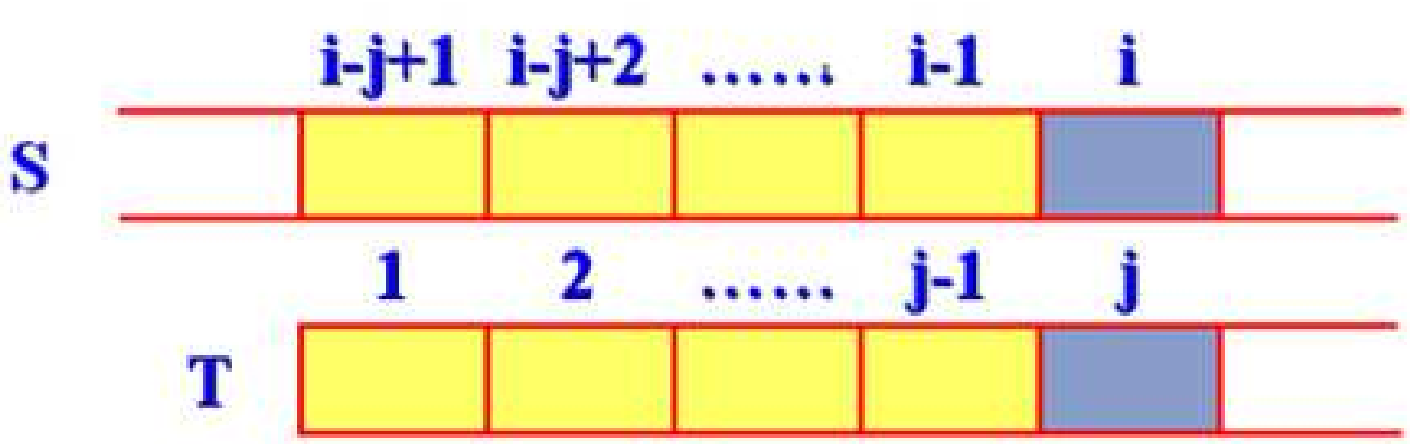
Index(S,T,pos)

- 将主串的第pos个字符和模式的第一种字符比较，
  若相等，继续逐一比较后续字符；
  若不等，从主串的下一字符起，重新与模式的第一种字符比较。

- 直到主串的一种连续子串字符序列与模式相等 。返回值为S中与T匹配的子序列第一种字符的序号，即匹配成功。
- 不然，匹配失败，返回值 0

BF算法描述（算法4.1）

```
int  Index(Sstring S,Sstring T,int pos){
    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; }
   if ( j>T[ 0 ])   return   i－T[0];
   else return 0;
}
```



BF算法时间复杂度

例： S='0000000001'，T='0001'，pos=1

若n为主串长度，m为子串长度，最坏情况是

- ✓主串前面n-m个位置都部分匹配到子串的最终一位，即这n-m位各比较了m次
- ✓最终m位也各比较了1次

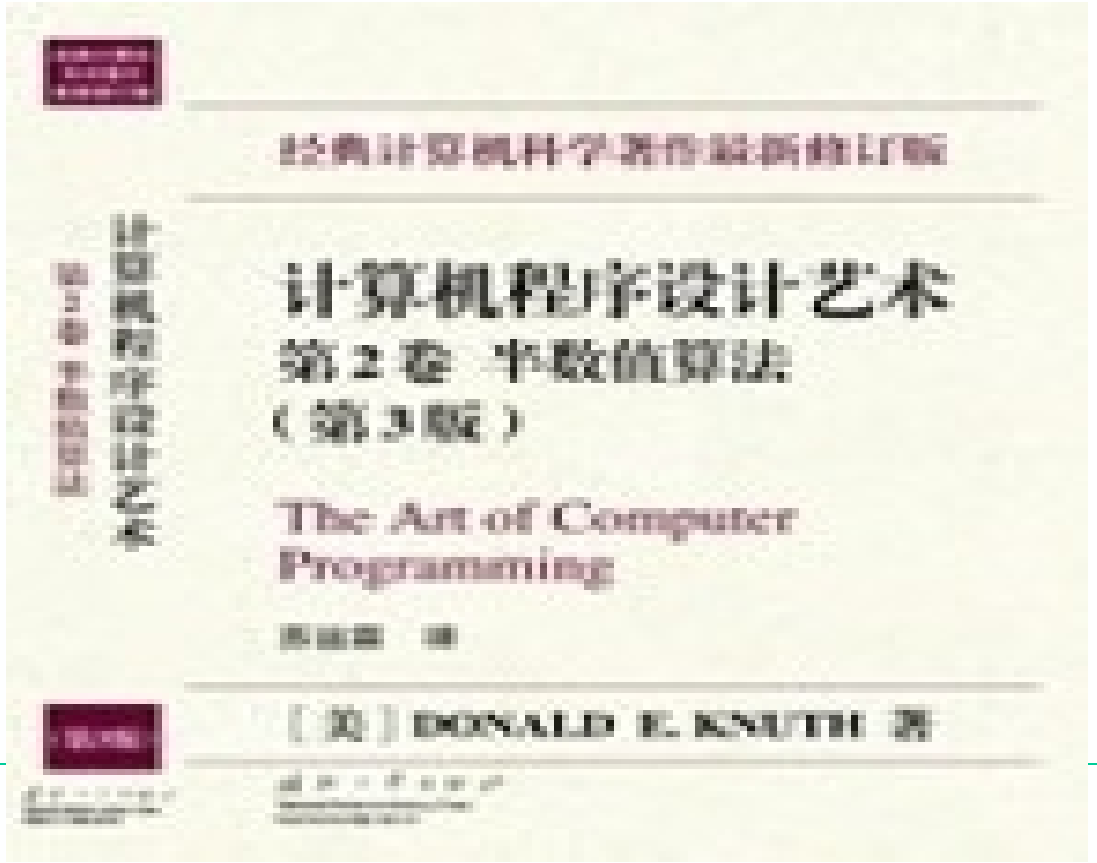
总次数为：(n-m)*m+m＝(n-m+1)*m

若m<<n，则算法复杂度O(n*m)

KMP（Knuth Morris Pratt）算法

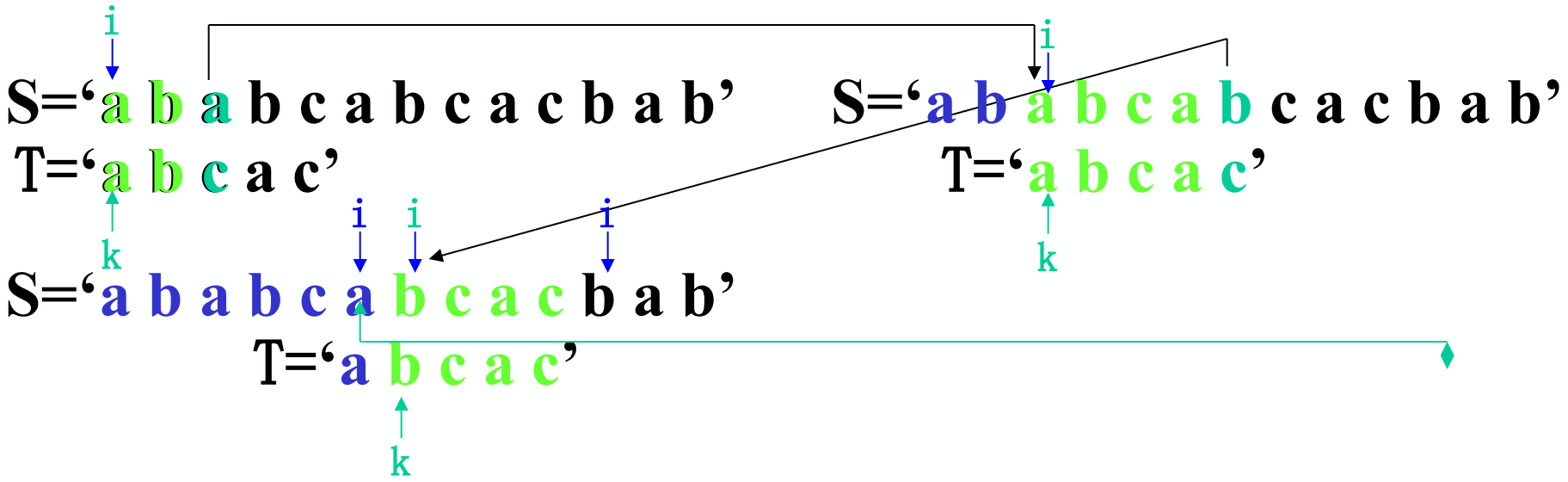
《计算机程序设计艺术 第1卷 基本算法》 98元
《计算机程序设计艺术 第2卷 半数值算法》 98元
《计算机程序设计艺术 第3卷 排序与查找》 98元

http://www-cs-faculty.stanford.edu/~knuth/



KMP算法设计思想（了解）

利用已经部分匹配的成果而加紧模式串的滑动速度？
且主串S的指针i不必回溯！ 可提速到O(n+m)！



串操作应用举例——文本编辑

- 文本可被看作一种字符串，称为文本串
- 页则是文本串的子串
- 行又是页的子串。

页表

页号	起始行号
⋮	

行表

行号	起始地址	长度
⋮		