华图网校
SINCE 2001
V.HUATU.COM
乐享品质　高效学习

专业课

计算机

数据结构

袁礼

华图网校

目录

数据结构.....1
1. 绪 论.....1
1.1 基本概念和术语.....2
1.2 算法：.....3
1.3 如何读程序.....4
2. 线性表.....6
— 线性表的类型定义.....6
— 线性表的顺序表示和实现.....6
— 线性表的链式表示和实现.....6
— 线性链表.....6
— 循环链表.....6
— 双向链表.....6
2.1 线性表的定义：.....6
2.2 线性表的逻辑结构.....6
2.3 线性表的顺序存储结构.....8
2.4 结构类型.....9
2.5 顺序表上实现的基本操作.....10
2.6 单链表.....13
2.7 双向链表.....14
3. 栈和队列.....15
3.1 栈.....16
3.2 队列.....19
4.串.....24
4.1 串和基本概念.....24
4.2 串的存储.....26
5. 数组.....26
5.1 数组的定义.....26
5.2 数组的顺序表示和实现.....27
5.3 矩阵的压缩存储.....27
6. 树和二叉树.....31
6.1 树的定义和基本术语.....31
6.2 二叉树.....32
6.3 什么是递归调用？.....37
7. 图.....40
7.1 图的定义和术语.....40
7.2 图的存储结构.....45
7.3 图的遍历.....46
7.4 图的连通性.....47
8. 查找.....48
8.1 基本概念.....48
8.2 静态查找表顺序表的查找.....49
8.3 动态查找表.....52

9. 排序.....55
9.1 排序的基本概念.....55
9.2 插入排序.....57
9.3 交换排序.....59

数据结构

1. 绪　论

数据结构基本概念和术语

抽象数据类型的表示与实现

算法设计

算法效率

数据结构在讨论什么问题？

1. 数据的逻辑结构
2. 数据的存储结构
3. 对数据结构的运算

以达到以下目的：

a) 提高计算速度

b) 节省存储空间

如何编出一个好程序？需要分析待处理的对象的特征及各对象之间存在的关系，这就是数据结构的研究主题。

数据结构是信息（数据）之间的结构关系。

例： 查询电话号码簿

（张三，12345678）

（李四，27654321）

……

（郑万，34567213）

基本概念和术语

1. 数据(Data)：是对信息的一种符号表示。在计算机科学中是指所有能输入到计算机中并被计算机程序处理的符号的总称。
2. 数据元素(Data Element)：是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。

一个数据元素可由若干个数据项组成。数据项是数据的不可分割的最小单位。

3. 数据对象(Data Object)：是性质相同的数据元素的集合。是数据的一个子集。

4. 数据结构(Data Structure)：是相互之间存在一种或多种特定关系的数据元素的集合。

1.1 基本概念和术语

数据结构主要指逻辑结构和物理结构

数据之间的相互关系称为逻辑结构。通常分为四类基本结构：

一、集合　结构中的数据元素除了同属于一种类型外，别无其它关系。

二、线性结构　结构中的数据元素之间存在一对一的关系。

三、树型结构　结构中的数据元素之间存在一对多的关系。

四、图状结构或网状结构　结构中的数据元素之间存在多对多的关系。

1. 线性关系：如火车各个车厢、排队的人、一摞扑克牌等。
2. 层次关系：如家族树、公司组织结构等。
3. 网状关系：如铁路交通网、计算机网络等。

数据结构的形式定义为：数据结构是一个二元组：

Data-Structure=(D，S)

其中：D 是数据元素的有限集，S 是 D 上关系的有限集。

数据结构在计算机中的表示称为数据的物理结构，又称为存储结构。

存储结构

数据的逻辑结构在计算机存储空间中的存放形式

常见存储结构：

(1)顺序存储结构：特点是借助于数据元素的相对存储位置来表示数据元素之间的逻辑结构.即逻辑上相邻,物理上也相邻.

(2)链式存储结构：特点是借助于指示数据元素地址的指针表示数据元素之间的逻辑结构。

数据对象可以是有限的，也可以是无限的。数据结构不同于数据类型，也不同于数据对象它不仅要描述数据类型的数据对象，而且要描述数据对象各元素之间的相互关系。

抽象数据类型：一个数学模型以及定义在该型上的一组操作。

抽象数据类型实际上就是对该数据结构定义。因为它定义了一个数据的逻辑结构以在此结构上的一组算法。

用三元组描述如下：

（D，S，P）

数据结构在计算机中有两种不同的表示方法：

顺序表示（线性）和非顺序表示（非线性）

由此得出两种不同的存储结构：顺序存储结构和链式存储结构

1. 顺序存储结构:用数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。
2. 链式存储结构：在每一个数据元素中增加一个存放地址的指针（ ），用此指针来表示数据元素之间的逻辑关系。

数据类型：在一种程序设计语言中，变量所具有的数据种类。

例 在 C 语言中

1. 数据类型：基本类型和构造类型
2. 基本类型：整型、浮点型、字符型
3. 构造类型：数组、结构、联合、指针、枚举型、自定义
4. 数据对象：某种数据类型元素的集合。

例 整数的数据对象是{…-3，-2，-1，0，1，2，3，…}

英文字符类型的数据对象是{A，B，C，D，E，F，…}

算法和算法分析

1.2 算法：

算法是对特定问题求解步骤的一种描述，是指令的有限序列，其中每一条指令表示一个或多个操作。

算法具有以下五个特性：

1）有穷性 一个算法必须总是在执行有穷步之后结束，且每一步都在有穷时间内完成。

2）确定性 算法中每一条指令必须有确切的含义。不存在二义性。且算法只有一个入口和一个出口。

3）可行性 一个算法是可行的。即算法描述的操作都是可以通过已经实现的基本运算执行有限次来实现的

4）输入 一个算法有零个或多个输入，这些输入取自于某个特定的对象集合。

5）输出 一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

算法设计的要求

评价一个好的算法有以下几个标准:

(1) 正确性(Correctness) 算法应满足具体问题的需求。

(2)可读性(Readability)　算法应该好读。以有利于阅读者对程序的理解。

(3)健壮性(Robustness) 算法应具有容错处理。当输入非法数据时，算法应对其作出反应，而不是产年莫名其妙的输出结果。

4)效率与存储量需求　效率指的是算法执行的时间；存储量需求指算法执行过程中所需要的最大存储空间。一般，这两者与问题的规模有关。

算法效率的度量

时间复杂度 是指执行算法所需要的计算工作量，是由算法执行的基本运算次数来度量。

空间复杂度是指算法在计算机内执行时所需的内存空间。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，算法的时间量度记作

T(n)=O(f(n))

称作算法的渐近时间复杂度。

例

1. {++x;s=0;}

2. for(i=1;i<=n;++i)

{++x;s+=x;}

3. for(j=1，j<=n;++j)

for(k=1;k<=n;++k)

{++x;s+=x;}

1.O(1)　2. O(n)　3. O(n2)

常数阶 线性阶　平方阶

常见时间复杂度比较：O(1)<O(logn)<O(n)<O(nlogn)<O(n2)<O(n3)

指数时间的关系为：

O(2n)<O(n!)<O(nn)

当 n 取得很大时，指数时间算法和多项式时间算法在所需时间上非常悬殊。

1.3 如何读程序

读程序如同看小说，看看角色有哪些，看看角色之间发生了什么事，推测结局将会是怎样的？

角色：变量

角色之间的故事：各行代码

常见套路：顺序、分支、循环

【练习】对一个算法的评价，不包括如下（B）方面的内容。

A．健壮性和可读性

B．并行性

C．正确性

D．时空复杂度

【练习】在数据结构中，从逻辑上可以把数据结构分为（C）

A．动态结构和静态结构

B．紧凑结构和非紧凑结构

C．线性结构和非线性结构

D．内部结构和外部结构

【练习】数据结构中，与所使用的计算机无关的是数据的结构 C

A. 存储

B. 物理

C. 逻辑

D. 物理和存储

【练习】算法分析的目的是： C

A. 找出数据结构的合理性

B. 研究算法中的输入和输出的关系

C. 分析算法的效率以求改进

D. 分析算法的易懂性和文档性

2. 线性表

- 线性表的类型定义
- 线性表的顺序表示和实现
- 线性表的链式表示和实现
- 线性链表
- 循环链表
- 双向链表

2.1 线性表的定义：

线性表直观理解：

一种线性结构，结点的个数称为表长，表长为 0 的线性表称为空表。

2.2 线性表的逻辑结构

线性表(Linear List) ：由 n(n≧)个数据元素(结点)a_1，a_2， …a_n 组成的有限序列。其中数据元素的个数 n 定义为表的长度。当 n=0 时称为空表，常常将非空的线性表(n>0)记作：

(a_1，a_2，…a_n)

这里的数据元素 a_i(1≦i≦n)只是一个抽象的符号，其具体含义在不同的情况下可以不同。

例　26 个英文字母组成的字母表

A，B，C、…、Z）

例　某校从 1978 年到 1983 年各种型号的计算机拥有量的变化情况。

（6，17，28，50，92，188）

例　学生基本信息表如下：

学号	姓名	性别	民族	政治面貌	出生日期
200901001	程鑫民	男	锡伯族	团员	1992-1-30
200901002	吴薇	女	壮族	群众	1992-7-9
200901003	李薪	女	汉族	团员	1990-12-3
200901004	张丽	女	白族	团员	1991-2-5
200901005	李斯	男	彝族	团员	1990-8-24

例　一副扑克的点数

（2，3，4，…，J，Q，K，A）

从以上例子可看出线性表的逻辑特征是：

在非空的线性表，有且仅有一个开始结点 a1，它没有直接前趋，而仅有一个直接后继 a2；

有且仅有一个终端结点 an，它没有直接后继，而仅有一个直接前趋 a n-1；

其余的内部结点 ai(2≦i≦n-1)都有且仅有一个直接前趋 a i-1 和一个直接后继 a　i+1。

线性表是一种典型的线性结构。

线性表的基本操作

数据的运算是定义在逻辑结构上的，而运算的具体实现则是在存储结构上进行的。

对于线性表，常用的运算有如下几种：

(1))置空表（结构初始化）

init_seqlist(L)；

操作结果：构造一个空的线性表 L。

⑵求线性表的长度。

list_length(L);

初始条件：线性表 L 已存在。

操作结果：返回 L 中元素个数。

⑶定位:访问线性表的第 i 个元素。

get_node(L,i);

初始条件：线性表 L 已存在。

操作结果：返回 L 中第 i 个元素的值或地址。

⑷查找 ：在线性表中查找满足某种条件的数据元素。

location_list(L,x);

初始条件：线性表 L 已存在。

操作结果：返回 L 中值为给定值 x 的数据元素。

⑸插入:在线性表的第 i 个元素之前，插入一个同类型的元素。

insert_list(L,i,x);

初始条件：线性表 L 已存在。

操作结果：在 L 的第 i 个位置插入一个值为 x 的新元素。

⑹删除 ：删除线性表中第 i 个元素。

delete_list(L,i);

初始条件：线性表 L 已存在。

操作结果：在 L 中删除序号为 i 的数据元素。

⑺清除表: 将已有线性表变为空表，即删除表中所有元素。

2.3 线性表的顺序存储结构

线性表

把线性表的结点按逻辑顺序依次存放在一组地址连续的存储单元里。用这种方法存储的线性表简称顺序表。

假设线性表的每个元素需占用 l 个存储单元，并以所占的第一个单元的存储地址作为数据元素的存储位置。则线性表中第 i+1 个数据元素的存储位置 LOC(a i+1)和第 i 个数据元素的存储位置 LOC(a i)之间满足下列关系：

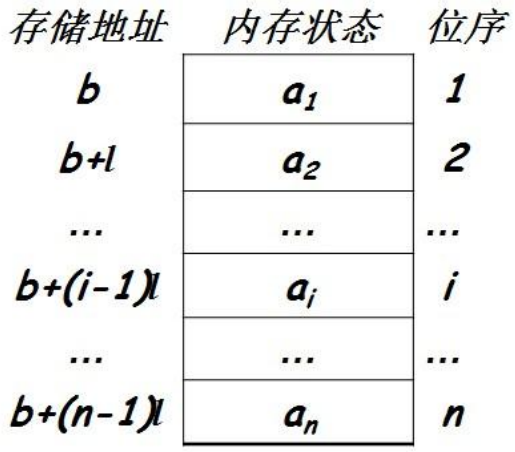
LOC(a i+1)=LOC(a i)+l

线性表的第 i 个数据元素 ai 的存储位置为：

LOC(ai)=LOC(a1)+(i-1)*l

逻辑结构中相邻的结点在存储结构中仍相邻。

线性表的顺序表示



b——数据元素 a1 的存储地址

l——每个数据元素所需的存储单元大小

由于 C 语言中的一维数组也是采用顺序存储表示，故可以用数组类型来描述顺序表。又因为除了用数组来存储线性表的元素之外，顺序表还应该用一个变量来表示线性表的长度属性，所以用结构类型来定义顺序表类型。

2.4 结构类型

什么是结构类型？

在实际问题中，一组数据往往具有不同的数据类型。例如， 在学生登记表中，姓名应为字符型；学号可为整型或字符型； 年龄应为整型；性别应为字符型；成绩可为整型或实型。

“结构”是一种构造类型，它是由若干“成员”组成的。 每一个成员可以是一个基本数据类型或者又是一个构造类型。 结构既是一种“构造”而成的数据类型， 那么在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

结构类型

结构的定义

定义一个结构的一般形式为：

```
struct  结构名
{
   成员表列     /*类型说明符  成员名*/
};
```

如：

```
struct STU
{
   int num;
   char name[20];
   char sex;
```

```
    float score;
};
```

STU boy1；

等同于：

```
struct STU
{
    int num;
    char name[20];
    char sex;
    float score;
}boy1;
```

C 语言中用结构类型来定义顺序表

```
struct  结构名
{
    成员表列     /*类型说明符  成员名*/
};
```

如：

```
        # define ListSize      100
        typedef    int    DataType; /*类型别名*/
        typedef    struc{
                        DataType    data[ListSize];    /*元素*/
                        int      length; /*长度*/
                } Sqlist;
```

【练习】线性表是具有 n 个（　）的有限序列。

A．字符

B．数据元素

C．数据项

D．表元素

2.5 顺序表上实现的基本操作

在顺序表存储结构中，很容易实现线性表的一些操作，如线性表的构造、第 i 个元素的访问。以下主要讨论线性表的插入和删除两种运算。

注意：C 语言中的数组下标从“0”开始，因此，若 L 是 Sqlist 类型的顺序表，则表中第 i 个元素是 L.data[i-1]。

线性表的插入运算

插入运算：在表的第 i(1<=i<=n+1)个位置上，插入一个新结点 x，使长度为 n 的线性表变成长度为 n+1 的线性表

插入算法的基本步骤：

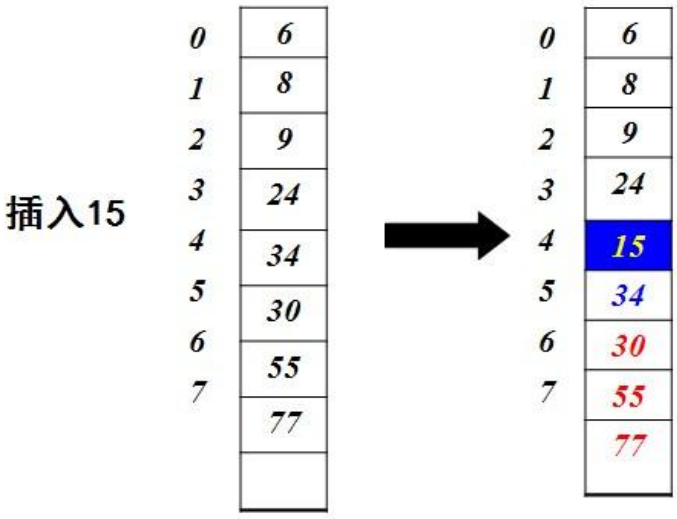
将结点 a_i，…,a_n 各后移一位；

将新结点 x 置入第 i 个位置；

表长加 1

插入一个元素所需平均移动次数：n/2

在线性表的第 5 个位置插入一个元素：




```
Void insertList(Sqlist*l , DataType x , int i)
{
   int j;
   if(i<1 || i>l.length+1) /*确保位置在正常范围*/
        printf( "Position error" );
        return ERROR
             if(l.length>=ListSize)
      printf( "overflow" );
      exit(overflow);
    for(j=l.length-1;j>=i-1;j--) /*最后一轮j为4*/
         l.data[j+1]=l.data[j];
     l.data[i-1]=x;
     l.length++;
}
```

插入15→

0 6
1 8
2 9
3 24
4 34
5 30
6 55
7 77

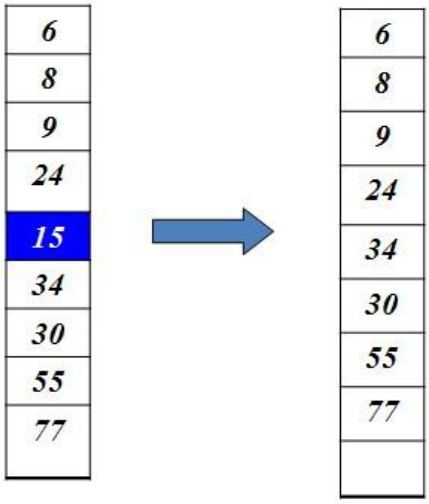
线性表的删除运算

删除运算:将表的第 i(1<=i<=n)个结点删去，使长度为 n 的线性表变成长度为 n-1 的线性表。

删除运算的基本步骤：结点 ai+1, …,an 依次前移一个位置；表长减 1

删除一个元素所需平均移动次数： (n-1)/2

删除顺序表中某个指定的元素：



线性表的链式表示和实现

线性表的顺序表示的特点是用物理位置上的邻接关系来表示结点间的逻辑关系，这一特点可以随机存取表中的任一结点，但它也使得插入和删除操作会移动大量的结点.为避免大量结点的移动，可以采用线性表的另一种存储方式，链式存储结构，简称为链表(Linked List)。

线性表的链式存储结构

线性表的链式表示：用一组任意的存储单元（可连续也可不连续）存储线性表的数据元素。在链式存储结构方式下，存储数据元素的结点空间可以不连续，各数据结点的存储顺序与数据元素之间的逻辑关系可以不一致，而数据元素之间的逻辑关系由指针域来确定。

每个节点都由两部分组成：数据域和指针域。

数据域存放元素本身的数据，

指针域存放指针。

data	***link***

（甲，乙，丙，丁，戊，己，庚）的链式表示

2.6 单链表

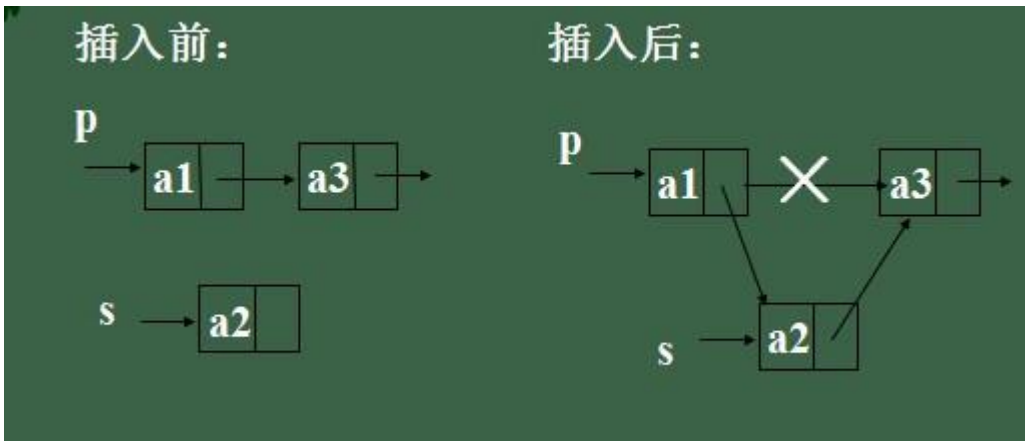
数据域(data)：用于存储线性表一个数据元素的信息。

指针域(next)：用于存放一个指针，该指针指向本结点的后继结点。

Head 称为头指针，指向链表中第一个结点；

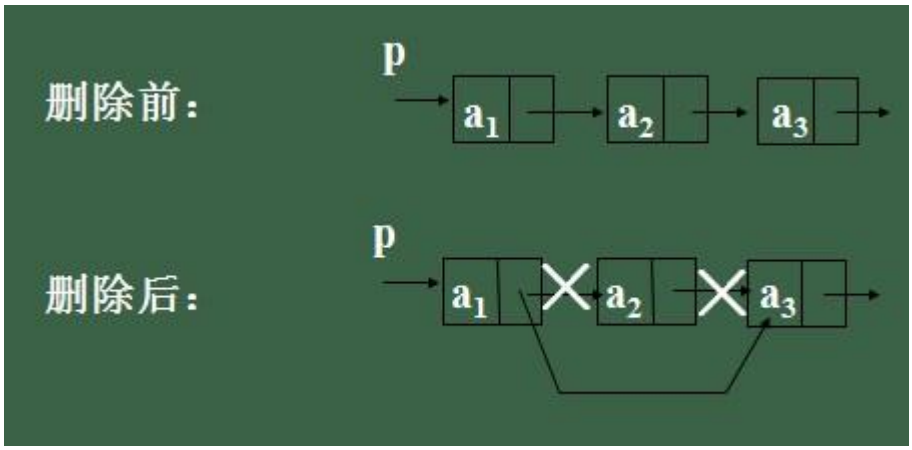
如果为 NULL，表示为空

单链表中结点的插入



（在指针 p 所指的结点后插入指针 s 所指的结点）

单链表中结点的删除：（删除指针 p 后面的结点）

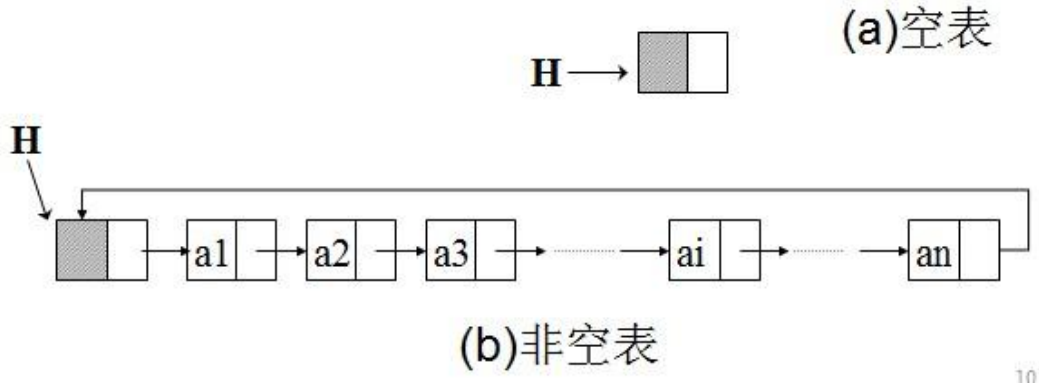


循环链表

循环链表的特点——表中的最后一个结点的指针域指向头结点，整个链表形成一个环。

从表的任意结点出发可以找到表中其它结点。

循环链表和线性表的操作差别：仅在于算法中的循环条件不是 L 或 L->next 是否为空，而是它们是否等于头指针。



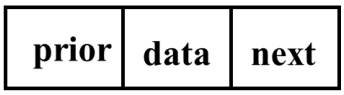
2.7 双向链表

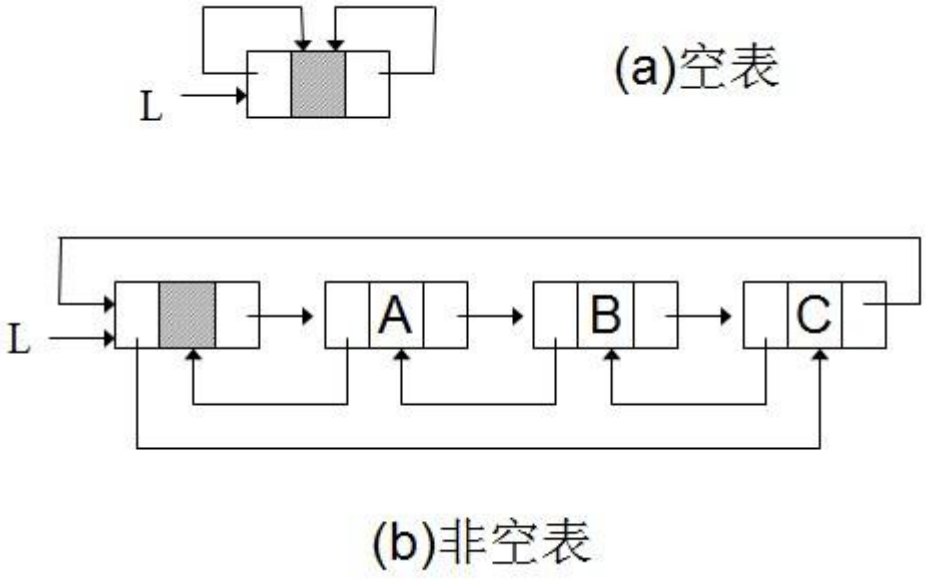
双向链表

双向链表的特点：表中的每个结点有两个指针域，一个指向后继结点，一个指向前趋结点，整个链表形成两个环。

从表的任意结点出发可以通过正向环（或反向环）找到表中其它结点。

结点：





【练习】下面关于线性表的叙述中，错误的是哪一个？ B

A 线性表采用顺序存储，必须占用一片连续的存储单元

B 线性表采用顺序存储，便于进行插入和删除操作

C 线性表采用链式存储，不必占用一片连续的存储单元

D 线性表采用链式存储，便于进行插入和删除操作。

【练习】若长度为 n 的线性表采用顺序存储结构，在其第 i 个位置插入一个新元素的算法的时间复杂度为 （ ）。

A．O(0) B．O(1)

C．O(n) D．$O(n^2)$

3. 栈和队列

栈的定义及基本运算

栈(Stack)是限制在表的一端进行插入和删除运算的线性表，通常称插入、删除的这一端为栈顶(Top)，另一端为栈底(Bottom)。当表中没有元素时称为空栈.

假设栈 S=(a1，a2，a3，…an)，则 a1 称为栈底元素，an 为栈顶元素。栈中元素按 a1，a2，a3，…an 的次序进栈，退栈的第一个元素应为栈顶元素。换句话说，栈的修改是按后进先出的原则进行的。因此，栈称为后进先出表（LIFO）。

3.1 栈

顺序栈

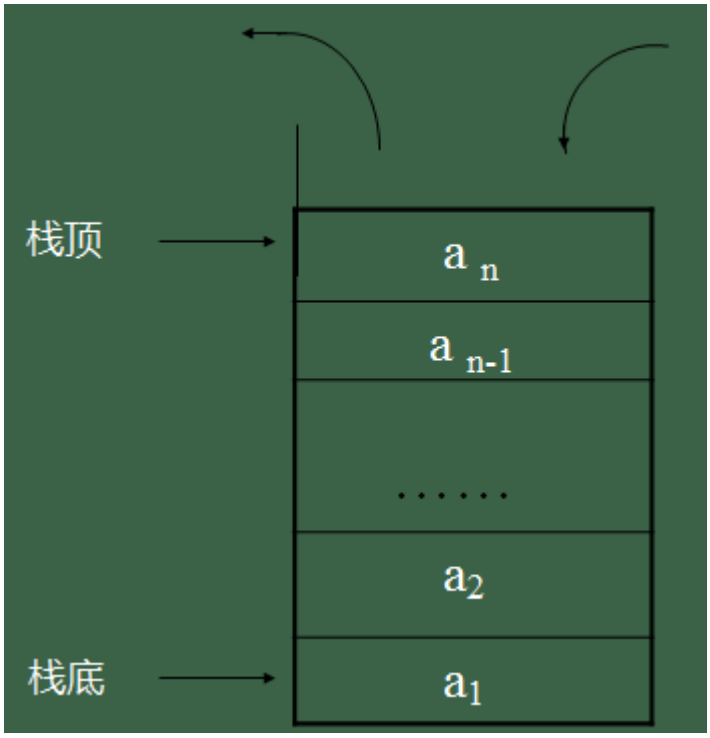
由于栈是运算受限的线性表，因此线性表的存储结构对栈也适应。

栈的顺序存储结构简称为顺序栈，它是运算受限的线性表。因此，可用数组来实现顺序栈。因为栈底位置是固定不变的，所以可以将栈底位置设置在数组的两端的任何一个端点；栈顶位置是随着进栈和退栈操作而变化的，故需用一个整型变量 top 来指示当前栈顶的位置，通常称 top 为栈顶指针。因此，顺序栈的类型定义只需将顺序表的类型定义中的长度属性改为 top 即可。

例一叠盘子

栈的抽象数据类型的定义如下：

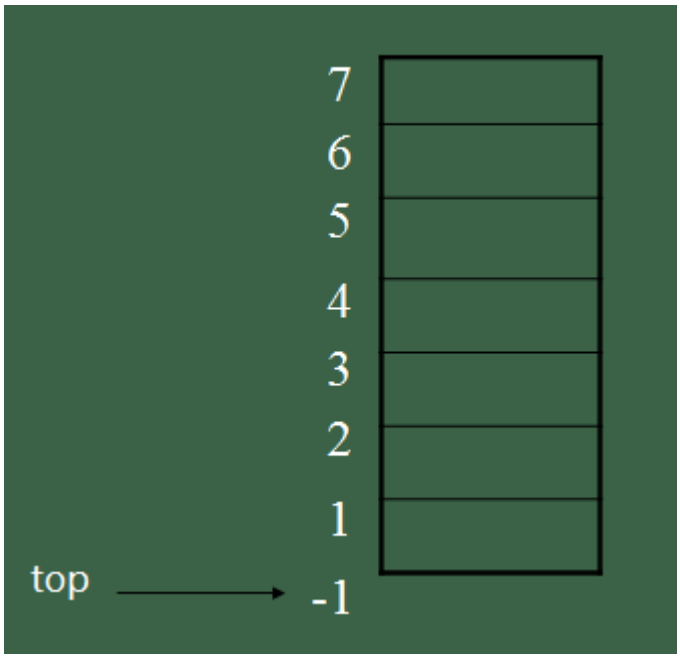
栈的逻辑含义：



常用 top 来指示当前栈顶的位置，通常称 top 为栈顶指针。因此，顺序栈的类型定义只需将

顺序表的类型定义中的长度属性改为 top 即可。顺序栈的类型定义如下：

```
# define StackSize 100
    typedef    char datatype;
    typedef struct {
            datatype    data[stacksize];
             int top;
      }seqstack;
```



设 S 是 SeqStack 类型的指针变量。若栈底位置在向量的低端，即 s－>data[0]是栈底元素，那么栈顶指针 s－>top 是正向增加的，即进栈时需将 s－>top 加 1，退栈时需将 s－>top 减 1。因此，s－>top<0 表示空栈， s－>top =stacksize-1 表示栈满。当栈满时再做进栈运算必定产生空间溢出，简称“上溢”;当栈空时再做退栈运算也将产生溢出，简称“下溢”。上溢是一种出错状态，应该设法避免之；下溢则可能是正常现象，因为栈在程序中使用时，其初态或终态都是空栈，所以下溢常常用来作为程序控制转移的条件。

常见栈操作

1、置空栈

```
void initstack(seqstack *s)
  {
   s－>top=-1;
   }
```

2、判断栈空

```
int stackempty(seqstack *s)
```

```
{
    return(s－>top==-1);
}
```

3、判断栈满

```
int stackfull(seqstack    *s)
{
    return(s－>top==stacksize-1);
}
```

4、进栈

```
void   push(seqstack   *s，datatype   x)
{
    if (stackfull(s))
        error(“stack overflow”);
    s－>data[++s－>top]=x;
}
```

5、退栈

```
datatype pop(seqstack *s)
{
    if(stackempty(s))
    error(“stack underflow”);
    x=s－>data[top];
    s－>top--;
    return(x)
//return(s－>data[s－>top--]);
}
```

6、取栈顶元素

```
Datatype   stacktop(seqstack   *s)
{
    if(stackempty(s)
        error(“stack is enpty”);
    return s－>data[s－>top];
```

}

链栈

栈的链式存储结构称为链栈，它是运算是受限的单链表，克插入和删除操作仅限制在表头位置上进行.由于只能在链表头部进行操作，故链表没有必要像单链表那样附加头结点。栈顶指针就是链表的头指针。

链栈的类型说明如下：

```
typedef    struct    stacknode{
        datatype    data;
        struct stacknode *next;
    }stacknode;
```

【练习】假设以 I 和 O 分别表示进栈和出栈操作，则对输入序列 a,b,c,d,e 进行一系列栈操作 IIOIOIIOOO 之后，得到的输出序列为（ A ）。

A b,c,e,d,a

B b,e,c,a,d

C e,c,b,d,a

D c,e,b,a,d

3.2 队列

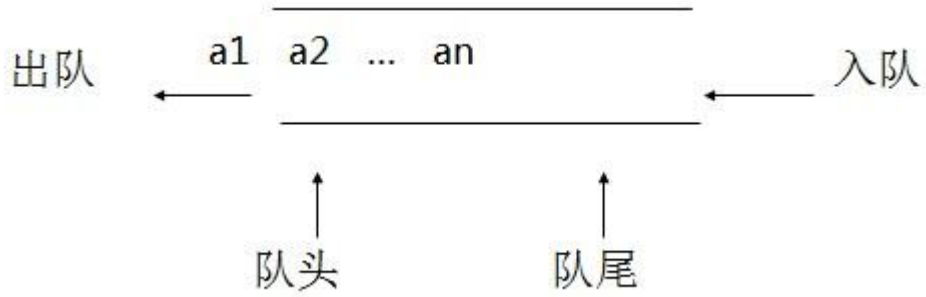
抽象数据类型队列的定义

队列(Queue)也是一种运算受限的线性表。它只允许在表的一端进行插入，而在另一端进行删除。允许删除的一端称为队头(front)，允许插入的一端称为队尾(rear)。

例如：排队购物。操作系统中的作业排队。先进入队列的成员总是先离开队列。因此队列亦称作先进先出(First In First Out)的线性表，简称 FIFO 表。

当队列中没有元素时称为空队列。在空队列中依次加入元素 a1,a2,…an 之后，a1 是队头元素，an 是队尾元素。显然退出队列的次序也只能是 a1,a2,…an ，也就是说队列的修改是依先进先出的原则进行的。

队列的示意图

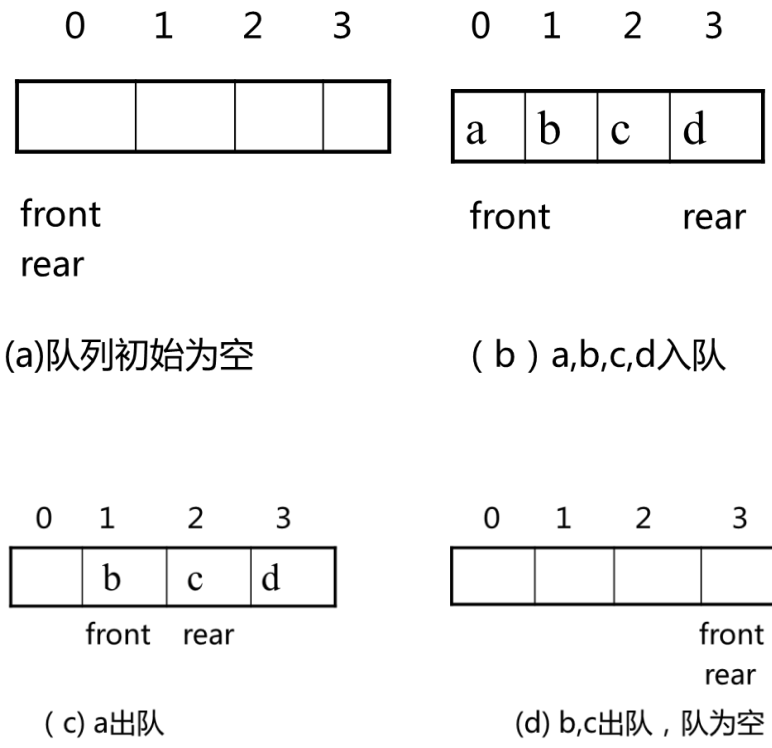


顺序队列

1.队列的顺序存储结构称为顺序队列，顺序队列实际上是运算受限的顺序表，和顺序表一样，顺序队列也是必须用一个向量空间来存放当前队列中的元素。由于队列的队头和队尾的位置是变化的，因而要设两个指针和分别指示队头和队尾元素在队列中的位置，它们的初始值地队列初始化时均应置为 0 。

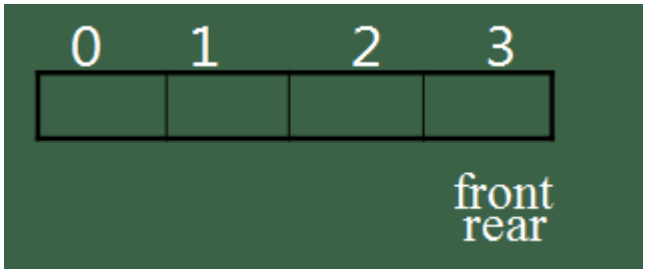
2.入队时将新元素插入所指的位置，然后将加 1 。出队时，删去所指的元素，然后将加 1 并返回被删元素。由此可见，当头尾指针相等时队列为空。

3.在非空队列里，头指针始终指向队头元素，而尾指针始终指向队尾元素的下一位置。



和栈类似，队列中亦有上溢和下溢现象。此外，顺序队列中还存在“假上溢”现象。

因为在入队和出队的操作中，头尾指针只增加不减小，致使被删除元素的空间永远无法重新利用。因此，尽管队列中实际的元素个数远远小于向量空间的规模，但也可能由于尾指针已超出向量空间的上界而不能做入队操作。该现象称为假上溢。



为充分利用向量空间。克服上述假上溢现象的方法是将向量空间想象为一个首尾相接的圆环，并称这种向量为循环向量，存储在其中的队列称为循环队列（Circular Queue)。

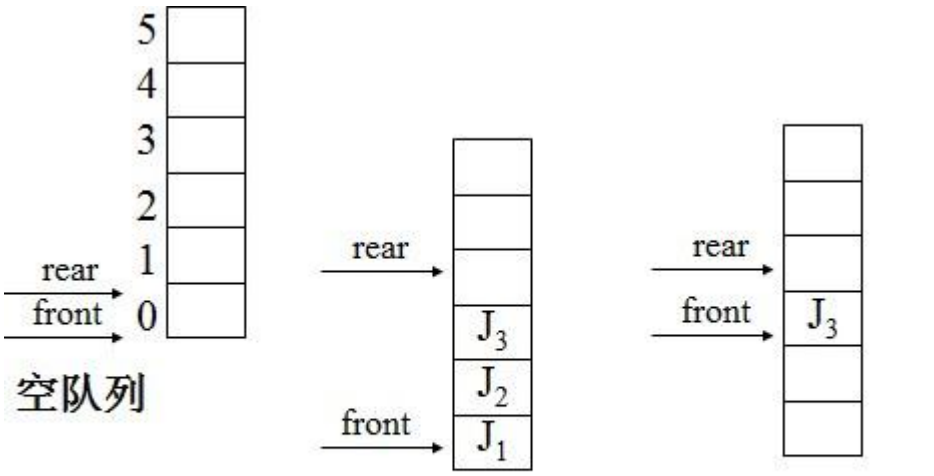
在循环队列中进行出队、入队操作时，头尾指针仍要加 1，朝前移动。只不过当头尾指针指向向量上界（QueueSize-1）时，其加 1 操作的结果是指向向量的下界 0。

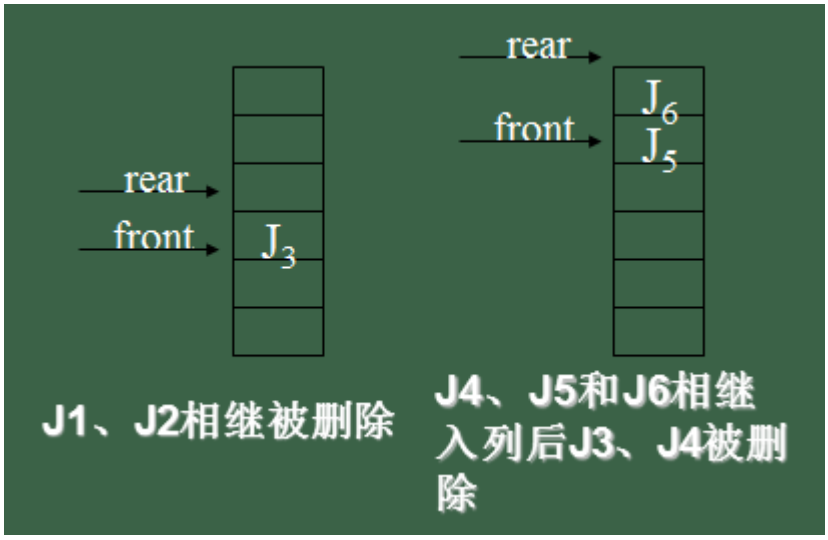
由于入队时尾指针向前追赶头指针，出队时头指针向前追赶尾指针，故队空和队满时头尾指针均相等。因此，无法通过 front=rear 来判断队列“空”还是“满”。

因此，需要判断循环队列中元素个数：

rear-front

队列示例





链队列

队列的链式存储结构简称为链队列，它是限制仅在表头删除和表尾插入的单链表。显然仅有单链表的头指针不便于在表尾做插入操作，为此再增加一个尾指针，指向链表的最后一个结点。于是，一个链队列由一个头指针唯一确定。和顺序队列类似，我们也是将这两个指针封装在一起，将链队列的类型 LinkQueue 定义为一个结构类型：

```
typedef struct queuenode{
    datatype    data;
    struct queuenode *next;
}queuenode;
```

```
typedef struct{
        queuenode   *front;
        queuenode   *rear;
    }linkqueue;
```

【练习】假设以 I 和 O 分别表示进栈和出栈操作，则对输入序列 a,b,c,d,e 进行一系列栈操作 IIOIOIIOOO 之后，得到的输出序列为（ A ）。

A b,c,e,d,a

B b,e,c,a,d

C e,c,b,d,a

D c,e,b,a,d

【练习】

栈和队列的共同特点是(A)。

A.只允许在端点处插入和删除元素

B.都是先进后出

C.都是先进先出

D.没有共同点

【练习】用链表存储的队列，在进行插入运算时(D).

A. 仅修改头指针

B. 头、尾指针都要修改

C. 仅修改尾指针

D.头、尾指针可能都要修改

【练习】设计一个判别表达式中左、右括号是否配对出现的算法，采用(D)数据结构最佳。

A．线性表的顺序存储结构

B．队列

C．线性表的链式存储结构

D．栈

【练习】一个队列的入队序列是 1，2，3，4，则队列的输出序列是（B）。

A．4，3，2，1

B．1，2，3，4

C．1，4，3，2

D．3，2，4，1

4.串

- 串的基本概念
- 串的基本操作
- 串的表示和实现

  --定长顺序存储

  --链式存储

4.1 串和基本概念

串类型的定义

1.串(String)是零个或多个字符组成的有限序列。一般记作 S=“a1a2a3…an”，其中 S 是串名，双引号括起来的字符序列是串值；ai(1≦i≦n)可以是字母、数字或其它字符；串中所包含的字符个数称为该串的长度。

2.长度为零的串称为空串(Empty String)，它不包含任何字符。通常将仅由一个或多个空格组成的串称为空白串(Blank String)

3.注意：空串和空白串的不同，例如“ ”和“”分别表示长度为 1 的空白串和长度为 0 的空串。

串中任意个连续字符组成的子序列称为该串的子串，包含子串的串相应地称为主串。通常将子串在主串中首次出现时的该子串的首字符对应的主串中的序号，定义为子串在主串中的序号（或位置）。例如，设 A 和 B 分别为

A=“This is a string” B=“is”

则 B 是 A 的子串，A 为主串。B 在 A 中出现了两次，其中首次出现所对应的主串位置是 3。因此，称 B 在 A 中的序号（或位置）为 3

特别地，空串是任意串的子串，任意串是其自身的子串。

串的基本操作

C 语言实现

定义下列几个变量：

```
char s1[20]=“dirtreeformat”,s2[20]=“file.mem”;
char s3[30],*p;
int result;
```

（1）求串长(length)

int strlen(char s);　//求串的长度

例如：

printf(“%d”,strlen(s1)); //输出 13

（2）串复制(copy)

char *strcpy(char to,char from);

该函数将串 from 复制到串 to 中，并且返回一个指向串 to 的开始处的指针。

例如：

strcpy(s3,s1);　//s3=“dirtreeformat”

(3)联接(concatenation)

char　strcat(char to,char from)

该函数将串 from 复制到串 to 的末尾，并且返回一个指向串 to 的开始处的指针。

例如：strcat(s3,”/”)

strcat(s3,s2);　//s3=“dirtreeformat/file.mem”

(4)串比较（compare)

int strcmp(chars1,char s2);

该函数比较串 s1 和串 s2 的大小，当返回值小于 0，等于 0 或大于 0 时分别表示 s1<s2\s1=s2 或 s1>s2

例如：result=strcmp(“baker”,”Baker”)　result>0

result=strcmp(“12”,”12”);　result=0

result=strcmp(“Joe”,”Joseph”);　result<0

（5）字符定位(index)

char strchr(char s,char c);

该函数是找 c 在字符串中第一次出现的位置，若找到则返回该位置，否则返回 NULL。

例如：

p=strchr(s2,”.”);　// p 指向“file”之后的位置

if(p) strcpy(p,”.cpp”);　s2=“file.cpp”

串的表示和实现

因为串是特殊的线性表，故其存储结构与线性表的存储结构类似。只不过由于组成串的结点是单个字符。故此有两种表示方式。

4.2 串的存储

定长顺序存储表示

定长顺序存储表示,也称为静态存储分配的顺序表。它是用一组连续的存储单元来存放串中的字符序列。所谓定长顺序存储结构，是直接使用定长的字符数组来定义，数组的上界预先给出：

```
#define maxstrlen 256
typedef char sstring[maxstrlen];
sstring s;     //s 是一个可容纳 255 个字符的顺序串。
```

一般可使用一个不会出现在串中的特殊字符在串值的尾部来表示串的结束。例如，C 语言中以字符＇\0′表示串值的终结。

这就是为什么在上述定义中，串空间最大值 maxstrlen 为 256，但最多只能存放 255 个字符的原因，因为必须留一个字节来存放＇\0′字符。

串的链式存储结构

顺序串上的插入和删除操作不方便，需要移动大量的字符。因此，可用单链表方式来存储串值，串的这种链式存储结构简称为链串。

```
typedef struct node{
        char data;
        struct node *next;
}lstring;
```

一个链串由头指针唯一确定。这种结构便于进行插入和删除运算，但存储空间利用率太低。

5. 数组

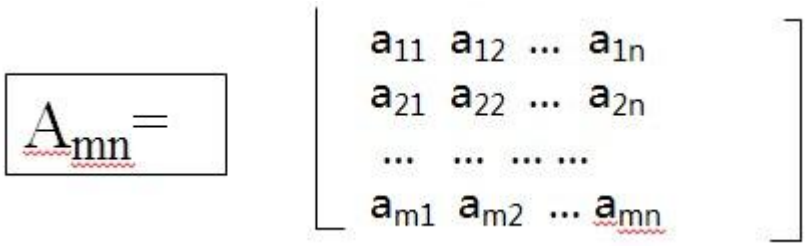
- 数组的定义
- 数组的顺序表示和实现
- 矩阵的压缩存储
- 特殊矩阵
- 稀疏矩阵

5.1 数组的定义

1. 数组可看成是一种特殊的线性表，其特殊在于，表中的数所代表的元素本身也是一种线性表。

2. 数组是我们最熟悉的数据类型，数组中各元素具有统一的类型，并且数组元素的下标一般具有固定的上界和下界，因此，数组的处理比其它复杂的结构更为简单。

3. 多维数组是向量的推广。例如，二维数组可以看成是由个行向量组成的向量，也可以看成是个列向量组成的向量。



在 C 语言中，一个二维数组类型可以定义为其分量类型为一维数组类型的一维数组类型，也就是说，

```
typedef    elemtype    array2[m][n];
```

等价于：

```
typedef elemtype array1[n];

typedef    array1 array2[m]; //每个分类都是一维数组
```

同理，一个维数组类型可以定义为其数据元素为维数组类型的一维序组类型。

数组一旦被定义，它的维数和维界就不再改变。因此，除了结构的初始化和销毁之外，数组只有存取元素和修改元素值的操作。

5.2 数组的顺序表示和实现

由于计算机的内存结构是一维的，因此用一维内存来表示多维数组，就必须按某种次序将数组元素排成一列序列，然后将这个线性序列存放在存储器中。

又由于对数组一般不做插入和删除操作，也就是说，数组一旦建立，结构中的元素个数和元素间的关系就不再发生变化。因此，一般都是采用顺序存储的方法来表示数组。

5.3 矩阵的压缩存储

在科学与工程计算问题中，矩阵是一种常用的数学对象，在高级语言编制程序时，将一个矩阵描述为一个二维数组。

矩阵在这种存储表示之下，可以对其元素进行随机存取。但是在矩阵中非零元素呈某种规律分布或者矩阵中出现大量的零元素的情况下，实际上占用了许多单元去存储重复的非零元

素或零元素，这对高阶矩阵会造成极大的浪费。

为了节省存储空间， 可以对这类矩阵进行压缩存储：即为多个相同的非零元素只分配一个存储空间；对零元素不分配空间。

特殊矩阵

所谓特殊矩阵是指非零元素或零元素的分布有一定规律的矩阵，下面我们讨论几种特殊矩阵的压缩存储。

1、对称矩阵

在一个 n 阶方阵 A 中，若元素满足下述性质：

$a_{ij}=a_{ji}$ $0 \leqq i,j \leqq n-1$

则称 A 为对称矩阵。

对称矩阵中的元素关于主对角线对称，故只要存储矩阵中上三角或下三角中的元素，让每两个对称的元素共享一个存储空间，这样，能节约近一半的存储空间。

按“行优先顺序”存储主对角线（包括对角线）以下的元素，其存储形式如图所示：

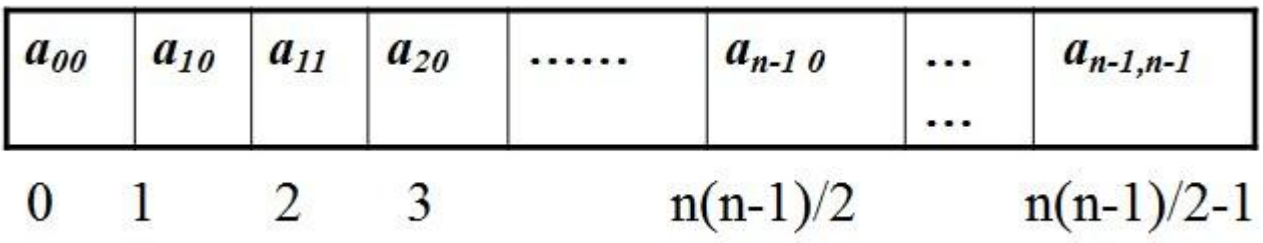
1	5	1	3	7	a00
5	0	8	0	0	a10 a 11
1	8	9	2	6	a20 a21 a23
3	0	2	5	1	……..
7	0	6	1	3	an-1 0 a n-1 1 a n-1 2 …a n-1 n-1

在这个下三角矩阵中，第 i 行恰有 i+1 个元素，元素总数为：

(i+1)=n(n+1)/2

对称矩阵的压缩

可以将这些元素存放在一个向量 sa[0..n(n+1)/2-1]中。



2、三角矩阵

以主对角线划分，三角矩阵有上三角和下三角两种。上三角矩阵如图所示，它的下三角（不包括主对角线）中的元素均为常数。下三角矩阵正好相反，它的主对角线上方均为

常数，如图所示。在大多数情况下，

三角矩阵常数为零。

$$\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0\,n-1} \\ c & a_{11} & \cdots & a_{1\,n-1} \\ & \cdots\cdots & & \\ c & c & \cdots & a_{n-1\,n-1} \end{pmatrix} \qquad \begin{pmatrix} a_{00} & c & \cdots & c \\ a_{10} & a_{11} & \cdots & c \\ & \cdots\cdots & & \\ a_{n-1\,0} & a_{n-1\,1} & \cdots & a_{n-1\,n-1} \end{pmatrix}$$

(a)上三角矩阵　　(b)下三角矩阵

三角矩阵中的重复元素 c 可共享一个存储空间，其余的元素正好有 n(n+1)/2 个，因此，三角矩阵可压缩存储到向量 sa[0..n(n+1)/2]中，其中 c 存放在向量的最后一个分量中。

a_{00}	a_{10}	a_{11}	a_{20}	……	$a_{n-1\,0}$	… …	$a_{n-1,n-1}$
0	1	2	3		n(n-1)/2		n(n-1)/2-1

3、稀疏矩阵

什么是稀疏矩阵？简单说，设矩阵 A 中有 s 个非零元素，若 s 远远小于矩阵元素的总数（即 s≦m×n），则称 A 为稀疏矩阵。

设在的矩阵 A 中，有 s 个非零元素。令 e=s/(m*n)，称 e 为矩阵的稀疏因子。通常认为 e≦0.05 时称之为稀疏矩阵。

在存储稀疏矩阵时，为了节省存储单元，很自然地想到使用压缩存储方法。

由于非零元素的分布一般是没有规律的，因此在存储非零元素的同时，还必须同时记下它所在的行和列的位置（i,j)。反之，一个三元组(i,j,aij)唯一确定了矩阵 A 的一个非零元。因此，稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。

例如，下列三元组表

((1 ,2,12)(1,3,9),(3,1,- 3),(3,6,14),(4,3,24), (5,2,18),(6,1,15),(6,4,-7))

加上(6,7)这一对行、列值便可作为下列矩阵 M 的一种描述。

$$M=\begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{pmatrix}$$

图　稀疏矩阵M

稀疏矩阵存储：三元组顺序表

假设以顺序存储结构来表示三元组表，则可得到稀疏矩阵的一种压缩存储方法——三元顺序表。

```
#define maxsize 10000
typedef int datatype;
typedef struct{
    int     i,j;
    datatype v;
 }triple;
```

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

【练习】若串 S=‘software’，其子串的数目是 B 。

A．8　B．37

C．36　D．9

【练习】串的长度是指 B 。

A．串中所含不同字母的个数

B．串中所含字符的个数

C．串中所含不同字符的个数

D．串中所含非空格字符的个数

【练习】数组 A 中，每个元素的长度为 3 个字节，行下标 i 从 1 到 8，列下标 j 从 1 到 10，从首地址 SA 开始连续存放的存储器内，该数组按行存放，元素 A[8][5]的起始地址为 C 。

A．SA＋141

B．SA＋144

C．SA＋222

D．SA＋225

【练习】对稀疏矩阵进行压缩存储目的是（ C）。

A．便于进行矩阵运算

B．便于输入和输出

C．节省存储空间

D．降低运算的时间复杂度

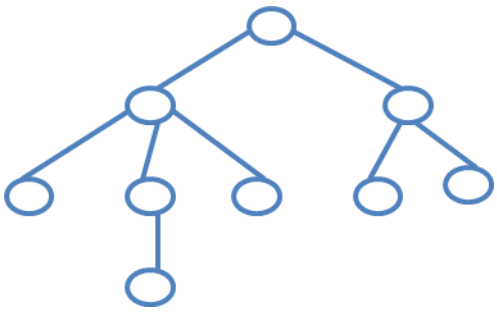
6. 树和二叉树

6.1 树的定义和基本术语

树是一类重要的非线性数据结构，是以分支关系定义的层次结构。

特点：非线性结构，一个直接前驱，但可能有多个直接后继（1：n）

- 公司组织结构
- 家
- 计算机中目录



结点：即树的数据元素

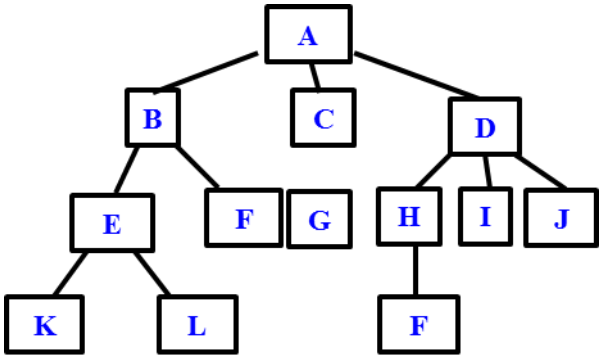
结点的度：一个结点所拥有的后继的个数（有几个直接后继就是几度）

根：即根结点(没有前驱)

叶子：即终端结点(没有后继)

树的度：所有结点度中的最大值（Max{各结点的度}）

树的深度(或高度)：指所有结点中最大的层数（Max{各结点的层次}）



上图中的结点数＝13 ；树的度＝3 ；树的深度＝4

定义：树(Tree)是 n(n>=0)个结点的有限集 T，T 为空时称为空树，否则它满足如下两个条件：

（1）有且仅有一个特定的称为根(Root)的结点；

(2)其余的结点可分为 m(m>=0)个互不相交的子集 T1,T2,T3…Tm，其中每个子集又是一棵树，并称其为子树(Subtree)。

6.2 二叉树

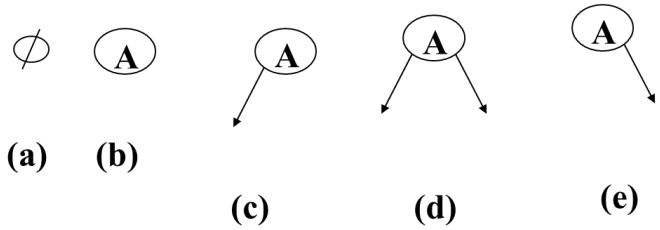
二叉树或为空树，或是由一个根结点加上两棵分别称为左子树和右子树的、互不交的二叉树组成。

基本特征:

① 每个结点最多只有两棵子树（不存在度大于 2 的结点）；

② 左子树和右子树次序不能颠倒。

二叉树的五种基本形态：



(a) 空二叉树

(b) 仅有根结点的二叉树

(c) 右子树为空的二叉树

(d) 左、右子树均为非空的二叉树

(e) 左子树为空的二叉树

二叉树的定义

定义：二叉树是由 n(n>=0)个结点的有限集合构成，此集合或者为空集，或者由一个根结点及两棵互不相交的左右子树组成，并且左右子树都是二叉树。

这也是一个递归定义。二叉树可以是空集合，根可以有空的左子树或空的右子树。二查树不是树的特殊情况，它们是两个概念。

两种特殊的二叉树

1.满二叉树：一颗树中所有叶结点均在同一阶层，而其它非终端结点的分支度均为 2，则此树为一颗满二叉树。

若该树的高度为 h，则此满二叉树的结点为 2h-1。

2. 完全二叉树：如果一颗二叉树只有最下一层结点数可能未达到最大，并且最下层结点都集中在该层的最左端，则称为完全二叉树；

二叉树的性质

性质 1

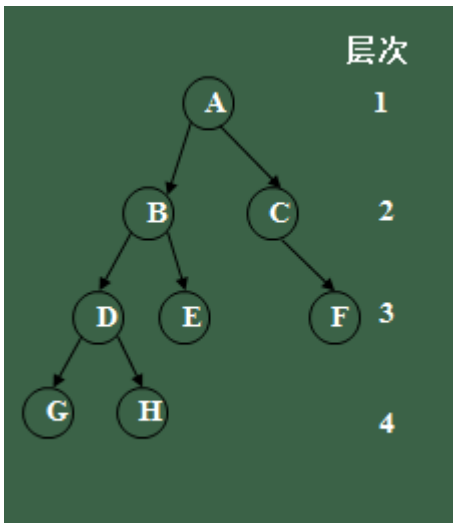
在二叉树的第 i 层上最多有 2i-1 个结点

性质 2

深度为 k 的二叉树最多有 2k-1 个结点

性质 3

设二叉树叶子结点数为 n0，度为 2 的结点 n2，则 n0 = n2 +1



性质 4 具有 n 个结点的完全二叉树的深度为：

$$\lfloor \log_2 n \rfloor + 1$$

对完全二叉树的结点编号：从上到下，从左到右

性质 5：在完全二叉树中编号为 i 的结点

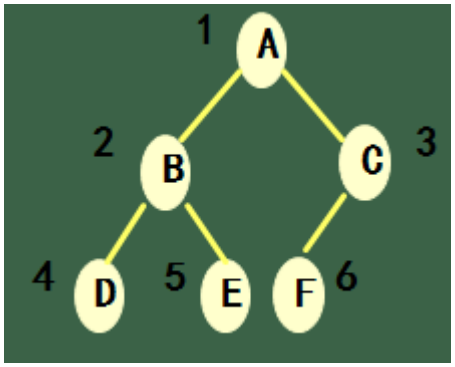
1）若有左孩子，则左孩编号为 2i

2）若有右孩子，则右孩子结点编

号为 2i+1

3）若有双亲，则双亲结点编号为

(i/2)



二叉树的存储结构

即用一组连续的存储单元存储二叉树的数据元素。因此，必须把二叉树的所有结点安排成为一个恰当的序列，结点在这个序列中的相互位置能反映出结点之间的逻辑关系，用编号的方法：

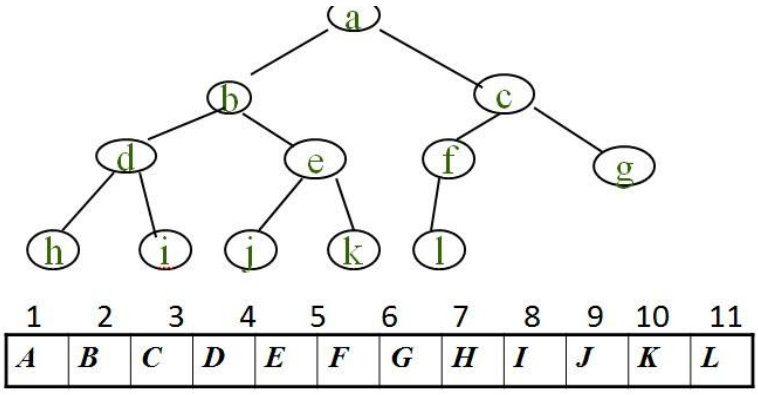
#define max-tree-size　100

Typedef telemtype sqbitree[max-tree-size];

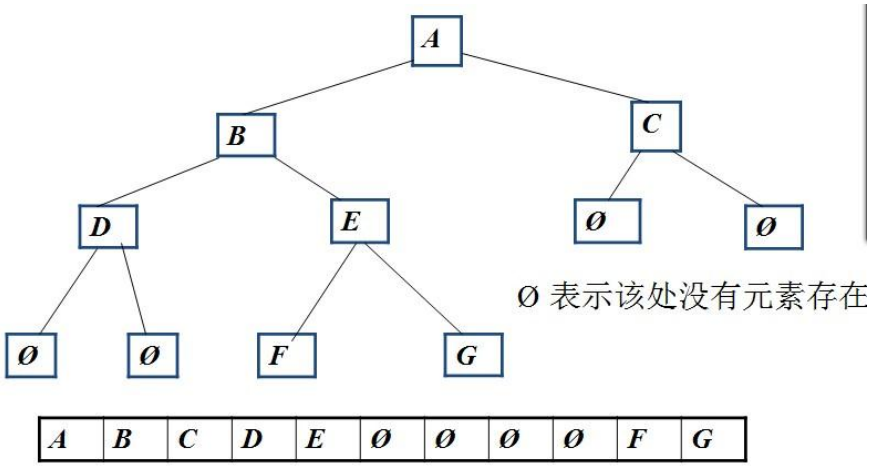
Sqbitree　bt

从树根起，自上层至下层，每层自左至右的给所有结点编号。

完全二叉树

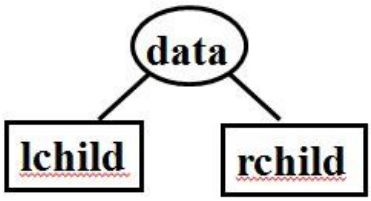


一般二叉树



A	B	C	D	E	Ø	Ø	Ø	Ø	F	G

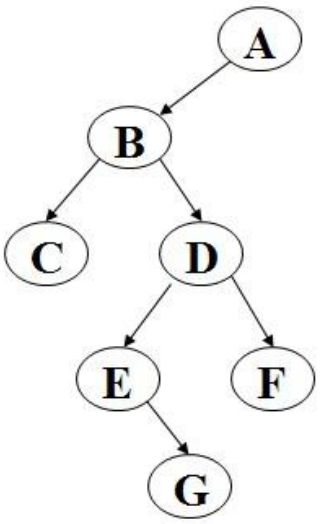
2. 二叉树的链式存储结构

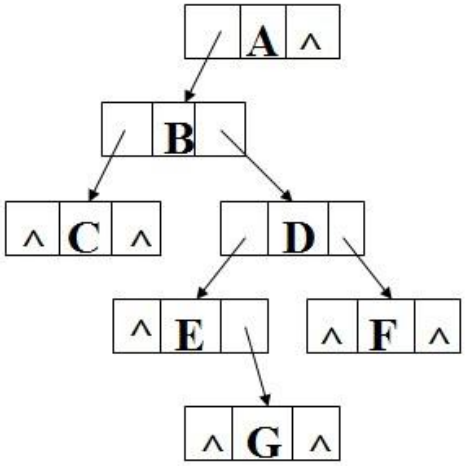


二叉链表中每个结点包含三个域：

1. 数据域
2. 左指针域
3. 右指针域

二叉链表表示





二叉树的链表存储表示

```
Typedef struct BiTNode   {
        TelemType data;
         struct BiTNode *lchild,*rchild;
} BiTNode,*BiTree;
```

有时也可用数组的下标来模拟指针,即开辟三个一维数组 Data ,lchild,rchild 分别存储结点的元素及其左,右指针域;

遍历二叉树

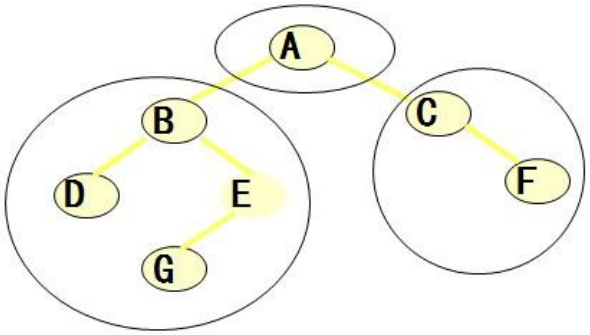
遍历：按某种搜索路径访问二叉树的每个结点，而且每个结点仅被访问一次。

二叉树的遍历方法

约定先左后右,有三种遍历方法：先序遍历、中序遍历、后序遍历

二叉树由根、左子树、右子树三部分组成

二叉树的遍历可以分解为：访问根，遍历左子树和遍历右子树



先序遍历（TLR）

若二叉树非空，

（1）访问根结点；

（2）先序遍历左子树；

（3）先序遍历右子树；

先序遍历序列：A,B,D,E,G,C,F

中序遍历（LTR）

若二叉树非空

（1）中序遍历左子树

（2）访问根结点

（3）中序遍历右子树

遍历序列： D,B,G,E,A,C,F

后序遍历（LRT）

若二叉树非空

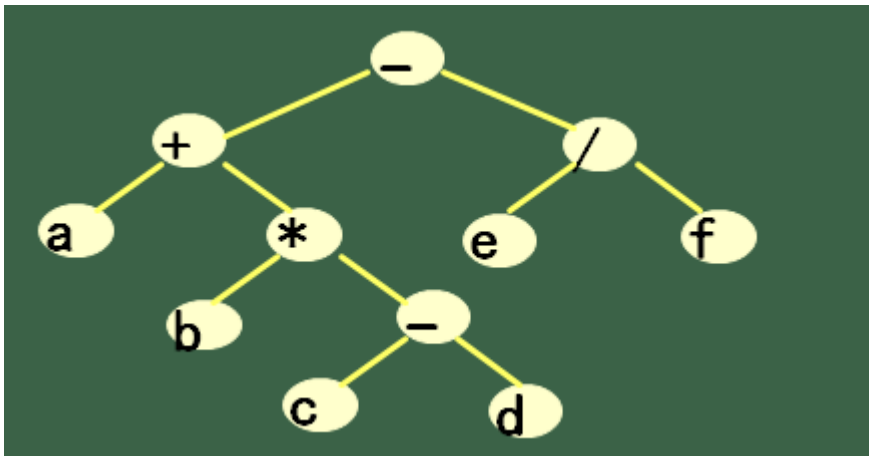
（1）后序遍历左子树

（2）后序遍历右子树

（3）访问根结点

后序遍历序列： D,G,E,B,F,C,A

例：先序遍历、中序遍历、后序遍历下图所示的二叉树



后序遍历序列：a,b,c,d,-,*,+,e,f,/,-

中序遍历序列：a,+,b,*,c,-,d,-,e,/,f

先序遍历序列：-,+,a,*,b,-,c,d,/,e,f

6.3 什么是递归调用？

递归调用就是函数自己调用自己，以实现逐层深入的访问。递归可以使代码更简洁清晰，

可读性更好。但对于初学者理解难度加大。

例：计算 n 的阶乘

```
long fact(long n)
    {
    if(n==0||n==1) return 1;
    else return n*fact(n-1);
    }
```

计算 3！

第一层

fact（3）

第二层

fact（2）

第三层

fact（1）

返回 1

第二层

返回 2×1=2

第一层

返回 3×2=6

通过递归建立树？

```
TREENODE *creat_tree() //创建树
{
                TREENODE   *t;
                 char c;
                 c=getchar();
   if(c= =‘#’)     return(NULL);
   else{
   t=(TREENODE *)malloc(sizeof(TREENODE))
   t  -  >data=c;
   t  - >lchild=create_tree();
```

```
    t  ->rchild=create  -tree(); }
       return(t);
  }
```

中序遍历算法:

```
#include<stdio.h>
#include<stdlib.h>
#define NULL 0
Typedef struct node{                    //树节点的定义
          char data;
          struct node *lchild,*rchild;
      }TREENODE;
TREENODE *root; //定义根节点
TREENODE *creat_tree(); //建立树
Void inorder(TREENODE *);
Void inorder(TREENODE *p)     //中序遍历过程
   {
         if(p!=NULL)
        { inorder(p ->lchild);
             printf("%c",p ->data)
            inorder(p ->rchild);
     }
}
```

【练习】树最适合用来表示 C 。

A．有序数据元素

B．无序数据元素

C．元素之间具有分支层次关系的数据

D．元素之间无联系的数据

【练习】深度为 5 的二叉树至多有 A 个结点。

A．16　　B．32

C．31　　D．10

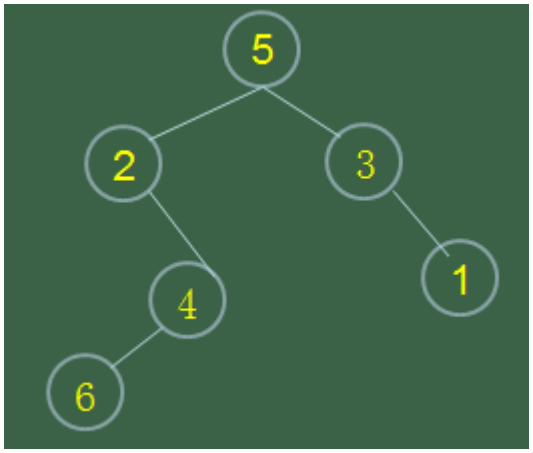
【练习】右图中序遍历的顺序是？（ ）

A. 264531

B. 462513

C. 524631

D. 523461



7. 图

.图状结构是一种比树形结构更复杂的非线性结构。在树状结构中，结点间具有分支层次关系，每一层上的结点只能和上一层中的至多一个结点相关，但可能和下一层的多个结点相关。

而在图状结构中，任意两个结点之间都可能相关，即结点之间的邻接关系可以是任意的。因此，图状结构被用于描述各种复杂的数据对象，在自然科学、社会科学和人文科学等许多领域有着非常广泛的应用。

7.1 图的定义和术语

1.基本术语

(1)顶点

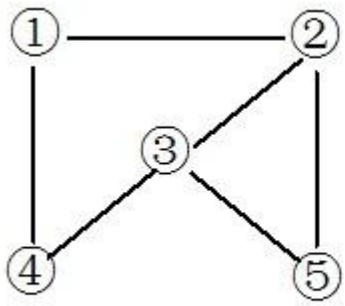
图中的数据元素通常称为顶点。V 是顶点的有穷非空集合；VR 是两个顶点之间的关系的集合。

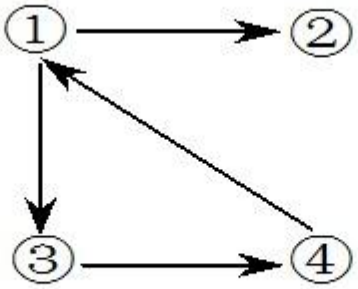
(2)有向图

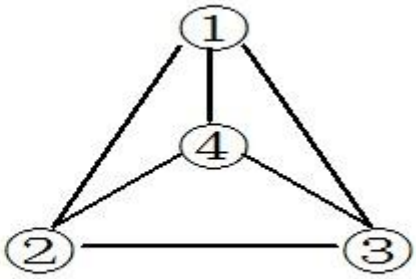
若称 <v,w>∈VR 表示从 v 到 w 的一条弧，且称 v 为弧尾或初始点，称 w 为弧头或终端结点，则该图称为有向图。

(3)无向图

若<v,w>∈VR，必有<w,v>∈VR，即 VR 是对称的，则以无序对（v,w)代替这两个有序对，表示 v 和 w 之间的一条边，则该图称为无向图。







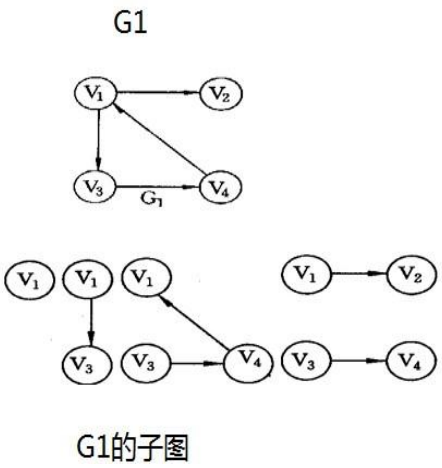
(4)权/网

有时图的边或弧具有与它相关的数，这些数称为权值（通常表示顶点间的距离或耗费），则带权值的图称为网。

(5)子图

假设有两个图 G=（V，{VR}）和 G’=（V’，{VR’ }），若 V’ 是 V 的子集，且 VR’ 是 VR 的子集，则称 G’ 为 G 的子图。

子图示例



(6)完全图

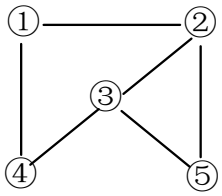
对于无向图，有 (n(n-1))/2 条边的无向图称为完全图。对于有向图，有 n(n-1) 条弧的有向图称为有向完全图。

(7)稀疏图/稠密图

边或弧很少的图称稀疏图，反之称稠密图。

(8)邻接点

对于无向图 G=(V,{E})，若边(v,v’)∈E，则称顶点 v 和 v’ 互为邻接点，即 v 和 v’ 相邻接。或称边（v,v’）依附于顶点 v 和 v’，或称（v,v’）和顶点 v 和 v’ 相关联。



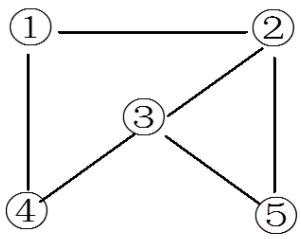
(9)顶点 v 的度(Degree)

是和 v 相关联的边的数目，记为 TD（v）。

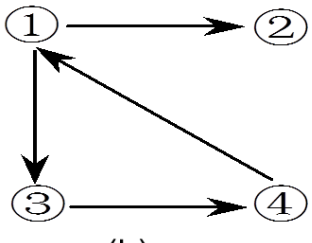
顶点的入度/出度

以顶点 v 为头的弧的数目称 v 的入度，记为 ID(v)；以顶点 v 为尾的弧的数目称 v 的出度，记为 OD(v)。

顶点 v 的度 TD（v）=ID（v）＋OD（v）



(a)

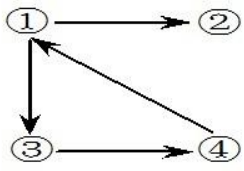


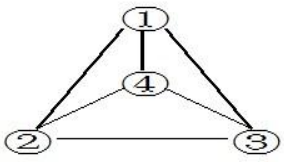
(b)

(10)路径(Path)

无向图 G=(V,{E})中，从顶点 v 到 v’的路径是顶点序列(v=vi0,vi1,…,vim=v’)，其中(vij-1 ,vij)∈E

，1≤j≤m。





若 G 是有向图，则路径也是有向的，顶点序列应满足：<vij-1 ,vij>∈E ，1≤j≤m。

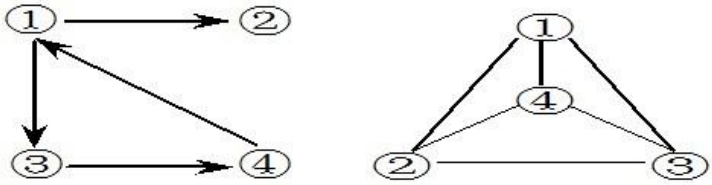
路径的长度是路径上的边或弧的数目。

(11)回路/环/简单路径

第一个顶点和最后一个顶点相同的路径称为回路/环。

序列中顶点不重复出现的路径称为简单路径。

除了第一个顶点和最后一个顶点之外，其余顶点不重复出现的回路，称为简单回路或简单环。



(12)连通图/连通分量

在无向图 G 中，如果从顶点 V 到顶点 V’ 有路径，则称 V 和 V’ 是连通的。

若图中任意两个顶点 vi、vj∈V，vi 和 vj 都是连通的，则称 G 是连通图。

无向图中的极大连通子图称之为连通分量。

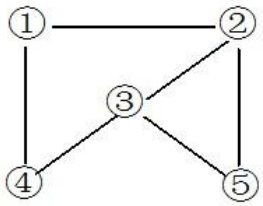


图1 连通图

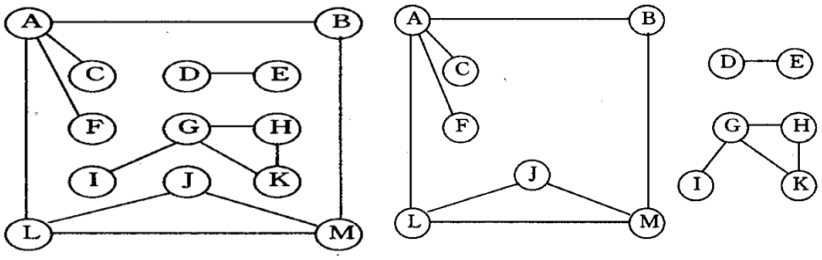
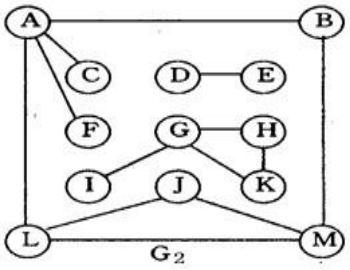


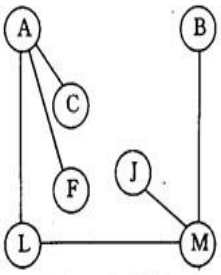
图2 非连通图，但有三个连通分量

(13)生成树

一个连通图的生成树是一个极小连通子图，它含有图中全部顶点，但只有足以构成一棵树的 n-1 条边。

如果在一棵生成树上添加一条边，必定构成一个环，因为这条边使得它依附的那两个顶点之间有了第二条路径。





左图的最大连通分量的一棵生成树

图的抽象类型定义

ADT Graph {

数据对象 V：V 是具有相同特性的数据元素的集合，称为顶点集。

数据关系 R：R={ VR }

VR={<v,w>|v,w∈V 且 P(v,w)，<v,w>表示从 v 到 w 的弧，谓词 P(v,w)定义了弧<v,w>的意义或信息 }

基本操作 P：

}ADT Graph

基本操作

CreateGraph(&G,V,VR); // 按 V 和 VR 的定义构造图 G

DestroyGraph(&G); // 销毁图 G

LocateVex(G, u); // 若 G 中存在顶点 u，则返回该顶点在图中位置；否则返回其它信息

GetVex(G, v); // 返回 v 的值

PutVex(&G, v, value); // 对 v 赋值 value

FirstAdjVex(G, v); // 返回 v 的第一个邻接点。若 v 在 G 中没有邻接点，则返回“空”

NextAdjVex(G, v, w); // 返回 v 的(相对于 w 的)下一个邻接点。若 w 是 v 的最后一个邻接点，则“空”

InsertVex(&G, v); // 在图 G 中增添新顶点 v。

DeleteVex(&G, v); // 删除 G 中顶点 v 及其相关的弧

InsertArc(&G, v, w); // 在 G 中增添弧<v,w>，若 G 是无向的，则还增添对称弧<w,v>

DeleteArc(&G, v, w); // 在 G 中删除弧<v,w>，若 G 是无向的，则还删除对称弧<w,v>

DFSTraverse(G, v, Visit());

// 从顶点 v 起深度优先遍历图 G，并对每个顶点调用

// 函数 Visit 一次且仅一次。

BFSTraverse(G, v, Visit());

// 从顶点 v 起广度优先遍历图 G，并对每个顶点调用

// 函数 Visit 一次且仅一次。

7.2 图的存储结构

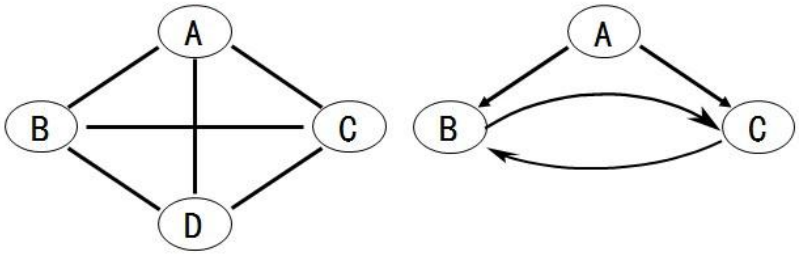
1. 图的数组(邻接矩阵)存储表示
2. 图的邻接表存储表示
3. 有向图的十字链表存储表示
4. 无向图的邻接多重表存储表示

图的数组(邻接矩阵)存储表示

邻接矩阵是用于描述图中顶点之间关系(即弧或边的权)的矩阵。

假设图中顶点数为 n，则邻接矩阵 An×n：

$$A[i][j]=\begin{cases}1 & \text{若 Vi 和 Vj 之间有弧或边}\\ 0 & \text{反之}\end{cases}$$



$$A=\begin{array}{c}\begin{matrix}A & B & C & D\end{matrix}\\ \begin{pmatrix}0&1&1&1\\1&0&1&1\\1&1&0&1\\1&1&1&0\end{pmatrix}\end{array}\begin{matrix}A\\B\\C\\D\end{matrix}\qquad B=\begin{array}{c}\begin{matrix}A & B & C\end{matrix}\\ \begin{pmatrix}0&1&1\\0&0&1\\0&1&0\end{pmatrix}\end{array}\begin{matrix}A\\B\\C\end{matrix}$$

注意：

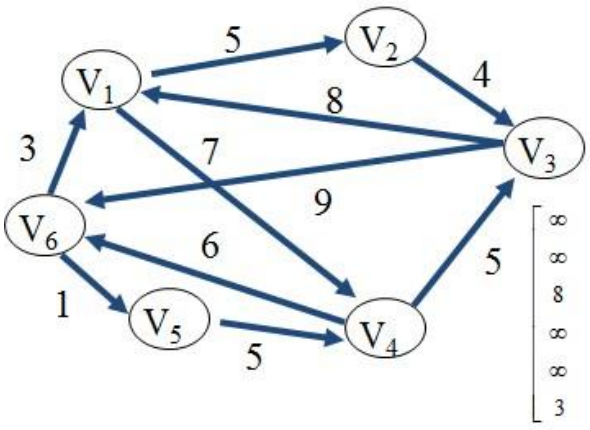
1) 图中无邻接到自身的弧，因此邻接矩阵主对角线为全零。

2) 无向图的邻接矩阵必然是对称矩阵。

3) 无向图中，顶点的度是邻接矩阵对应行（或列）的非零元素之和

网的邻接矩阵

$$A[i][j] = \begin{cases} \infty & \text{反之} \\ \text{权值} & \text{若 Vi 和 Vj 之间有弧或边} \end{cases}$$



$$\begin{bmatrix} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{bmatrix}$$

图的数组（邻接矩阵）存储表示

7.3 图的遍历

从图中某一顶点出发访遍图中其余顶点，且使每一个顶点仅被访问一次。这一过程就叫做图的遍历。

通常有两条遍历图的路径：

- 深度优先搜索
- 广度优先搜索

1.深度优先搜索(DFS)

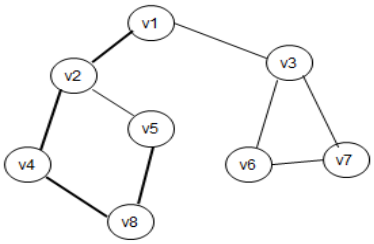
基本思想：

从图中某顶点 V0 出发，访问此顶点，然后依次从 V0 的各个未被访问的邻接点出发深度优先搜索遍历图，直至图中所有和 V0 有路径相通的顶点都被访问到；

若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点；

重复上述过程，直至图中所有顶点都被访问到为止。

从顶点v1出发，DFS下图。



顶点访问序列为：v1,v2,v4,v8,v5,v3,v6,v7
或 v1,v2,v5,v8,v4,v3,v6,v7
或 v1,v3,v7,v6,v2,v4,v8,v5

2.广度优先搜索(BFS)

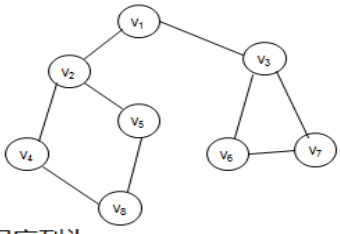
基本思想：

从图中某个顶点 V0 出发，并在访问此顶点后依次访问 V0 的所有未被访问过的邻接点，之后按这些顶点被访问的先后次序依次访问它们的邻接点，直至图中所有和 V0 有路径相通的顶点都被访问到；

若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点；

重复上述过程，直至图中所有顶点都被访问到为止。

从顶点v_1出发，BFS下图。



顶点访问序列为：$v_1,v_2,v_3,v_4,v_5,v_6,v_7,v_8$
或 v1,v3,v2,v6,v7,v5,v4,v8

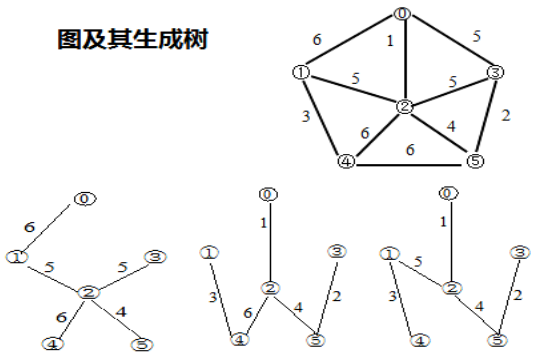
7.4 图的连通性

基本概念

连通分量的顶点集：即从该连通分量的某一顶点出发进行搜索所得到的顶点访问序列；

生成树：某连通分量的极小连通子图；

生成森林：非连通图的各个连通分量的极小连通子图构成的集合。



设 E(G)为连通子图 G 中所有边的集合，则从图中任一顶点出发遍历图时，必定将 E(G)分成两个集合 T(G)和 B(G)，其中 T(G)是遍历过程中历经的边的集合。

显然，T(G)和图 G 中所有顶点一起构成连通图 G 的极小连通子图，按照 7.1 节的定义，它是连通图的一棵生成树，并且称由深度优先搜索得到的为深度优先生成树；由广度优先搜索得到的为广度优先生成树。

【练习】连通图 G 中的边集 E={(a，b)，(a，e)，(a，c)，(b，e)，(e，d)，(d，f)，(f，c)}，则从顶点 a 出发可以得到一种深度优先遍历的顶点序列为（ ）。

(A) abedfc

(B) acfebd

(C) aebdfc

(D) aedfcb

8. 查找

8.1 基本概念

1.查找的定义

给定一个值 k，在含有 n 个结点的表（或文件）中找出关键字等于给定值 k 的结点，若找到，则查找成功，输出该结点在表中的位置；否则查找失败，输出查找失败的信息。

2.查找表

查找表是由具有同一类型（属性）的数据元素（记录）组成的集合。分静态查找表和动态查找表两类。

静态查找表：仅对查找表进行查找操作，而不改变查找表中的数据元素；

动态查找表：对查找表除进行查找操作外，可能还要进行向表中插入数据元素，或删除表中数据元素的表。

平均查找长度(Average Search Length)

为确定记录在查找表中的位置，需和给定值进行比较的关键字个数的期望值称为查找算法在查找成功时的平均查找长度（ASL）。

对于含有 n 个记录的表，查找成功时的平均查找长度为：

$$ASL = \sum_{i=1}^{n} cipi$$

Ci:表中关键字与给定值相等的第 i 个记录时，和给定值已进行过比较的关键字个数

Pi:查找表中第 i 个记录的概率，各记录查找概率相等时， Pi=1/n

8.2 静态查找表顺序表的查找

1.查找过程：

对给定的一关键字 K，从线性表的一端开始，逐个进行记录的关键字和 K 的比较，直到找到关键字等于 K 的记录或到达表的另一端。

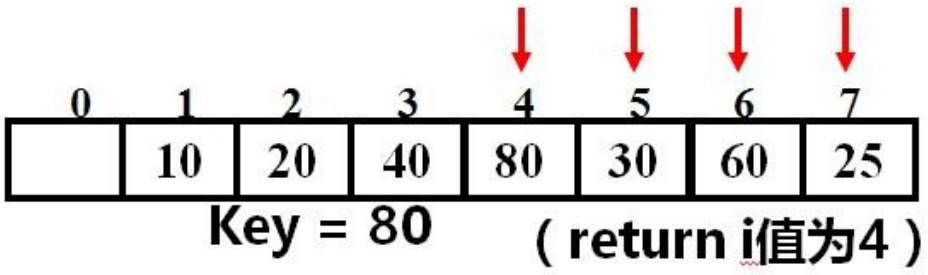
2.实现顺序查找的数据结构

```
typedef   struct{
   int      key;   //关键字域
   ……   //其他域
}SSTable;
```

3.顺序查找的算法

```
int seqsearch(SSTable ST[ ], int n, int key){
    int i=n;
    while(ST[i].key!=key)&&(i>=1)   i- -;
    return   i;
}
```

（return i 值为 4）



4.算法分析

查找成功时的平均查找次数为：

ASL=(1+2+3+4+……+n)/n = (n+1)/2 ①

查找不成功时的比较次数为：

ASL=(n(n+1))/n = n+1 ②

则顺序查找的平均查找长度为：

ASL==(① + ②)/2 = 3(n+1)/4

优点：算法简单，无需排序，采用顺序和链式存储均可。

缺点：平均查找长度较大。

有序表的查找——二分查找

可用折半查找（二分法查找）来实现。

1.查找思想

先确定待查找记录所在的范围，然后逐步缩小范围，直到找到或确认找不到该记录为止。

前提条件：

必须在具有顺序存储结构的有序表中进行。

二分查找

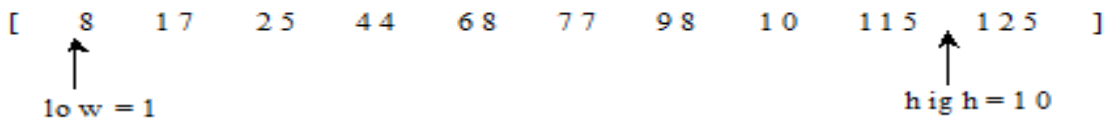
首先将待查找的 k 值和有序表 R[0]到 R[n－1]的中间位置 mid 上的结点的关键字进行比较，若相等，则查找完成；否则，若 R[mid].key＞k，则说明待查找的结点在左子表 R[0]到 R[mid－1]中， 只要在左子表中继续进行二分查找；若 R[mid].key＜k，则说明待查找的结点在右子表 R[mid＋1]到 R[n－1]中，只要在右子表中继续进行二分查找。

这样，经过一次关键字的比较就缩小一半查找区间。如此进行下去，直到找到关键字为 k 的结点。

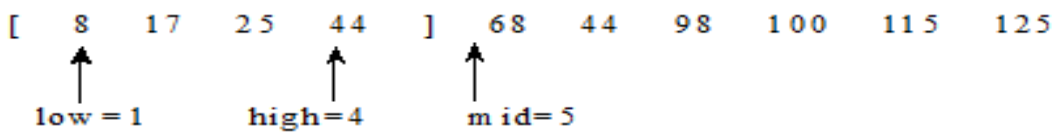
二分查找要求线性表是有序表，即表中的结点按关键字有序，并且要用顺序表作为表的存储结构。

例如，假设给定有序表中 10 个关键字为 8,17,25,44,68,77,98,100,115,125，将查找 K=17 和 K=120 的情况描述为图 1 及图 2 形式。

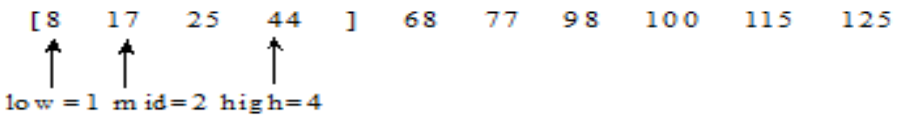
查找 K=17 mid ＝ （1＋10）/2＝ 5



(a) 初始情形

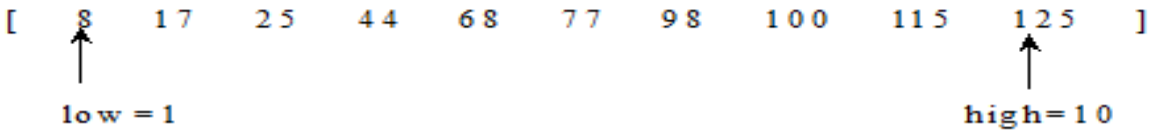


(b) 经过一次比较后的情形

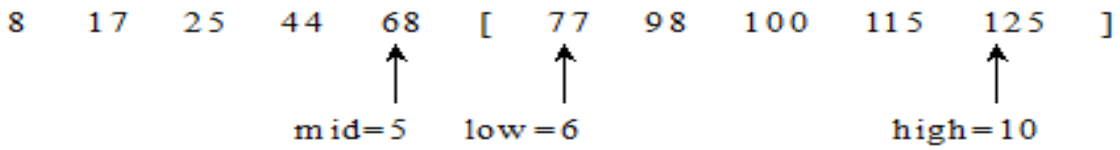


(c) 经过二次比较后的情形 (v[mid].key=17)

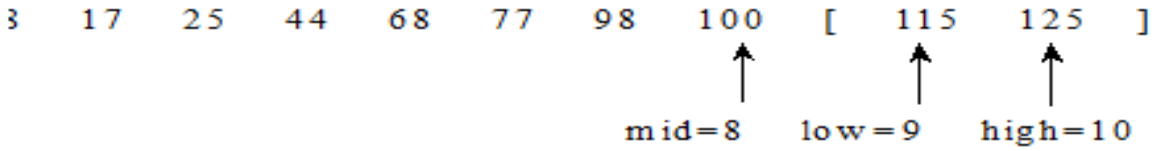
图 1 查找 K=17 的示意图 (查找成功)



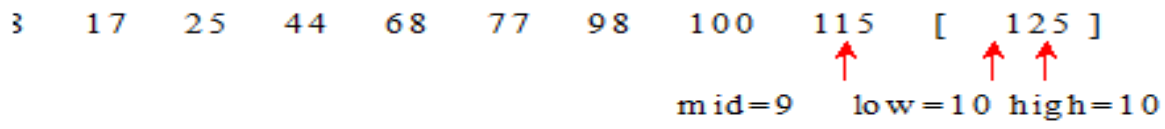
(a) 初始情形



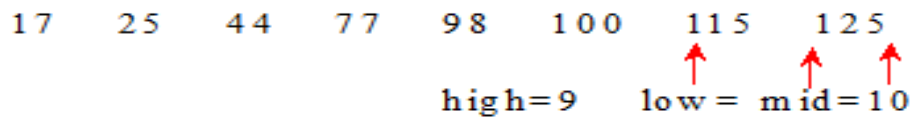
(b) 经过一次比较后的情形



(c) 经过二次比较后的情形



(d) 经过三次比较后的情形



(e) 经过四次比较后的情形(high<low)

图 2 查找 K=120 的示意图 (查找不成功)

2. 二分查找的算法

```
int Search_Bin( SSTable ST[ ], int n, int    key)
{   int low, high, mid;
   low=1; high=n;
   while(low<=high)
   {     mid=(low+high)/2;
         if(ST[mid].key= = key)    return mid;
         else
              if( key< ST[mid].key)    high=mid-1;
              else    low=mid+1;
   }
   return 0;
}
```

8.3 动态查找表

特点

表结构在查找过程中动态生成。

要求

对于给定值 key，若表中存在其关键字等于 key 的记录，则查找成功返回；否则插入关键字等于 key 的记录。

动态查找表主要有二叉树结构和树结构两种类型。二叉树结构有二叉排序树、平衡二叉树等。树结构有 B-树、B+树等。

二叉排序树(Binary Sort Tree)

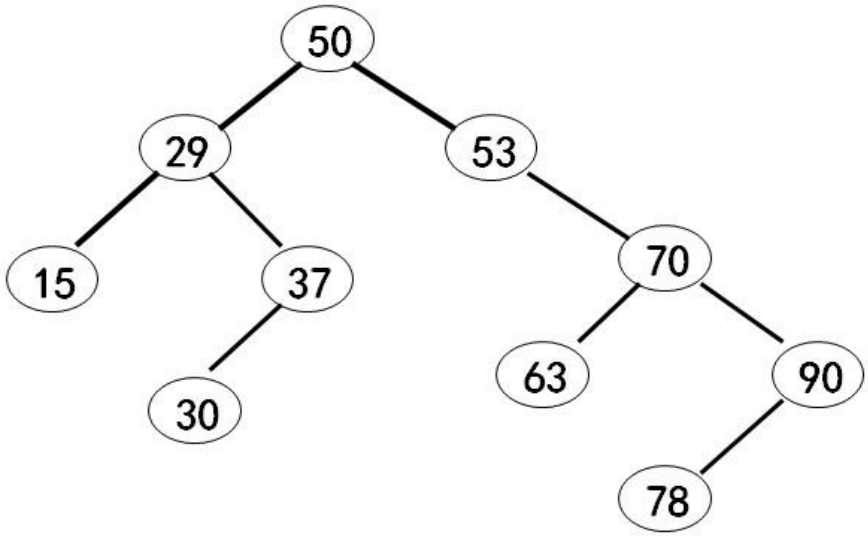
一棵空树；或是具有下列性质的二叉树：

（1）若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值；

（2）若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值；

（3）它的左、右子树也分别为二叉排序树。

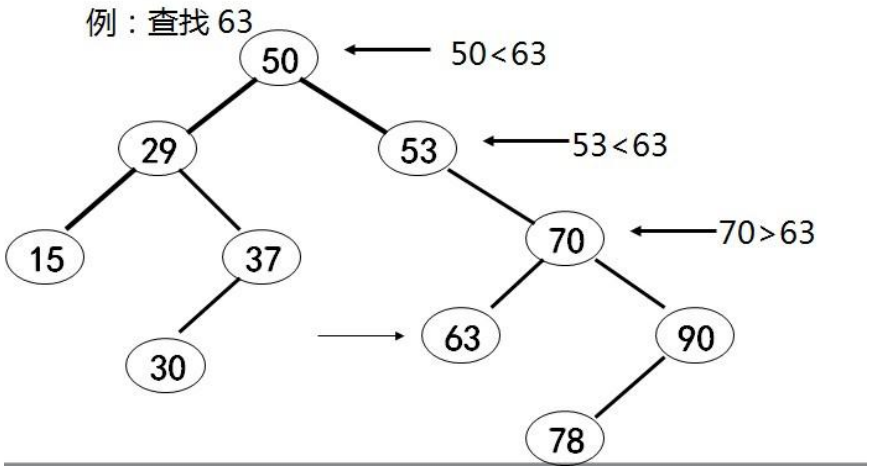
二叉排序树实例：



二叉排序树的数据类型描述

```
struct    BtreeNode {
    ElemType    key;    //关键字
    BtreeNode *lchild,*rchild;//左、右孩子
} BtreeNode,    *Bitree;
```

二叉排序树的查找



说明：为查找 63，在二叉排序树中做以下比较：

50<63

53<63

70>63

二叉排序树的查找算法

```
BiTree SearchBST (BiTree T, KeyType key)
{
   if( (T==NULL)|| (key==T->key))   return (T); //查找结束
    else
        if (key < T->key)
                return(SearchBST(T->lchild, key));
                                        //在左子树中继续查找
        else
              return(SearchBST(T->rchild, key));
                                        // 在右子树中继续查找
}//SearchBST
```

【练习】对于长度为 9 的有序顺序表，若采用折半搜索，在等概率情况下搜索成功的平均搜索长度为(A)的值除以 9。

A、20

B、18

C、25

D、22

提示：对各个位置查找的次数：3 2 3 4 1 3 2 3 4，其和为 25

9. 排序

9.1 排序的基本概念

1.排序

设含有 n 个记录的文件｛R_1，R_2，…R_n｝，相应的关键字为｛K_1，K_2，…K_n｝，需确定一种排列 P(1),P(2)…P(n)使其相应的关键字满足递增(或递减)关系：

$K_{P(1)} \leqslant K_{P(2)} \leqslant \cdots K_{P(n)}$ 或 $K_{P(1)} \geqslant K_{P(2)} \geqslant \cdots K_{P(n)}$

使上述文件成为一个按其关键字线性有序的文件｛$R_{P(1)}$，$R_{P(2)}$ ，…$R_{P(n)}$｝，这种运算就称为排序。

将数据元素的无序序列调整为有序序列的过程。

2.排序算法的稳定性

排序码（Key）

作为排序依据的记录中的一个属性。它可以是任何一种可比的有序数据类型，它可以是记录的关键字，也可以是任何非关键字。

如果待排序的序列中，存在多个具有相同排序码的数据元素，若经过排序这些数据元素的相对次序保持不变，则称这种排序算法是稳定的，若经过排序，这些数据元素的相对次序发生了改变，则称这种排序算法是不稳定的。

3.内部排序与外部排序

内部排序

指当文件的数据量不太大时，全部信息放内存中处理的排序方法。

外部排序

当文件的数据量较大时，排序过程中需要在内、外存之间不断地进行数据交换才能达到排序的目的，这种排序称为外排序。

4.内部排序的方法

内部排序的过程是一个逐步扩大记录的有序区长度的过程。在排序的过程中，参与排序的记录序列中存在两个区域：有序区和无序区。使有序区中记录的数目增加一个或几个的操作称为一趟排序。

内部排序的方法很多，每种方法都有各自的优缺点，适合在不同的环境(如记录的初始排列状态等)下使用。

按内部排序过程中所需的工作量来区分：

排序:

简单排序方法，其时间复杂度为 O(n2)

先进排序方法，其时间复杂度为 O(nlogn)

按排序过程中依据的原则分

插入类（直插排序、二分排序、希尔排序）

交换类（冒泡排序、快速排序）

选择类（直选排序、树型排序、堆排序）

归并类（二路归并排序、多路归并排序）

分配类（多关键字排序、基数排序）

平均时间复杂度由高到低为

冒泡排序 $O(n^2)$

插入排序 $O(n^2)$

选择排序 $O(n^2)$

归并排序 O(nlogn)

堆排序 O(nlogn)

快速排序 O(nlogn)

希尔排序 $O(n^{1.25})$

基数排序 O(n)

对待排序记录类型的顺序表描述:

```
#define MAXSIZE   20     //一个顺序表的最大长度
typedef int KeyType;   //定义关键字为整数类型
typedef struct{
    KeyType key;          //关键字项
    InfoType otherinfo;       //其他数据项
}RedType;   //记录类型
typedef struct{
    RedType r[MAXSIZE+1];   //r[0]闲置或用作哨兵单元
```

```
    int length;          //顺序表长度
}SqList; //顺序表类型
```

9.2 插入排序

— 直接插入排序

— 折半插入排序

1. 直接插入排序

1）一趟直接插入排序的基本思想

将记录 L.r[i]插入到有序子序列 L.r[1..i-1]中，使记录的有序序列从 L.r[1..i-1]变为 L.r[1..i]。

完成这个“插入”分三步进行：

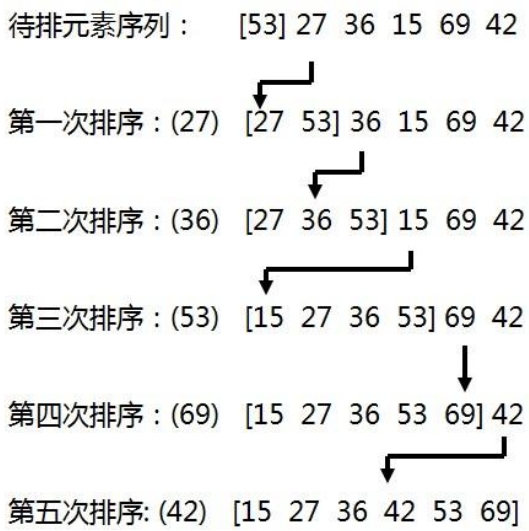
1．查找 L.r[i]在有序子序列 L.r [1..i-1]中的插入位置 j；

2．将 L.r [j..i-1]中的记录后移一个位置；

3．将 L.r [i]复制到 L.r [j]的位置上。

整个排序过程进行 n－1 趟插入，即：先将序列中的第 1 个记录着成一个有序的子序列，然后从第 2 个记录起逐个插入，直至整个序列变成接关键字非递减有序序列为止。

直接插入排序示例（插入操作要进行 n-1 次）



2）直接插入排序算法描述

```
void InsertionSort ( SqList &L )
{// 对记录序列 R[1..L.length]作直接插入排序。
  for ( i=2; i<=L.length; ++i )
   {   L.r[0] = L.r[i]; // 复制为监视哨
       for(j=i-1;L.r[0].key < L.r[j].key; j-- )
```

```
            L.r[j+1] = L.r[j]; //  记录后移
        L.r[j+1] = L.r[0]; //  插入到正确位置
    }
} // InsertSort
```

3）直接插入排序性能分析

实现排序的基本操作有：

（1）“比较” 关键字的大小

（2）“移动”记录

对于直接插入排序：

最好情况“比较”次数：n-1；“移动”次数：2(n-1)

最坏的情况“比较”和“移动”的次数均达到最大值，分别为：(n+2)(n-1)/2；(n+4)(n-1)/2

由于待排记录序列是随机的，取上述二值的平均值。所以直接插入排序的时间复杂度为O(n2)。

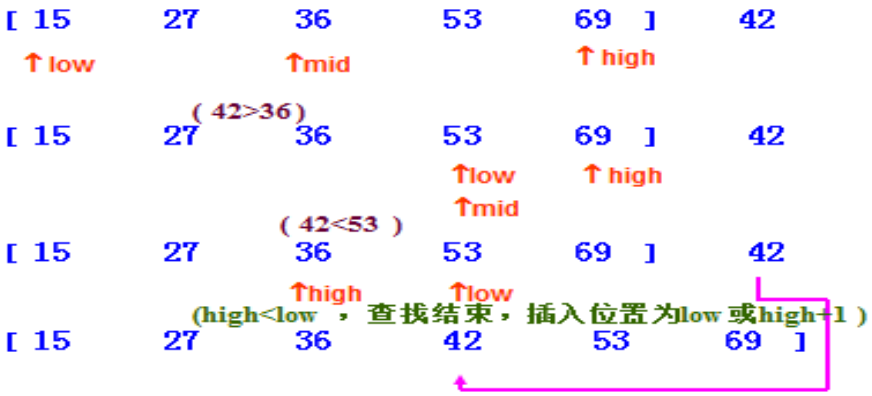
直接插入排序是“稳定的”：关键码相同的两个记录，在整个排序过程中，不会通过比较而相互交换。

2.折半插入排序

1）基本思想

考虑到 L.r[1..i-1] 是按关键字有序的有序序列，则可以利用折半查找实现“ L.r[1…i-1] 中查找 L.r[i] 的插入位置”如此实现的插入排序为折半插入排序。

例：6 个记录中，前 5 个已经排好序，现在对第 6 个记录排序



折半插入排序在寻找插入位置时，不是逐个比较而是利用折半查找的原理寻找插入位置。带排序元素越多，改进效果越明显。

2)折半插入排序算法

```
void BinsertSort(SqList &L)
{   int i,low,high,mid;
```

```
    for(i=2; i<= L.length; ++i)
    {     L.r[0]=L.r[i];
          low=1;    high=i-1;
          While(low<=high)
          {   mid=(low+high)/2;
              if (L.r[0].key< L.r[mid].key) high=mid-1;
              else low=mid+1;     }
        for( j=i-1; j>=low; j           )    L.r[j+1]=L.r[j];
        L.r[low]=L.r[0];
    }
}
```

3)折半插入排序性能分析

折半插入排序减少了关键字的比较次数，但记录的移动次数不变，其时间复杂度与直接插入排序相同,时间复杂度为 O(n2) 。折半插入排序是“稳定的”

9.3 交换排序

1）冒泡排序

小的浮起，大的沉底

具体做法：

第一趟：第 1 个与第 2 个比较，大则交换；第 2 个与第 3 个
比较，大则交换，… 关键字最大的记录交换
到最后一个位置上；

25	**25**	**25**	**25**	**11**	**11**	**11**
56	**49**	**49**	**11**	**25**	**25**	**25**
49	**56**	**11**	**49**	**41**	**36**	**36**
78	**11**	**56**	**41**	**36**	**41**	
11	**65**	**41**	**36**	**49**		
65	**41**	**36**	**56**			
41	**36**	**65**				

36	**78**					
初始状态	**第一趟**	**第二趟**	**第三趟**	**第四趟**	**第五趟**	**第六趟**

第二趟：对前 n-1 个记录进行同样的操作，关键字次大

的记录交换到第 n-1 个位置上；

依次类推，则完成排序。

2）冒泡排序算法分析

正序：

只需进行一趟排序，在排序过程中进行 n-1 次关键字间的比较，且不移动记录；时间复杂度为 O(n) 。

逆序：

需要进行 n-1 趟排序，需要进行 n(n-1)/2 次比较，并作等数量级的记录移动。总的时间复杂度为 O(n2) 。

冒泡排序方法是稳定的。适合于数据较少的情况。

【练习】排序方法中，从未排序序列中依次取出元素与已排序序列中的元素进行比较，将其放入已排序序列的正确位置上的方法，称为 C 。

A.希尔排序

B.冒泡排序

C.插入排序

D.选择排序

【练习】冒泡排序的平均时间复杂度为 D 。（n 为元素个数）

A．O（n)

B．O(log2n)

C．O(nlog2n)

D．O(n2）

华图网校
SINCE 2001 V.HUATU.COM
乐享品质 高效学习

■ 华图网校介绍

华图网校（V.HUATU.COM）于2007年3月由华图教育投资创立，是华图教育旗下的远程教育高端品牌。她专注于公职培训， 目前拥有遍及全国各地500万注册用户，已成为公职类考生学习提高的专业门户网站。

华图网校是教育部中国远程教育理事单位。她拥有全球最尖端高清录播互动技术和国际领先的网络课程设计思想，融汇华图教育十余年公职辅导模块教学法，凭借强大师资力量与教学资源、利用教育与互联网的完美结合，真正为考生带来“乐享品质”的学习体验，通过“高效学习”成就品质人生。

华图网校课程丰富多元，涵盖公务员、事业单位、招警、法院、检察院、军转干、选调生、村官、政法干警、三支一扶、乡镇公务员、党政公选等热门考试、晋升及选拔。同时，华图网校坚持以人为本的原则，不断吸引清华、北大等高端人才加入经营管理，优化课程学习平台，提升用户体验，探索网络教育新技术和教学思想，力争为考生提供高效、个性、互动、智能的高品质课程和服务。

华图网校将秉承“以教育推动社会进步”的使命，加快网站国际化进程，打造全球一流的网络学习平台。

我们的使命：以教育推动社会进步

我们的愿景：德聚最优秀人才，仁就基业长青的教育机构

我们的价值观：诚信为根、质量为本、知难而进、开拓创新。

■ **咨询电话：400-678-1009**
■ **听课网址：v.huatu.com（华图网校）**