



# 数据结构

(C语言版) (第2版)

## 串、数组和广义表


### 串的概念及存储结构

**主讲教师：汪红松**





# 教学目标

- 01**  
OPTION 熟悉串的实现和表示，包括顺序存储和链式存储表示
  - 02**  
OPTION 熟悉数组的存储方法
  - 03**  
OPTION 了解特殊矩阵和稀疏矩阵的压缩存储，稀疏矩阵的转置运算
  - 04**  
OPTION 了解广义表的逻辑结构和存储结构
- 

# 教学内容 Contents

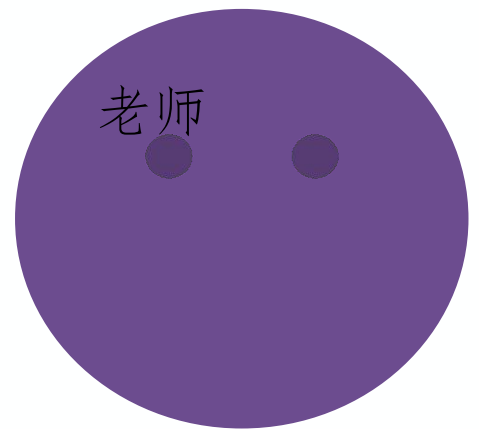
1

串的概念及存储结构

2

数组与广义表

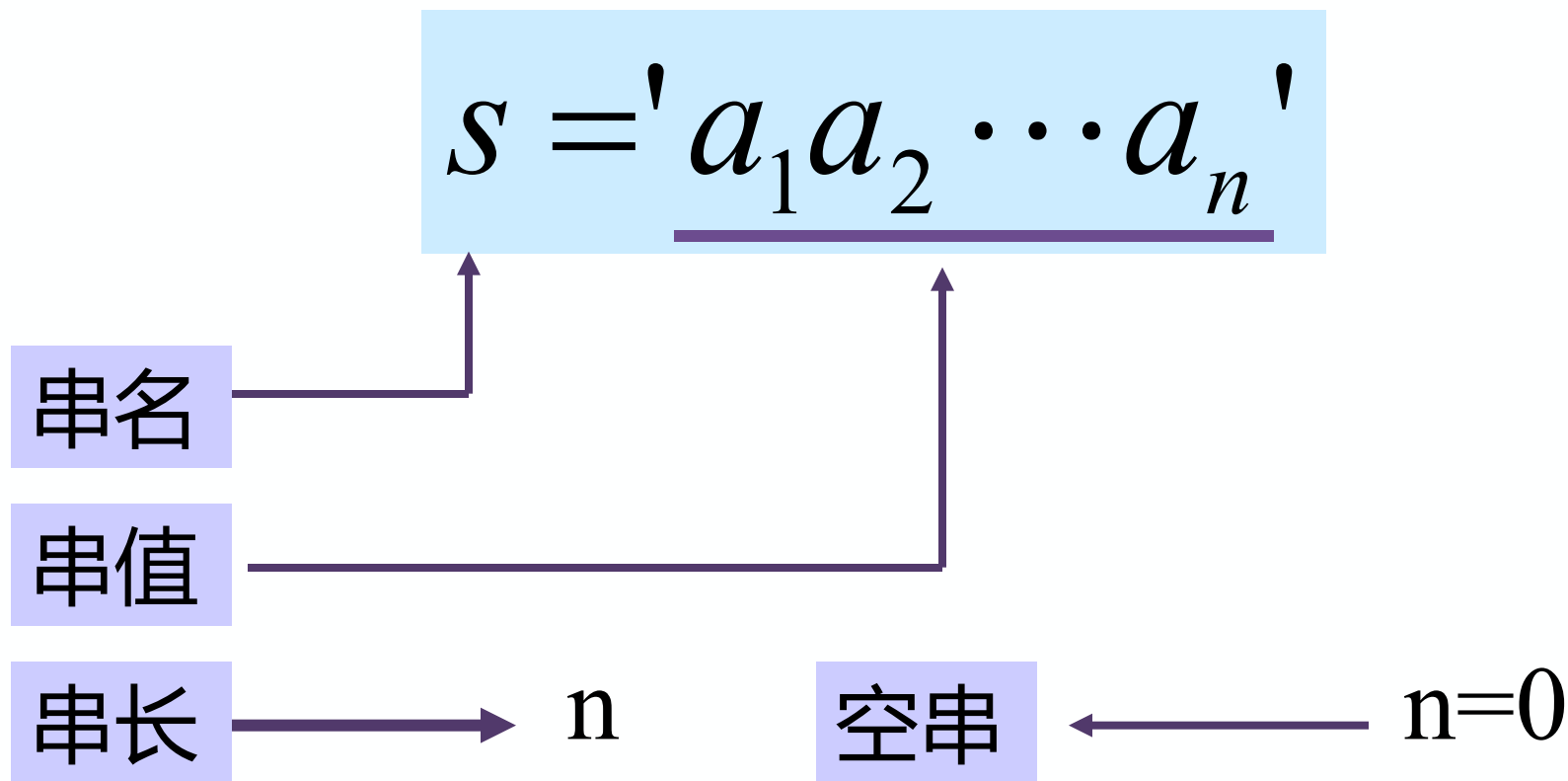
- 一、串的定义
- 二、串的存储结构
- 三、串的模式匹配BF算法



## ▶▶▶ 一、串的定义

串(String)----零个或多个字符组成的有限序列

---



## ▶▶▶ 一、串的定义

a='BEI'Ø,  
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

# ▶▶▶ 一、串的定义

## 1.串的类型定义、存储结构及运算

ADT String {

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

数据关系:  $R_1 = \{ \langle a_{i-1}, a_i \rangle \mid a_{i-1}, a_i \in D, i = 1, 2, \dots, n \}$

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

## ▶▶▶ 一、串的定义

### 1.串的类型定义、存储结构及运算

(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)               //串替换

(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



## ▶▶▶ 二、串的存储结构



顺序存储

The diagram features two large circles. The left circle is dark gray and contains the text '顺序存储' (Sequential Storage). The right circle is purple and contains the text '链式存储' (Linked Storage). Both circles are surrounded by several smaller, semi-transparent circles of varying shades of purple and gray, creating a decorative effect.

链式存储

## 二、串的存储结构

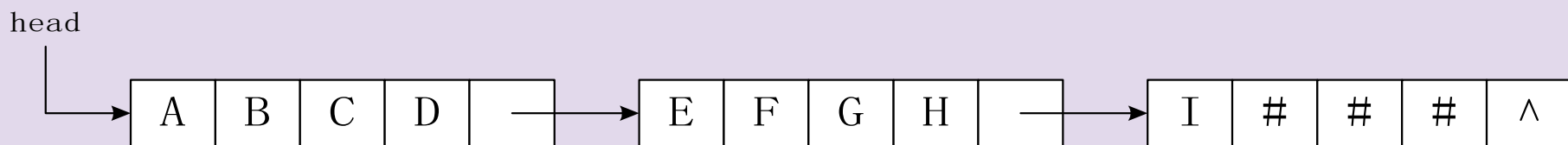
### 1.顺序存储表示

#### 定长顺序存储结构

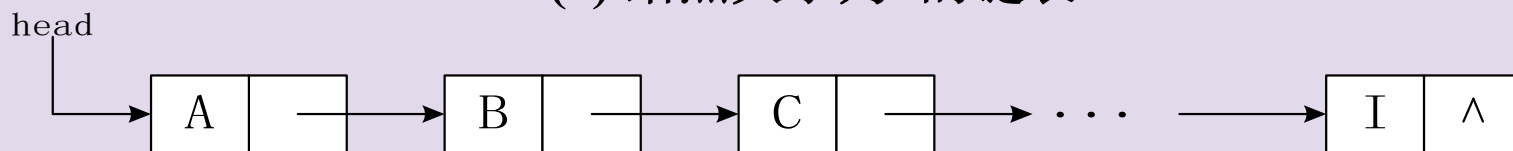
```
#define MAXLEN 255 //用户可在255以内定义最大串长
typedef struct{
    char ch[MAXLEN+1]; //存储串的一维数组
    int length;         //串长度
}SString;
```

#### 堆式顺序存储结构

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



(a) 结点大小为4的链表

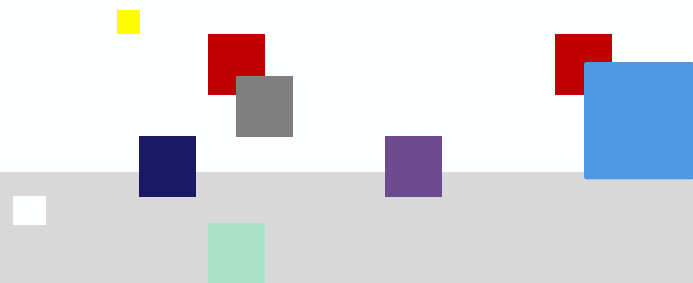


(b) 结点大小为1的链表

```
#define CHUNKSIZE 80      //可由用户定义的块大小

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;           //串的当前长度
}LString;
```



## 二、串的存储结构

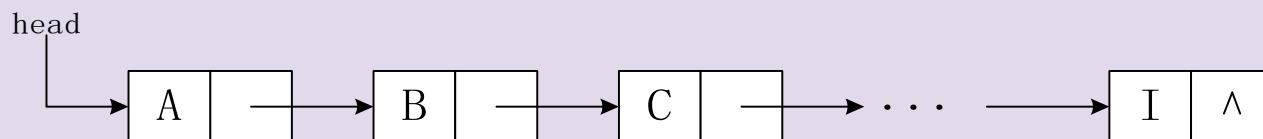
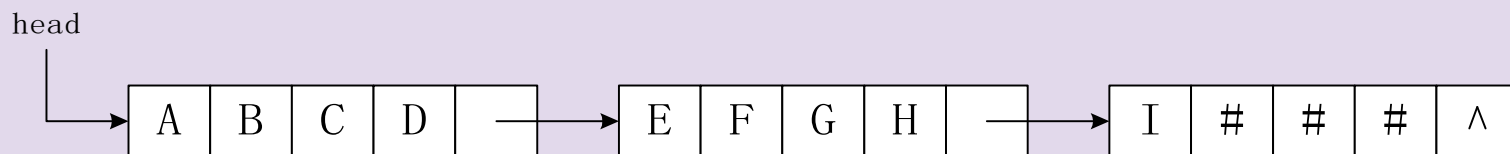
### 2.链式存储表示

优点：操作方便

缺点：存储密度较低

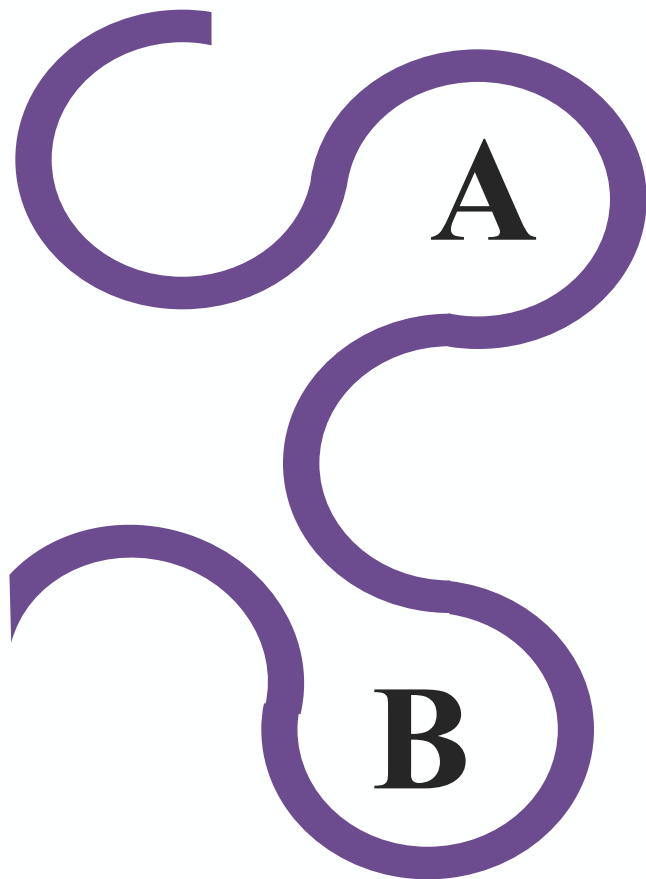
$$\text{存储密度} = \frac{\text{串值所占的存储位}}{\text{实际分配的存储位}}$$

可将多个字符存放在一个结点中，以克服其缺点



## ▶▶三、串的模式匹配BF算法

### 1.串的模式匹配算法



#### 算法目的：

确定主串中所含子串第一次出现的位置（定位）。

#### 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）

第一趟匹配：

S : a b a b c a b c a c b a b

T : a b c

Blue arrows point to the 1st, 2nd, and 3rd characters of S.  $i=3$  is above the 3rd character.

Red arrows point to the 1st, 2nd, and 3rd characters of T.  $j=3$  is to the right of the 3rd character.

A callout bubble points to the 3rd character of S with the text: 指针回溯

第二趟匹配：

S : a b a b c a b c a c b a b

T : a b c

A red arrow points to the 2nd character of S.  $i=2$  is above it.

A blue arrow points to the 1st character of T.  $j=1$  is to the right of it.

第三趟匹配：

S : a b a b c a b c a c b a b

T : a b c

Red arrows point to the 4th, 5th, 6th, and 7th characters of S.  $i=6$  is above the 6th character.

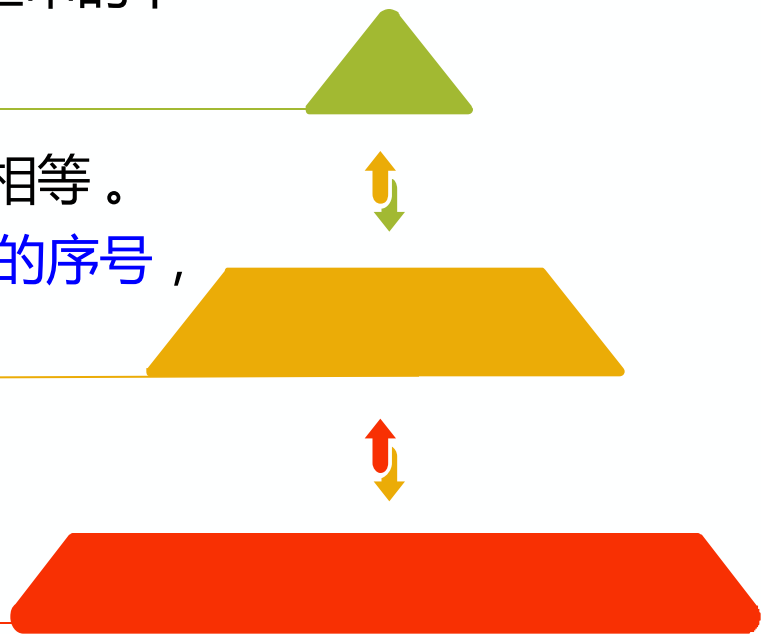
Blue arrows point to the 1st, 2nd, 3rd, and 4th characters of T.  $j=4$  is to the right of the 4th character.

$\text{Index}(S,T,\text{pos})$

将主串的第pos个字符和模式的第一个字符比较，若相等，继续逐个比较后续字符；若不等，从主串的下一字符起，重新与模式的第一个字符比较。

直到主串的一个连续子串字符序列与模式相等。  
返回值为S中与T匹配的子序列第一个字符的序号，即匹配成功。

否则，匹配失败，返回值 0





## 三、串的模式匹配BF算法

## 3.BF算法描述

```
int Index(Sstring S,Sstring T,int pos){  
    i=pos; j=1;  
    while (i <= S.length && j <=T[ 0 ]){  
        if ( S.ch[ i ]=T.ch[ j ]) {++i; ++j; }  
        else{ i=i-j+2; j=1; }  
        if ( j > S.length) return i - S.length;  
        else return 0;  
    }  
}
```



### 三、串的模式匹配BF算法

#### 4.BF算法时间复杂度

例：S='00000000001'，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后一位，即这n-m位各比较了m次；
- ✓最后m位也各比较了1次。

总次数为： $(n-m)*m+m = (n-m+1)*m$

若 $m \ll n$ ，则算法复杂度 $O(n*m)$ 。

1. 串的类型定义
2. 串的顺序存储结构和链式存储结构
3. 串的模式匹配BF算法