

数据结构

适用班级：软件设计师

主讲：张红

网址：[www.bitpx.com](http://www.bitpx.com)

E-Mail:bitpx@163.com

分值说明：早上考3-6分, 下午考15分

比特培训中心

贵州·贵阳

目 录

[第1节 序论 1](#_Toc478744058)

[1.1 什么是数据? 1](#_Toc478744059)

[1.2 什么是数据元素? 1](#_Toc478744060)

[1.3 什么是数据结构及种类? 1](#_Toc478744061)

[1.4 数据的逻辑结构 1](#_Toc478744062)

[1.5 数据的物理结构 1](#_Toc478744063)

[1.6 算法和算法分析 1](#_Toc478744064)

[1.7 算法的五个特性 1](#_Toc478744065)

[1.8 算法设计的要求 2](#_Toc478744066)

[1.9 算法效率的度量 2](#_Toc478744067)

[第2节 线性表 3](#_Toc478744068)

[2.1 线性表举例 3](#_Toc478744069)

[2.2 线性表的存储 4](#_Toc478744070)

[2.3 线性表-栈 4](#_Toc478744071)

[2.4 队列 4](#_Toc478744072)

[2.5 双端队列 6](#_Toc478744073)

[第3节 树和二叉树 6](#_Toc478744074)

[3.1 树 6](#_Toc478744075)

[1.1.2 树的基本概念 6](#_Toc478744076)

[1.1.3 树的常用存储结构 6](#_Toc478744077)

[1.1.4 树的遍历 7](#_Toc478744078)

[3.2 二叉树 7](#_Toc478744079)

[3.3 广义表 13](#_Toc478744093)

[3.4 矩阵的压缩存储 14](#_Toc478744094)

[3.4.1 特殊矩阵 14](#_Toc478744095)

[3.4.2 压缩存储 14](#_Toc478744096)

[第4节 历年真题讲解 15](#_Toc478744097)

[第5节 图 22](#_Toc478744113)

[5.1 图的基本概念 22](#_Toc478744114)

[5.2 图的存储结构 23](#_Toc478744115)

[5.3 图的遍历 25](#_Toc478744116)

[5.4 最小生成树 25](#_Toc478744117)

[5.5 最短路径 26](#_Toc478744118)

[5.6 拓扑排序 27](#_Toc478744119)

[5.7 关键路径 28](#_Toc478744120)

[第6节 历年真题讲解 28](#_Toc478744121)

[第7节 排序 32](#_Toc478744122)

[7.1 排序的功能 32](#_Toc478744123)

[7.2 排序的稳定性 32](#_Toc478744124)

[7.3 排序的类别 32](#_Toc478744125)

[7.4 插入排序 33](#_Toc478744126)

[7.4.1 直接插入排序 33](#_Toc478744127)

[7.4.2 希尔排序 33](#_Toc478744128)

[7.5 选择排序 33](#_Toc478744129)

[7.5.1 直接选择排序 33](#_Toc478744130)

[7.5.2 堆排序 33](#_Toc478744131)

[7.6 交换排序 34](#_Toc478744132)

[7.6.1 冒泡排序 34](#_Toc478744133)

[7.6.2 快速排序 34](#_Toc478744134)

[7.7 归并排序 34](#_Toc478744135)

[7.8 基数排序 35](#_Toc478744136)

[7.8.1 链式基数排序 35](#_Toc478744137)

[7.9 各种内部排序方法的比较 35](#_Toc478744138)

[第8节 查找 35](#_Toc478744139)

[8.1 8.1顺序查找 35](#_Toc478744140)

[8.2 二分查找(有序表的查找) 36](#_Toc478744141)

[8.3 分块查找 37](#_Toc478744142)

[8.4 散列表 38](#_Toc478744143)

[8.4.1 散列函数的构造方法 38](#_Toc478744144)

[8.4.2 冲突的解决 38](#_Toc478744145)

[第9节 历年真题讲解 39](#_Toc478744146)

# 序论

## 什么是数据?

所有能输入到计算机中并能够被计算机程序处理的符号的总称,它是计算机程序加工的原料。

## 什么是数据元素?

数据的基本单位,在计算机程序中通常作为一个整体来进行考虑和处理。如数组中一个存储单元里面的数或者链表中一个结点。

## 什么是数据结构及种类?

数据元素相互之间存在的一种或多种特定关系的集合。主要研究数据逻辑结构和存储结构及其运算的实现。

数据结构的种类:

* 集合：结构中的数据元素之间除了“同属于一个集合”的关系外，别无其它关系，如图 1.1所示。

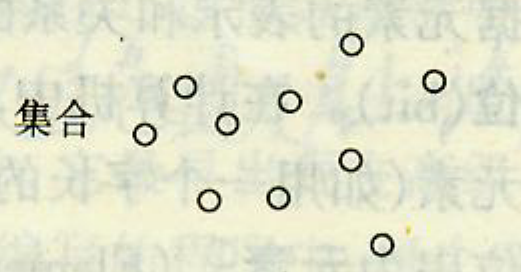


图 1.1 集合

* 线性结构：结构中的数据元素之间存在一个对一个的关系，如图 1.2所示。

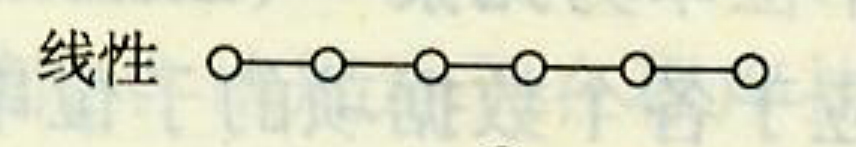


图 1.2 线性结构

* 树形结构：结构中的数据元素之间存在一个对多个的关系，如图 1.3所示。

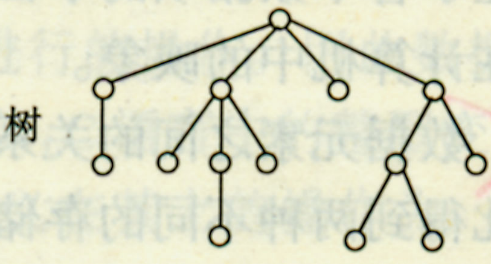


图 1.3树结构

* 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，如图 1.4所示。

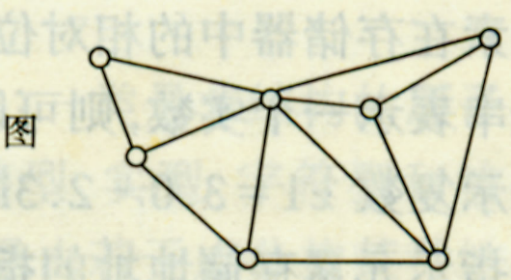


图 1.4 图

## 数据的逻辑结构

结构定义中的“关系”描述的是数据元素之间的逻辑关系,又称为逻辑结构。比如平常教学中所画的内存图,数组等为数据的逻辑结构。

## 数据的物理结构

数据结构在计算机中的实际表示形式称为数据的物理结构,又称为物理存储。

## 算法和算法分析

计算机算法是对特定问题求解步骤的描述，它是指令的有限序列。为解决某问题的算法与为该问题编写的程序含义是相同的。

常用的表示算法的语言有：自然语言、流程图、盒图、程序设计语言和伪代码。

## 算法的五个特性

* + - 1. 有限性：算法必须在执行有限条指令之后结束，每条指令执行的时间也必须是有限的。
      2. 确定性：算法中每一条指令必须有确切的含义，读者和计算机在理解时不会产生二义性，并且在相同条件下，相同的输入只能得到相同的输出。
      3. 可行性：算法能把问题真正的解决。即不能是理论正确但无法在计算机上实现的算法。
      4. 输入：一个算法有零个或多个输入。
      5. 输出：一个算法有一个或多个输出。

●(2004年下)下面的程序段违反了算法的 (54) 原则。

void sam()

{

int n=2；

while(!odd(n))

n+=2；

printf(n)；

}

(54)A.有穷性 B.确定性 C.可行性 D.健壮性

解析:odd：判断是否是奇数!

## 算法设计的要求

* 正确性：算法应当满足具体问题的需求。
* 可读性：算法应该能让人读懂，能被计算机运行。
* 健壮性：算法应该具有容错处理能力，不容易被击垮。
* 时间高效率与低存储量要求：效率指程序的执行时间(越短越好)，算法要占用计算机一定的存储量(越小越好)。

## 算法效率的度量

* + - 1. 时间复杂度

根据不同的输入，将算法的时间复杂度分为三种情况：

###### 最佳情况：使算法执行时间最少的输入。一般不进行算法在最佳情况下的时间复杂度分析。

###### 最坏情况：使算法执行时间最多的输入。一般会进行算法在最坏时间复杂度的分析，因为最坏情况是在任何输入下运行时间的一个上限，而且对于某些算法来说，最坏情况是相当频繁的。

###### 平均情况：算法的平均运行时间，可按三个步骤进行分析：将所有的输入按其执行时间分类；确定每类输入发生的概率；确定每类输入的执行时间。

###### 时间复杂度的表示

算法时间复杂度符号O、Ω、Θ的定义分别如下。

* O记号：给出一个函数的渐进上界。给定一个函数g(n)，O(g(n))表示为一个函数集合的{f(n)：存在正常数c和n0，使得对所有的n≥n0，有O≤f(n)≤c·g(n)}。
* Ω记号：给出一个函数的渐进下界。给定一个函数g(n),Ω(g(n))表示为一个函数集合的{f(n)：存在正常数c和n0，使得对所有的n≥n0，有O≤c·g(n)≤f(n)。
* Θ记号：给出一个函数的渐进上界和下界，即渐进确界。给定一个函数g(n)，Θ(g(n))表示为一个函数集合{f(n)：存在正常数c1、c2和n0，使得对所有的n≥n0，有O≤c1·g(n)≤f(n)≤c2·g(n)}。

常见的时间复杂度如图 1.5所示。



图 1.5 常见时间复杂度的比较

●(2012年上)以下关于渐进符号的表示中，不正确的是 (62) 。

(62)A.n2=(n2) B.n2=O(n2) C.n2=O(n) D.n2=O(n3)

解析：等号左边为{f(n):…}中值,等号右边为O(g(n))的理解方式。

●(2004年下)下面函数中渐进时间最小的是(53)。

(53)A.T1(n)=n+nlogn　B.T2(n)=2n+nlogn C.T3(n)=n2-logn D.T4(n)=n+100logn

#### 空间复杂度

一个算法的空间复杂度是指程序运行从开始到结束所需的存储量。主要包含算法自身实现所需要的辅助存储空间，不包含程序中原存储数据的空间。

* 固定空间：在硬盘上的存储空间。
* 可变空间：程序运行时占用的内存空间。

●(2007年下)关于算法与数据结构的关系，(64)是正确的。

(64)A.算法的实现依赖于数据结构的设计B.算法的效率与数据结构无关

C.数据结构越复杂，算法的效率越高D.数据结构越简单，算法的效率越高

●(2014年上)某个算法的时间复杂度递归式T(n)=T(n-l)+n，其中n为问题的规模，则该算法的渐进时间复杂度为(62)，若问题的规模增加了16倍，则运行时间增加(63)倍。

(62)A.Θ(n) B.Θ(nlgn) C.Θ(n2) D.Θ(n2lgn)

(63)A.16 B.64 C.256 D.1024

●(2006年上)设某算法的计算时间可用递推关系式T(n)=2T(n/2)+n表示，则该算法的时间复杂度为(59)。

A．O(lgn) B．O(nlgn) C．O(n) D．O(n2)

●(2008年下)设某算法的计算时间表示为递推关系式T(n)= T(n-1)+ n(n>0)及T(0)=1，则该算法的时间复杂度为(65)。

(65)A.O(lgn) B.O(nlgn) C.O(n) D.O(n2)

●(2010年上)若某算法在问题规模为n时，其基本操作的重复次数可由下式表示，则该算法的时间复杂度为(64)。

T(n)=



(64)A.O(n) B.O(n2) C.O(logn) D.O(nlogn)

●(2009年下)某算法的时间复杂度表达式为T(n)=an2+bnlgn+cn+d，其中，n为问题规模，a、b、c和d为常数，用O表示其渐进意义时间复杂度为(63)。

(63)A.O(n2) B.O(n) C.O(nlgn) D. O(1)

●(2010年上)若对一个链表最常用的操作是在末尾插入结点和删除结点，则采用仅设尾指针的单向循环链表(不含头结点)时，(65)。

(65)A.插入和删除操作的时间复杂度都为O(1) B.插入和删除操作的时间复杂度都为O(n)

C.插入操作的时间复杂度为O(1)，删除操作的时间复杂度为O(n)

D.插入操作的时间复杂度为O(n)，删除操作的时间复杂度为O(1)

●(2011年下)对n个元素值分别为-1、0或1的整列数组A进行升序排序的算法描述如下：统计A中-1、0和1的个数，设分别为n1、n2和n3，然后将A中的前n1个元素赋值为-1，第n1+1到n1+n2个元素赋值为0，最后n3个元素赋值为1。该算法的时间复杂度和空间复杂度分别为(64) 。

(64)A. B. C. D.

●(2012年上)现要对n个实数(仅包含正实数和负实数)组成的数组A进行重新排列，使得其中所有的负实数都位于正实数之前。求解该问题的算法的伪代码如下所示，则该算法的时间和空间复杂度分别为 (65) 。

i=0；j=n-l；

while i<j do

while A[i]<0 do

i= i+l：

while A[j]>O do

j =j-l；

if i<j do

交换A[i]和A[j]

(65)A.  (n)和(n) B． (1)和(n) C.  (n)和 (1) D． (1)和 (1)

●(2013年上)采用顺序表和单链表存储长度为n的线性序列，根据序号查找元素，其时间复杂度分别为(51)。

(51)A.0(l)、0(l) B.0(l)、0(n) C.0(n)、0(l) D.0(n)、0(n)

●(2015年上) (53) 算法采用模拟生物进化的三个基本过程“繁殖(选择)→交叉(重组)→变异(突变)”。

(53)A.粒子群 B.人工神经网络 C.遗传 D.蚁群

# 线性表

## 线性表举例

线性表是由相同数据类型的结点组成的有限序列，如数组、链表(单链表、双链表、循环单链表，循环双链表)、栈、队列,双端队列等。

* + - 1. 数组

int a[5]={1,2,3,4,5};

a[0] a[1] a[2] a[3] a[4]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

* + - 1. 单链表

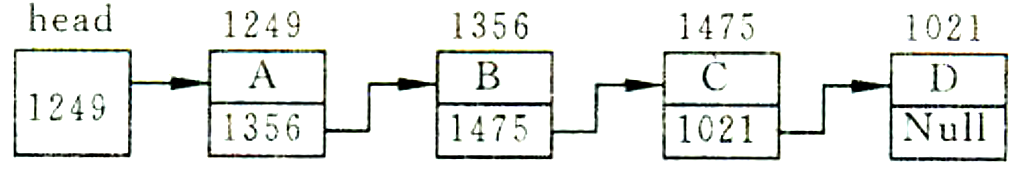


图 2.1只有头指针的单链表

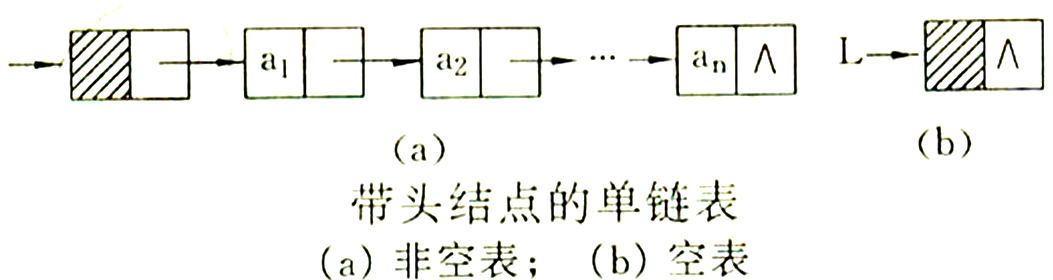


图 2.2有头指针及头结点的非空链表及空链表

* + - 1. 单循环链表

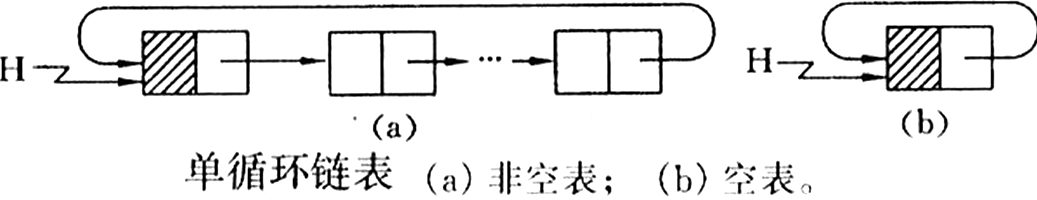


图 2.3非空表与空表

* + - 1. 双向循环链表

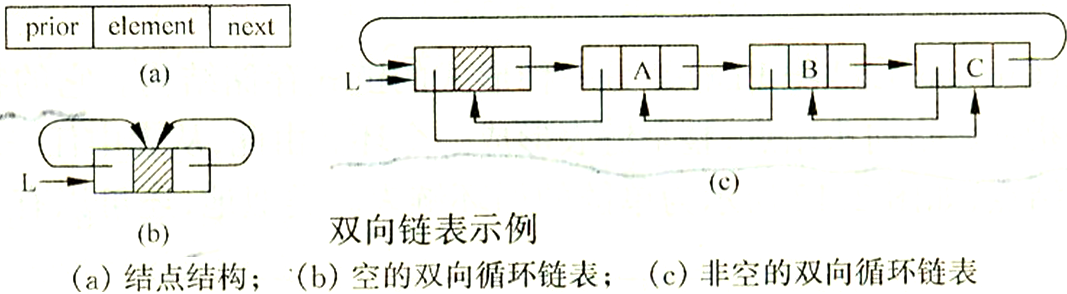


图 2.4 双向链表（a）结点结构 （b）空的双向循环链表 （c）非空的双向循环链表

* 使用头结点的好处:不管链表是否为空，头结点都不为空，这使得对链表的各种操作(查找和修改)得到统一。

## 线性表的存储

#### 顺序存储

用地址连续的数据结构如数组来存储线性表。

* 优点：能随机存取线性表中的任何结点。
* 缺点：数组的大小通常是固定的，不利于任意增加或减少线性表的个数；插入和删除线性表的结点时，要移动数组中的其他元素，操作复杂。

#### 链接存储

用链表存储线性表(链表)，最简单的是单向链表。

* 优点：表的每个结点的实际存储位置是任意的，这给线性表的插入和删除带来方便，只要改变链表有关结点的后继指针就能完成插入或删除的操作，不需移动任何表元。
* 缺点：每个结点增加了一个后继指针成分，要花费更多的存储空间；不方便随机访问线性表的任一结点。

## 线性表-栈

栈是一种特殊的线性表，栈只允许在同一端进行插入和删除运算。允许插入和删除的一端称为栈顶，另一端称为栈底。称栈的结点插入为进栈，结点删除为出栈。因为最后进栈的结点必定最先出栈，所以栈具有后进先出的特征*。*



图 2.5(a)盏的示意图 (b)用铁路调度站表示盏

栈有顺序存储和链式存储两种存储方式。主要应用在数制转换、括号匹配的检验、行编辑程序、迷宫求解、表达式求值等。

●若push、pop分别表示入栈、出栈操作，初始栈为空且元素1、2、3依次进栈，则经过操作序列push、push、pop、pop、push、pop之后，得到的出栈序列为 (29) 。

(29)A.321 B.213 　C.231 D.123

●可以用栈来检查算术表达式中的括号是否匹配。分析算术表达式时，初始栈为空，从左到右扫描字符，遇到字符“(”就将其入栈，遇到“)”就执行出栈操作。对算术表达式“(a+b\*(a+b))/c)+(a+b)”,检查时， (33) ；对算术表达式“((a+b/(a+b)-c/a)/b”,检查时， (34) 。这两种情况都表明所检查的算术表达式括号不匹配。

(33)A.栈为空却要进行出栈操作　　　　　　 B.栈已满却要进行入栈操作

C.表达式处理已结束，栈中仍留下有字符“(” D.表达式处理已结束，栈中仍留下有字符“)”

(34)A.栈为空却要进行出栈操作　　　　　　 B.栈已满却要进行入栈操作

C.表达式处理已结束，栈中仍留下有字符“(” D.表达式处理已结束，栈中仍留下有字符“)”

* 逆波兰式：是一种算术表达式的另一种表示，也称为后缀表示，定义为把运算符放在两个运算对象的后面。
* 中缀算术表达式（又名波兰式）转换成对应的后缀算术表达式：把每个运算符都移到运算对象的后面，然后去掉括号即可。

●表达式采用逆波兰式表示时可以不用括号，而且可以用基于 （1）的求值过程进行计算，与逆波兰式ab+cd+\*对应的中缀表达式为： （2） 。

1．A.栈 B.队列 C.符号表 D.散列表

2．A．a+b+c\*d B.(a+b)\*c+d C.(a+b)\*(c+d) D.a+b\*c+d

●表达式a\*(b+c)-d的后缀表达形式为\_\_\_\_.

(39)A.abcd\*+- B.abc+\*d- C.abc\*+d- D.-+\*abcd

## 队列

队列也是一种特殊的线性表，只允许在一端进行插入，另一端进行删除运算。允许删除运算的那一端称为队首，允许插入运算的一端称为队尾。称队列的结点插入为进队，结点删除为出队。因最先进入队列的结点将最先出队，所以队列具有先进先出的特征。

* + - 1. 顺序存储队列

可以用顺序存储线性表来表示队列，为了指明当前执行出队运算的队首位置，需要一个指针变量head(称为队头指针)，为了指明当前执行进队运算的队尾位置，也需要一个指针变量tail(称为尾指针)，如图 2.6所示。



图 2.6顺序存储队列

若用有N个元素的数组表示队列，随着一系列进队和出队运算，队列的结点移向存放队列的数组的尾端，会出现数组的前端空着，而队列空间已用完的情况。一种可行的解决办法是当发生这样的情况时，把队列中的结点移到数组的前端，修改头指针和尾指针。另一种更好的解决办法是采用循环队列。

循环队列就是将实现队列的数组a的第一个元素a[0]与最后一个元素a[n-1]连接起来。队空的初态为head=tail=O。在循环队列中，当tail赶上head时，队列满。反之，当head赶上tail时，队列变为空。这样队空和队满的条件都同为head=tail，这会给程序判别队空或队满带来不便。因此，可采用当队列只剩下一个空闲结点的空间时，就认为队列已满，用这种简单办法来区别队空和队满。

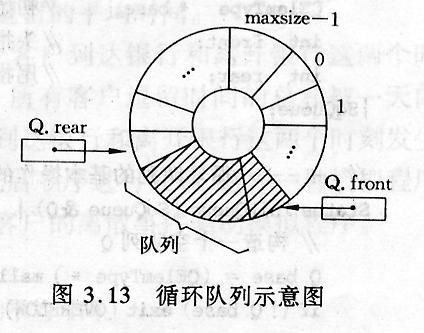


图 2.7循环队列

即队空的判别条件是head=tail，队满的判别条件是head=(tail+1) mod n。

* + - 1. 链接存储队列

队列也可以用链接存储线性表实现，用链表实现的队列称为链接队列。链表的第一个结点是队列首结点，链表的末尾结点是队列的队尾结点，队尾结点的链接指针值为NULL。队列的头指针font指向链表的首结点，队列的尾指针rear指向链表的尾结点。当font==rear,队列为空。

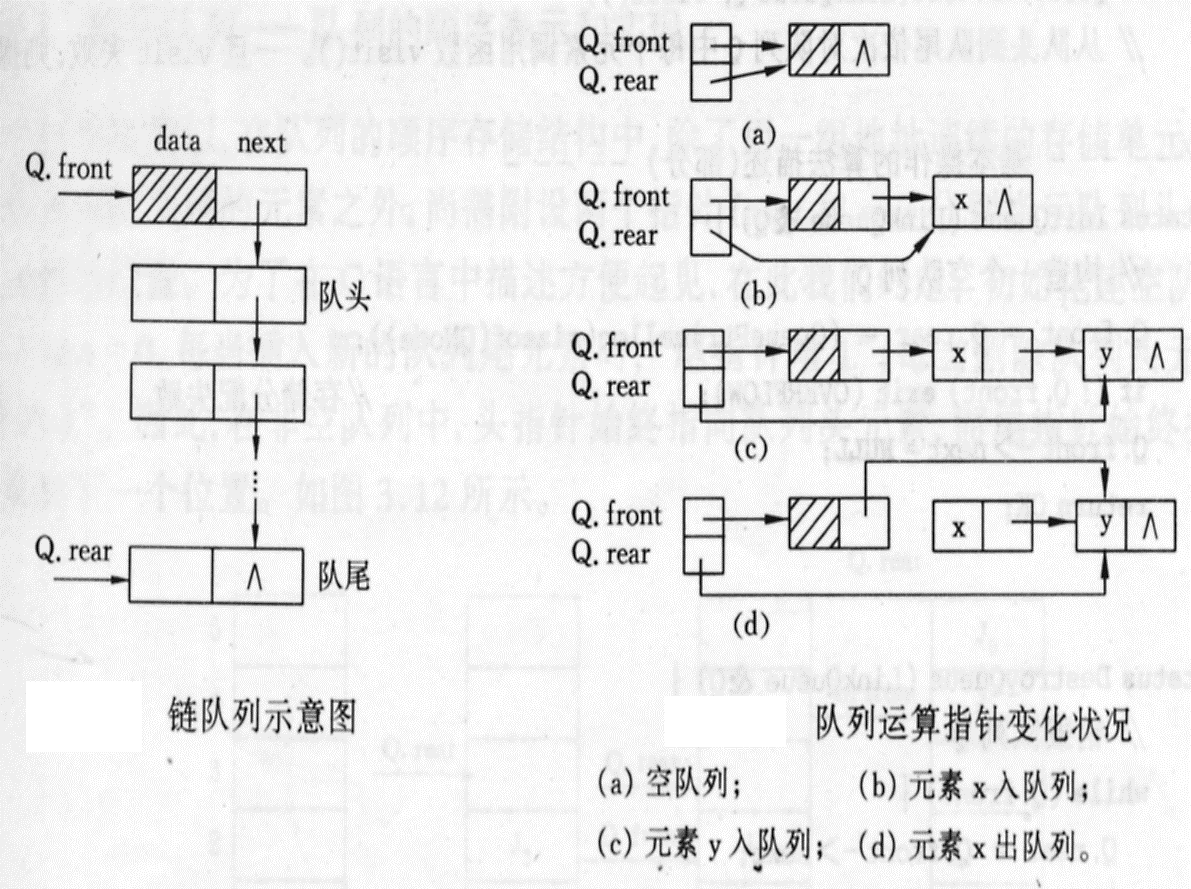


图 2.8 链式存储队列

例题分析：

●判断“链式队列为空”的条件是：\_\_\_\_(front为头指针，rear为尾指针)

A．front ＝ NULL B.rear==NULL C. front == rear D.front!=rear

● 若in、out分别表示入、出队操作，初始队列为空且元素a、b、c依次入队，则经过操作序列in、in、out、out、in、out之后，得到的出队序列为 (30) 。

(30)A.cba B.bac C.bca D.abc

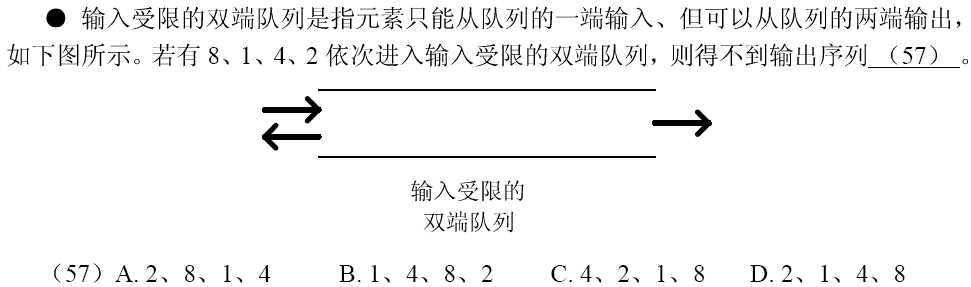
## 双端队列

限定插入和删除操作在表的两端进行的线性表(注意其与队列的区别)。



图 2.9双端队列

●输入受限的双端队列是指元素只能从队列的一端输入、但可以从队列的两端输出，如下图所示。若有8、1、4、2依次进入输入受限的双端队列，则得不到输出序列 (57) 。



(57)A.2、8、4、1 B.1、4、8、2 C.4、2、1、8 D.2、1、4、8

# 树和二叉树

## 树

* + 1. 树的基本概念

树是由零个或多个结点组成的有限集合T。树的几个概念：树的结点、结点的度(又叫次数)、树的度(又叫次数)、叶子结点(终端结点)、非终端结点(分支结点)、孩子、双亲、兄弟、树的深度等，如图 3.1所示。

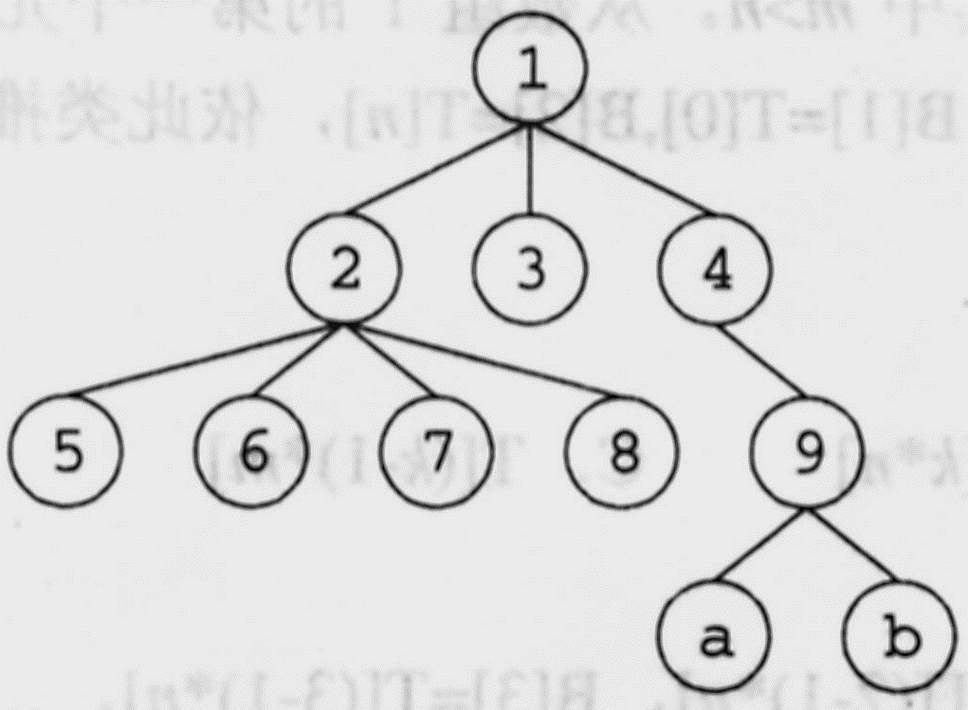


图 3.1树

根结点的度数为3，结点2的度数为4，结点4的度数为1，结点9的成数为2，其他结点的度数为0，该树的度数4，该树的深度为4。

定义一棵树的根结点所在的层次为1(这里暂且定义1 ，也可以把它定义0层，考试时题目会有明确的规定)，其他结点所在的层次等于它的父结点所在的层次加1。树中各结点的层次的最大值称为树的层次。

* + 1. 树的常用存储结构
       1. 标准存储结构

在树的标准存储结构中，树中的结点内容可分成两部分，分别为结点的数据和指向孩子结点的指针数组。对于N度树，在其标准存储结构中指针数组有N个元素。

例如，设树的次数为5，树的结点数据仅限于字符，用c代码描述树结点的标准存储结的数据类型如下：

#define N 5

typedef struct tnode{

char data；//树结点的数据信息

struct tnode \*child[N]；//树结点的子结点指针

}TNODE； //树结点的数据类型

* + - 1. 带逆存储结构

带逆存储结构在标准存储结构的基础上增加了一个指向其父结点的指针，用c代码描述树结点的带逆存储结构的数据类型如下：

#define N 5

typedef struct rtnode{

char data：//树的结点数据信息

struc rtnode \*child[N]；//树结点的子结点指针

struct rtnode \*parent； //p父结点指引

}RTNODE；//树结点的数据类型

* + 1. 树的遍历
* 前序遍历：首先访问根结点，然后从左到右按前序遍历根结点的各棵子树。
* 后序遍历：首先从左到右按后序遍历根结点的各棵子树，然后访问根结点。
* 层次遍历：首先访问处于第1层上的根结点，然后从左到右依次访问处于第2层上的结点，然后在从左到右依次访问处于第3层上的结点等等，即自上而下从左到右逐层访问树各层上的结点。

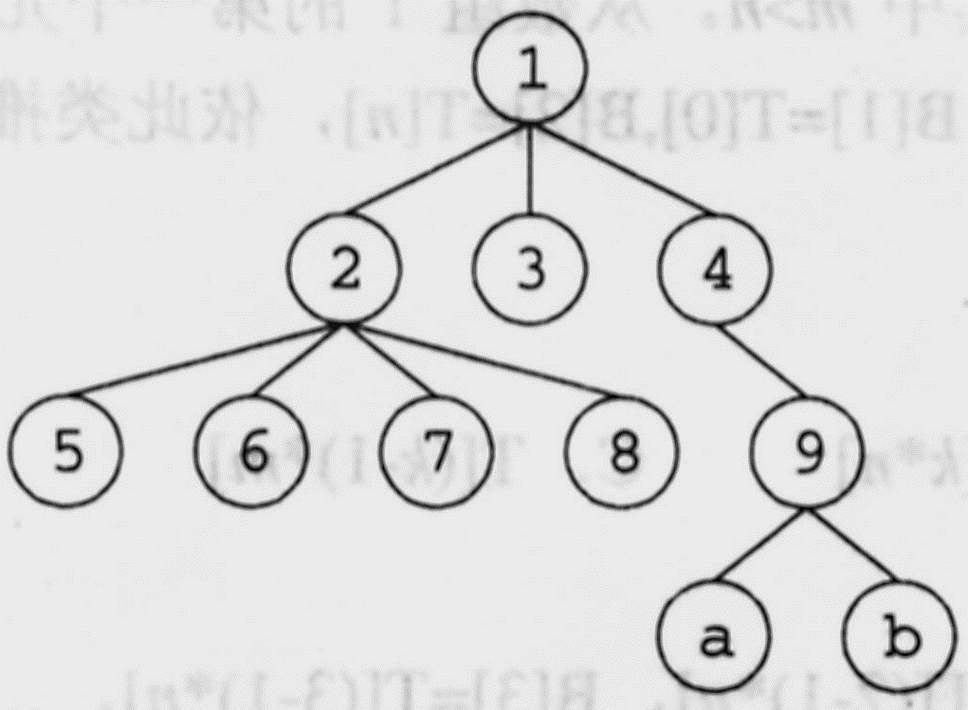


图 3.2树

上述各种遍历结果如下：

* 前序：1，2，5，6，7，8，3，4，9，a，b。
* 后序：5，6，7，8，2，3，a，b，9，4，1。
* 层次：1，2，3，4，5，6，7，8，9，a，b。

注意：树没有中序遍历。

## 二叉树

* + 1. 二叉树的基本概念

二叉树是一个有限的结点集合，该集合或者为空，或者是由一个根结点及其两棵互不相交的左、右叉子树所组成的。二叉树的结点中有两棵子二叉树，分别称为左子树和右子树。因二叉树可以为空，所以二叉树中的结点可能没有孩子结点，也可能只有一个左子结点(或右子结点)，也可能同时有左右两个了结点。如图 3.3所示的是二叉树的4种可能形态(如果把空树计算在内，则共有5种状态)。

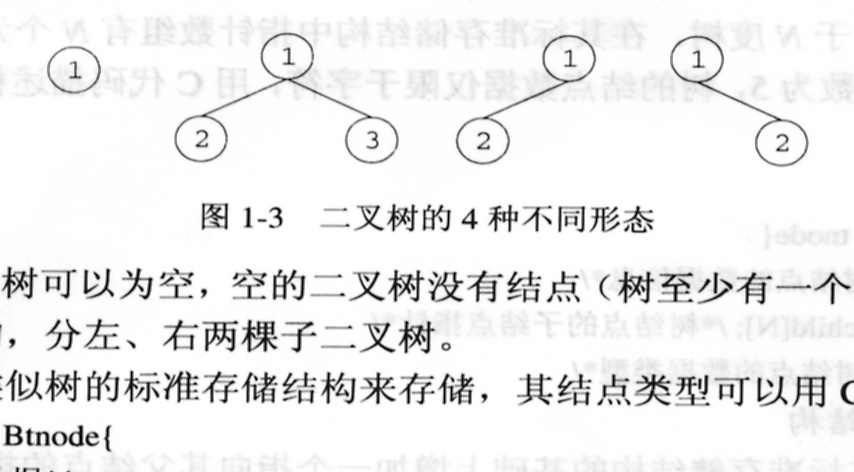


图 3.3二叉树的四种不同形态

* + 1. 二叉树与树的区别
* 二叉树中，结点的子树是有序的，分左右两棵子树。单就树来说，一个结点可能拥有更多的孩子，所以没有左右之分。

问题：二叉树是否为一种特殊的树？答案：不是。

* + 1. 树及森林转到二叉树
* 最左孩子结点变左孩子；
* 兄弟结点变右孩子。

#### 树转成二叉树

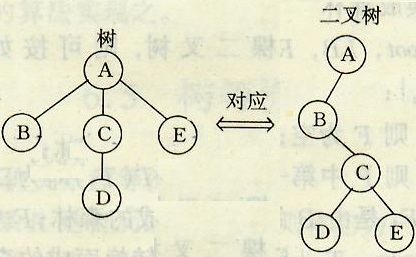
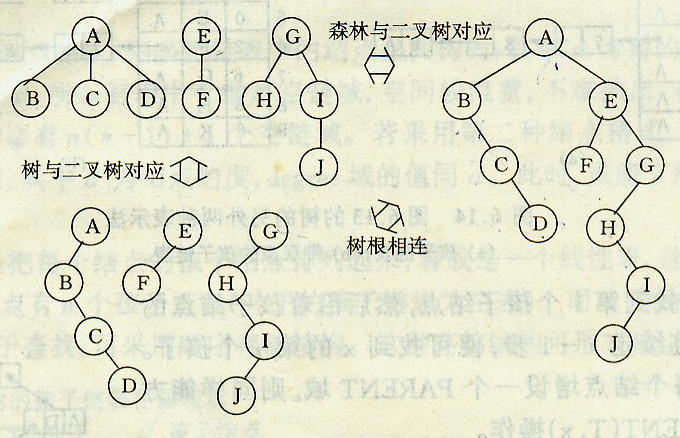


图 3.4树转成二叉树

#### 森林转成二叉树

给各棵树的根都定一个总双亲，让各棵树变成子树，然后按照树转二叉树的方法转，最后去掉总双亲。



* + 1. 二叉树的性质
* 性质1：在二叉树的第i层上至多有个结点(i≥1);
* 性质2：深度为k的二叉树最多有个结点(k≥1);
* 性质3：对任何一棵二叉树，如果其叶子结点个数为,度为2的节点数为,则= +1。

补充:假设度为1的结点个数为,则二叉树的结点个数n= + + ,且= +1。

* + 1. 满二叉树

一棵深度为k且有个结点的二叉树称为满二叉树。满二叉树中不存在度数为1的结点。

如图 3.5所示就是一棵满二叉树，对结点进行顺序编号。

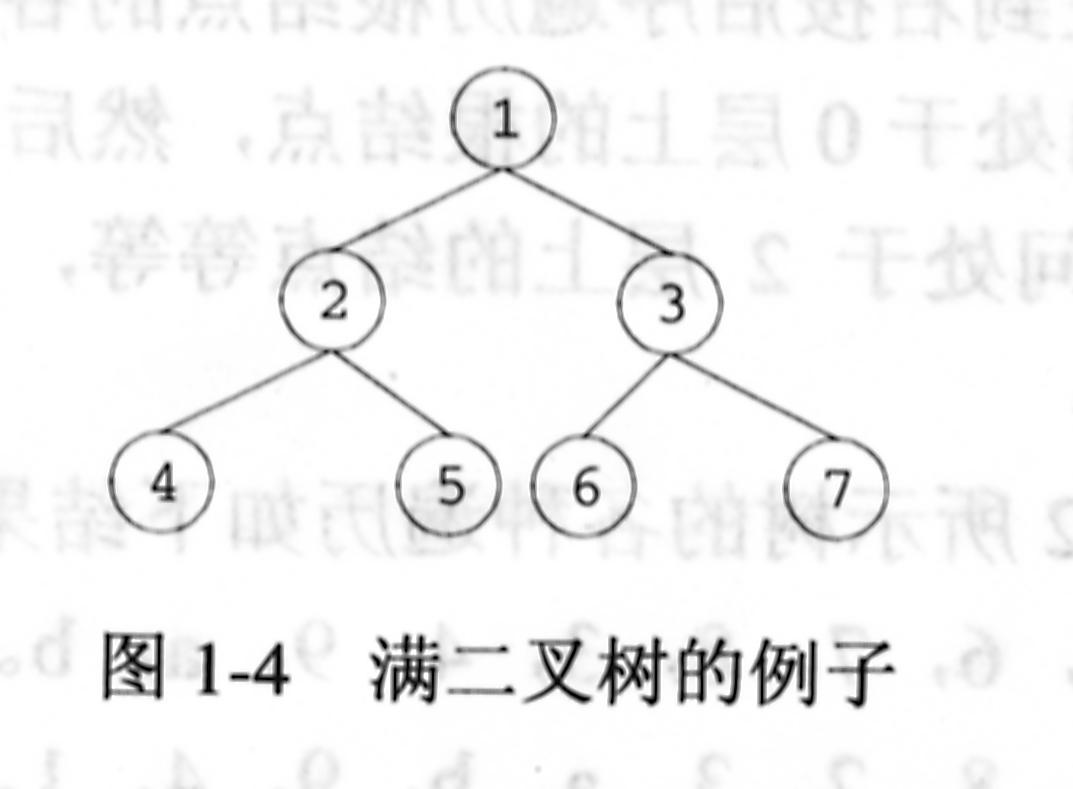


图 3.5满二叉树

### 完全二叉树

如果深度为k，有n个结点的二叉树中的结点能够与深度为k的顺序编号的满二叉树从1到n标号的结点相对应，则称这样的二叉树为完全二叉树。

* 在一棵深度为k (k≥1)的完全二叉树中，所有的叶子结点都出现在第k层或第k-1层(最后两层)。
* 在完全二叉树中，若一个结点没有左子结点，则它必定没有右子结点，所以一定是叶子结点。
* 完全二叉树中度为1的结点数只能为0或1。

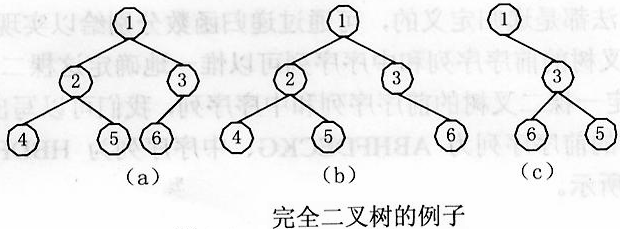


图 3.6完全二叉树

图 3.6中，（a）是，（b）、（c）不是，根据完全二叉树的定义。

注意:满二叉树是完全二叉树,但完全二叉树不一定是满二叉树。

### 完全二叉树的性质

* 性质4：具有n个结点的完全二叉树的深度为+1(其中为下取整)。

例子:在一棵完全二叉树中，其根的序号为1， (33) 可判定序号为p和q的两个结点是否在同一层。

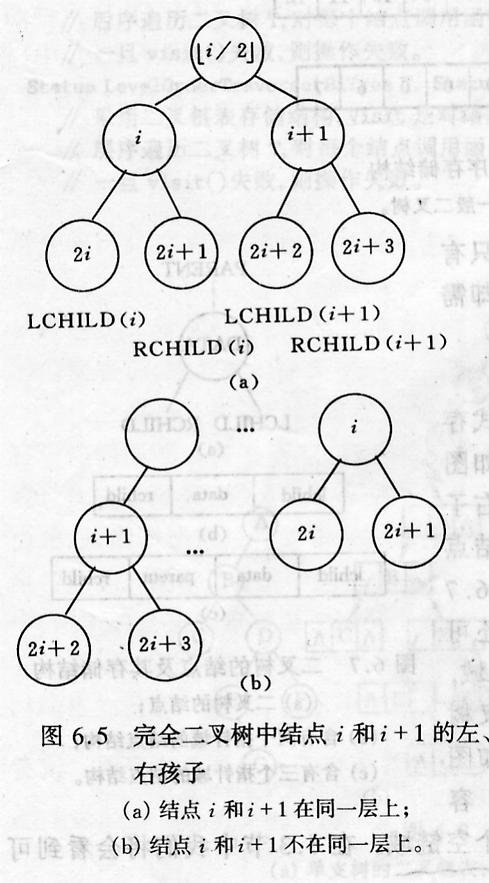
(33)A．= B．= C．+1= D．= +1

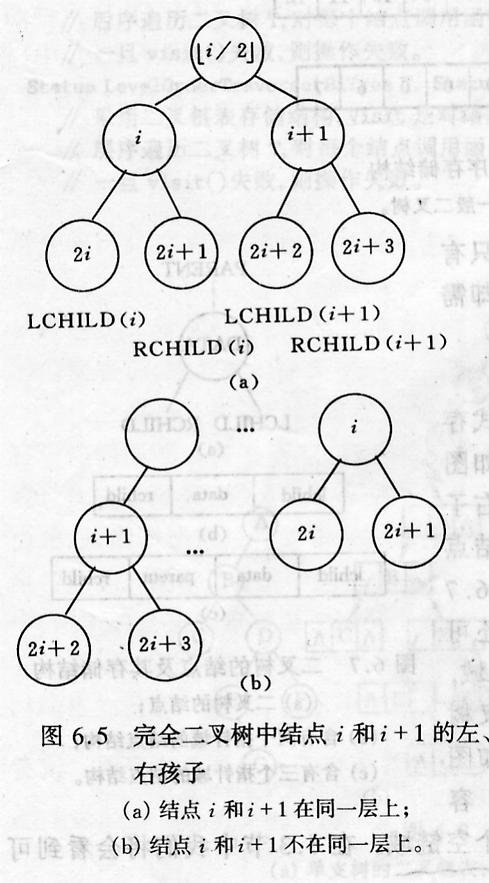
* 性质5：如果对一棵n个结点的完全二叉树的结点按层次编号(从第一层到第+1层，每层从左到右)，则对任一结点i(1≤i≤n),有：

###### 如果i=1,则结点i无双亲，是二叉树的根结点；如果i>1,则其双亲结点是;

###### 如果2i〉n,则结点i为叶子结点，无左孩子，否则，其左孩子是2i;

###### 2i+1〉n,则结点i无右孩子；否则，其右孩子是结点2i+1。





注意:性质1、性质2、性质3适用于所有类型的二叉树，而性质4、性质5只适用于满二叉树和完全二叉树。

### 二叉树的四种遍历

###### 前序遍历(先根遍历、先序遍历)：根-左-右 根一定是遍历序列的第一个元素。

###### 中序遍历(中根遍历)：左- 根-右。

###### 后序遍历(后根遍历)：左-右-根 根一定是遍历序列的最后一个。

###### 层次遍历：根一定是遍历序列的第一个。

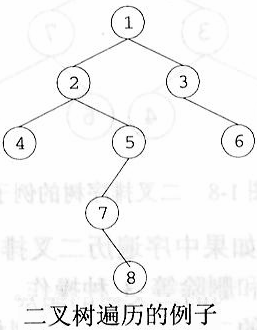


图 3.7二叉树遍历

* 先序：12457836
* 中序：42785136
* 后序：48752631
* 层次：12345678

注意：前序遍历、中序遍历、后续遍历属于深度优先遍历，层次遍历属于广度优先遍历。都采用递归定义。

* 性质：一棵二叉树的前序序列与中序序列可以唯一地确定这棵树。

例：前序序列：ABHFDECKG；中序序列：HBDFAEKCG，则构造二叉树的过程为：

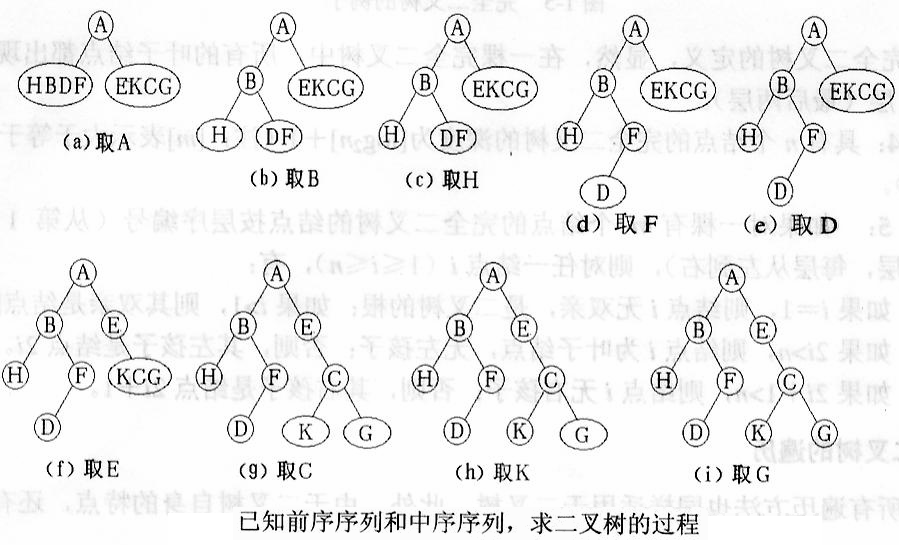


图 3.8已知前序序列和中序序列,求二叉树的过程

问题：

* 什么情况下，二叉树的前序遍历序列与中序遍历序列相同？

答：只有根结点的二叉树或只有右子树的二叉树。

* 什么情况下，二叉树的前序遍历序列与后序遍历序列相同？

答：只有根结点的二叉树。

●(2003真题)结论“ (7) ”是正确的。

(07)A．二叉树的度为2 B．树中结点的度可以小于2

C．二叉树中至少有一个结点的度为2 D．二叉树中任何一个结点的度都为2

### 二叉排序树

二叉排序树也叫最优查找树(二叉查找树)：或是一棵空树，或者是具有下列性质(BST)的二叉树：

* 若它的左子树非空，则左子树上的所有结点的值均小于根结点；
* 若它的右子树非空,则右子树上的所有结点均大于根结点;
* 左、右子树本身又各是一棵二叉排序树。

特点简记：左小右大

在二叉查找树中，由根结点到所有其他结点的路径长度总和称为内部路径长度。具有最小内部路径长度的树称为丰满树，对丰满查找树进行插入或者删除操作后，会产生一棵非丰满树,如图 3.9所示：

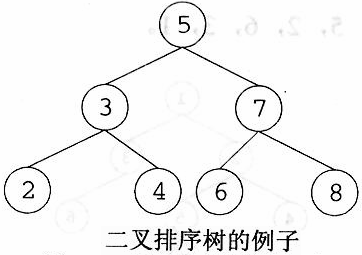


图 3.9二叉排序树

* 二叉排序树的插入、删除和查找时间复杂度均为：O()。

●【2006年下半年】结点数目为n的二叉查找树(二叉排序树)的最小高度为(52)、最大高度为(53)。

(52)A.n B. C.[] D.

(53)A.n B. C.[] D.

### 平衡二叉树

平衡二叉树又称AVL树。它或者是一棵空树，或者是具有下列性质的二叉树：它的左子树和右子树都是平衡二叉树，且左子树和右子树的深度之差的绝对值不超过1。

平衡二叉树的的高度总为：，基本操作的平均时间为O()，最坏情况下也为O()。

二叉树上结点的平衡因子BF：左子树的深度减去它的右子树的深度。平衡二叉树上所有结点的平衡因子只可能是－1、0、1。只要二叉树上有一个平衡因子的绝对值大于1，则该二叉树就是不平衡的,如图 3.10所示。

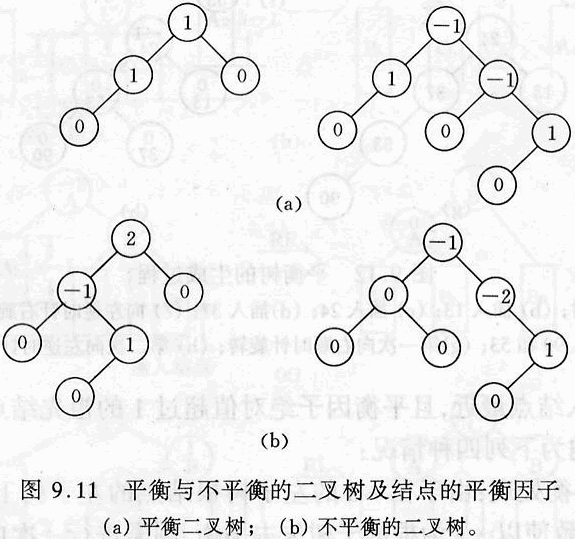


图 3.10(a)平衡二叉树(b)非平衡二叉树

●【2005年下半年】由元素序列(27,16,75,38,51)构造平衡二叉树，则首次出现的最小不平衡子树的根(即离插入结点最近且平衡因子的绝对值为2的结点)为 (46) 。

(46)A.27 B.38 C.51 D.75

注意：构建以后的平衡二叉树是二叉排序树。

### m阶B-树

m阶B-树是一种平衡的m叉树，具有如下的性质：

* 树中每个结点至多有m棵子树；
* 除了根结点和叶子结点之外，每个结点的孩子个数大于等于)；
* 具有k个孩子的非叶结点含有k-1个键值
* 所有叶结点在同一层上，而且不附有信息。

### 最优二叉树

* 树的(内部)路径长度: 从树根到树中每一个结点的路径长度之和。在结点数相同的二叉树中,丰满树的(内部)路径长度最短。
* 结点的权：可以赋予树中结点一个有某种意义的实数,这些数字称为结点的权。
* 结点的带权路径长度：结点到树根之间的路径长度与该结点上权的乘积,称为结点的带权路径长度.
* 树的带权路径长度：树中所有叶子结点的带权路径长度之和,称为树的带权路径长度(树的代价),通过记为:WPL =。
* 带权路径长度最小(即代价最小)的二叉树称为最优二叉树或赫夫曼树。

构造一棵赫夫曼树的过程如下:



图 3.11赫夫曼树的构造过程

* 赫夫曼树是严格的二叉树，没有度数为1的结点。

●【2004年】若一棵赫夫曼树共有9个结点，则其叶子结点为：\_\_\_\_\_\_。

A.4 B.5 C.6 D.7

* 赫夫曼编码：左分支表示’0’,右分支表示’1’。

在数据压缩编码的应用中，赫夫曼算法可以用来构造具有最优前缀码的二叉树，这是一种贪心的算法。

●由权值为9，2，5，7的四个叶子构造一棵哈夫曼树，该树的带权路径长度为\_\_\_\_\_\_\_\_。

(50)A.23 B.37 C.44 D.46

### 二叉树的存储结构

#### 顺序存储

用一组地址连续的存储单元依次自上而下、自左而右存储完全二叉树上的结点元素，即将二叉树上编号为i的节点元素存储在一维数组中。对于一般的二叉树，则将其每个结点与完全二叉树上的结点相对照，存储在一维数组的相应分量中。

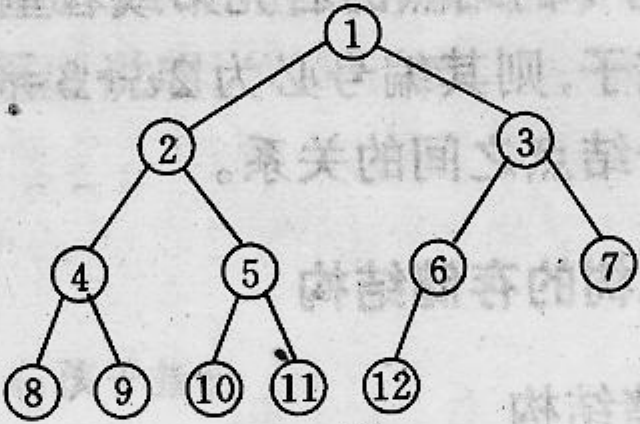


图 3.12二叉树1

图 3.12所示的二叉树顺序存储为：



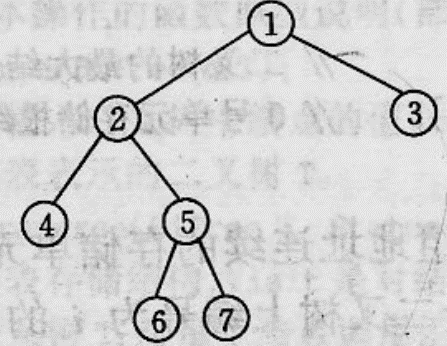


图 3.13二叉树2

图 3.13所示的二叉树顺序存储为：



图中以“0”表示不存在此结点。这种顺序结构仅适用于完全二叉树。因为，在最坏得情况下，一个深度为k且有k个结点的单支树(树中不存在度为2的结点)确需要长度为2k-1的一维数组，造成很大的空间浪费。

* + - 1. 链式存储

二叉树常采用类似树的存储结构，其结点类型可以用C语言定义如下：

typedef struct Btnode{

char data；//数据

struct Btnode \*Ichild；//左孩子

struct Btnode \*rchild；//右孩子

}BYNODE；

●【2005年下半年】在二叉树的顺序存储中，每个结点的存储位置与其父结点、左右子树结点的位置都存在一个简单的映射关系，因此可与三叉链表对应。若某二叉树共有n个结点，采用三叉链表存储时，每个结点的数据域需要d个字节，每个指针域占用4个字节，若采用顺序存储，则最后一个结点下标为k(起始下标为1),那么\_\_\_(39)\_\_\_时采用顺序存储更节省空间。

(39) 

## 广义表

也称为列表。广义表通常用括号括起来，用逗号分隔广义表中的元素，原定用小写字母表示原子，用大写字母表示广义表：

###### 广义表LS=(,,,…,)

###### 广义表的长度：该广义表中包含的元素(包括原子和子表)的个数。

###### 广义表的深度：表展开后所含括号的层数。

###### 表头和表尾：称第一个元素为LS的表头，称其余元素组成的表(,,…,)是LS的表尾。

* A=():A是一个空广义表，长度为0，深度为1
* B=(e):B只有一个原子e,长度为1，深度为1。
* C=(a，(b，c，d)):列表C长度为2，两个元素分别是a和子表(b，c，d)，深度为2。
* D=(A,B,C):列表D的长度为3，三个元素都是列表。显然，将子表的值带入后，则有D＝(()，(e)，(a，(b，c，d)))，深度为3。
* E=(a,E):这是一个递归的表，它的长度为2。E相当于一个无限的列表E＝(a,(a,(a,…..)))，深度为。

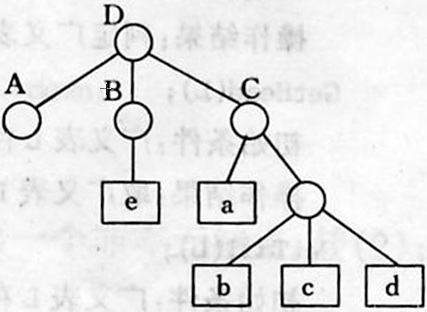


图 3.14列表D的图

GetHead(B) = e GetTail(B) = ();

GetHead(D) = A GetTail(B) = (B,C);

* G = (())：长度为1，深度为2，可分解得到其表头、表尾均为空表()；

●若广义表L=((1,2,3))，则L的长度和深度分别为 (36) 。

(36)A．1和1 B．1和2 C．1和3 D．2和2

## 矩阵的压缩存储

### 特殊矩阵

* + - 1. 对称矩阵

若n阶矩阵A中的元满足= (1i,jn),则称为对称矩阵。如三阶对称矩阵：

* + - 1. 对角矩阵

图 3.15所示的矩阵为对角矩阵。

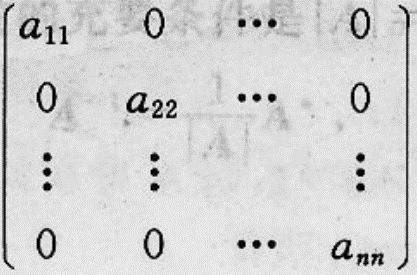


图 3.15对角矩阵

* + - 1. 上(下)三角矩阵



图 3.16上（下）三角矩阵

* + - 1. 单位矩阵

主对角线上的元素全为1，其余元素全为0的n阶矩阵。

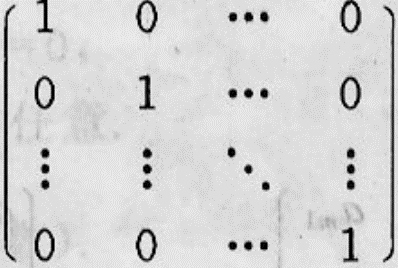


图 3.17单位矩阵

* + - 1. 三对角矩阵

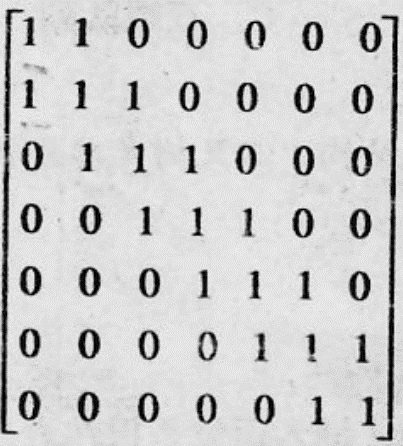


图 3.18三对角矩阵

### 压缩存储

为多个值相同的元只分配一个存储空间，对零元不分配空间。压缩存储的目的是节约存储空间。特殊矩阵通常可以采用压缩存储的方法，使用的是一维数组。

这个对阵矩阵可以压缩到以下数组中(按行存储)：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a11 | a21 | a22 | a31 | a32 | a33 |

等同于：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | -1 | 2 | 3 | 0 | 1 |

# 历年真题讲解

## 2009年上半年

●下面关于二叉排序树的叙述，错误的是 (59)。

(59)A.对二叉排序树进行中序遍历，必定得到结点关键字的有序列

B.依据关键字无序的序列建立二叉排序树，也可能构造出单支树

C.若构造二叉排序树时进行平衡化处理，则根结点的左子树高度与右子树高度的差值一定不超过1。

D.若构造二叉排序树时进行平衡化处理，则根节点的左子树高度与右子树高度的差值一定不超过1.

●下面关于栈和队列的叙述，错误的是 (60)。

(60)A.栈和队列都是操作受限的线性表

B.队列采用单循环链表存储时，只需设置队尾指针就可使入队和出队操作的时间复杂度都为O(1)

C.若队列的数据规模n可以确定，则采用顺序存储结构比链式存储结构效率更高

D.利用两个栈可以模拟一个队列的操作，反之亦可

●下面关于二叉树的叙述，正确的是 (61)。

(61)A.完全二叉树的高度h与其结点数n之间存在确定的关系

B.在二叉树的顺序存储和链式存储结构中，完全二叉树更适合采用链式存储结构

C.完全二叉树中一定不存在度为1的结点

D.完全二叉树中必定有偶数个叶子结点

● 设L为广义表，将head(L)定义为取非空广义表的第一个元素，tail(L)定义为取非空广义表除第一个元素外剩余元素构成的广义表。若广义表定义为取L=((x,y,z),a,(u,t,w))，则从L中取出原子项y的运算是 (62)。

(62)A. head(tail(tail(L))) B. tail(head(head(L))) C. head(tail(head(L))) D. tail(tail(head(L)))

## 2009年下半年

●已知一个二叉树的先序遍历序列为①、②、③、④、⑤，中序遍历序列为②、①、④、③、⑤，则该二叉树的后序遍历序列为(57)。对于任意一棵二叉树，叙述错误的是(58)。

(57)A ②、③、①、⑤、④ B.①、②、③、④、⑤ C.②、④、⑤、③、① D. ④、⑤、③、②、①

(58)A.由其后序遍历序列和中序遍历序列可以构造该二叉树的先序遍历序列

B.由其先序遍历序列和后序遍历序列可以构造该二叉树的中序遍历序列

C.由其层序遍历序列和中序遍历序列可以构造该二叉树的先序遍历序列

D.由其层序遍历序列和中序遍历序列不能构成该二叉树的先序遍历序列

●单向链表中往往含有一个头结点，该结点不存储数据元素，一般令链表的头指针指向该结点，而该结点指针域的值为第一个元素结点的指针。以下关于单链表头结点的叙述中，错误的是(60)。

(60)A.若在头结点中存入链表长度值，则求链表长度运算的时间复杂度为O(1)

B.在链表的任何一个元素前后进行插入和删除操作可用一致的方式进行处理

C.加入头结点后，代表链表的头指针不因为链表为空而改变

D.加入头结点后，在链表中进行查找运算的时间复杂度为O(1)

●对于长度为m(m>1)的指定序列， 通过初始为空的一个栈、一个队列后，错误的叙述是 (61 )。

(61)A.若入栈和入队的序列相同，则出栈序列和出队序列可能相同

B.若入栈和入队的序列相同，则出栈序列和出队序列可以互为逆序

C.入队序列与出队序列关系为1:1，而入栈序列与出栈序列关系是1:n(n≥1)

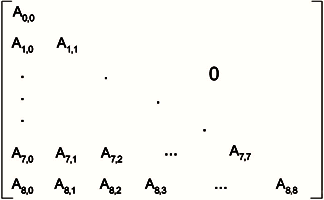
D.入栈序列与出栈序列关系为1:1，而入队序列与出队序列关系是1:n(n≥1)

●字符串采用链表存储方式时，每个结点存储多个字符有助于提高存储密度。若采用结点大小相同的链表存储串，则串比较、求子串、串连接、串替换等串的基本运算中，(62)。

(62) A.进行串的比较运算最不方便 B.进行求子串运算最不方便 C. 进行串连接最不方便 D.进行串替换最不方便

## 2010年上半年

●设有如下所示的下三角矩阵A[0..8,0..8]，将该三角矩阵的非零元素(即行下标不小于列下标的所有元素)按行优先压缩存储在数组M[1..m]中，则元素A[ij](o≤i≤8，j≤i)存储在数组M的(58)中。



(58)A. B. C. D.

●若用n个权值构造一棵最优二叉树(哈夫曼树)，则该二叉树的结点总数为(59)。

(59)A.2n B.2n-1 C.2n+1 D.2n+2

●栈是一种按“后进先出”原则进行插入和删除操作的数据结构，因此，(60)必须用栈。

(60)A.实现函数或过程的递归调用及返回处理时 B.将一个元素序列进行逆置

C.链表结点的申请和释放 D.可执行程序的装入和卸载

●若对一个链表最常用的操作是在末尾插入结点和删除尾结点，则采用仅设尾指针的单向循环链表(不含头结点)时，(65)。

(65)A.插入删除操作的时间复杂度都为O(1) B.插入和删除操作的时间复杂度都为O(n)

C.插入操作的时间复杂度为O(1),删除操作的时间复杂度为O(n)D.插入操作的时间复杂度为O(n),删除操作的时间复杂度为O(1)

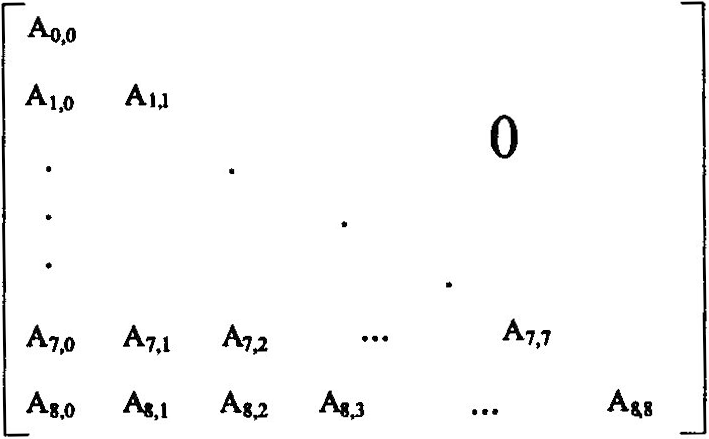
## 2011年上半年

● 算术表达式采用逆波兰式表示时不用括号，可以利用 (20) 进行求值，与逆波兰式ab-cd+\*对应的中缀表达式是 (21)。

(20)A. 数组 B. 栈 C. 队列 D. 散列表

(21)A. a-b+c\*d B. (a-b)\*c+d C. (a-b)\*(c+d) D. a-b\*c+d

● 设下三角矩阵(上三角部分的元素值都为0)A[0..n,0..n]如下所示，将该三角矩阵的所有非零元素(即行下标不小于列下标的元素)按行优先压缩存储在容量足够大的数组M[]中(下标从1开始)，则元素A[I,j](0≤i≤n,j≤i)存储在数组M的 (57) 中。



(57)A.  B.  C.  D. 

● 在 (59) 中，任意一个结点的左、右子树的高度之差的绝对值不超过1。

(59)A. 完全二叉树 B. 二叉排序树 C. 线索二叉树 D. 最优二叉树

## 2011年下半年

● 若二维数组arr[1..M, 1..N]的首地址为base，数组元素按列存储且每个元素占用K个存储单元，则元素arr[I,j]在该数组空间的地址为 (21) 。

(21)A. base+((i-1)\*M+j-1)\*K B. base+((i-1)\*N+j-1)\*K C. base+((j-1)\*M+i-1)\*K D. base+((j-1)\*N+i-1)\*K

● 对于线性表(由n个同类元素构成的线性序列)，采用单向循环链表存储的特点之一是 (58) 。

(58)A. 从表中任意结点出发都能遍历整个链表 B. 对表中的任意结点可以进行随机访问

C. 对于表中的任意一个结点，访问其直接前驱和直接后继结点所用时间相同

D. 第一个结点必须是头结点

● 一棵满二叉树，其每一层结点个数都达到最大值，对其中的结点从1开始顺序编号，即根结点编号为1，其左、右孩子结点编号分别为2和3，再下一层从左到右的编号为4、5、6、7，依此类推，每一层都从左到右依次编号，直到最后的叶子结点层为止，则用 (60) 可判定编号为m和n的两个结点是否在同一层。

(60)A. log2 m = log2 n B. = C. +1= D. =+1

● (61)是由权值集合{8，5，6，2}构造的哈夫曼树(最优二叉树)。

(61)A. B. C. D.



## 2012年上半年

●对于二维数组a[l．．N，l．．N】中的一个元素a[i,j](1≤i,j≤N)，存储在a[i,j]之前的元素个数 (21) 。

(21)A．与按行存储或按列存储方式无关 B．在i-j时与按行存储或按列存储方式无关

C．在按行存储方式下比按列存储方式下要多 D．在按行存储方式下比按列存储方式下要少

●对于一个长度大于l且不存在重复元素的序列，令其所有元素依次通过一个初始为空的队列后，再通过一个初始为空的栈。设队列和栈的容量都足够大，一个序列通过队列(栈)的含义是序列的每个元素都入队列(栈)且出队列(栈)一次且仅一次。对于该序列在上述队列和栈上的操作，正确的叙述是 (57) 。

(57)A．出队序列和出栈序列一定相同 B．出队序列和出栈序列一定互为逆序

C．入队序列与出队序列一定相同，入栈序列与出栈序列不一定相同

D．入栈序列与出栈序列一定互为逆序，入队序列与出队序列不一定互为逆序

●若n2、n1、n0分别表示一个二叉树中度为2、度为1和叶子结点的数目(结点的度定义为结点的子树数目)，则对于任何一个非空的二叉树， (59) 。

(59)A.n2一定大于n1 B.n1-定大于n0 C.n2-定大于n0 D. n0-定大于n2

## 2012年下半年

● 若某二叉树的后序遍历序列为KBFDCAE，中序遍历序列为BKEFACD，则该二叉树为 (58) 。



● 下图所示为一棵M阶B-树，M最有可能的值为 (61) 。



(61) A．I B．2 C．3 D．4

● 霍夫曼编码将频繁出现的字符采用短编码，出现频率较低的字符采用长编码。具体的操作过程为：i)以每个字符的出现频率作为关键字构建最小优先级队列；ii)取出关键字最小的两个结点生成子树，根节点的关键字为孩子节点关键字之和，并将根节点插入到最小优先级队列中，直至得到一颖最优编码树。

霍夫曼编码方案是基于 (64) 策略的。用该方案对包含a到f六个字符的文件进行编码，文件包含100,000个字符，每个字符的出现频率(用百分比表示)如下表所示，则与固定长度编码相比，该编码方案节省了 (65) 存储空间。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 字 符 | a | b | c | d | e | f |
| 出现频率 | 18 | 32 | 4 | 8 | 12 | 26 |

(64) A．分治 B．贪心 C．动态规划 D．回溯

(65) A．21% B．27% C．18% D．36%

## 2013年上半年

●设元素序列a、b、c、d、e、f经过初始为空的栈S后，得到出栈序列c e d f b a，则栈S的最小容量为(52)。

(52) A.3 B.4 C.5 D.6

● 输出受限的双端队列是指元素可以从队列的两端输入、但只能从队列的一端输出，如下图所示。若有el、e2、e3、e4依次进入输出受限的双端队列，则得不到输出序列(53)。



(53) A.e4、e3、e2、el B.e4、e2、el、e3 C.e4、e3、el、e2 D.e4、e2、C3、C1

● 一个高度为h的满二叉树的结点总数为2h一1，从根结点开始，自上而下、同层次结点从左至右，对结点按照顺序依次编号，即根结点编号为1，其左、右孩子结点编号分别为2和3，再下一层从左到右的编号为4、5、6、7，依此类推。那么，在一棵满二叉树中，对于编号为m和n的两个结点，若n=2m+1，则(64)结点。

(64) A.m是n的左孩子 B.m是n的右孩子 C.n是m的左孩子 D.n是m的右孩子

## 2013年下半年

●算数表达式a+(b-c)\*d的后缀式是(22)(－、＋、\*表示算数的减加乘运算，运算符的优先级和结合性遵循惯例)。

(22)A.bc－d\*a＋ B.abc－d\*+ C.ab＋c－d\* D.abcd－\*＋

●以下关于线性表存储结构的叙述，正确的是(57)

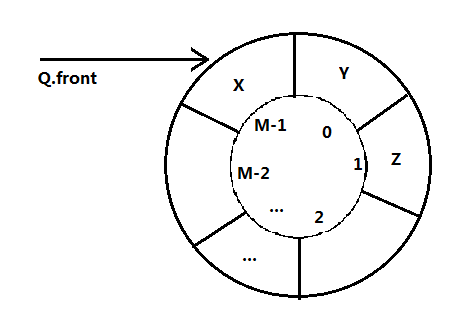
(57)A.线性表采用顺序存储结构时，访问表中任意一个指定序号元素的时间复杂度为常量级

B.线性表采用顺序存储结构时，在表中任意位置插入新元素的运算时间复杂度为常量级

C.线性表采用链式存储结构时，访问表中任意一个指定序号元素的时间复杂度为常量级

D.线性表采用链式存储结构时，在表中任意位置插入新元素的运算时间复杂度为常量级

●设循环队列的定义中有front和size两个域变量，其中front表示队头元素的指针，size表示队列的长度，如下图所示(队列长度为3，对头元素为X，队尾元素为Z)，设队列的存储空间容量为M，则队尾元素的指针为(58)



(58)A.(Q.front+Q.size-1) B.(Q.front+Q.size-1+M)%M C.(Q.front-Q.size) D.(Q.front-Q.size+M)%M

●以下关于哈夫曼树的叙述，正确的是(60)

(60)A.哈夫曼树一定是满二叉树，其每层结点数都达到最大值

B.哈夫曼树一定是平衡二叉树，其每个结点左右子树的高度差为-1,0,1

C.哈夫曼树中左孩子结点的权值小于父节点、右孩子结点的权值大于父节点

D.哈夫曼树中叶子结点的权值越小则距离树根越远、叶子结点的权值越大则距离树根越近

## 2014年上半年

●若对线性表的最常用操作是访问任意指定序号的元素，并在表尾加入和删除元素，则适宜采用 (57) 存储。

(57)A．顺序表 B．单链表 C．双向链表 D．哈希表

●某二叉树如图所示，若进行顺序存储(即用一堆数组元素存储该二叉树中的结点且通过下标反映结点间的关系，例如，对于下标为i的结点，其左孩子的下标为2i、右孩子的下标为2i+1)，则该数组的大小至少为(58)；若采用三叉链表存储该二叉树(各个结点包括结点的数据、父结点指针、左孩子指针、右孩子指针)，则该链表的所有结点中空指针的数目为(59)。



(58)A.6 B.10 C.12 D.15

(59)A.6 B.8 C.12 D.14

●某双端队列如下图所示，要求元素进出队列必须在同一端口，即从A端进入的元素必须从A端出、从B端进入的元素必须从B端出，则对于4个元素的序列el、 e2、e3、 e4，若要求前2个元素(el、 e2)从A端口按次序全部进入队列，后两个元素(e3、e4)从B端口按次序全部进入队列，则可能得到的出队序列是(60)。



(60)A. el、 e2、 e3、 e4 B. e2、 e3、 e4、 el C. e3、 e4、el、 e2 D. e4、 e3、 e2、 el

## 2014年下半年

●对于线性表，相对于顺序存储：采用链表存储的缺点是 (57) 。

(57)A.数据元素之间的关系需要占用存储空间，导致存储密度不高 B.表中结点必须占用地址连续的存储单元，存储密度不高

C.插入新元素时需要遍历整个链表，运算的时间效率不高 D.删除元素时需要遍历整个链表，运算的时间效率不高

●若一个栈初始为空，其输入序列是1，2，3…，n-1，n，其输出序列的第一个元素为k(1≤k≤)，则输出序列的最后一个元素是(58)。

(58)A.值为n的元素 B.值为l的元素 C.值为n-k的元素 D.不确定的

●在某个二叉查找树(即二叉排序树)中进行查找时，效率最差的情形是该二叉查找树是 (59) 。

(59)A.完全二叉树 B.平衡二叉树 C.单枝树 D.满二叉树

●已知一个文件中出现的各字符及其对应的频率如下表所示。若采用定长编码，则该文件中字符的码长应为 (64) 。若采用Huffman编码，则字符序列“face”的编码应为 (65) 。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 字符 | a | b | c | d | e | f |
| 频率(%) | 45 | 13 | 12 | 16 | 9 | 5 |

(64)A.2 B.3 C.4 D.5

(65)A.110001001101 B.001110110011 C.101000010100 D.010111101011

## 2015年上半年

●(2015年上半年)与算术表达式“(a+(b-c))\*d”对应的树是 (21) 。



●设某循环队列Q的定义中有front和rear两个域变量，其中，front指示队头元素的位置，rear指示队尾元素之后的位置，如下图所示。若该队列的容量为M，则其长度为 (57) 。



(57)A.(Q.rear-Q.front+1) B.(Q.rear-Q.front+M) C.(Q.rear-Q.front+1)%M D.(Q.rear-Q.front+M)%M

●设栈S和队列Q的初始状态为空，元素a b c d e f g依次进入栈S。要求每个元素出栈后立即进入队列Q，若7个元素出队列的顺序为b d f e c a g，则栈S的容量最小应该是 (58) 。

(58)A.5 B.4 C.3 D.2

●某二叉树的先序遍历序列为c a b f e d g，中序遍历序列为a b c d e f g，则该二叉树是 (59) 。

(59)A.完全二叉树 B.最优二叉树 C.平衡二叉树 D.满二叉树

●优先队列通常采用 (62) 数据结构实现，向优先队列中插入一个元素的时间复杂度为 (63) 。

(62)A.堆 B.栈 C.队列 D.线性表

(63)A.Θ(n) B.Θ(1) C.Θ(lgn) D.Θ(n2)

## 2015年下半年

●表达式采用逆波兰式表示时，利用(22)进行求值。

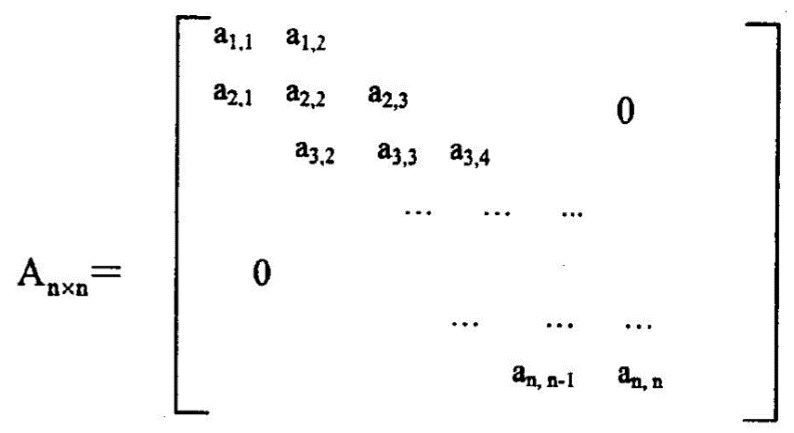
(22)A.栈 B.队列 C.符号表 D.散列表

●对于一个长度为n(n>1)且元素互异的序列，令其所有元素依次通过一个初始为空的栈后，再通过一个初始为空的队列。假设队列和栈的容量都足够大，且只要栈非空就可以进行出栈操作，只要队列非空就可以进行出队操作，那么以下叙述中，正确的是(57)。

(57)A.出队序列和出栈序列一定互为逆序 B.出队序列和出栈序列一定相同

C.入栈序列与入队序列一定相同 D.入栈序列与入队序列一定互为逆序

●设某n阶三对角矩阵An\*n的示意图如下图所示。若将该三对角矩阵的非零元素按行存储在一维数组B[k](1<=k<=3\*n-2)中，则k与i、j的对应关系是(58)。



(58)A.k=2i+j-2 B.k=2i-j+2 C.k=3i+j-1 D.k=3i-j+2

●对于非空的二叉树，设D代表根结点，L代表根结点的左子树，R代表根结点的右子树。若对下图所示的二叉树进行遍历后的结点序列为7 6 5 4 3 2 1,则遍历方式是(59)。



(59)A.LRD B.DRL C.RLD D.RDL

## 2016年上半年

●若元素以a、b，c、d、e的顺序进入一个初始为空的栈中，每个元素进栈、出栈各1次，要求出栈的第一个元素为d，则合法的出栈序列共有(57)种。

(57)A.4 B.5 C.6 D.24

●设有二叉排序树(或二叉查找树)如下图所示，建立该二叉树的关键码序列不可能是(58)。



(58)A.23 31 17 19 11 27 13 90 61 B.23 17 19 31 27 90 61 11 13

C.23 17 27 19 31 13 11 90 61 D.23 31 90 61 27 17 19 11 13

●若一棵二叉树的高度(即层数)为h，则该二叉树(59)。

(59)A.有2h个结点 B.有2h-1个结点 C.最少有2h-1个结点 D.最多有2h-1个结点

## 2016年下半年

●二维数组a[l..N,1..N]可以按行存储或按列存储。对于数组元素a[i,j](1,j),当(22)时，在按行和按列两种存储方式下，其偏移量相同。

(22)A.ij B.i=j C.i>j D.i<j

●设有一个包含n个元素的有序线性表。在等概率情况下删除其中的一个元素，若采用顺序存储结构，则平均需要移动(58)个元素；若采用单链表存储，则平均需要移动(59)个元素。

(58)A.1 B.(n-1)/2 C.log n D.n

(59)A.0 B.1 C. (n-1)/2 D. n/2

●具有3个结点的二又树有\_(60)\_的种形态。

(60)A.2 B.3 C.5 D.7

●以下关于二叉排序树(或二叉查找树、二叉检索树)的叙述中，正确的是\_(61)\_。

(61)A.对二叉排序树进行先序、中序和后序遍历，都得到结点关键字的有序序列

B.含有n个结点的二叉排序树高度为+1

C.从根到任意一个叶子结点的路径上，结点的关键字呈现有序排列的特点

D.从左到右排列同层次的结点，其关键字呈现有序排列的特点

●下表为某文件中字符的出现频率，采用霍夫曼编码对下列字符编码，则字符序列“bee”的编码为\_(62)\_；编码“110001001101”对应的字符序列为\_(63)\_。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 字符 | a | b | c | d | e | f |
| 频率(%) | 45 | 13 | 12 | 16 | 9 | 5 |

(62) A.10111011101 B.10111001100 C.001100100 D.110011011

(63) A.bad B.bee C.face D.bace

## 2017年上半年

●已知栈S 初始为空，用1表示入栈、0表示出栈，若入栈序列为，则通过栈S得到出栈序列的合法操作序列 (58) 。

(58)A.11O11O1OOO B.1O1O1O1O1O C.1OO11O1O1O D.11OO1O1OOO

●某二叉树的先序遍历序列为ABCDEF，中序遍历序列为BADCFE，则该二叉树的高度(即层数)为 (59) 。

(59)A.3 B.4 C.5 D.6

## 2017年下半年

●设S是一个长度为n的非空字符串，其中的字符各不相同，则其互异的非平凡子串(非空且不同于S本身)个数为 (57) 。 (57)A.2n-1 B. C.n(n+1)/2 D.(n+2)(n-1)/2

●假设某消息中只包含7个字符{a，b，c，d，e，f，g}，这7个字符在消息中出现的次数为{5, 24, 8, 17, 34, 4, 13}，利用哈夫曼树(最优二叉树)为该消息中的字符构造符合前缀编码要求的不等长编码。各字符的编码长度分别为(58)。

(58)A.a:4, b:2, c:3, d:3, e:2, f:4, g:3 B.a:6, b:2, c:5, d:3, e:1, f:6, g:4

C.a:3, b:3, c:3, d:3, e:3, f:2, g:3 D.a:2, b:6, c:3, d:5, e:6, f:1, g:4

●设某二叉树采用二叉链表表示(即结点的两个指针分别指示左、右孩子)。当该二叉树包含k个结点时，其二叉链表结点中必有 (59) 个空的孩子指针。

(59) A k-1 B. k C. k+l D. 2k

## 2018年上半年

●队列的特点是先进先出，若用循环单链表表示队列，则 (57) 。

(57)A.入队列和出队列操作都不需要遍历链表 B.入队列和出队列操作都需要遍历链表

C.入队列操作需要遍历链表而出队列操作不需要 D.入队列操作不需要遍历链表而出队列操作需要

●设有n阶三对角矩阵A，即非零元素都位于主对角线以及与主对角线平行且紧邻的两条对角线上，现对该矩阵进行按行压缩存储，若其压储空间用数组B表示，A的元素下标从0开始，B的元素下标从1开始。已知A[0,0]存储在B[1],A[n-l,n-l]存储在B[3n-2],那么非零元素A[i,j](0≤i<n,0≤j<n,|i-j|≤1)存储在B[ (58) ]。

(58)A.2i+j-i B.2i+j C.2i+j+i D.3i-j+i

●对下面的二叉树进行顺序存储(用数组MEM表示），已知结点A、B、C在MEM中对应元素的下标分别为1、2、3,那么结点D、E、F对应的数组元素下标为 (59) 。



(59)A.4、5、6 B.4、7、10 C.6、7、8 D.6、7、14

## 2018年下半年

●栈的特点是后进先出，若用单链表作为栈的存储结构，并用头指针作为栈顶指针，则 (57) 。

(57)A.入栈和出栈操作都不需要遍历链表 B.入栈和出栈操作都需要遍历链表

C.入栈操作需要遍历链表而出栈操作不需要 D.入栈操作不需要遍历链表而出栈操作需要

●已知某二叉树的先序遍历序列为ABCDEF、中序遍历序列为BADCFE，则可以确定该二叉树 (58) 。

(58)A.是单支树(即非叶子结点都只有一个孩子) B.高度为4(即结点分布在4层上)

C.根结点的左子树为空 D.根结点的右子树为空

●可以构造出下图所示二叉排序树(二叉检索树、二叉查找树)的关键码序列是 (59) 。



(59)A.10 13 17 19 23 27 31 40 65 91 B.23 40 91 17 19 10 31 65 27 13

C.23 19 40 27 17 13 10 91 65 31 D.27 31 40 65 91 13 10 17 23 19

## 2019年上半年

●某n阶的三对角矩阵A如下图所示，按行将元素存储在一维数组M中，设存储在M[l],那么 (l<=i,j<=n且位于三条对角线中)存储在M (57) 。

(57)A.i+2j B.2i+j C.i+2j-2 D.2i+j-2

●具有3个结点的二叉树有5种，可推测出具有4个结点的二叉树有 (58) 种。

(58)A.10 B.11 C.14 D.15

●双端队列是指在队列的两个端口都可以加入和删除元素，如下图所示。现在要求元素进队列和出队列必须在同一端口，即从A端进队的元素必须从A端出、从B端进队的元素必须从B端出，则对于4个元素的序列a、b、c、d,若要求前2个元素(a、b)从 A端口按次序全部进入队列，后两个元素(c、d)从B端口按次序全部进入队列，则不可能得到的出队序列是 (59) 。



(59)A.d、a、b、c B.d、c、b、a C.b、a、d、c D.b、d、c、a

# 图

在线性结构(例如队列和栈)中，除第一个结点没有前驱，最后一个结点没有后继之外，每一个结点都有唯一的前驱和后继。在树形结构(例如树和二叉树)中，除根结点没有前驱外，一个结点只有一个前驱结点，但可以有若干个后继。在图结构中，一个结点的前驱和后继的个数是任意的。

## 图的基本概念

* + - 1. 图的种类

图分为有向图和无向图两种,如图 5.1所示。

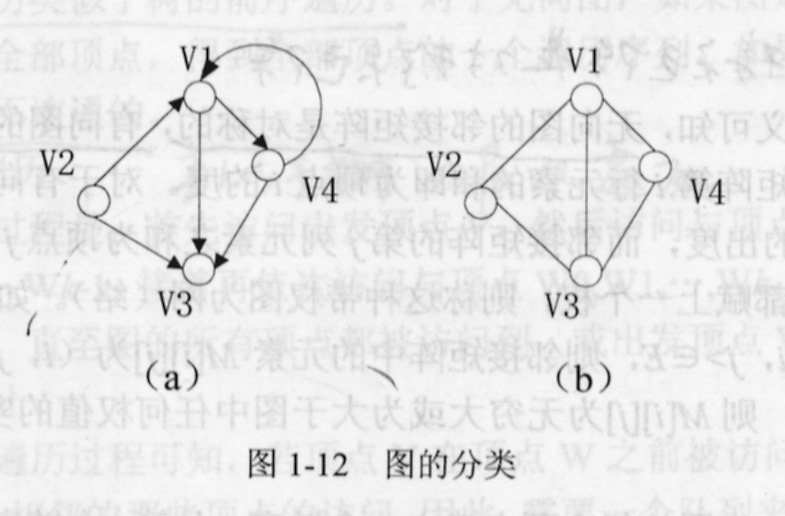


图 5.1 a是一个有向图，b是一个无向图

* + - 1. 图的表示方法

图G由两个集合V和E组成，记为G=(V，E)。通常，也将图G的顶点集和边集分别记为V(G)和 E(G)。E(G)可以是空集。若E(G)为空，则图G只有顶点而没有边。

###### 有向图边的表示方法

在有向图中，一条有向边是由两个顶点组成的有序对，有序对通常用尖括号表示。<，>表示一条有向边，是边的始点(起点)，是边的终点，<,>和< >是两条不同的有向边。

有向边也称为弧，边的始点称为弧尾，终点称为弧头。

###### 无向图边的表示方法

无向图中的边均是顶点的无序对，无序对通常用圆括号表示。在无向图G中，如果i≠j，i、j∈V，(i，j) ∈E，即i和j是G的两个不同的顶点，(i，j)是G中一条边，顶点i和顶点j相邻顶点，边(i，j)是与顶点i 和j相关联的边。

* + - 1. 回路或环

第一个顶点和最后一个顶点相同的路径称为回路或环。

* + - 1. 无向完全图

如果限定任何一条边的两个顶点都不相同，则有n个顶点的无向图至多有n(n-1)／2条边，这样的无向图称为无向完全图(图中每两个顶点之间都有无向边)。

* + - 1. 有向完全图

一个有向图至多有n(n-1)条弧，这样的有向图称为有向完全图(图中每两个顶点之间都有有向边)。

* + - 1. 顶点的度

在无向图中，一个顶点的度等于与其相邻接的顶点个数。

在有向图中，一个顶点的入度等于邻接到该顶点的顶点个数，其出度等于邻接于该顶点的个数。

* + - 1. 路径

在图G=(V，E)中，如果存在顶点序列(，……)其中=P，=Q，且(，)，(，)，…，(，)都在E中，则称顶点P到顶点Q有－条路径，并用(，，……………，)表示这条路径，路径的长度是路径的边数，这条路径的长度为k。若G是有向图，则路径也是有向的。

* + - 1. 连通图

在无向图中，如果从顶点到顶点有路径，称和是连通的。如果对于图中任意两个顶点、V,和都是连通的，则称G是连通图。

* + - 1. 连通分量

无向图中的极大连通子图(加入任何一个其它的结点便不连通的子图),如图 5.2所示为无向图及其连同分量。

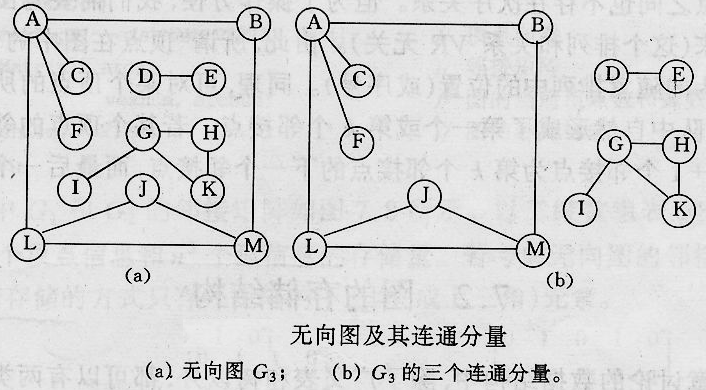


图 5.2(a)无向图，(b)的三个联通分量

* + - 1. 强连通图

有向图G中，若对于V(G)中任意两个不同的顶点和，都存在从到及从到的路径，则称G是强连通图。

* + - 1. 强连通分量

有向图中的极大强连通子图称为有向图的强连通分量，如图 5.3所示。

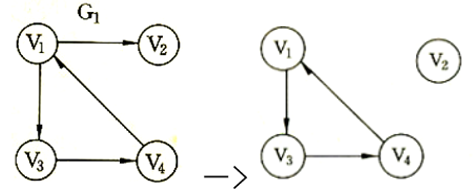


图 5.3的两个强连同分量

## 图的存储结构

最常用的图的存储结构有邻接矩阵和邻接表。

* + - 1. 邻接矩阵

邻接矩阵反映顶点邻接关系．设G=(V,E)是具有n(n≥1)个顶点的图，G的邻接矩阵M是一个n行n列的矩阵，并有若(i，j)或<i，j>∈E，则M[i][j]=1，否则，M[i][j]＝0。

如图 5.4所示的有向图和无向图对应的邻接矩阵分别为和所示。

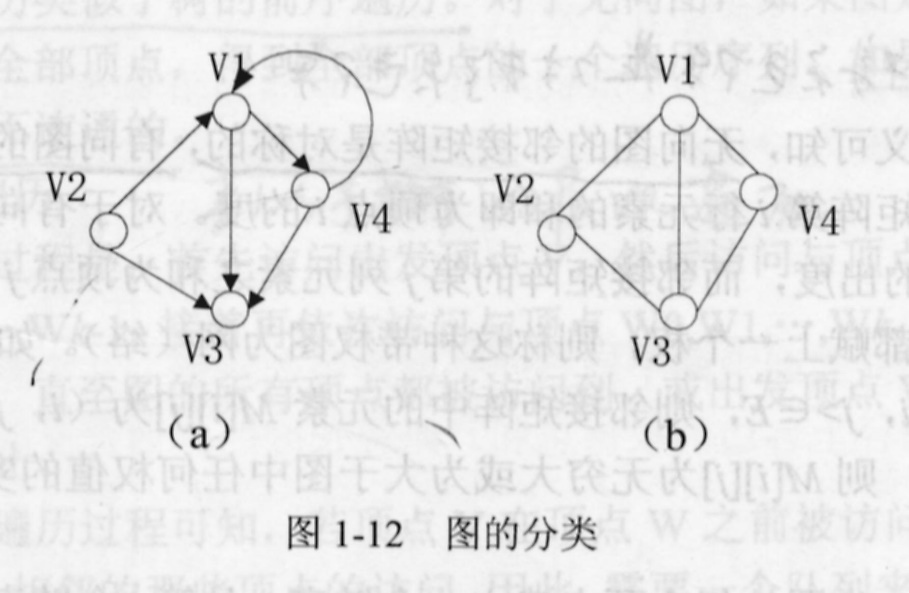


图 5.4 有向图和无向图

＝ ＝

* 无向图的邻接矩阵是对称的，且非0元素的个数为边数的2倍；有向图的邻接矩阵不一定对称，且非0元素的个数等于边数。
* 对于无向图，其邻接矩阵第i行元素的之和即为顶点i的度。
* 对于有向图，其邻接矩阵的第i行元素之和为顶点i的出度，而邻接矩阵的第j列元素之和为顶点j的入度。

若将图的每条边都赋上一个权，则称这种带权图为网(络)。如果图 G＝(V，E)是一个网，若(i，j)或<i,j>属于E，则邻接矩阵中的元素M[i][j]为(i,j)或<i，j>上的权。若(i，j)或<i,j>不属于E，则M[i][j]为无穷大或为大于图中任何权值的实数。

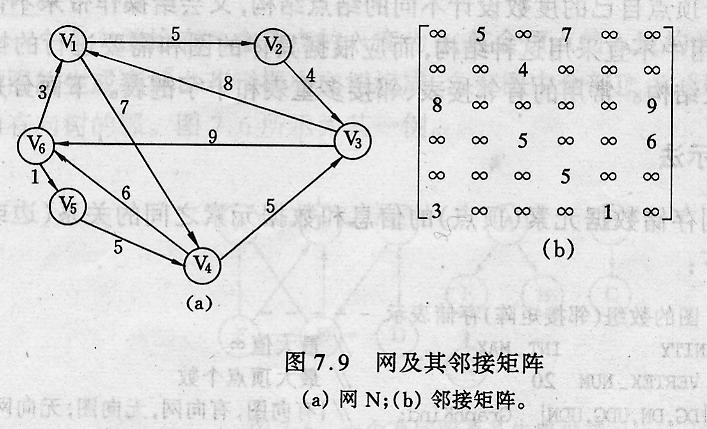


图 5.5网及其邻接矩阵 (a)网N;(b)邻接矩阵

* + - 1. 邻接表

在图的邻接表表示中，为图的每个顶点建立一个链表，且第i个链表中的结点代表与顶点i相关联的一条边或由顶点i出发的一条弧。有n个顶点的图，需用n个链表表示，这n个链表的头指针通常由顺序线性表存储。

如图 5.6所示的图对应的邻接矩阵如图 5.7所示。

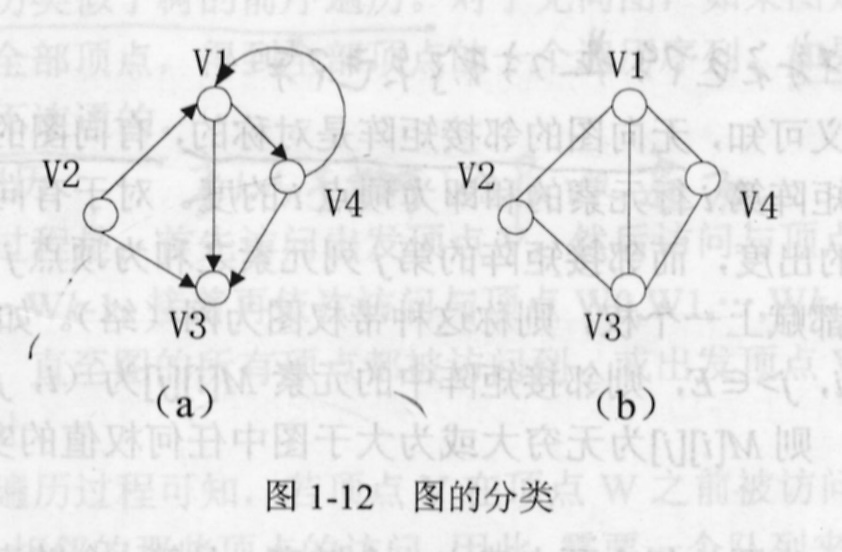


图 5.6图的分类

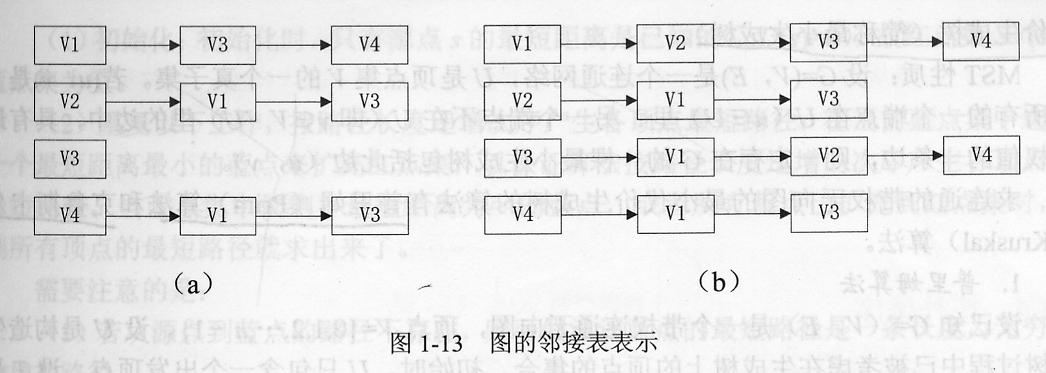


图 5.7图的邻接表表示

* 在无向图的邻接表中，对应某结点的链表的结点个数就是该顶点的度。链表中结点数为边数的2倍。
* 在有向图的邻接表中，对应某结点的链表的结点个数就是该顶点的出度。

## 图的遍历

* + - 1. 深度优先遍历

图的深度优先遍历类似于树的前序遍历。对于无向图，如果图是连通的，那么按深度优先遍历时，可遍历全部顶点，得到全部顶点的一个遍历序列。如果遍历序列没有包含所有顶点，那么该图是不连通的。

* + - 1. 广度优先遍历

类似于树的层次优先遍历。广度优先的遍历过程是：首先访问出发顶点V，然后访问与顶点V邻接的全部未被访问过的顶点、、…… ；接着再依次访问与顶点、、…… 邻接的全部未被访问过的顶点。依此类推，直至图的所有顶点都被访问到，或出发顶点V所在的连通分量的全部顶点都被访问到为止。

注意：图的遍历运算是按照某种策略访问图中的每一个顶点，是通过边或狐找邻接点的过程，不论广度搜索还是深度搜索，时间复杂度都一样。在邻接矩阵中遍历图的时间为O()，在邻接表中遍历的时间复杂度为O(n+e)。

## 最小生成树

如果连通图G的一个子图是一棵包含G的所有顶点的树，则该子图称为G的生成树。生成树是连通图的包含图中的所有顶点的极小连通子图。值得注意的是，图的生成树并不唯一。从不同的顶点出发进行遍历，可以得到不同的生成树。

含有n个顶点的连通图的生成树有n个顶点和n-1条边。对一个带权的图(网)，在一棵生成树中，各条边的权值之和称为这棵生成树的代价。其中代价最小的生成树称为最小代价生成树(简称最小生成树)。

MST性质：设G=(V，E)是一个连通网络，U是顶点集V的一个真子集。若(u，v)是G中所有的一个端点在U(u∈U)里，另一个端点不在U(即v∈V-U)里的边中的具有最小权值的一条边，则一定存在 G的一棵最小生成树包括此边(u，v)。

求连通的带权无向图的最小代价生成树的算法有普里姆(Prim)算法和克鲁斯卡尔(Kruskal)算法。

* + - 1. 普里姆算法

算法从U＝{}(€V),TE={}开始，重复执行下述操作：在所有u€U，v€V－U的边(u，v)€E中找一条代价最小的边(，)并入集合TE，同时并入U,直至U＝V为止。此时TE中必有n-1条边，则T＝(V，TE)为N的最小生成树。

过程描述：

###### U={},V－U={,,,,}, TE={}

###### U={v1,v3},V－U={v2,v4,v5,v6},TE={(v1,v3)}

###### U={ v1,v3,v6},V－U={v2,v4,v5},TE={(v1,v3),(v3,v6)}

###### U={v1,v3,v6,v4,}, V－U={v2,v5},TE={(v1,v3),(v3,v6),(v6,v4)}

###### U={ v1,v3,v6,v4,v2,}, V－U={v5},TE={(v1,v3),(v3,v6),(v6,v4),(v3,v2)}

###### U={ v1,v3,v6,v4,v2,v5},V－U={},TE={(v1,v3),(v3,v6),(v6,v4),(v3,v2),(v2,v5)}

普里姆算法的特点是当前形成的集合TE始终是一颗树。因为每次添加的边是使树中的权尽可能小，因此这是一种**贪心**的策略。

普里姆算法的时间复杂度为*O*()，与图中边数无关，所以适用于稠密图。

* 稀疏图：有很少条边或弧(如e<nlogn)的图
* 稠密图：有很多条边或弧(如e>nlogn)的图

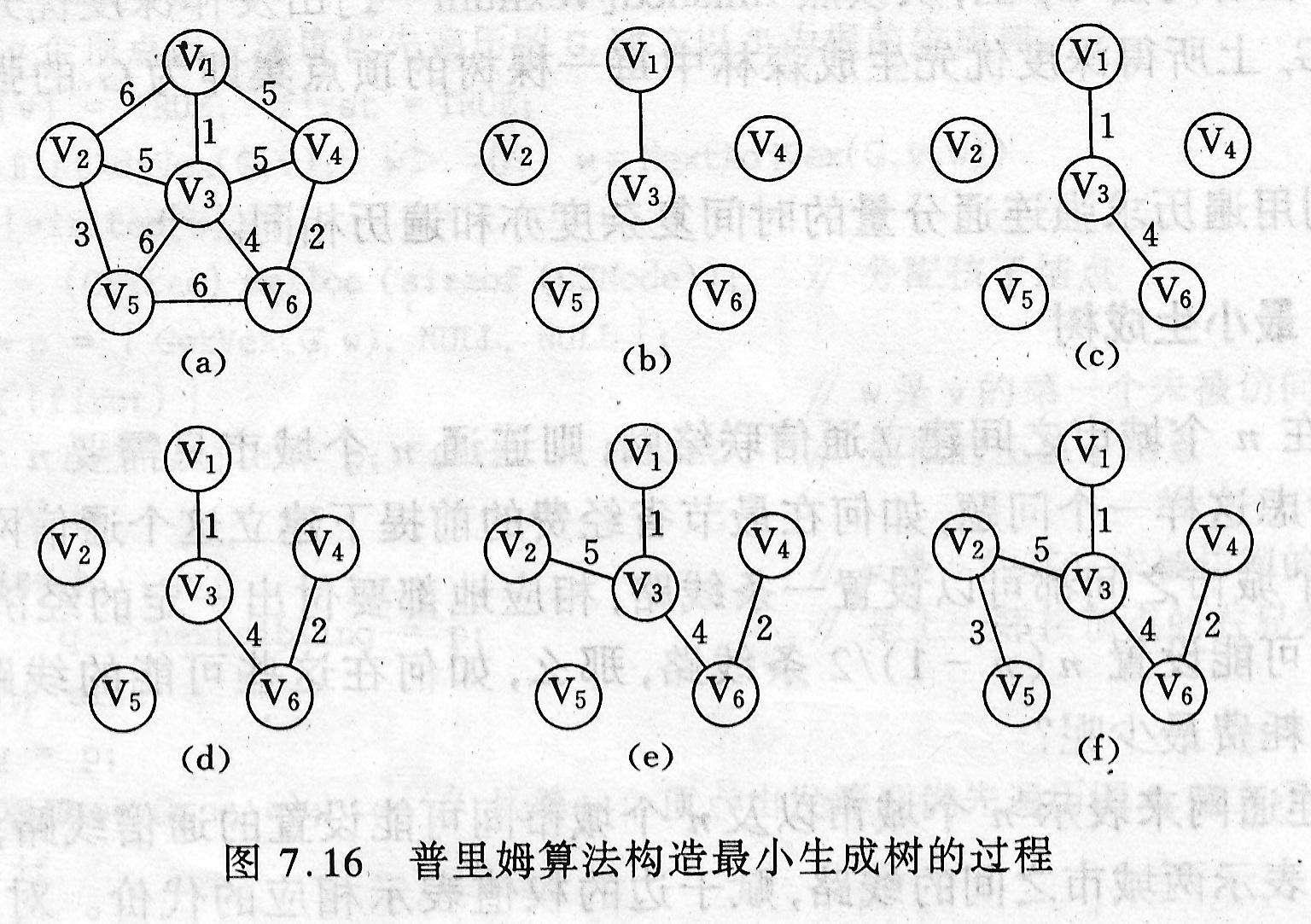


图 5.8普里姆算法构造最小生成树的过程

* + - 1. 克鲁斯卡尔算法

设T的初始状态只有n个顶点而无边的森林T＝(V，*Ф*),按边长递增的顺序选择E中n-1安全边(u，v)并加入T，生成最小生成树。所谓安全边是指两个端点分别是森林T里两棵树中的顶点的边。加入安全边，可将森林中的两棵树连成一棵更大的树，因为每一次添加到T中的边均是当前权值最小的安全边，MST性质也能保证最终的T 是一棵最小生成树。

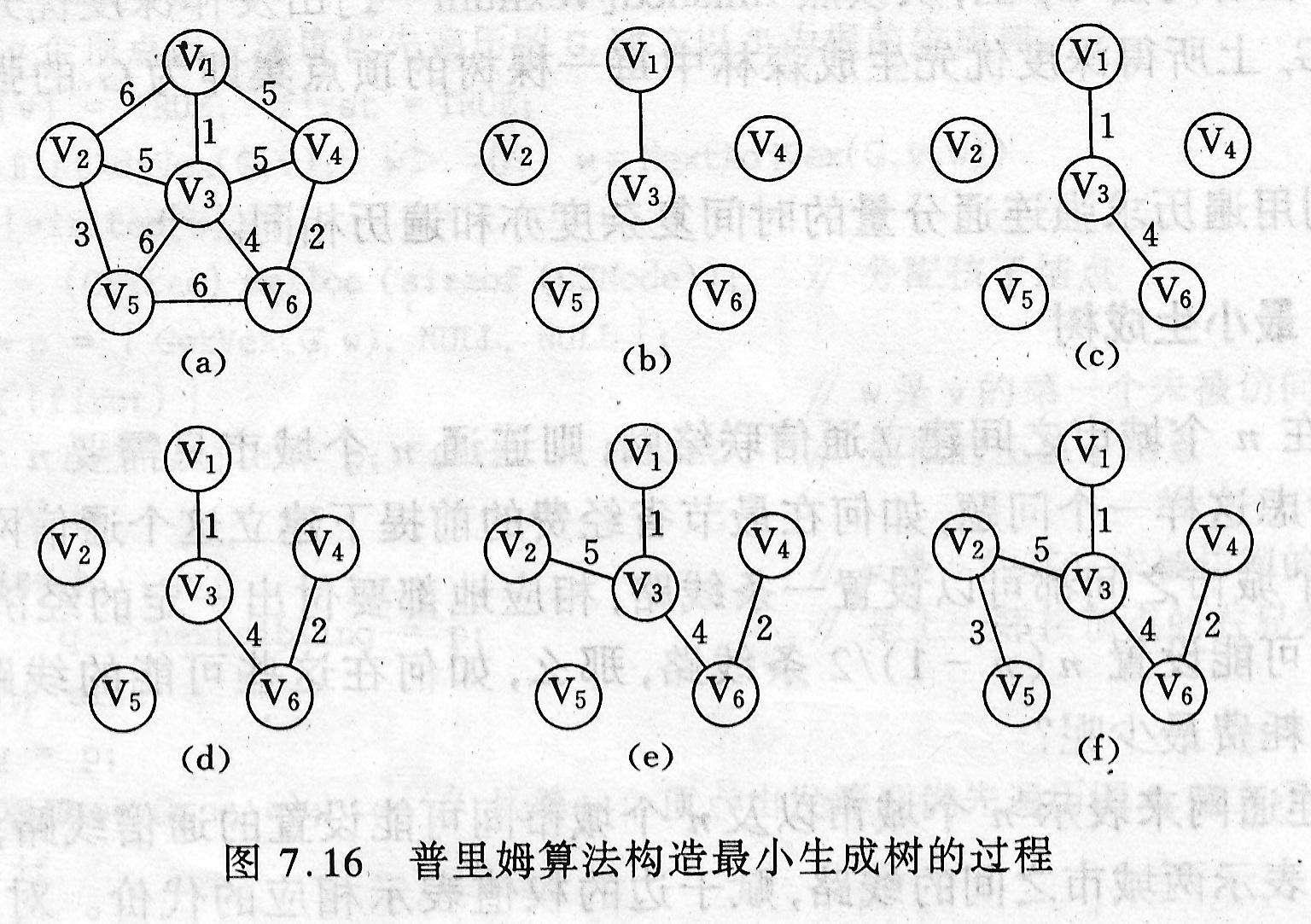


图 5.9图T

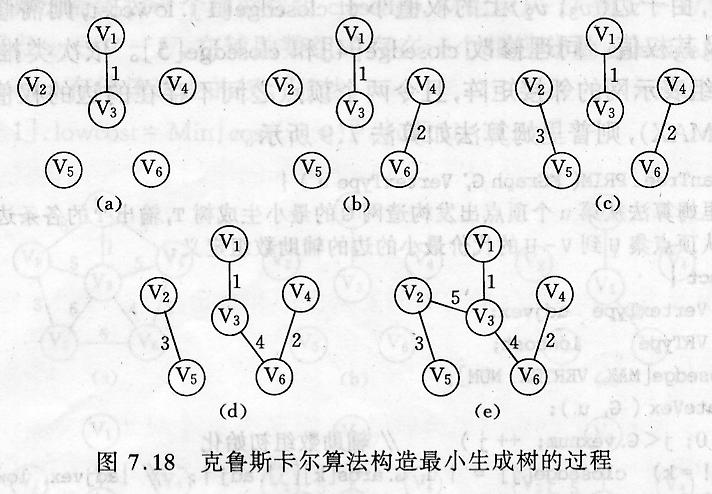


图 5.10卡鲁斯卡尔算法构造最小生成树的过程

克鲁斯卡尔算法的特点是当前形式的集合T除最后的结果外，始终是一个森林。克鲁斯卡尔算法的时间复杂度为O()，与图中的顶点数无关，所以较适合于稀疏图。

## 最短路径

带权图的最短路径问题即求两个顶点间长度最短的路径。其中路径长度不是指路径上边数的总和，而是指路径上各边的权值总和。路径长度的的具体含义取决于边上权值所代表的意义。

* + - 1. 单源最短路径

已知有向带权图(简称有向网)G=(V，E)，找出从某个源点s∈V到V中其余各顶点的最短路径，称为单源最短路径。

求单源最短路径主要使用迪杰斯特拉(Dijkstra)提出的一种按路径长度递增产生各顶点最短路径的算法，采用的是贪心算法。

图 5.11所示带权有向图中从到其余各顶点之间的最短路径。

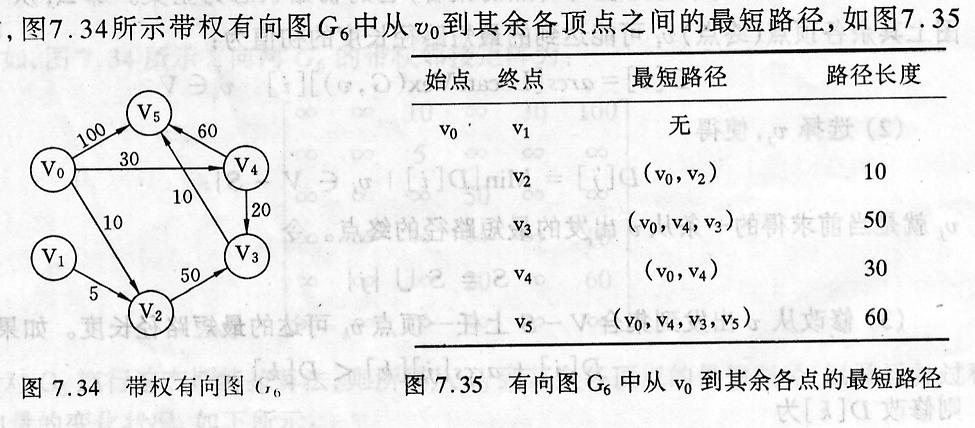


图 5.11图中从到其余各顶点之间的最短路径

* + - 1. 每一对顶点之间的最短路径

对图中每对顶点u和v，找出u到v的最短路径问题。这一问题可用每个顶点作为源点调用一次单源最短路径问题的迪杰斯特拉算法予以解决。

但更常用的是弗洛尹德(Folyd)提出的求每一对顶点之间的最短路径的算法。

## 拓扑排序

对一个有向无环图G进行拓扑排序，是将G中所有顶点排成一个线性序列，使得图中任意一对顶点u和v，若<u，v>∈E(G)，则u在线性序列中出现在v之前。这样的线性序列称为满足拓扑次序的序列，简称拓扑序列。

要注意的是：

* 若将图中顶点按拓扑次序排成一行，则图中所有的有向边均是从左指向右的。
* 若图中存在有向环，则不可能使顶点满足拓扑次序。
* 一个有向无环图可能有多个拓扑序列。
* 当有向图中存在有向环时，拓扑序列不存在。

一个大工程有许多项目组，有些项目的实行则存在先后关系，某些项目必须在其它一些项目完成之后才能开始实行。工程项目实行的先后关系可以用一个有向图来表示，工程的项目称为活动，有向图的顶点表示活动，有向边表示活动之间开始的先后关系。这种有向图称为用表活动网络，简称AOV网络，如图 5.12所示。

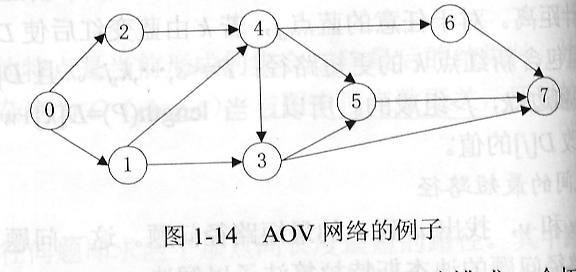


图 5.12 AOV网络的例子

对AOV网络的顶点进行拓扑排序，就是对全部活动排成一个拓扑序列，使得在AOV网络中存在一条弧<i，j>，则活动i排在活动j之前。例如，对图 5.12中有向图的顶点进行拓扑排序，可以得到多个不同的拓扑序列。

拓扑排序方法：

###### 在有向图中选择一个没有前驱的顶点并输出。

###### 从图中删除该顶点和所有以它为尾的弧。

###### 重复上述步骤，直至全部顶点均已输出，或者当前图中不存在无前驱的顶点为止。

图 5.13所示为一个拓扑排序过程:

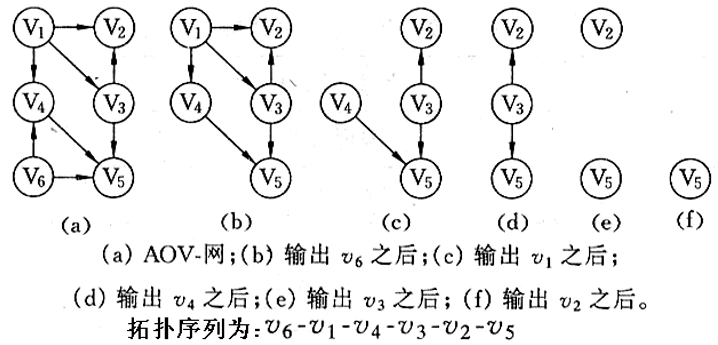


图 5.13AOV-网及其拓扑有序序列产生的过程

如图 5.12中的拓扑序列有：02143567，02143657，01243567等。

## 关键路径

在AOV网络中，如果边上的权表示完成该活动所需时间，则称这样的AOV为AOE网络。图 5.14表示一个具有10个活动的某个工程的AOE网络。图中有7个顶点，分别表示事件1～7，其中1表示工程开始状态，7表示工程结束状态，边上的权表示完成该活动所需的时间。

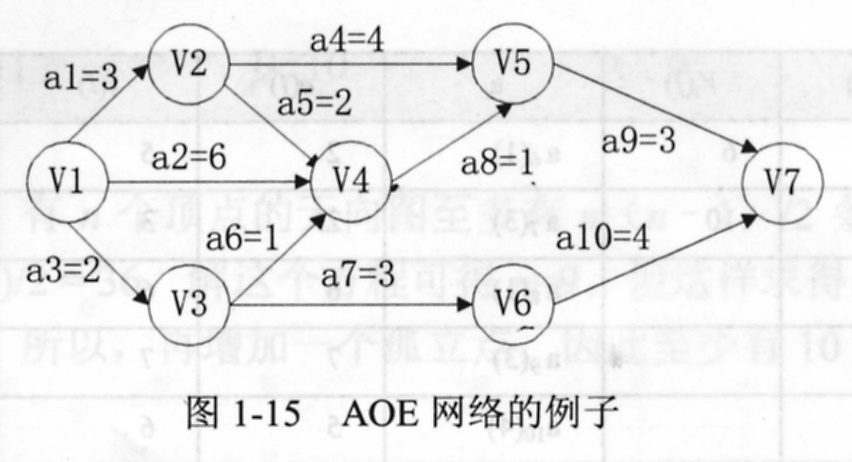


图 5.14 AOE网络的例子

因AOE网络中某些活动可以并行地进行，所以完成工程的最少时间是从开始顶点到结束顶点的最长路径长度，称从开始顶点到结束顶点的最长路径为关键路径(临界路径)。关键路径上的活动为关键活动。

# 历年真题讲解

●【1999年年】给定数据结构(V，E)，V为结点的有限集合，V＝｛V1,V2,V3,V4,V5,V6,V7,V8｝,Ｅ是Ｖ上边的集合。E＝｛<V1,V2>, <V3,V4>, <V5,V8>,<V5,V6>,

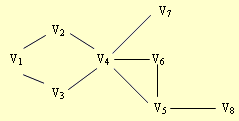
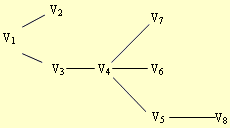
<V1,V3>, <V4,V7>, <V4,V5>, <V2,V4>, <V4,V6>｝

它所对应的图形是 Ａ ，这是 Ｂ 。图的存储结构主要有邻接表和 C ，若用邻接表来存储一个图，则需要保存一个 D 存储的结点表和若干个 E 存储的关系表(又称边表)。

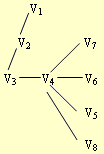
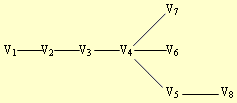
E＝｛<V1,V2>, <V3,V4>, <V5,V8>,<V5,V6>,

<V1,V3>, <V4,V7>, <V4,V5>, <V2,V4>, <V4,V6>｝

A：① ②

③ ④

B：①树 ②无向图 ③有向图 ④无向图  C：①转移矩阵 ②邻接矩阵 ③状态矩阵 ④优先矩阵

D：①顺序 ②链接 ③散列 ④分块 E：①顺序 ②链接 ③散列 ④索引

●【2003年上半年】关键路径是指AOE(Activity On Edge)网中 (4) 。

(4) A. 最长的回路 B. 最短的回路 C. 从源点到汇点(结束顶点)的最长路径 D. 从源点到汇点(结束顶点)的最短路径

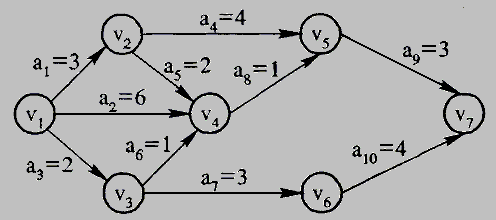
●【2003年上半年】若G是一个具有36条边的非连通无向图(不含自回路和多重边)，则图G至少有\_\_(8)\_个顶点。

(8) A. 11 B. 10 C. 9 D. 8

●【2003年上半年】已知AOE网中顶点v1～v7分别表示7个事件，弧al～a10分别表示10个活动，弧上的数值表示每个活动花费的时间，如下图所示。那么，该网的关键路径的长度为\_\_(10)\_\_，活动a6的松驰时间(活动的最迟开始时间-活动的最早开始时间)为\_\_(11)\_\_。

(10) A. 7 B. 9 C. 10 D.11

(11) A. 3 B. 2 C. 1 D.0



●【2004年上半年】若采用邻接矩阵来存储简单有向图，则其某一个顶点i的入度等于该矩阵 (8) 。

(8) A．第i行中值为1的元素个数　　 B．所有值为1的元素总数

C．第i行及第i列中值为1的元素总个数 D．第i列中值为1的元素个数

●【2005年上半年】具有n(n>0)个顶点的无向图最多含有(37) 条边。

(37)A. n(n-1) B.n(n+1)/2 C. n(n-1)/2 D. n(n+1)

●【2005年上半年】无向图中一个顶点的度是指图中 41 。

(41)A.通过该顶点的简单路径数 B.通过该顶点的回路数C.与该顶点相邻接的顶点数D.与该顶点连通的顶点数

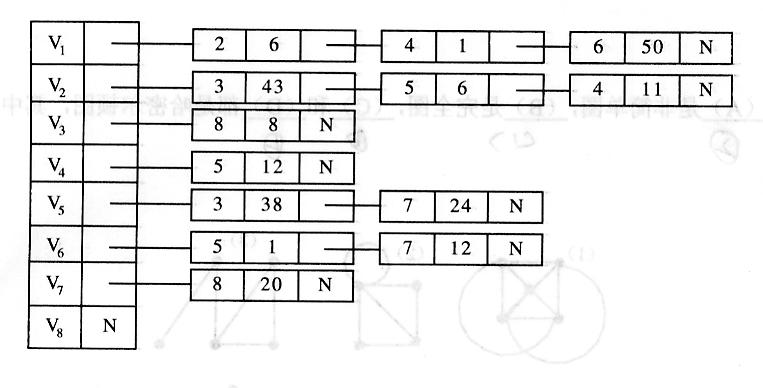
●【2005年上半年】一个具有n(n>0)个顶点的连通无向图至少有(49)条边。

(49) A.n+1 B.n C.n/2 D.n-1

●【2005年上半年】一个含有n个顶点和e条边的简单无向图，在其邻接矩阵存储结构中共有 (50) 个零元素。

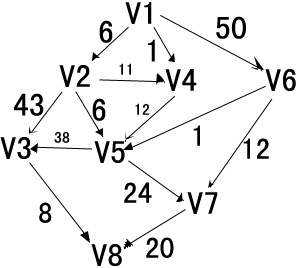
(50)A.e B.2e C.n2-e D.n2-2e

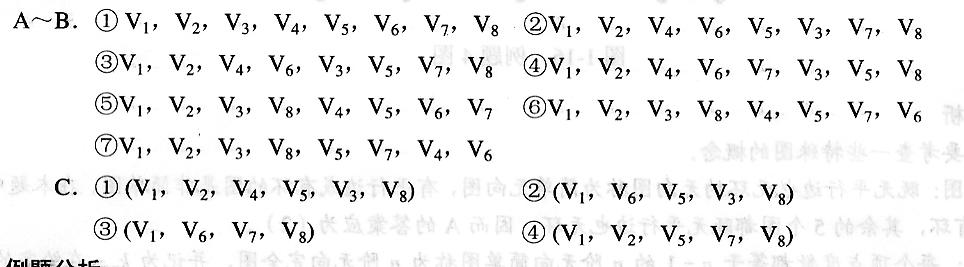
●【2005年上半年】下图所示为带权的有向图G的邻接表示法，以结点V1出发，深度遍历图G所得的结点序列为 A ,广度遍历图G的结点序列是 B ;从结点V1到V8的最短路径是: C 。



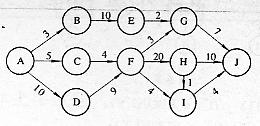
A.⑺ B.⑶ C. ⑷

图为：





●【2005年下半年】【2005年下半年】在活动图中，结点表示项目中各个工作阶段的里程碑，连接各个结点的边表示活动，边上的数字表示活动持续的时间。在下面的活动图中，从A到J的关键路径是 (16) ，关键路径长度是 (17)，从E开始的活动启动的最早时间是 (18) 。



(16)A.ABEGJ B.ADFHJ C.ACFGJ D.ADFIJ

(17)A.22 B.49 C.19 D.35

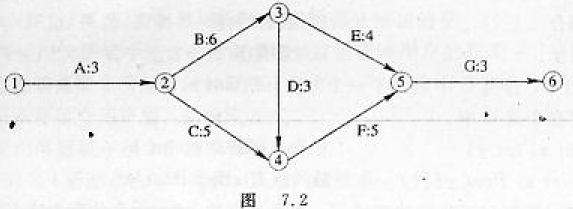
(18)A.10 B.12 C.13 D.15

●【2005年下半年】简单无向图的邻接矩阵是对称的，可以对其进行压缩存储。若无向图G有n个结点，其邻接矩阵为A[1..n,1..n],且压缩存储在B[1..k]中，则k的值至少为 (40) 。若按行压缩存储对称矩阵的上三角元素，则当n等于10时，边(V6，V3)的信息存储在B[ (41) ]中。

(40)A. B. C. D. 

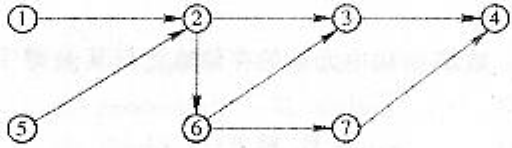
(41)A.18 B.19 C.20 D.21

●【2006年上半年】某工程计划图如图所示，弧上的标记为作业编码及其需要的完成时间(天)，作业E最迟应在第 (27) 天开始。



(27)A.7 B.9 C.12 D.13

●【2006年上半年】拓扑排序是无环有向图中所有顶点的一个线性序列，图中任意路径中的各个顶点图的拓扑序列中保持先后关系，(52)为图所示有向图的一个拓扑序列。

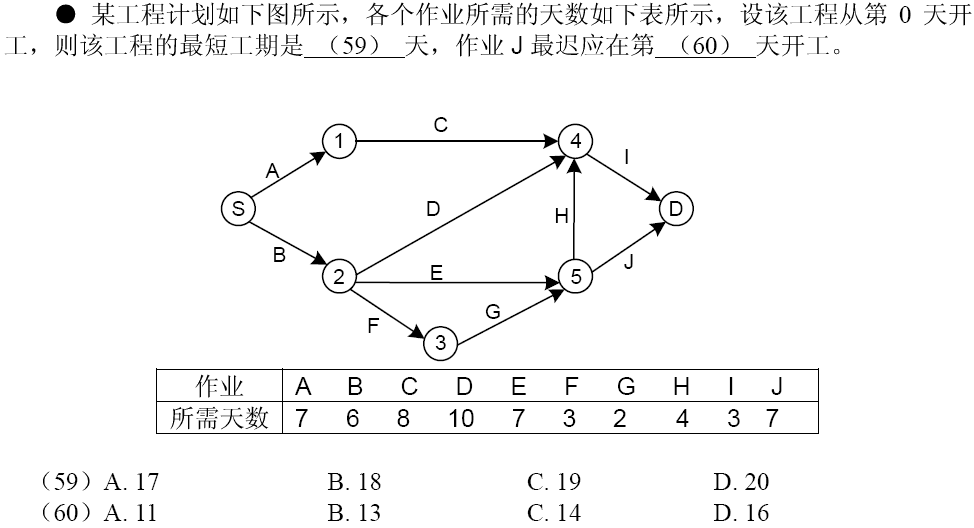


(52)A.1234567 B.1526374 C.5126347 D.5123764

●【2006年下半年】求单源点最短路径的迪杰拉斯特(Dijkstra)算法是按(57)顺序求源点到各顶点的最短路径的。

(57)A.路径长度递减 B.路径长度递增 C.顶点编号递减 D.顶点编号递增

●【2007年上半年】某工程计划如下图所示，各个作业所需的天数如下表所示，设该工程从第0天开工，则该工程的最短工期是 (59) 天，作业J最迟应在第 (60) 天开工。



(59) A.17 B.18 C.19 D.20

(60) A.11 B.13 C.14 D.16

●【2007年下半年】拓扑排序是指有向图中的所有顶点排成一个线性序列的过程，若在有向图中从顶点vi到vj有一条路径，则在该线性序列中，顶点vi必然在顶点vj之前。因此，若不能得到全部顶点的拓扑排序序列，则说明该有向图一定 (57) 。

(57)A.包含回路 B.是强连通图 C.是完全图 D.是有向树

●【2007年下半年】迪杰斯特拉(Dijkstra)算法按照路径长度递增的方式求解单源点最短路径问题，该算法运用了 (63) 算法策略。

(63)A.贪心B.分而治之C.动态规划D.试探＋回溯

●【2008年上半年】设一个包含N个顶点、E条边的简单有向图采用邻接矩阵存储结构(矩阵元素A[i][j]等于1/0分别表示顶点i与顶点j之间有/无弧)，则该矩阵的元素数目为 60 ,其中非零元素数目为 61 .

(60)A.E2 B.N2 C. N2-E2 D. N2+E2

(61)A.N B.N+E C.E D.N-E

●【2008年下半年】(59)的邻接矩阵是一个对称矩阵。

(59)A.无向图 B.AOV网 C.AOE网 D.有向图

●【2008年下半年】具有n个顶点、e条边的图采用邻接表存储结构，进行深度优先遍历和广度优先遍历运算的时间复杂度均为(63)。

(63)A.O(n2) B.O(e2) C.O(n\*e) D.O(n+e)

●【2009年上半年】下面关于图(网)的叙述，正确的是(58)。

(58)A.连通无向网的最小生成树中，顶点数恰好比边数多1 B.若有向图是强连通的，则其边数至少是顶点数的2倍

C.可以采用AOV网估算工程的工期 D.关键路径是AOE网中源点至汇点的最短路径

●【2009年下半年】邻接矩阵和邻接表是图(网)的两种基本存储结构，对于具有n个顶点、e条边的图，(59)。

(59)A.进行深度优先遍历运算所消耗的时间与采用哪一种存储结构无关

B.进行广度优先遍历运算所消耗的时间与采用哪一种存储结构无关

C.采用邻接表示图时，查找所有顶点的邻接顶点的时间复杂度为O(n\*e)

D.采用邻接矩阵表示图时，查找所有顶点的邻接顶点的时间复杂度为O(n2)

●【2011年上半年】设一个包含N个顶点、E条边的简单无向图采用邻接矩阵存储结构(矩阵元素A[i][j]等于1/0分别表示顶点i与顶点j之间有/无边)，则该矩阵中的非零元素数目为 (60) 。

(60)A. N B. E C. 2E D. N+E

●【2011年下半年】设一个包含N个顶点的度是指图中与该顶点相邻接的顶点数。若无向图G中的顶点数为n，边数为e,则所有顶点的度数之和为\_\_\_\_\_\_\_\_\_

A.n\*e B.n+e C.2n D.2e

●【2011年下半年】迪杰斯特拉算法用于求解图上的单源点最短路径。该算法按路径长度递增次序产生最短路径，本质上说，该算法是一种基于（62）策略的算法。

(62)A．分治 B.动态规则 C.贪心 D.回溯

●【2012年下半年】拓扑排序是将有向图中所有顶点排成一个线性序列的过程，并且该序列满足：若在AOV网中从顶点vi到vj有一条路径，则顶点vi必然在顶点vj之前。对于下面所示的有向图， (60) 是其拓扑序列。



(60)A．1234576 B．1235467 C．2135476 D．2134567

●【2013年下半年】在一个有时间图G的拓扑序列中，定点Vi排列在Vj之前，说明图G中(59)

(59)A.一定存在弧(Vi，Vj) B.一定存在弧(Vj, Vi)

C.可能存在Vi到Vj的路径，而不可能存在Vj到Vi的路径 D.可能存在Vj到Vi的路径，而不可能存在Vi到Vj的路径

●【2014年上半年】Prim算法和Kruscal算法都是无向连通网的最小生成树的算法，Prim算法从一个顶点开始，每次从剩余的顶点中加入一个顶点，该顶点与当前的生成树中的顶点的连边权重最小，直到得到一颗最小生成树；Kruscal算法从权重最小的边开始，每次从不在当前的生成树顶点中选择权重最小的边加入，直到得到一颗最小生成树，这两个算法都采用了 (64) 设计策略，且 (65) 。

(64)A．分治 B．贪心 C．动态规划 D．回溯

(65)A．若网较稠密，则Prim算法更好 B．两个算法得到的最小生成树是一样的

C．Prim算法比Kruscal算法效率更高 D．Kruscal算法比Prim算法效率更高

●【2015年下半年】设一个包含n个顶点、e条弧的简单有向图采用邻接矩阵存储结构(即矩阵元素A[i][j]等于1或0，分别表示顶点i与顶点j之间有弧或无弧，则该矩阵的非零元素数目为(61)。

(61)A.e B.2e C.n-e D.n+e

●【2016年上半年】以下关于图的遍历的叙述中，正确的是(61)。

(61)A.图的遍历是从给定的源点出发对每一个顶点仅访问一次的过程 B.图的深度优先遍历方法不适用于无向图

C.使用队列对图进行广度优先遍历 D.图中有回路时则无法进行遍历

●【2016年下半年】拓扑序列是有向无环图中所有顶点的一个线性序列，若有向图中有存在弧<v，w>或存在从顶点v到w的路径，则在该有向图的任一拓扑序列中，v一定在w之前。下面有向图的拓扑序列是 (57) 。



(57)A.41235 B.43125 C.42135 D.41325

●【2017年下半年】以下关于无向连通图G的叙述中，不正确的是 (60) 。

(60)A.G中任意两个顶点之间均有边存在 B. G中任意两个顶点之间存在路径

C.从G中任意顶点出发可遍历图中所有顶点 D. G的邻接矩阵是对称矩阵

●【2018年上半年】对有n个结点、e条边且采用数组表示法(即邻接矩阵存储）的无向图进行深度优先遍历，时间复杂度为 (61) 。

(61)A.0() B.0() C.0(n+e) D.O(n\*e)

●【2018年下半年】图G的邻接矩阵如下图所示(顶点依次表示为v0,vl,v2,v3,v4,v5)}G是 (60) 。对G进行广度优先遍历(从v0开始)，可能的遍历序列为 (61) 。

(60)A.无向图 B.有向图 C.完全图 D.强连通图

(61)A.v0、v1、v2、v3、v4、v5 B.v0、v2、v4、v5、v1、v3 C.v0、v1、v3、v5、v2、v4 D.v0、v2、v4、v3、v5、v1

# 排序

听课建议：请各位学员作好笔记，因为课件里面没有排序过程。

## 排序的功能

将一个数据元素(或记录)的任意序列，重新排序成一个按关键字有序的序列。

## 排序的稳定性

假设= (1≤i≤n，1≤j≤n，i≠j)，且在排序前的序列中领先于(i<j)。若在排序后的序列中任然领先于，则称所用的排序方法是稳定的；反之，若可能是排序后的序列中领先于，则称所用的排序方法是不稳定的。

## 排序的类别

分为内部排序和外部排序。

* 内部排序：待排序记录存放在计算机随机存储器中进行的排序过程。
* 外部排序：指的是待排序记录的数量很大，以至内存一次不能容纳全部记录，在排序过程中尚需对外存进行访问的排序过程。即对外存中大于内存允许空间的需排序的数据，通过多次内外存间的交换实现排序。

## 插入排序

### 直接插入排序

void InsertSort(SqList &L)

{//对顺序表L做直接插入排序

for(i=2;i<=L.length;++i)

if(L[i]<L[i-1]){

L[0] = L[i];L[i] = L[i-1];

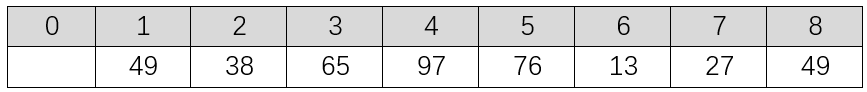
for(j=i-2;L[0]**<**L[j];--j)L[j+1]=L[j];

L[j+1]=L[0];

}

}

例子：



排序过程见视频

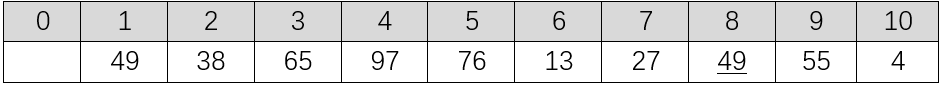
* 使用插入排序，对于具有n个记录的文件，要进行n-1趟排序，是稳定的插入排序。
* 直接插入排序：从未排序的序列中依次取出一个元素与已排序列中的元素进行比较，然后将其放在已排序序列的合适位置上。

### 希尔排序

先将整个待排序列分割成为若干个子序列分别进行直接插入排序，待整个序列中的记录“基本有序”时，再对整个记录进行一次直接插入排序，其实希尔排序就是采用分治策略对每一个分组进行直接插入排序,。

取＝，＝,如果结果为偶数，则加1，保证是一个奇数，且最后一个增量一定等于1。

例子：



排序过程见视频,希尔排序是不稳定的排序，希尔排序的执行时间依赖于增量序列，其平均时间复杂度为O()。

## 选择排序

每步从待排的记录中选出排序码最小的记录，顺序存放在已排序的记录序列的后面，直到全部排序。选择排序中主要有直接选择排序和堆排序。

### 直接选择排序

首先在所有记录中选出码最小的记录，把它与第1个记录交换，然后在其余的记录内选出排序码最小的记录，与第2个记录交换…依次类推，直到所有记录排好序。直接选择排序的平均时间复杂度O()，是不稳定的排序。算法如下：

void SelectSort(int data[],int n)

/\*将数组data中n个整数按非递减有序的方式进行排序\*/

{ int i,j,k,temp;

for(i=0;i<n-1;i++){

k=i; //data[k]表示当前找到的最小数

for(j=i+1;j<n;j++)

{if(data[j]<data[k])k=j; }

if(k!=i)

{ temp=data[i];

data[i]=data[k];

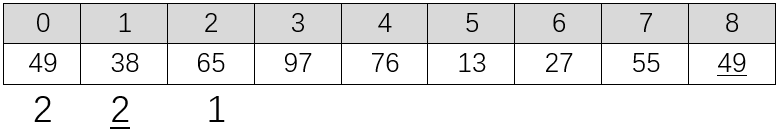
data[k]=temp;}

}

}

第一趟下来，第1个一定是最小的或者最大关键字。

例子：



排序过程见视频,直接选择排序的平均时间复杂度O()，是不稳定的排序。第一趟下来，第1个一定是最小的或者最大关键字。

### 堆排序

堆是一种树形结构，堆排序是对直接选择排序的有效改进。

实现堆排序需要解决三个问题：

#### 构建完全二叉树：先把待排序序列构建成一棵完全二叉树。

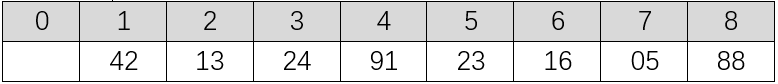
#### 把无序序列建成的完全二叉树调成一个有序堆。根据以下定义建堆：

n个元素的序列{,, …, }当且满足下述关系时，称之为堆。

或 (i=1,2,…,)

前者称为小顶堆(小根堆)，后者称为大顶堆(大根堆)。从一个无序序列建堆的过程就是一个反复“筛选”的过程。“筛选”只需从第,个元素开始(完全二叉树中最后一个非终端结点)。如何输出堆顶元素之后，调整剩余元素成为一个新的堆。

例子：



排序过程见视频

###### 大顶堆的情况

###### 小顶堆情况

堆排序的最坏时间复杂度为O()，它的平均性能较接近于最坏情况。在记录数较少的情况下，不适宜用堆排序。堆排序是就地排序，辅助空间为O()，它是不稳定的排序。

●(2004年下半年)堆是一种数据结构， (34)是堆。

(34)A.(10,50,80,30,60,20,15,18) B.(10,18,15,20,50,80,30,60) C.(10,15,18,50,80,30,60,20) D.(10,30,60,20,15,18,50,80)

●(2004年下半年) (35) 从二叉树的任一结点出发到根的路径上，所经过的结点序列必按其关键字降序排列。

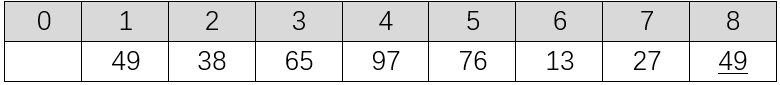
(35)A.二叉排序树 B.大顶堆C.小顶堆 D.平衡二叉树

## 交换排序

两两比较待排序记录的排序码，并交换不满足顺序要求的那些偶对，直到满足条件为止。交换排序的主要方法有冒泡排序和快速排序。

### 冒泡排序

例子：



排序过程见视频,冒泡排序最好的时间复杂度是O(n)。反之，最坏情况下为O()。冒泡排序是就地排序，且它是稳定的。第一趟得到的最后位置上的关键字一定是最大的或者最小的。

### 快速排序

采用一种分治的方法，通常又称为分治法。思想：将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。关键算法部分：

函数调用:partition(L,1,8)

int partition(sqlist &l,int low,int high)

{

L[0] = L[low];

while(low<high)

{

while(low<high&& L[hight]>= L[0])

{

--high;

}

L[low] = L[high]

while(low<hight&& L[low]<= L[0])++low;

L[high] = L[low]

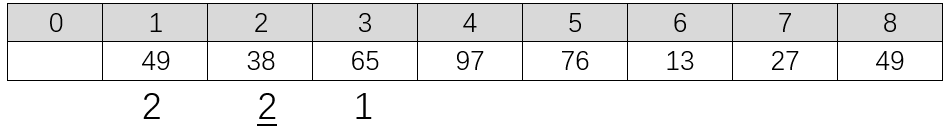
}

L[low] = L[0];

return low;

}

例子：



排序过程见视频,第一趟排序结果为：27 38 13 49 76 97 65 49

总的关键字比较次数为O(),最坏情况下时间复杂度为O()，最好情况下的时间复杂度为O().快速排序是不稳定的排序。最坏情况下需要的栈空间为O(n)，其他需要O()。

## 归并排序

归并排序是将两个或两个以上的有序子表合并成一个新的有序表。初始时，把含有n个结点的待排序序列看做是由n个长度为1的有序表所组成，然后两两归并，得到个长度为2或1的有序子序列；再两两合并,…,如此重复，直至得到一个长度为n的有序序列为止。

时间复杂度为O()，所需辅助空间为O(n)。一般情况下，对m个元素进行k路归并时，归并的趟数为：s＝

例子：49 38 65 97 76 13 27 49 排序过程见视频

归并排序采用的是一种分治策略。赫夫曼树、普里母算法使用贪心策略。

## 基数排序

这是一种和前述算法完全不同的排序方法。前述算法都要进行关键字的比较，而基数排序不需要进行记录关键字之间的比较。

### 链式基数排序

思想：从低位到高位一次对待排序的关键码进行分配和收集，经过d趟分配和收集，就可以得到一个有序序列。

#### 若关键字是十进制整数，则按个、十、百等位进行分解，基数r＝10，＝0，＝9，d为最长整数的位数。

#### 若关键字是小写的英文字，则r＝26，＝‘a’，＝‘z’，d为字符串的最大长度。

例子：288 371 260 531 287 235 56 299 18 23,r＝10,＝0,＝9,d＝3

排序过程见视频

## 各种内部排序方法的比较

* 简单排序:除希尔排序之外的所有插入排序、冒泡排序和直接选择排序，其中以直接插入排序最为简单。
* 插入排序:直接插入排序(稳定)、希尔排序(不稳定)
* 选择排序:直接选择排序(不稳定)、堆排序(不稳定)
* 交换排序:冒泡排序(稳定)、快速排序(不稳定)。
* 归并排序:稳定
* 基数排序:链式的基数排序(稳定)，不需要进行关键字之间的比较。

在平均情况下和最坏情况下的时间复杂度是一样的排序方法有：简单排序(除希尔排序之外的所有插入排序、冒泡排序和直接选择排序，其中以直接插入排序最为简单)――O()、堆排序(O(n))、归并排序O(n))、基数排序(O(d(r＋n)))。

1.从平均时间性能而言，快速排序最佳，其所需要的时间最省，但快速排序在最坏情况下的时间性能不如堆排序和归并排序。而后两者相比较的结果是，在n较大时，归并排序所需要的时间较堆排序省，但它所需要的辅助存储量最多。

2.一般来说，排序过程中的“比较”是在“相邻的两个记录关键字”间进行的排序方法是稳定的。

3.任何一个借助“比较”进行排序的算法，在最坏情况下所需要进行的比较次数至少为。然而，这只是一个理论上的下届，一般的排序算法在n>4时所需进行的比较次数均大于此值。归并排序在n<11时所用的比较次数为。当需排序的数很多时，接近于O(nlog2n))。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 类别 | 排序方法 | 平均情况 | 最好情况 | 最坏情况 | 辅助存储 | 稳定性 |
| 插入排序 | 直接插入 | O() | O(n) | O() | O(1) | 稳定 |
| 希尔排序 | O() | O(n) | O() | O(1) | 不稳定 |
| 选择排序 | 直接选择 | O() | O) | O() | O(1) | 不稳定 |
| 堆排序 | O(n) | O(n) | O(n) | O(1) | 不稳定 |
| 交换排序 | 冒泡排序 | O() | O(n) | O() | O(1) | 稳定 |
| 快速排序 | O(n) | O(n) | O() | O(1) | 不稳定 |
| 归并排序 | | O(n) | O(n) | O(n) | O(n) | 稳定 |
| 基数排序 | | O(d(n+rd)) | O(d(n+rd)) | O(d(n+rd)) | O(rd) | 稳定 |

# 查找

查找是指给定一个值K，在含有n个结点的表中找出关键字等于给定值k的结点。若找到，则查找成功，返回该结点的信息或该结点在表中的位置；否则查找失败，返回相关的指示信息。若在查找的同时对表作修改操作(例如插入或删除)，则相应的表称之为动态查找表，否则称之为静态查找表。

查找运算的主要操作是关键字的比较，所以通常把查找过程中对关键字需要执行的平均比较次数(也称为平均查找长度)作为衡量一个查找算法优劣的标准。平均查找长度为ASL定义为：ASL=。

其中n是结点的个数，是查找第i个结点的概率。若不特别声明，认为每个结点的查找概率相等，即*＝＝……＝*1/n；是找到第i个结点所需进行的比较次数。

若每个结点的查找概率相等，即*＝＝……＝*1/n，则ASL==(+++…+)。

## 8.1顺序查找

顺序查找的基本思想是：从表的一端开始，顺序扫描线性表，依次将扫描到的结点关键字和给定值k相比较。若当前扫描到的结点关键字和k相等，则查找成功；若扫描结束后，仍未找到关键字等于k的结点，则查找失败。顺序查找方法既适用于线性表的顺序存储结构，也适用于线性表的链式存储结构。

在等概率的情况下，平均查找长度为(n+1)/2,即查找成功时的平均比较次数约为表长的一半。若k值不在表中，则需要进行n+1次比较之后才能确定查找失败。

顺序查找的优点是算法简单，且对表的结构无任何要求，无论是用向量还是用链表来存放结点，也无论结点之间是否按关键字有序，它都同样适用。缺点是查找效率低，因此，当n较大时不宜采用顺序查找。

## 二分查找(有序表的查找)

二分法查找又称折半查找，它是一种效率较高的查找方法。二分法查找要求线性表是有序表，即表中结点按关键字有序。二分查找算法只适合于顺序存储的线性结构。

二分法查找的基本思想是(设R[low,…,high]是当前的查找区间)：

##### 确定该区间的中点位置:mid=;

##### 将待查的k值与R[mid].key比较，若相等，则查找成功并返回此位置，否则须确定新的查找区间，继续二分查找，具体方法如下：

###### 若R[mid].key>k,则由表的有序性可知R[mid…n].key均大于k，因此若表中存在关键字等于k的结点，则该结点必定是在位置mid左边的子表R[low,…,mid-1]中。因此，新的查找区间是左子表R[low,…,high]，其中high=mid-1。

###### 若R[mid].key<k，则要查找的k必在mid的右子表R[mid+1,…,high]中，即新的查找区间是右子表R[low,…,high]，其中low=mid+1。

###### 若R[mid].key=k，则查找成功，算法结束。

##### 下一次查找是针对新的查找区间进行，重复步骤1,2。

##### 在查找过程中，low逐步增加，而high逐步减小。如果high<low，则查找失败，算法结束。

因此，从初始的查找区间R[1…n]开始，每经过一次与当前查找区间的中点位置上的结点关键字的比较，就可确定查找是否成功，不成功则当前的查找区间就缩小一半。这一过程重复直至找到关键字为K的结点，或者直至当前的查找区间为空(即查找失败)时为止。

算法(用循环实现)如下：

int SearchBin(SSTable ST,KeyType key)

{

Low=1; high=ST.length;

while(low<=high)

{

mid = (low+high)/2;

if(ST[mid]==key)return mid;

else if(key<ST[mid])high=mid-1;

else low=mid+1;

}

return -1;//查找失败

}

用递归思想实现如下：

#include <stdio.h>

main()

{

int a[11]={0,10,20,30,40,50,60,70,80,90,100};

int SearchBin(int array[],int,int,int);

int result;

result = SearchBin(a,90,1,10);

printf("%d",result);

}

int SearchBin(int array[],int key,int low,int high)

{

int mid;

if(high<low)return -1;//查找失败

mid = (low+high)/2;

if(array[mid]==key) return mid; //查找成功

else if(key<array[mid])SearchBin(array,key,low,mid-1);

else if(key>array[mid])SearchBin(array,key,mid+1,high);

}

例：有序表 5，14，18，21，23，29，31，35，38，42，46，49，52。在表中查找值为14和22的数据元素。

###### 查找关键字为14的过程如图 8.1所示。



图 8.1查找14

###### 查找关键字为22的过程如图 8.2所示。

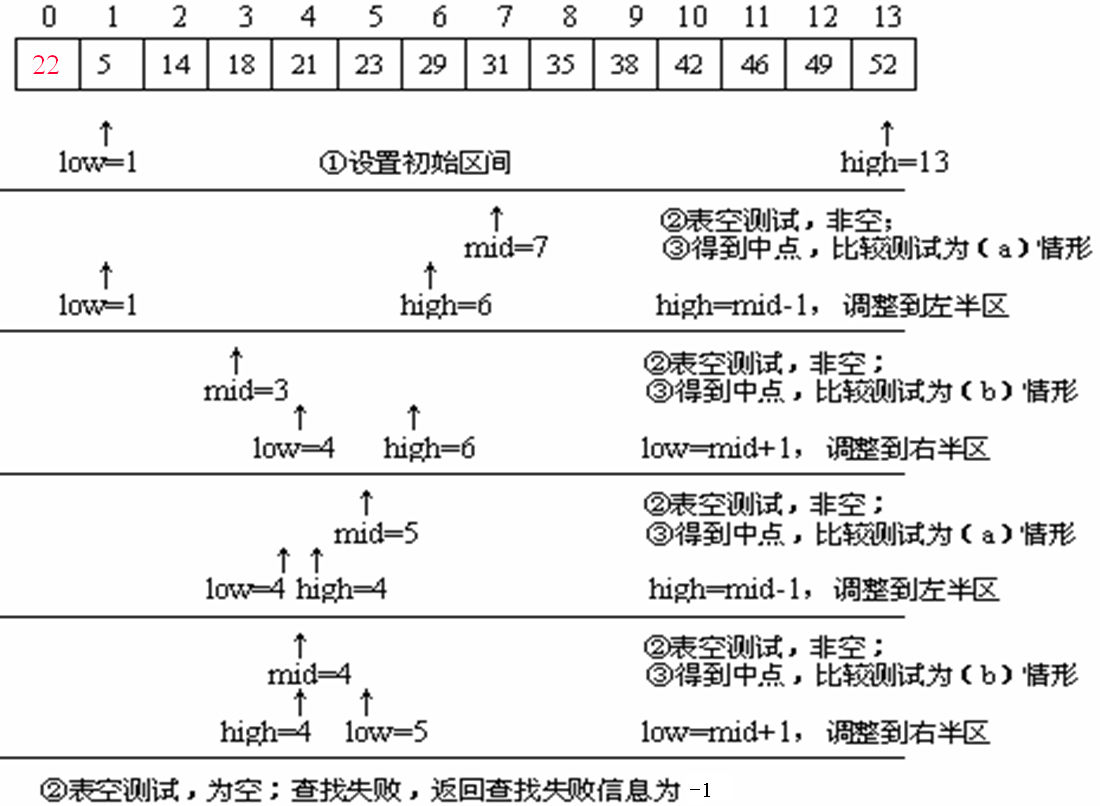


图 8.2查找22

二分法查找过程可用二叉树来描述：把当前查找区间的中间位置上的结点作为根，左子表和右子表中的结点分别作为根的左子树和右子树。由此得到二叉树，称为描述二分查找的判定树或比较树或查找树。要注意的是：判定树的形态只与表结点个数n相关，而与输入实例中R[1，…，n]的取值无关。

在等概率假设下，二分法查找成功的平均查找长度是。在最坏情况下查找成功的比较次数为，故其时间复杂度为。

●(200２年)用递归算法实现n个相异元素构成的有序序列的二分查找，采用一个递归工作栈时，该栈的最小容量应为 (15) 。

(15) A. n 　B. [n/2] 　C. [Log2n] 　D. [Log2(n+1)]

二分法查找只适用于顺序存储结构。为保持表的有序性，在顺序结构里插入和删除都必须移动大量的结点。因此，二分法查找特别适用于那种一经建立就很少改动，而又经常需要查找的线性表。

对那些查找少而又经常需要改动的线性表，可采用链表作为存储结构，进行顺序查找。链表上无法实现二分法查找。

## 分块查找

分块查找又称索引顺序查找。它是一种性能介于顺序查找和二分查找之间的方法。在此查找表中，除表本身以外，尚需建立一个“索引表”。

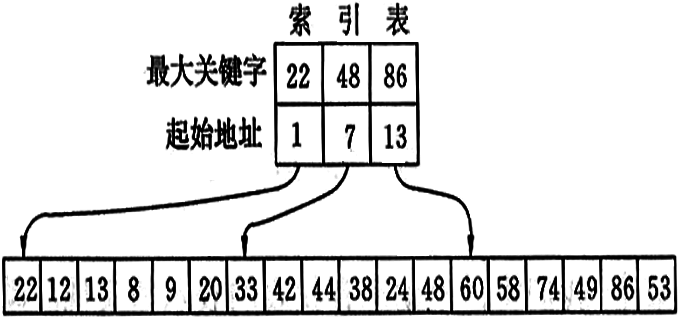


图 8.3表及其索引表

索引表中每个元素包括两项内容：关键字项(其值为该子表内的最大关键字)和指针项(指示该子表的第一个记录在在表中的位置)。索引表按关键字有序，子表或者有序或者分块有序(指的是第二个子表中的所有记录的关键字均大于第一个子表中的最大关键字，第三个子表中的关键字都大于第二个子表中的所有关键字，。。。。，依次类推)。

因此，分块查找需要两步进行，先确定待查记录所在的块(子表)，然后在块中顺序查找。

由于索引项组成的索引表按关键字有序，则确定块的查找可以用顺序查找，也可以用折半查找，而块中的记录是任意排列的，所以块中只能用顺序查找。

上述的线性表中查找关键字为38的查找过程：

由于38大于索引表里面的22，而22又为第一块子表里的最大数，所以28肯定不在第一块子表里面，所以38只能继续在索引表里面继续往后查找以确定其所在的块。由于38小于48，而48为第二块子表里的最大数，所以38如果在表里面，则肯定在第二块子表，于是进入第二块子表进行查找。

在第二子表里面，采用顺序查找。首先和33比较，不等，再继续往后查找，直到找到返回38所在表中的位置10，查找成功。

由此，分块查找的算法即为两部分查找算法的简单合成。

平均查找长度=+，其中为查找索引表确定所在块的平均查找长度，为在块中查找元素的平均查找长度。

一般情况下，为进行分块查找，可以将长度为n的表均匀分成b块，每块含有s个记录，即b＝，则有：

* 以二分查找来确定块，则分块查找成功时的平均查找长度为：。
* 以顺序查找确定块，分块查找成功时的平均查找长度为：。
* 当s＝时取极小值+1。

分块查找的优点是：在表中插入或删除一个记录时，只要找到该记录所属的块，就在该块内进行插入和删除运算；因块内记录的存放是任意的，所以插入和删除比较容易，无需移动大量记录。

分块查找的主要代价是增加一个辅助数组的存储空间和将初始表分块排序的运算。

## 散列表

散列表又叫杂凑表，还叫哈希表，是一种非常实用的查找技术，能在O(1)时间内完成查找。

* h：散列函数(哈希函数)，是一个映象。

此函数的设定很灵活，只要使得任何关键字由此所得的散列函数值都落在散列表长允许的范围内即可。

对不同的关键字可能得到同一个散列地址，即≠，但是H()＝H()，这种现象称为散列冲突。我们应该尽量避免散列冲突。

根据设定的散列函数H(key)和处理冲突的方法将一组关键字映象到一个有限的连续的地址集(区间)上，并以关键字在地址集中的“象”作为记录在表中的存储位置，这种表叫做散列表。这一映象过程称为散列，所得到的存储位置称散列地址或哈希地址。

### 散列函数的构造方法

###### 除余法：取关键字被某个不大于哈希表长m的数p除后的余数为哈希地址，如H＝key%p。

###### 其他:基数转换法,平方取中法,折叠法,移位法,随机数法。

### 冲突的解决

##### 开放地址法：开发地址法又分为：线性探查法和双散列函数法。

###### 线性探查法：

线性探查法将散列表T(0，。。。，m-1)看成是一个循环向量，若初始探查的地址为d(即h(key)＝d)，则探查序列为d，d＋1，。。。，m－1，0，1，。。。，d-1。即探查时从地址d开始，首先探查T[d],然后探查T[d+1]，。。。，T[m-1]，此后又循环到T[0]，T[1]，。。。，T[d-1]。

例子：设记录关键码为(4 11 16 54 28 34 21)，取m＝10，p＝7，h＝key％p，p值一般为不大于n(数的个数)且最接近n的质数，则用线性探查法处理冲突的过程如图 8.4所示。

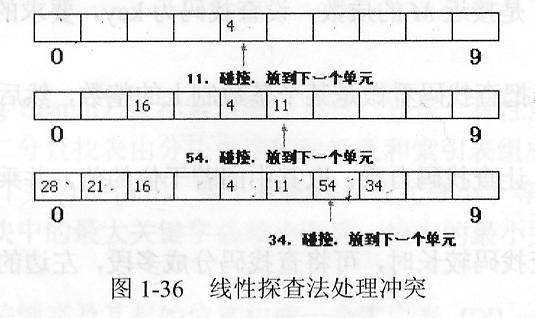


图 8.4线性探查法处理冲突

###### 双散列函数法

在发生冲突的时候再散列一次得到一个新得散列值。这是最好的开放定址法之一。

例子：

序列为：4 11 16 54 28 34，p＝7，h＝key％7

第一次：h=key%p

其它：(h(key)+i\*key%(p-1))%p，(1im-1)

##### 拉链法

用拉链法处理冲突，要求散列表的每个结点增加一个指针字段，用于连接散列值相同的子表，链表中的结点都是同义词。凡是散列地址为i的结点，均插入到以T[i]为头指针的单链表中。

例子：4 11 16 54 28 34 21，p=5，h=key％p

详细过程见视频。

# 历年真题讲解

●(2002年)堆是一种特殊的数据结构， A 是一个堆，堆排序是一种 B 排序，m个元素进行堆排序时，其时间复杂性为 C 。排序的算法很多，若按排序的稳定性和不稳定性分类，则 D 是不稳定排序。外排序是指 E 。

Ａ:(1)19,75,34,26,97,56 (2)97,26,34,75,19,56 (3)19,56,26,97,34,75 (4)19,34,26,97,56,75

B:(1)归并 (2)交换 (3)选择 (4)插入 C:(1)o(m) (2)o(m2) (3)o(log2m) (4)o(mlog2m)

Ｄ:(1)冒泡排序 (2)归并排序 (3)直接插入排序 (4)希尔(shell)排序

E:(1)用机器指令直接对硬盘中需排序数据排序 (2)把需排序数据，用其他大容量机器排序

(3)把外存中需排序数据一次性调入内存，排好序后，再输回外存

(4)对外存中大于内存允许空间的需排序的数据，通过多次内外存间的交换实现排序。

●(2002年) (4) 的特点是数据结构中元素的存储地址与其关键字之间存在某种映射关系。

(04)A.树形存储结构　B.链式存储结构　C.索引存储结构　D.散列存储结构

●(2002年)已知一个线性表(38，25，74，63，52，48)，采用的散列函数为H(Key)=Key mod 7，将元素散列到表长为7的哈希表中存储。若采用线性探测的开放定址法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为 (11) ；若利用拉链法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为 (12) 。

(11)A、1.5 B、1.7 C、2.0 D、2.3

(12)A、1.0 B、7/6 C、4/3 D、3/2

●(2002年)快速排序算法采用的设计方法是 (12) 。

(12)A.动态规划法 (Dynamic Programming) B.分治法 (Divide and Conquer)

C.回溯法 (Backtracking) D.分枝定界法 (Branch and Bound)

●(2002年)用递归算法实现 n 个相异元素构成的有序序列的二分查找，采用一个递归工作栈时，该栈的最小容量应为 (15) 。

(15) A.n B. [n/2] C. [Log2n] D. [Log2(n+1)]

●(2003年)以下序列中不符合堆定义的是(5) 。

(05)A.(102，87，100，79，82，62，84，42，22，12，68) B.(102，100，87，84，82，79，68，62，42，22，12)

C.(12，22，42，62，68，79，82，84，87，100，102) D.(102，87，42，79，82，62，68，100，84，12，22)

●(2003年)将两个长度为 n 的递增有序表归并成一个长度为 2n 的递增有序表，最少需要进行关键字比较\_\_(9)\_\_次。

(9) A. I B. n-1 C. n D. 2n

●(2003年)对n个元素进行快速排序时，最坏情况下的时间复杂度为 (64) 。

(64) A.O(1og2n) B.O(n) C.O(nlog2n) D. 0(n2)

●(2003年)任何一个基于“比较”的内部排序的算法，若对6个元素进行排序，则在最坏情况下所需的比较次数至少为 (65) 。

(65) A.10 B.11 C. 21 D. 36

解析：任何一个借助“比较”进行排序的算法，在最坏情况下所需要进行的比较次数至少为。然而，这只是一个理论上的下届，一般的排序算法在n>4时所需进行的比较次数均大于此值。归并排序在n<11时所用的比较次数为。当需排序的数很多时，接近于O(nlog2n))。

●(2004年上)设顺序存储的某线性表共有123个元素，按分块查找的要求等分为3块。若对索引表采用顺序查找方法来确定子块，且在确定的子块中也采用顺序查找方法，则在等概率的情况下，分块查找成功的平均查找长度为 (11) 。

(11)A.21　　B.23　　　C.41　　　D.62

解析：如果以二分查找来确定块，则分块查找成功时的平均查找长度为ASLBSlog2(+1)+

如果以顺序查找确定块，分块查找成功时的平均查找长度为ASLbs=+1。

●(2004年下)在第一趟排序之后，一定能把数据表中最大或最小元素放在其最终位置上的排序算法是 (40) 。

(40)A.冒泡排序 B.基数排序 C.快速排序D.归并排序

●(2004年下)堆是一种数据结构， (34)是堆。

(34)A.(10,50,80,30,60,20,15,18) B.(10,18,15,20,50,80,30,60) C.(10,15,18,50,80,30,60,20) D.(10,30,60,20,15,18,50,80)

●\_(35)\_从二叉树的任一结点出发到根的路径上，所经过的结点序列必按其关键字降序排列。

(35)A.二叉排序树 B.大顶堆　　 C.小顶堆　　　 D.平衡二叉树

●(2004年下)若对27个元素只进行三趟多路归并排序，则选取的归并路数为 (37) 。

(37)A.2　 　B.3　　　C.4　　　　D.5

解析：一般情况下，对m个元素进行k路归并时，归并的趟数为：s＝,3=logk27 =>27=K3 =>K=3。

●(2005年上)从未排序的序列中依次取出一个元素与已排序列中的元素进行比较，然后将其放在已排序序列的合适位置上，该排序方法称为(39)

(39)A.插入排序 B.选择排序 C.希尔排序 D.归并排序

●(2005年上)利用逐点插入建立序列(50,72,43,85,75,20,35,45,65,30)对应的二叉排序树以后，查找元素30要进行 42 次元素间的比较。

(42) A. 4 B.5 C. 6 D.7

●(2005年上)在最好和最坏情况下的时间复杂度均为O(nlogn)且稳定的排序方法是 （51） 。

(51)A.基数排序 B.快速排序 C.堆排序 D.归并排序

●(2005年上)已知一个线性表(38，25，74，63，52，48)，假定采用散列函数h(key)=key%7计算散列地址，并散列存储在散列表A[0..6]中，若采用线性探测方法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为 （52） 。

(52)A.1.5 B.1.7 C.2.0 D.2.3

●(2005年上)以比较为基础的排序算法在最坏情况下的计算时间下界为\_\_\_\_\_\_\_\_。

(55)A.O(n) B.O(n2) C.O(logn) D.O(nlogn)

●(2006年上) (60) 在其最好情况下的算法时间复杂度为0(n)。

(60)A.插入排序 B.归并排序 C.快速排序 D.堆排序

●(2006年上)在(56)存储结构中，数据结构中元素的存储地址与其关键字之间存在某种映象关系。

(56)A.顺序(Sequence) B.链表(Link) C.索引(Index) D.散列(Hash)

●(2006年下)对于n个元素的关键字序列{k1,k2,…,kn},当且仅当满足关系ki<=k2i且ki<=k2i+1(2i<=n,2i+1<=n)称其为小根堆，反之则为大根堆。以下序列中，(56)不符合堆的定义。

(56)A.(4,10,15,72,39,23,18) B.(58,27,36,12,8,23,9) C.(4,10,18,72,39,23,15) D.(58,36,27,12,8,23,9)

●(2006年下)对于具有n个元素的一个数据序列，若只需要得到其中第k个元素的部分排序，最好采用(59)，使用分治(Divide and Conquer)策略的是(60)算法。

(59)A.希尔排序 B.直接插入排序 C.快速排序 D.堆排序

(60)A.冒泡排序 B.插入排序 C.快速排序 D.堆排序

●(2007年上)对n个元素的数组进行 (63) ,其平均时间复杂度和最坏情况下的时间复杂度都是.

(63)A.希尔排序 B.快速排序 C.堆排序 D.选择排序

●(2008年上)已知一个线性表(16,25,35,43,51,62,87,93),采用散列函数H(Key)=Key mod 7将元素散列到表长为9的散列表中。若采用线性探查的开放定址法解决冲突(顺序地探查可用存储单元)，则构造的哈希表为 (57) ，在该散列表上进行等概率成功查找的平均查找长度为 (58) (为确定记录在查找表中的位置，需和给定关键字值进行比较的次数的期望值称为查找算法在查找成功时的平均查找长度)。

A.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 35 | 43 | 16 | 51 | 25 |  | 62 | 87 | 93 |

B.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 35 | 43 | 16 | 93 | 25 | 51 | 62 | 87 |  |

C.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 35 | 43 | 16 | 51 | 25 | 87 | 62 | 93 |  |

D.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 35 | 43 | 16 | 51 | 25 | 87 | 62 |  | 93 |

(58)A.(5\*1+2+3+6)/8 B.(5\*1+2+3+6)/9 C.(8\*1)/8 D.(8\*1)/9

●(2008年上)若总是以待排序列的第一个元素作为基准元素进行快速排序，那么最好情况下的时间复杂度为 (65) 。

(65)A.O(log2n) B.O(n) C.O(nlog2n) D.O(n2)

●(2008年下)某一维数组中依次存放了数据元素12,23,30,38,41,52,54,76,85在用折半(二分)查找方法(向上取整)查找元素54时，所经历“比较”运算的数据元素依次为 (62) 。

(62)A.41, 52, 54 B.41, 76, 54 C.41, 76, 52, 54 D.41, 30, 76, 54

●(2009年上)下面关于查找运算及查找表的叙述，错误的是(57)。

(57)A.哈希表可以动态创建 B.二叉排序树属于动态查找表

C.二分查找要求查找表采用顺序存储结构或循环链表结构 D.顺序查找方法既适用于顺序存储结构，也适用于链表结构

●(2009年下)以下关于快速排序算法的描述中，错误的是(64)。在快速排序过程中，需要设立基准元素并划分序列来进行排序。若序列由元素{12,25,30,45,52,67,85}构成，则初始排列为(65)时，排序效率最高(令序列的第一个元素为基准元素)。

(64)A.快速排序算法是不稳定的排序算法 B.快速排序算法在最坏情况下的时间复杂度为O(nlgn)

C.快速排序算法是一种分治算法 D.当输入数据基本有序时，快速排序算法具有最坏情况下的时间复杂度

(65)A.45,12,30,25,67,52,85 B.85,67,52,45,30,25,12 C.12,25,30,45,52,67,85 D.45,12,25,30,85,67,52

●(2010年上)对n个元素的有序表A[1..n]进行二分(折半)查找(除2取商时向下取整)，查找元素A[i](1≤i≤n)时，最多与A中的(57)个元素进行比较。

(57)A.n B. C.n/2 D.

●(2010年上)对以下四个序列用直接插入排序方法由小到大进行排序时，元素比较次数最少的是 (61)。

(61)A.89，27，35，78，41，15 B.27，35，41，16，89，70 C.15，27，46，40，64，85 D.90，80，45，38，30，25

●(2010年上)对于哈希表，如果将装填因子α定义为表中装入的记录数与表的长度之比，那么向表中加入新记录时， (62)。

(62)A.α的值随冲突次数的增加而递减 B.α越大发生冲突的可能性就越大

C.α等于1时不会再发生冲突 D.α低于0.5时不会发生冲突

●(2010年下)某一维数组中依次存放了数据元素15,23,38,47,55,62,88,95,102,123，采用折半(二分)法查找元素95时，依次与(60)进行了比较。

(60)A.62,88,95 B.62,95 C.55,88,95 D.55,95

●(2010年下)已知一棵度为3的树(一个结点的度是指其子树的数目，树的度是指该树中所有结点的度最大值)中有5个度为1的结点，4个度为2的结点，2个度为3的结点，那么，该树中的叶子结点数目为(61)

(61)A.10 B.9 C.8 D.7

●(2011年上)对n个元素的有序表A[1..n]进行顺序查找，其成功查找的平均查找长度(即在查找表中找到指定关键码的元素时，所进行比较的表中元素个数的期望值)为 (58) 。

(58)A.n B. (n+1)/2 C.log2n D.n2

●(2011年上)对于关键字序列(26,25,72,38,8,18,59)，采用散列函数H(Key)=Key mod 13构造散列表(哈希表)。若采用线性探测的开放定址法解决冲突(顺序地探查可用存储单元)，则关键字59所在散列表中的地址为 (61) 。

(61)A. 6 B. 7 C. 8 D. 9

●(2011年上)用插入排序和归并排序算法对数组<3,1,4,1,5,9,6,5>进行从小到大排序，则分别需要进行 (65) 次数组元素之间的比较。

(65)A. 12,14 B. 10,14 C. 12,16 D. 10,16

●(2012年上)递增序列A｛,,…,｝和B｛,,…,｝的元素互不相同，若需将它们合并为一个长度为2n的递增序列，则当最终的排列结果为 (61) 时，归并过程中元素的比较次数最多。

(61)A.,,…,,,,…,B.,,…,,,,…,

C.,,,,…,,,…,,D.,,…,/2,,,…,,+1,+2,…,,,,…,

●(2012年下)在13个元素构成的有序表M[1..13]中进行折半查找(向下取整)，若找到的元素为M[4]，则被比较的元素依次为(59) 。

(59)A.M[7]、M[3]、M[5]、M[4] B.M[7]、M[5]、M[4] C.M[7]、M[6]、M[4] D.M[7]、M[4]

●(2012年下)将数组{1，1，2，4，7，5}从小到大排序，若采用 (62) 排序算法，则元素之间需要进行的比较次数最少，共需要进行(63)次元素之间的比较。

(62) A.直接插入 B.归并 C.堆 D.快速

(63) A.5 B.6 C.7 D.8

●(2013年上)采用顺序表和单链表存储长度为n的线性序列，根据序号查找元素，其时间复杂度分别为(51)。

(51)A.0(1)、0(1) B.0(1)、0(n) C.0(n)、0(1) D.0(n)、0(n)

●(2013年上)给定n个整数构成的数组A｛,，…，｝和整数x，判断A中是否存在两个元素和，使得+=x。为了求解该问题，首先用归并排序算法对数组A进行从小到大排序；然后判断是否存在+=x,具体的方法如下列伪代码所示。则求解该问题时排序算法应用了 (62) 算法设计策略，整个算法的时间复杂度为(63) 。

…

i=1；j=n

while i<j

if ai+aj=x return true

else if ai+aj>x

j--；

else

i++；

return false；

…

(62) A.分治 B.贪心 C.动态规划 D.回溯

(63) A.0(n) B.0(nlgn) C.O(n2) D.O(nlg2n)

●(2013年上)以下关于哈希(Hash，散列)查找的叙述中，正确的是(65)。

(65)A.哈希函数应尽可能复杂些，以消除冲突 B.构造哈希函数时应尽量使关键字的所有组成部分都能起作用

C.进行哈希查找时，不再需要与查找表中的元素进行比较 D.在哈希表中只能添加元素不能删除元素

●(2013年下)某哈希表(散列表)的长度为N，设散列函数为H(Key) mod p，采用线性探测法解决冲突。以下关于P值的叙述中，正确的是(61)。

(61)A.P的值一般为不大于n且最接近n的质数 B.P的值一般为不大于n的任意数

C.P的值必须小于n的合数 D.P的值必须等于n

●(2013年下)对n个基本有序的整数进行排序，若采用插入排序算法，则时间和空间复杂度为(62)；若采用快速排序算法，则时间和空间复杂度分别为(63)。

(62)A.0()和0(n) B.0(n)和0(n) C.0()和0(1) D.0(n)和0(1)

(63)A.0()和0(n) B.0(nlgn)和0(n) C.0()和0(1) D.0(nlgn)和0(1)

●(2014年上)实现二分查找(折半查找)时，要求查找表 (61) 。

(61)A.顺序存储，关键码无序排列 B.顺序存储，关键码有序排列 C.双向链表存储，关键码无序排列 D.双向链表存储，关键码有序排列

●(2014年下)快速排序算法在排序过程中，在待排序数组中确定一个元素为基准元素，根据基准元素把待排序数组划分成两个部分，前面一部分元素值小于等于基准元素，而后面一部分元素值大于基准元素。然后再分别对前后两个部分进一步进行划分。根据上述描述，快速排序算法采用了 (61) 算法设计策略。已知确定基准元素操作的时间复杂度为Θ(n)，则快速排序算法的最好和最坏情况下的时间复杂度为 (62) 。

(61)A.分治 B.动态规划 C.贪心 D.回溯

(62)A.Θ(n)和Θ(nlgn) B.Θ(n)和Θ() C.Θ(nlgn)和Θ(nlgn) D.Θ(nlgn)和Θ()

●(2014年下)对一待排序序列分别进行直接插入排序和简单选择排序，若待排序序列中有两个元素的值相同，则 (63) 保证这两个元素在排序前后的相对位置不变。

(63)A.直接插入排序和简单选择排序都可以 B.直接插入排序和简单选择排序都不能

C.只有直接插入排序可以 D.只有简单选择排序可以

●(2015年上)对某有序顺序表进行折半查找时，(60)不可能构成查找过程中关键字的比较序列。

(60)A.45，10，30，18，25 B.45，30，18，25，10

C.10，45，18，30，25 D.10，18，25，30，45

●(2015年上)用某排序方法对一元素序列进行非递减排序时，若该方法可保证在排序前后排序码相同者的相对位置不变，则称该排序方法是稳定的。简单选择排序方法是不稳定的， (61) 可以说明这个性质。

(61)A.21 48 63 17 B.17 21 48 63

C.63 21 48 17 D. 17 48 63 21

●(2015年上)在n个数的数组中确定其第i(1≤i≤n)小的数时，可以采用快速排序算法中的划分思想，对n个元素划分，先确定第k小的数，根据i和k的大小关系，进一步处理，最终得到第i小的数。划分过程中，最佳的基准元素选择的方法是选择待划分数组的 (64) 元素，此时，算法在最坏情况下的时间复杂度为(不考虑所有元素均相等的情况) (65) 。

(64)A.第一个 B.最后一个 C.中位数 D.随机一个

(65)A.Θ(n) B.Θ(lgn) C.Θ(nlgn) D.Θ()

●(2015年下)在55个互异元素构成的有序表A[1..55]中进行折半查找(或二分查找，向下取整)。若需查找的元素等于A[19]，则在查找过程中参与比较的元素依次为(60)、A[19]。

(60)A.A[28]、A[30]、A[15]、A[20] B.A[28]、A[14]、A[21]、A[17] C.A[28]、A[15]、A[22]、A[18] D.A[28]、A[18]、A[22]、A[20]

●(2015年下)在某应用中，需要先排序一组大规模的记录，其关键字为整数。若这组记录的关键字基本上有序，则适宜采用(64)排序算法。若这组记录的关键字的取值均在0到9之间(含)，则适宜采用(65)排序算法。

(64)A.插入 B.归并 C.快速 D.计数

(65)A.插入 B.归并 C.快速 D.计数

●(2016年上)在13个元素构成的有序表A[1…13]中进行折半查找(或称为二分查找，向下取整)。那么以下叙述中，错误的是 (60) 。

(60)A.无论要查找哪个元素，都是先与A[7]进行比较 B.若要查找的元素等于A[9]，则分别需与A[7]、A[11]、A[9]进行比较

C.无论要查找的元素是否在A[]中，最多与表中的4个元素比较即可

D.若待查找的元素不在A[]中，最少需要与表中的3个元素进行比较

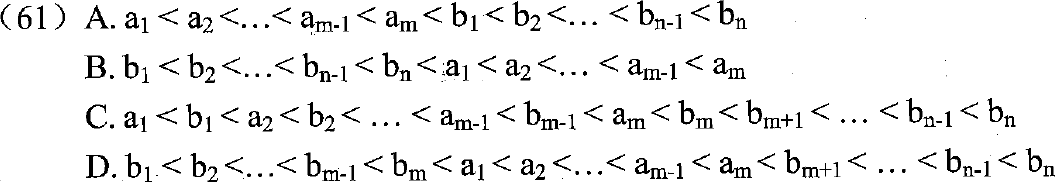
●(2017年上)对于n个元素的关键字序列{……}，当且仅当满足关系≤且≤{i=1,2,……,[n/2]}时称其为小根堆(小顶堆)。以下序列中， (60) 不是小根堆。

(60)A.16,25,40,55,30,50,45 B.16,40,25,50,45,30,55 C.16,25,39,41,45,43,50 D.16,40,25,53,39,55,45

●(2017年上)在12个互异元素构成的有序数组a[1..12]中进行二分查找(即折半查找，向下取整)，若待查找的元素正好等于a[9]，则在此过程中，依次与数组中的 (61) 比较后，查找成功结束。

(61)A.a[6]、a[7]、a[8]、a[9] B.a[6]、a[9] C.a[6]、a[7]、a[9] D.a[6]、a[8]、a[9]

●(2017年下)两个递增序列A和B的长度分别为m和n（m<n且m与n接近），将二者归并为一个长度为m+n的递增序列。当元素关系为 （61） 时，归并过程中元素的比较次数最少。（答案：A）



●(2017年下)现需要对一个基本有序的数组进行排序。此时最适宜采用的算法为 (64) 排序算法，时间复杂度为 (65) 。

(64)A.插入 B.快速 C.归并 D.堆

(65)A.0(n) B.O(nlgn) C.O() D.O(1gn)

●(2018年上)用哈希表存储元素时，需要进行冲突(碰撞）处理，冲突是指 (60) 。

(60)A.关键字被依次映射到地址编号连续的存储位置 B.关键字不同的元素被映射到相同的存储位置

C.关键字相同的元素被映射到不同的存储位置 D.关键字被映射到哈希表之外的位置

●(2019年上)设散列函数为 H(key)=key%11,对于关键码序列(23,40,91,17,19,10,31,65,26), 用线性探查法解决冲突构造的哈希表为 (60) 。

(60)A.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 哈希地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 关键码 | 10 | 23 |  | 91 | 26 |  | 17 | 40 | 19 | 31 | 65 |

B.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 哈希地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 关键码 | 65 | 23 |  | 91 | 26 |  | 17 | 40 | 19 | 31 | 10 |

C.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 哈希地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 关键码 |  | 23 | 10 | 91 | 26 |  | 17 | 40 | 19 | 31 | 65 |

D.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 哈希地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 关键码 |  | 23 | 65 | 91 | 26 |  | 17 | 40 | 19 | 31 | 10 |

●(2019年上)对于有序表(8,15,19,23,26,31,40,65,91),用二分法进行查找时，可能的关键字比较顺序为 (61) 。

(61)A.26,23,19 B.26,8,19 C.26,40,65 D.26,31,40