作者：希赛教育软考学院　来源：希赛网　2015年07月13日

线性表

第1章　数据结构基础

数据结构是指数据对象及其相互关系和构造方法，一个数据结构S可以用一个二元组表示为：S=（D,R）。其中，D是数据结构中的数据的非空有限集合，R是定义在D上的关系的非空有限集合。在数据结构中，结点及结点间的相互关系称为数据的逻辑结构，数据在计算机中的存储形式称为数据的存储结构。

数据结构按逻辑结构的不同分为线性结构和非线性结构两大类，其中非线性结构又可分为树形结构和图结构，而树形结构又可分为树结构和二叉树结构。

按照考试大纲的要求，在数据结构与算法方面，要求考生掌握以下知识点。

1.常用数据结构

数组（静态数组、动态数组）、线性表、链表（单向链表、双向链表、循环链表）、队列、栈、树（二叉树、查找树、平衡树、线索树、堆）、图等的定义、存储和操作。

Hash（存储地址计算，冲突处理）。

2.常用算法

排序算法、查找算法、数值计算方法、字符串处理方法、数据压缩算法、递归算法、图的相关算法。

算法与数据结构的关系、算法效率、算法设计、算法描述（流程图、伪代码、决策表）、算法的复杂性。

本章主要讨论有关数据结构的问题。

1.1　线性表

线性表是最简单、最常用的一种数据结构，它是由相同类型的结点组成的有限序列。一个由n个结点a0,a1,…，an−1组成的线性表可记为（a0,a1,…，an−1）。线性表的结点个数为线性表的长度，长度为0的线性表称为空表。对于非空线性表，a0是线性表的第一个结点，an−1是线性表的最后一个结点。线性表的结点构成一个序列，对序列中两相邻结点ai和ai+1,称ai是ai+1的前驱结点，ai+1是ai的后继结点。其中a0没有前驱结点，an−1没有后继结点。

线性表中结点之间的关系可由结点在线性表中的位置确定，通常用（ai,ai+1）（0≤i≤n−2）表示两个结点之间的先后关系。例如，如果两个线性表有相同的数据结点，但它们的结点在线性表中出现的顺序不同，则它们是两个不同的线性表。

线性表的结点可由若干成分组成，其中能唯一标识该结点的成分称为关键字，或简称键。为了讨论方便，往往只考虑结点的关键字，而忽略其他成分。

1.线性表的基本运算

线性表包含的结点个数可以动态增加或减少，可以在任何位置插入或删除结点。线性表常用的运算可分成几类，每类有若干种运算。

1）查找运算

在线性表中查找具有给定键值的结点。

2）插入运算

在线性表的第i（0≤i≤n–1）个结点的前面或后面插入一个新结点。

3）删除运算

删除线性表的第i（0≤i≤n –1）个结点。

4）其他运算

统计线性表中结点的个数；

输出线性表各结点的值；

复制线性表；

线性表分拆；

线性表合并；

线性表排序；

按某种规则整理线性表。

2.线性表的存储

线性表常用的存储方式有顺序存储和链接存储。

1）顺序存储

顺序存储是最简单的存储方式，通常用一个数组，从数组的第一个元素开始，将线性表的结点依次存储在数组中，即线性表的第i个结点存储在数组的第i（0≤i≤n–1）个元素中，用数组元素的顺序存储来体现线性表中结点的先后次序关系。

顺序存储线性表的最大优点就是能随机存取线性表中的任何一个结点，缺点主要有两个，一是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结点个数；二是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。

2）链接存储

链接存储是用链表存储线性表（链表），最简单的是用单向链表，即从链表的第一个结点开始，将线性表的结点依次存储在链表的各结点中。链表的每个结点不但要存储线性表结点的信息，还要用一个域存储其后继结点的指针。单向链表通过链接指针来体现线性表中结点的先后次序关系。

链表存储线性表的优点是线性表中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操作带来了方便，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。链表存储方式的缺点主要有两个，一是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；二是不便随机访问线性表的任一结点。

3.线性表上的查找

线性表上的查找运算是指在线性表中找某个键值的结点。根据线性表中的存储形式和线性表本身的性质差异，有多种查找算法，例如顺序查找、二分法查找、分块查找、散列查找等。其中二分法查找要求线性表是一个有序序列。

4.在线性表中插入新结点

1）顺序存储

设线性表结点的类型为整型，插入之前有n个结点，把值为x的新结点插在线性表的第i（0≤i≤n）个位置上。完成插入主要有以下步骤：

检查插入要求的有关参数的合理性；

把原来的第n–1个结点至第i个结点依次往后移一个数组元素位置；

把新结点放在第i个位置上；

修正线性表的结点个数。

在具有n个结点的线性表上插入新结点，其时间主要花费在移动结点的循环上。若插入任一位置的概率相等，则在顺序存储线性表中插入一个新结点，平均移动次数为n/2.

2）链接存储

在链接存储线性表中插入一个键值为x的新结点，分为以下4种情况：

在某指针p所指结点之后插入；

插在首结点之前，使待插入结点成为新的首结点；

接在线性表的末尾；

在有序链表中插入，使新的线性表仍然有序。

5.删除线性表的结点

1）顺序存储

在有n个结点的线性表中，删除第i（0≤i≤n–1）个结点。删除时应将第i+1个结点至第n–1个结点依次向前移一个数组元素位置，共移动n–i–1个结点。完成删除主要有以下几个步骤：

检查删除要求的有关参数的合理性；

把原来第i+1个表元至第n–1个结点依次向前移一个数组元素位置；

修正线性表表元个数。

在具有n个结点的线性表上删除结点，其时间主要花费在移动表元的循环上。若删除任一表元的概率相等，则在顺序存储线性表中删除一个结点，平均移动次数为（n-1）/2.

2）链接存储

对于链表上删除指定值结点的删除运算，需考虑几种情况，一是链表为空链表，不执行删除操作；二是要删除的结点恰为链表的首结点，应将链表头指针改为指向原首结点的后继指点；其他情况，先要在链表中寻找要删除的结点，从链表首结点开始顺序寻找。若找到，执行删除操作，若直至链表末尾没有指定值的结点，则不执行删除操作。完成删除由以下几个步骤组成：

如链表为空链表，则不执行删除操作；

若链表的首结点的值为指定值，更改链表的头指针为指向首结点的后继结点；

在链表中寻找指定值的结点；

将找到的结点删除。

本书简介　　下一节

栈

1.1.1 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。

1.顺序存储

可以用顺序存储线性表来表示栈，为了指明当前执行插入和删除运算的栈顶位置，需要一个地址变量 top指出栈顶结点在数组中的下标。

2.链接存储栈

栈也可以用链表实现，用链表实现的栈称为链接栈。链表的第一个结点为顶结点，链表的首结点就是栈顶指针top,top为NULL的链表是空栈。

　作者：希赛教育软考学院　来源：希赛网　2014年01月24日

队列

1.1.2 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

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）。

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[N]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=0.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，就认为队列已满的简单办法，以区别队空和队满。即队空的判别条件是head=tail,队满的判别条件是head=tail+1。

2.链接存储

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head指向链表的首结点，队列的尾指针tail指向链表的尾结点。当队列的头指针head值为NULL时，队列为空。

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年01月24日

稀疏矩阵

1.1.3 稀疏矩阵

在计算机中存储一个矩阵时，可使用二维数组。例如，M×N阶矩阵可用一个数组a[M][N]来存储（可按照行优先或列优先的顺序）。如果一个矩阵的元素绝大部分为零，则称为稀疏矩阵。若直接用一个二维数组表示稀疏矩阵，则会因存储太多的零元素而浪费大量的内存空间。因此，通常采用三元组数组或十字链表两种方法来存储稀疏矩阵。

1.三元组数组

稀疏矩阵的每个非零元素用一个三元组来表示，即非零元素的行号、列号和它的值。然后按某种顺序将全部非零元素的三元组存于一个数组中。

如果只对稀疏矩阵的某些单个元素进行处理，则宜用三元组表示。

2.十字链表

在十字链表中，矩阵的非零元素是一个结点，同一行的结点和同一列的结点分别顺序循环链接，每个结点既在它所在行的循环链表中，又在它所在列的循环链表中。每个结点含5个域，分别为结点对应的矩阵元素的行号、列号、值，以及该结点所在行链表后继结点指针、所在列链表后继结点指针。

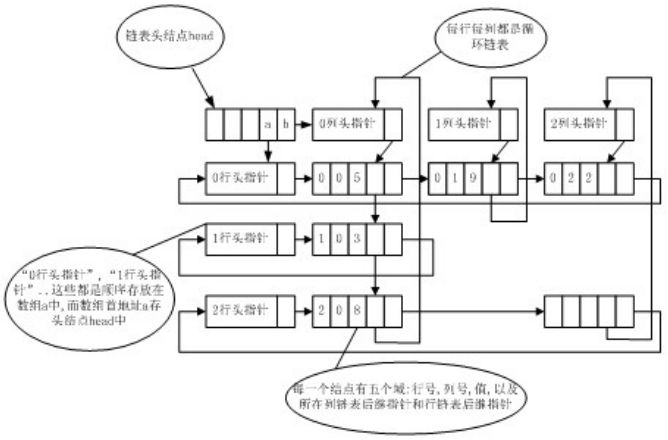
为了处理方便，通常对每个行链表和列链表分别设置一个表头结点，并使它们构成带表头结点的循环链表。为了引用某行某列的方便，全部行链表的表头结点和全部列链表的表头结点分别组成数组，这两个数组的首结点指针存于一个十字链表的头结点中，最后由一个指针指向该头结点。

例如有矩阵A如图1-1所示。

则其十字链表如图1-2所示。

$$\begin{bmatrix} 5 & 9 & 2 \\ 3 & 0 & 0 \\ 8 & 0 & 0 \end{bmatrix}$$

图1-1 矩阵A示意图图1-2



矩阵A十字链表存储示意图

如果对稀疏矩阵某行或某列整体做某种处理，可能会使原来为零的元素变为非零，而原来非零的元素变成零。对于这种场合，稀疏矩阵宜用十字链表来表示。

字符串

1.1.4 字符串

字符串是由某字符集上的字符所组成的任何有限字符序列。当一个字符串不包含任何字符时，称它为空字符串。一个字符串所包含的有效字符个数称为这个字符串的长度。一个字符串中任一连续的子序列称为该字符串的子串。

字符串通常存于足够大的字符数组中，每个字符串的最后一个有效字符之后有一个字符串结束标志，记为"\0".通常由系统提供的库函数形成的字符串的末尾会自动添加"\0",但当由用户的应用程序来形成字符串时，必须由程序自行负责在最后一个有效字符之后添加"\0",以形成字符串。

对字符串的操作通常有：

统计字符串中有效字符的个数；

把一个字符串的内容复制到另一个字符串中；

把一个字符串的内容连接到另一个足够大的字符串的末尾；

在一个字符串中查找另一个字符串或字符；

按字典顺序比较两个字符串的大小。

树

1.2 树和二叉树

1.2.1 树

1.树的基本概念

树是由一个或多个结点组成的有限集合T,它满足以下两个条件：

（1）有一个特定的结点，称为根结点；

（2）其余的结点分成m（m≥0）个互不相交的有限集合。其中每个集合又都是一棵树，称T1,T2,...，

Tm–1为根结点的子树。

显然，以上定义是递归的，即一棵树由子树构成，子树又由更小的子树构成。由条件（1）可知，一棵树至少有一个结点（根结点）。一个结点的子树数目称为该结点的度（次数），树中各结点的度的最大值称为树的度（树的次数）。度为0的结点称为叶子结点（树叶），除叶子结点外的所有结点称为分支结点，根以外的分支结点称为内部结点。例如，在图1-3所示的树中，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的度数为2,其他结点的度数为0,该树的度数4。

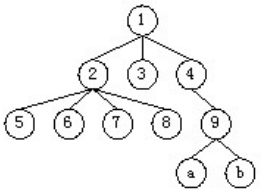


图1-3 树的例子

在用图形表示的树中，对两个用线段连接的相关联的结点而言，称位于上端的结点是位于下端的结点的父结点或双亲结点，称位于下端的结点是位于上端的结点的（孩）子结点，称同一父结点的多个子结点为兄弟结点，称处于同一层次上、不同父结点的子结点为堂兄弟结点。例如在图1-3中，结点1是结点2,3,4的父结点。反之，结点2,3,4都是结点1的子结点。结点2,3,4是兄弟结点，而结点5,6,7,8,9是堂兄弟结点。

定义一棵树的根结点所在的层次为1,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。

2.树的常用存储结构

因为树是非线性的结构，为了存储树，必须要把树中结点之间的关系反映在存储结构中。最常用树的存储结构有标准存储和带逆存储形式。

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语言描述树结点的标准存储结构的数据类型如下：

```
#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*/
```

2）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一个指向其父结点的指针，用C语言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*/
struct rtnode *child[N]; /*树结点的子结点指针*/
struct rtnode *parent; /*父结点指针*/
}RTNODE; /*树结点的数据类型*/
```

3.树的遍历

按照某种顺序逐个获得树中全部结点的信息，称为树的遍历。常用的树的遍历方法主要有以下3种。

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0层上的根结点，然后从左到右依次访问处于1层上的结点，然后从左到右依次访问处于2层上的结点等，即自上而下、从左到右逐层访问树中各层上的结点。

按上述遍历的定义，图1-3所示的树的各种遍历结果如下。

前序遍历：1,2,5,6,7,8,3,4,9,a,b.

后序遍历：5,6,7,8,2,3,a,b,9,4,1.

层次遍历：1,2,3,4,5,6,7,8,9,a,b.

二叉树

1.2.2 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，或者由一个根结点及其两棵互不相交的左、右二叉子树所组成。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。因为二叉树可以为空，所以二叉树中的结点可能没有子结点，也可能只有一个左子结点（右子结点），也可能同时有左右两个子结点。如图1-4所示是二叉树的4种可能形态（如果把空树计算在内，则共有5种形态）。

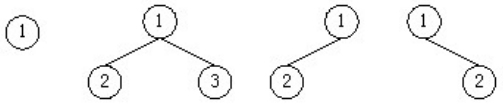


图1-4 二叉树的4种不同形态

与树相比，二叉树可以为空，空的二叉树没有结点（树至少有一个结点）。在二叉树中，结点的子树是有序的，分左、右两棵子二叉树。

二叉树常采用类似树的标准存储结构来存储，其结点类型可以用C语言定义如下：

```
typedef struct Btnode{
char data; /*数据*/
strucrt Btnode *lchild; /*左孩子*/
struct Btnode *rchild; /*右孩子*/
}BTNODE;
```

2.二叉树的性质

二叉树具有下列重要性质（此处省略了推导过程，有兴趣的读者可自行推导）。

性质1：在二叉树的第i层上至多有2^{i-1}个结点（i≥1）。

性质2：深度为k的二叉树至多有2^k-1个结点（k≥1）。

性质3：对任何一棵二叉树，如果其叶子结点数为n_0，度为2的结点数为n_2，则$n_0=n_2+1$。

一棵深度为k且有2^k-1（k≥1）个结点的二叉树称为满二叉树。如图1-5所示就是一棵满二叉

树，对结点进行了顺序编号。

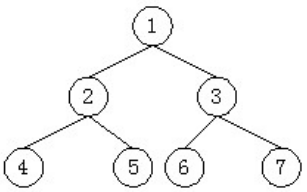


图1-5 满二叉树的例子

如果深度为k、有n个结点的二叉树中各结点能够与深度为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。如图1-6（a）所示是一棵完全二叉树，而（b）、（c）是两棵非完全二叉树。显然，满二叉树是完全二叉树的特例。

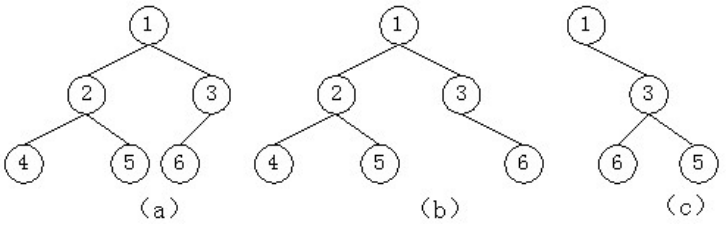


图1-6 完全二叉树和非完全二叉树

根据完全二叉树的定义，显然，在一棵完全二叉树中，所有的叶子结点都出现在第k层或k–1层（最后两层）。

性质4：具有n（n>0）个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor +1$。（注：$\lfloor\ \rfloor$符号为向下取整运算符，$\lceil\ \rceil$为向上取整运算符，$\lfloor m \rfloor$表示不大于m的最大整数，反之，$\lceil m \rceil$表示不小于m的最小整数）

性质5：如果对一棵有n个结点的完全二叉树的结点按层序编号（从第1层到第$\lfloor \log_2 n \rfloor +1$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；如果i>1,则其双亲是结点$\lfloor i/2 \rfloor$。

如果2i>n，则结点i为叶子结点，无左孩子；否则，其左孩子是结点2i。

如果2i+1>n，则结点i无右孩子；否则，其右孩子是结点2i+1。

3.二叉树的遍历

树的所有遍历方法也同样适用于二叉树，此外，由于二叉树自身的特点，还有中序遍历方法。

前序遍历（先根遍历，先序遍历）：首先访问根结点，然后按前序遍历根结点的左子树，再按前序遍历根结点的右子树。

中序遍历（中根遍历）：首先按中序遍历根结点的左子树，然后访问根结点，再按中序遍历根结点的右子树。

后序遍历（后根遍历，后序遍历）：首先按后序遍历根结点的左子树，然后按后序遍历根结点的右子树，再访问根结点。

例如如图1-7所示的二叉树，其前序遍历、中序遍历和后序遍历结果分别如下。

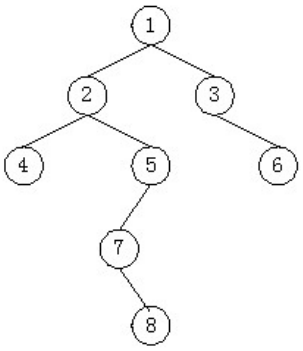


图1-7 二叉树遍历的例子

前序遍历：1,2,4,5,7,8,3,6。

中序遍历：4,2,7,8,5,1,3,6。

后序遍历：4,8,7,5,2,6,3,1。

以上3种遍历方法都是递归定义的，可通过递归函数分别加以实现。

性质6：一棵二叉树的前序序列和中序序列可以唯一地确定这棵二叉树。

根据性质6，给定一棵二叉树的前序序列和中序序列，我们可以写出该二叉树的后序序列。例如，某二叉树的前序序列为 ABHFDECKG，中序序列为HBDFAEKCG，则构造二叉树的过程如图1-8所示。

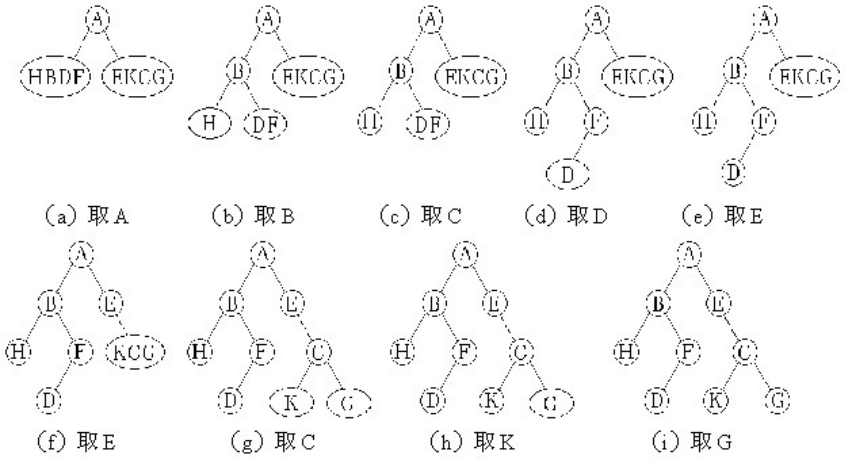


图1-8 已知前序序列和中序序列，求二叉树的过程

上一节 本书简介 下一节

作者：希赛教育软考学院 来源：希赛网 2014年08月25日

二叉排序树

1.2.3 二叉排序树

二叉排序树又称为二叉查找树，其定义为二叉排序树或者是一棵空二叉树，或者是具有如下性质（BST性质）的二叉树：

（1）若它的左子树非空，则左子树上所有结点的值均小于根结点；

（2）若它的右左子树非空，则右子树上所有结点的值均大于根结点；

（3）左、右子树本身又各是一棵二叉排序树。

例如，如图1-9所示就是一棵二叉排序树。

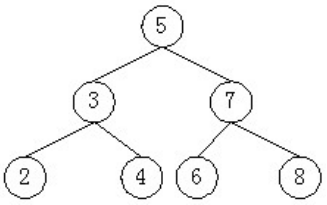


图1-9 二叉排序树的例子

根据二叉排序树的定义可知，如果中序遍历二叉排序树，就能得到一个排好序的结点序列。二叉排序树上有查找、插入和删除等3种操作。下面，我们假设二叉排序树的结点只存储结点的键值，其类型与前面的二叉树的结点类型相同。

1.静态查找

静态查找是在二叉排序树上查找键值为key的结点是否存在，这可按以下步骤在二叉排序树ST上找值为key的结点：

如果二叉排序树ST为空二叉树，则查找失败，结束查找；

如果二叉排序树的根结点的键值等于key，则查找成功，结束查找；

如果key小于根结点的键值，则沿着根结点的左子树查找，即将根结点的左子树作为新的二叉排序树ST继续查找；

如果key大于根结点的键值，则沿着根结点的右子树查找，即将根结点的右子树作为新的二叉排序树ST继续查找。

2.动态查找

在二叉排序树上，为插入和删除操作需要而使用的查找称为动态查找，动态查找应得到两个指针，一个指向键值为key的结点，另一个指向该结点的父结点。为此，查找函数可设4个参数，查找树的根结点指针root，待查找值key，存储键值为key结点的父结点的指针pre，存储键值为key结点的指针p，但函数要考虑以下几种不同情况。

（1）二叉排序树为空，查找失败，函数使*p=NULL，*pre=NULL；

（2）二叉排序树中没有键值为key的结点，函数一直寻找至查找路径的最后一个结点，*pre指向该结点，*p=NULL，如果插入键值为key的结点，就插在该结点下；

（3）查找成功，*p指向键值为key的结点，*p指向它的父结点。

3.插入结点

将利用动态查找函数确定新结点的插入位置，然后分以下几种情况进行相应的处理。

（1）如果相同键值的结点已在二叉排序树中，则不再插入；

（2）如果二叉排序树为空树，则以新结点为二叉排序树；

（3）将要插入结点的键值与插入后的父结点的键值比较，就能确定新结点是父结点的左子结点，还是右子结点，并进行相应插入。

4.删除结点

删除二叉排序树上键值为key的结点的操作如下。

（1）调用查找函数确定被删结点的位置；

（2）如被删结点不在二叉排序树上，则函数返回。否则，按以下情况分别处理。

1）如果被删除的结点是根结点，又可分两种情况：

（i）被删除结点无左子树，则以被删除结点的右子树作为删除后的二叉排序树；

（ii）被删除结点有左子树，则以被删除结点的左子结点作为根结点，并把被删除结点的右子树作为被删除结点的左子树按中序遍历的最后一个结点的右子树。

2）如果被删除结点不是根结点，且被删除结点无左子结点：

（i）被删除结点是它的父结点的左子结点，则把被删除结点的右子树作为被删除结点的父结点的左子树；

（ii）被删除结点是它的父结点的右子结点，则把被删除结点的右子树作为被删除结点的父结点的右子树；

3）如果被删除结点不是根结点，且被删除结点有左子结点，则被删除结点的右子树作为被删除结点的左子树按中序遍历的最后一个结点的右子树，同时进行以下操作：

（i）被删除结点是它的父结点的左子结点，则把被删除结点的左子树作为被删结点的父结点的左子树；

（ii）被删除结点是它的父结点的右子结点，则把被删除结点的左子树作为被删除结点的父结点的右子树。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年03月13日

平衡二叉树

1.2.4 平衡二叉树

为了保证二叉排序树的高度为$\log_2 n$,从而保证二叉排序树上实现的插入、删除和查找等基本操作的平均时间为O（$\log_2 n$），在往树中插入或删除结点时，要调整树的形态来保持树的"平衡".使之既保持BST性质不变，又保证树的高度在任何情况下均为$\log_2 n$,从而确保树上的基本操作在最坏情况下的时间均为O（$\log_2 n$）。

平衡二叉树（Balanced Binary Tree或Height-Balanced Tree）又称为AVL树，是指树中任一结点的左右子树的高度大致相同。如果任一结点的左右子树的高度均相同（如满二叉树），则二叉树是完全平衡的。通常，只要二叉树的高度为O（1$\log_2 n$），就可看做是平衡的。

平衡的二叉排序树指满足BST性质的平衡二叉树。AVL树中任一结点的左、右子树的高度之差的绝对值不超过1.若将二叉树上结点的平衡因子定义为该结点的左子树的深度减去它的右子树的深度，则平衡二叉树上所有结点的平衡因子只可能是–1、0和1。

在最坏情况下，n个结点的AVL树的高度约为1.44$\log_2 n$.而完全平衡的二叉树度高约为$\log_2 n$,AVL树是接近最优的。

上一节　本书简介　下一节

线索树

1.2.5 线索树

二叉树在一般情况下无法直接找到某结点在某种遍历序列中的前驱和后继结点。若增加指针域来存放结点的前驱和后继结点信息，将大大降低存储空间的利用率。考查n个结点的二叉树，其中有n+1个空指针域，它们可以被用来存放"线索",增加了线索的二叉树称为线索树（穿线树）。

设有一棵采用标准形式存储的二叉树BT,对于BT中的每个结点k,如它没有左（或右）子结点，而k_1是k的按中序遍历的前面（或后面）结点，则置结点k的左（或右）指针为k_1结点的指针。为了与k结点的真正子结点指针区别，另需在结点上增加两个标志域ltag和rtag.如此改造后的线索树的结点类型定义如下：

```
typedef stuct BTnode{ / *穿线树结点类型 * /
char data;
struct node *lchild,*rchild;
int 1tag,rtag;
}BTNODE;
```

当ltag=0时，表示lchild指针指向其左孩子结点；当ltag=1时，表示lchild指针指向其前驱结点。当rtag=0时，表示rchild指针指向其右孩子结点；当rtag=1时，表示rchild指针指向其后继结点。

上一节 本书简介 下一节

最优二叉树

1.2.6 最优二叉树

树的路径长度是从树根到树中每一结点的路径长度之和。在结点数目相同的二叉树中，完全二叉树的路径长度最短。在一些应用中，赋予树中结点的一个有某种意义的实数，这些数字称为结点的权。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。树中所有叶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），通常记为：

$$\mathrm{WPL}=\sum_{i=1}^{n} w_i l_i$$

其中n表示叶子结点的数目，w_i和l_i分别表示叶结点k_i的权值和根到结点k_i之间的路径长度。

在权值为$w_1,w_2,\ldots,w_n$的n个叶子所构成的所有二叉树中，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哈夫曼树。

假设有n个权值，则构造出的哈夫曼树有n个叶子结点。 n个权值分别设为$w_1,w_2,\ldots,w_n$,则哈夫曼树的构造规则为：

①将$w_1,w_2,\ldots,w_n$看成是有n棵树的森林（每棵树仅有一个结点）；

②在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和；

③从森林中删除选取两棵树，并将新树加入森林；

④重复第②和③步，直到森林中只剩一棵树为止，该树即为所求的哈夫曼树。

例如如果叶子结点的权值分别为1,2,3,4,5,6,则构造哈夫曼树的过程如图1-10所示。

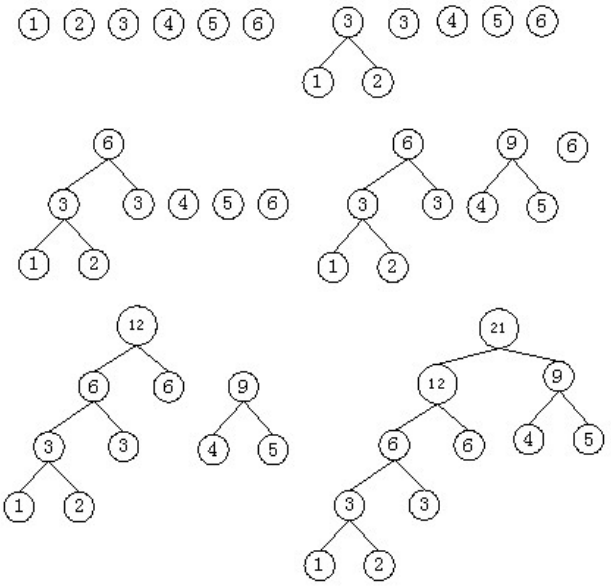


图1-10 哈夫曼树的构造过程

在构造哈夫曼树的过程中，每次都是选取两棵最小权值的二叉树进行合并，因此使用的是贪心算法。

给定结点序列<c_i,p_i>（c_i为编码字符，p_i为c_i的频度），哈夫曼编码的过程如下：

用字符c_i作为叶子，p_i作为c_i的权，构造一棵哈夫曼树，并将树中左分支和右分支分别标记为0和1;

将从根到叶子的路径上的标号依次相连，作为该叶子所表示字符的编码。该编码即为最优前缀码。

给定字符集的哈夫曼树生成后，求哈夫曼编码的具体实现过程是依次以叶子结点C[i]（0≤i≤n–1）为出发点，向上回溯至根为止。上溯时走左分支则生成代码0,走右分支则生成代码1.需要注意以下几个问题。

由于生成的编码与要求的编码反序，将生成的代码先从后往前依次存放在一个临时串中，并设一个指针start指示编码在该串中的起始位置（start初始时指示串的结束位置）。

当某字符编码完成时，从临时串的start处将编码复制到该字符相应的位串bits中即可。

因为字符集大小为n,故变长编码的长度不会超过n,加上一个结束符"\0",bits的大小应为n+1.

给定一个序列的集合，若不存在一个序列是另一个序列的前缀，则该序列集合称为前缀码。相反，给定一个序列的集合，若不存在一个序列是另一个序列的后缀，则该序列集合称为后缀码。平均码长或文件总长最小的前缀编码称为最优的前缀码，最优的前缀码对文件的压缩效果亦最佳。

$$\text{平均码长} = \sum_{i=1}^{n} p_i \, l_i$$

其中p_i为第i个字符的概率，l_i为码长。

利用哈夫曼树很容易求出给定字符集及其概率（或频度）分布的最优前缀码。哈夫曼编码是一种应用广泛且非常有效的数据压缩技术，该技术一般可将数据文件压缩掉20%至90%,其压缩效率取

决于被压缩文件的特征。

图

1.3 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一个前驱和后继。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都是任意的。

图的基础知识

1.3.1 图的基础知识

1.图的基本概念

图G由两个集合V和E组成，记为G=（V,E），其中V是顶点的有穷非空集合，E是V中顶点偶对（称为边）的有穷集合。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。

图分为有向图和无向图两种。如图1-10（a）所示是一个有向图，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。$<V_i,V_j>$表示一条有向边，Vi是边的始点（起点），Vj是边的终点，$<V_i,V_j>$和$<V_j,V_i>$是两条不同的有向边。例如，在图1-10（a）中，$<V_1,V_4>$和$<V_4,V_1>$是两条不同的边。有向边也称为弧，边的始点称为弧尾，终点称为弧头。

如图1-10（b）所示是一个无向图，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是相邻的顶点，边（i,j）是与顶点i和j相关联的边。

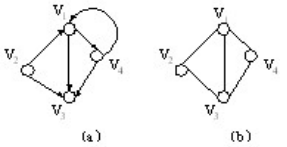


图 1-10　图的分类

如果限定任何一条边或弧的两个顶点都不相同，则有n个顶点的无向图最多有n（n–1）/2条边，这样的无向图称为无向完全图。一个有向图最多有n（n–1）条弧，这样的有向图称为有向完全

图。

如果同为无向图或同为有向图的两个图$G_1=(V_1,E_1)$和$G_2=(V_2,E_2)$，满足$V_2 \subseteq V_1$且$E_2 \subseteq E_1$,则称图G_2是图G_1的子图。

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。

在图G=（V,E）中，如果存在顶点序列（$V_0,V_1,\ldots,V_k$），其中$V_0=P,V_k=Q$,且（V_0,V_1），（V_1,V_2），…，（V_k-1,V_k）都在E中，则称顶点P到顶点Q有一条路径，并用（$V_0,V_1,\ldots,V_k$）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。

在有向图G中，若对于V（G）中任意两个不同的顶点Vi和Vj,都存在从V_i到V_j及从V_j到V_i的路径，则称G是强连通图。

有向图的极大强连通子图称为G的强连通分量。强连通图只有一个强连通分量，即其自身。非强连通的有向图有多个强连通分量。

2.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

邻接矩阵反映顶点间的邻接关系，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，若（i,j）或<i,j>∈E,则M[i][j]=1;否则，M[i][j]=0.例如，图1-10（a）和图1-10（b）的邻接矩阵分别如下。

$$M_a=\begin{pmatrix}0&0&1&1\\1&0&1&0\\0&0&0&0\\1&0&1&0\end{pmatrix}\qquad M_b=\begin{pmatrix}0&1&1&1\\1&0&1&0\\1&1&0&1\\1&0&1&0\end{pmatrix}$$

由邻接矩阵的定义可知，无向图的邻接矩阵是对称的，有向图的邻接矩阵不一定对称。对于无向图，其邻接矩阵第i行元素的和即为顶点i的度。对于有向图，其邻接矩阵第i行元素之和为顶点i的出度，而邻接矩阵第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。

2）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。例如，图1-10（a）和图1-10（b）的邻接表分别如图1-11（a）和图1-11（b）所示。

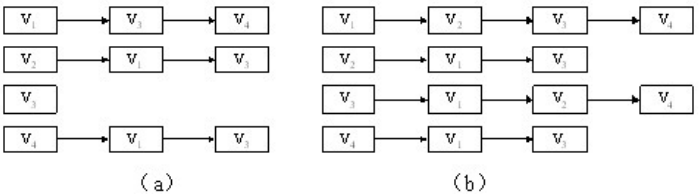


图1-11 图的邻接表表示

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。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。

3.图的遍历

和树的遍历类似，图的遍历也是从某个顶点出发，沿着某条搜索路径对图中每个顶点各做一次且仅做一次访问。它是许多图的算法的基础。深度优先遍历和广度优先遍历是最为重要的两种遍历图的方法。它们对无向图和有向图均适用。

1）深度优先遍历

在G中任选择一顶点V为初始出发点（源点），则深度优先遍历可以定义如下：首先，访问出发点V,并将其标记为已访问过；然后，依次从V出发搜索V的每个邻接点W.若W未曾访问过，则以W为新的出发点继续进行深度优先遍历，直至图中所有和源点V有路径相通的顶点（亦称为从源点可达的顶点）均已被访问为止。若此时图中仍有未访问的顶点，则另选一个尚未访问的顶点作为新的源点重复上述过程，直至图中所有的顶点均已被访问为止。

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。

2）广度优先遍历

广度优先遍历的过程是：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W_0,W_1,...，W_{k-1};接着再依次访问与顶点W_0,W_1,...，W_{k-1}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。

从广度优先搜索遍历过程知，若顶点V在顶点W之前被访问，则对V相邻顶点的访问就先于只与W相邻的那些顶点的访问。因此，需要一个队列来存放被访问过的顶点，以便按顶点的访问顺序依次访问这些顶点相邻接的其他还未被访问过的顶点。

上一节　本书简介　下一节

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

最小生成树

1.3.2 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，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

设已知G=（V,E）是一个带权连通无向图，顶点V={0,1,2,…，n−1}.设U是构造生成树过程中已被考虑在生成树上的顶点的集合。初始时，U只包含一个出发顶点。设T是构造生成树过程中已被考虑在生成树上的边的集合，初始时T为空。如果边（i,j）具有最小代价，且i∈U,j∈（V−U），那么最小代价生成树应包含边（i,j）。把j加到U中，把（i,j）加到T中。重复上述过程，直到U等于V为止。这时，T即为要求的最小代价生成树的边的集合。

普里姆算法的特点是当前形成的集合T始终是一棵树。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（n2），与图中边数无关，所以适合于稠密图。

2.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,¢），按边长递增的顺序选择E中的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接成一棵更大的树，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。

克鲁斯卡尔算法的特点是当前形成的集合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（$elog_2e$），与图中顶点数无关，所以较适合于稀疏图。

上一节　本书简介　下一节

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

最短路径

1.3.3　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，而是指路径上各边权值的总和。路径长度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。

目前，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序产生各顶点最短路径的算法。若按长度递增的次序生成从源点s到其他顶点的最短路径，则当前正在生成的最短路径上除终点以外，其余顶点的最短路径均已生成（将源点的最短路径看成是已生成的源点到其自身的长度为0的路径）。

迪杰斯特拉算法的基本思想是：设S为最短距离已确定的顶点集（看成红点集），V-S是最短距离尚未确定的顶点集（看成蓝点集）。

（1）初始化：初始化时，只有源点s的最短距离是已知的（SD（s）=0），故红点集S={s},蓝点集为空。

（2）重复以下工作，按路径长度递增的次序产生各顶点的最短路径：在当前蓝点集中选择一个最短距离最小的蓝点来扩充红点集，以保证算法按路径长度递增的次序产生各顶点的最短路径。当蓝点集中仅剩下最短距离为∞的蓝点，或者所有的蓝点已扩充到红点集时，s到所有顶点的最短路径就求出来了。

需要注意的是：

（1）若从源点到蓝点的路径不存在，则可假设该蓝点的最短路径是一条长度为无穷大的虚拟路径。

（2）从源点s到终点v的最短路径简称为v的最短路径；从s到v的最短路径长度简称为v的最短距离，并记为SD（v）。

根据按长度递增的次序产生最短路径的思想，当前最短距离最小的蓝点k的最短路径是：

源点，红点1,红点2,……，红点n,蓝点k

距离为：

源点到红点n最短距离 + <红点n,蓝点k>的边长。

为求解方便，可设置一个向量D[0…n−1],对于每个蓝点v∈V−S,用D[v]记录从源点s到达v且除v外中间不经过任何蓝点（若有中间点，则必为红点）的"最短"路径长度（简称估计距离）。若k是蓝点集中估计距离最小的顶点，则k的估计距离就是最短距离，即若D[k]=min{D[i] i∈V−S},则D[k]=SD（k）。

初始时，每个蓝点v的D[c]值应为权w<s,v>,且从s到v的路径上没有中间点，因为该路径仅含一条边<s,v>.

将k扩充到红点后，剩余蓝点集的估计距离可能由于增加了新红点k而减小，此时必须调整相应蓝点的估计距离。对于任意的蓝点j,若k由蓝变红后使D[j]变小，则必定是由于存在一条从s到j且包含新红点k的更短路径P=<s,…，k,j>.且D[j]减小的新路径P只可能是由于路径<s,…，k>和边<k,j>组成。所以，当length（P）=D[k]+w<k,j>小于D[j]时，应该用P的长度来修改D[j]的值。

2.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。

但更常用的是弗洛伊德（Folyd）提出的求每一对顶点之间的最短路径的算法。设G=（V,E）是有n个顶点的有向图，顶点集合V={0,1,…，n−1}.图用邻接矩阵M表示。Floyd算法的基本思想是递推地产生一个矩阵序列C_0,C_1,C_2,…，Cn,其中C_0是已知的带权邻接矩阵a,C_k（i,j）（0≤i,j<n）表示从顶点i到顶点j的中间顶点序号不大于k的最短路径长度。如果i到j的路径没有中间顶点，则对于0≤k<n,有C_k（i,j）= C_0（i,j）= a[i][j].递推地产生C_1,C_2,…，C_n的过程就是逐步将可能是最短路径上的顶点作为路径上的中间顶点进行试探，直到为全部路径都找遍了所有可能成为最短路径上的中间顶点，所有的最短路径也就全部求出，算法就此结束。

设在第k次递推之前已求得C_{k-1}（i,j）（0≤i,j<n），为求C_k（i,j）需考虑以下两种情况。

（1）如果从顶点i到顶点j的最短路径不经过顶点k,则由C_k（i,j）定义知，从i到j中间的顶点序号不大于k的最短路径长度还是C_{k-1}（i,j）即C_k（i,j）=C_{k-1}（i,j）。

（2）如果从顶点i到顶点j的最短路径要经过顶点k,则这样的一条路径是由 i到k和由k到j的两条路径所组成的。由于C_{k-1}（i,k）和C_{k-1}（k,j）分别表示i到k和从k到j的中间顶点序号不大于k-1的最短

路径长度，若$C_{k-1}(i,k)+C_{k-1}(k,j)<C_{k-1}(i,j)$，$C_{k-1}(i,k)+C_{k-1}(k,j)$必定是i到j的中间顶点序号不大于是k的最短路径的长度，则$C_k(i,j)=C_{k-1}(i,k)+C_{k-1}(k,j)$。

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年01月24日

拓扑排序

1.3.4 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，某些项目必须在其他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1-13所示。

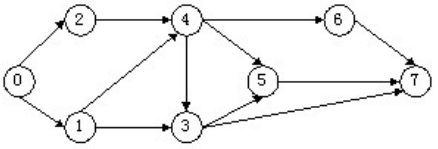


图 1-13　AOV 网络的例子

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1-35中的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，如02143567,02143657, 01243567等。

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

关键路径

1.3.5 关键路径

在AOV网络中，如果边上的权表示完成该活动所需的时间，则称这样的AOV为AOE网络。例如

图1-14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。

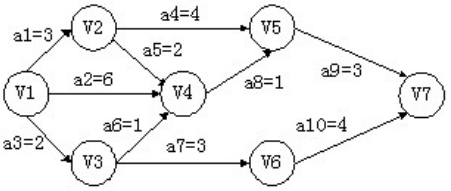


图 1-14　AOE 网络的例子

因AOE网络中的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。

为了找出给定的AOE网络的关键活动，从而找出关键路径，先定义几个重要的量。

$V_e(j)$、$V_l(j)$:顶点j事件最早、最迟发生时间。

e(i)、l(i)： 活动i最早、最迟开始时间。

从源点V_1到某顶点V_j的最长路径长度，称为事件V_j的最早发生时间，记为$V_e(j)$。$V_e(j)$也是以Vj为起点的出边<V_j,V_k>所表示的活动a_i的最早开始时间e(i)。

在不推迟整个工程完成的前提下，一个事件Vj允许的最迟发生时间，记为$V_l(j)$。显然，l(i)=Vl(j)–(a_i所需时间），其中j为a_i活动的终点。满足条件l(i)=e(i)的活动为关键活动。

求顶点V_j的$V_e(j)$和$V_l(j)$可按以下两步来做。

①由源点开始向汇点递推

$$\begin{cases} V_e(1)=0 \\ V_e(j)=\mathrm{MAX}\{V_e(i)+\mathrm{d}(i,j)\}, <V_i,\ V_j>\in E_1, 2\leqslant j\leqslant n \end{cases}$$

其中，E1是网络中以Vj为终点的入边集合。

②由汇点开始向源点递推

$$\begin{cases} V_l(n)=V_e(n) \\ V_l(j)=\mathrm{MIN}\{V_l(k)-d(j,k)\}, <V_j,\ V_k>\in E_2, 2\leqslant j\leqslant n-1 \end{cases}$$

其中，E2是网络中以Vj为起点的出边集合。

要求一个AOE的关键路径，一般需要根据以上变量列出一张表格，逐个检查。例如求图1-14所示的AOE关键路径的过程如表1-1所示。

表1-1　求关键路径的过程

顶点	$V_e(j)$	$V_l(j)$	边	e(i)	l(i)	l(i)-e(i)
V_1	0	0	$a_1(3)$	0	0	0
V_2	3	3	$a_2(6)$	0	0	0
V_3	2	3	$a_3(2)$	0	1	1
V_4	6	6	$a_4(4)$	3	3	0
V_5	7	7	$a_5(2)$	3	4	1
V_6	5	6	$a_6(1)$	2	5	3
V_7	10	10	$a_7(3)$	2	3	1
			$a_8(1)$	6	6	0
			$a_9(3)$	7	7	0
			$a_{10}(4)$	5	6	1

因此，图1-14的关键活动为a_1,a_2,a_4,a_8和a_9（即表1-1阴影所示活动），其对应的关键路径有两条，分别为（V_1,V_2, V_5,V_7）和（V_1,V_4,V_5,V_7），长度都是10。

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

排序

1.4 排　序

所谓排序，就是要整理文件中的记录，使之按关键字递增（或递减）次序排列起来。当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。

在待排序的文件中，若存在多个关键字相同的记录，经过排序后这些具有相同关键字的记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称这种排序方法是不稳定的。

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

插入排序

1.4.1 插入排序

插入排序的基本思想是，每步将一个待排序的记录按其排序码值的大小，插到前面已经排好的文件中的适当位置，直到全部插入完为止。插入排序方法主要有直接插入排序和希尔排序。

1.直接插入排序

直接插入排序的过程是，在插入第i个记录时，$R_1,R_2,\ldots,R_{i-1}$已经排好序，将第i个记录的排序码ki依次和$R_1,R_2,\ldots,R_{i-1}$的排序码逐个进行比较，找到适当的位置。

使用直接插入排序，对于具有n个记录的文件，要进行n−1趟排序。各种状态下的时间复杂度如表1-1所示。

表1-2 直接插入排序有关数据

初始文件状态	正　序	反　序	无序(平均)
第 i 趟的关键字比较次数	1	$i+1$	$(i-2)/2$
总关键字比较次数	$n-1$	$(n+2)(n-1)/2$	$n^2/4$
第 i 趟记录移动次数	0	$i+2$	$(i-2)/2$
总的记录移动次数	0	$(n-1)(n+4)/2$	$n^2/4$
时间复杂度	$O(n)$	$O(n^2)$	$O(n^2)$

说明：初始文件按关键字递增排序，简称"正序",初始文件按关键字递减排序，简称"反序".

2.希尔排序

希尔（Shell）排序的基本思想是，先取一个小于n的整数d1作为第一个增量，把文件的全部记录分成d1个组。所有距离为dl的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量$d_2<d_1$重复上述的分组和排序，直至所取的增量$d_t=1$（$d_t<d_{t-l}<\ldots<d_2<d_1$），即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

一般取$d_1=n/2$,$d_{i+1}=d_i/2$.如果结果为偶数，则加1,保证d_i为奇数。

例如，要对关键码{72,28,51,17,96,62,87,33,45,24}进行排序，这里n=10,首先取$d_1=n/2=5$.则排序过程如图1-12所示。

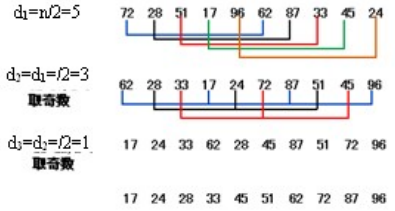


图1-15 希尔排序的过程

希尔排序是不稳定的，希尔排序的执行时间依赖于增量序列，有人在大量实验的基础上指出，当n在某个特定范围内时，希尔排序的平均时间复杂度为O（$n^{1.3}$）。

上一节 本书简介 下一节

第 1 章：数据结构基础 作者：希赛教育软考学院 来源：希赛网 2014年03月13日

选择排序

1.4.2 选择排序

选择排序的基本思想是每步从待排序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，直到全部排完。选择排序中主要使用直接选择排序和堆排序。

1.直接选择排序

直接选择排序的过程是，首先在所有记录中选出排序码最小的记录，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，直到所有记录排完为止。

无论文件初始状态如何，在第i趟排序中选出最小关键字的记录，需做n–i次比较，因此，总的比较次数为n（n–1）/2=O（n^2）。当初始文件为正序时，移动次数为0;文件初态为反序时，每趟排序均要执行交换操作，总的移动次数取最大值3（n–1）。直接选择排序的平均时间复杂度为O（n^2）。直接选择排序是不稳定的。

2.堆排序

堆排序是一种树形选择排序，是对直接选择排序的有效改进。n个关键字序列$K_l,K_2,\ldots,K_n$称为堆，当且仅当该序列满足（$K_i\le K_{2i}$且$K_i\le K_{2i+1}$）或（$K_i\ge K_{2i}$且$K_i\ge K_{2i+1}$），（$1\le i\le \lfloor n/2 \rfloor$）。根结点（堆顶）的关键字是堆里所有结点关键字中最小者，称为小根堆；根结点的关键字是堆里所有结点关键字中最大者，称为大根堆。

若将此序列所存储的向量R[1..n]看做是一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树，即树中任一非叶结点的关键字均不大于（或不小于）其左右孩子（若存在）结点的关键字。

堆排序的关键步骤有两个，一是如何建立初始堆；二是当堆的根结点与堆的最后一个结点交换

后，如何对少了一个结点后的结点序列做调整，使之重新成为堆。

下面，我们通过一个例子来具体说明建立初始堆和调整堆的过程。假设关键字序列为{42,13,24,91,23,16,05,88},则第一次建立的二叉树如图1-16（a）所示。

①从i = [n/2]开始比较父结点和子结点的关系，如果不满足堆的定义，就进行调整。我们假设需要建立大根堆，本题n=8,所以从第4个元素（91）开始调整。

因为91大于其子结点88,所以不需要调整；

再看第3个元素（24），同样，因为24大于其子结点16和05,也不需要调整；

再看第2个元素（13），13小于其子结点23和91,需要把13和91交换（把父结点与关键值最大的那个子结点交换）。这时，13比其子结点88要小，又需要交换。结果如图1-16（b）所示。

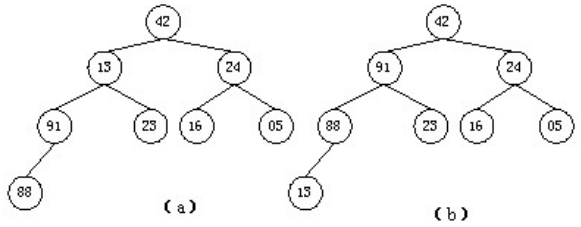


图1-16 建立堆的过程

再看第1个元素（42），因为42小于其左子结点91,需要交换。

这时，42还小于其左子结点88,又需要交换42和88的值。建堆过程结束，所建立的初始堆如图1-17（a）所示。

②在初始堆的基础上，把第一个元素（91）和最后一个元素（13）交换，输出91.这时，如图1-17（b）所示。

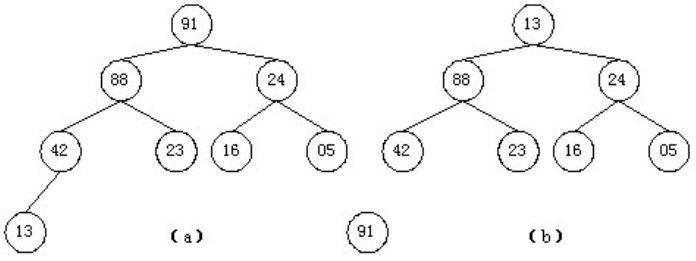


图1-17 初始堆及调整1

③在图1-17（b）的基础上，因13小于其左右子结点88和24,则和88交换，交换后，13还小于其左右子结点42和23,则和42再交换，如图1-18（a）所示。

④图1-18（a）又是一个n–1个元素的堆，把第一个元素（88）和最后一个元素（05）交换，输出88.这时，如图1-18（b）所示。

⑤在图1-18（b）的基础上，因05小于其左右子结点42和24,则和42交换，交换后，05还小于其左右子结点13和23,则和23再交换，如图1-19（a）所示。

⑥图1-19（a）又是一个n–2个元素的堆，把第一个元素（42）和最后一个元素（16）交换，输出42.这时，如图1-19（b）所示。

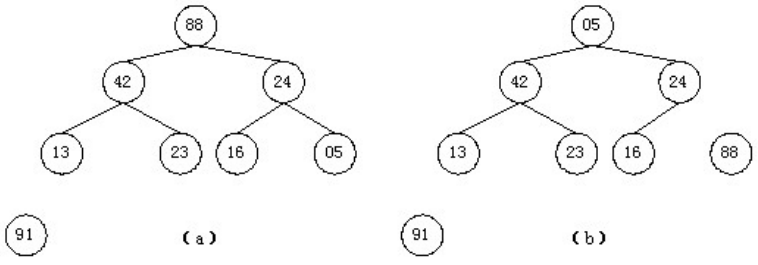


图1-18 调整堆的过程之1

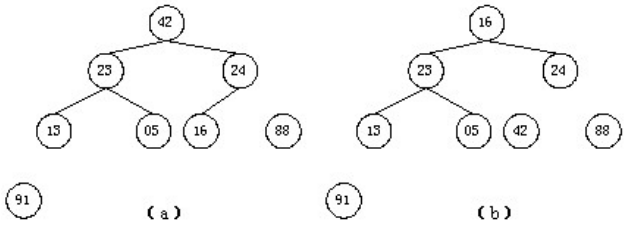


图1-19 调整堆的过程之2

⑦在图1-19（b）的基础上，因16小于其左右子结点23和24,则和24交换。如图1-20（a）所示。

⑧图1-20（a）又是一个n–3个元素的堆，把第一个元素（24）和最后一个元素（05）交换，输出24.这时，如图1-20（b）所示。

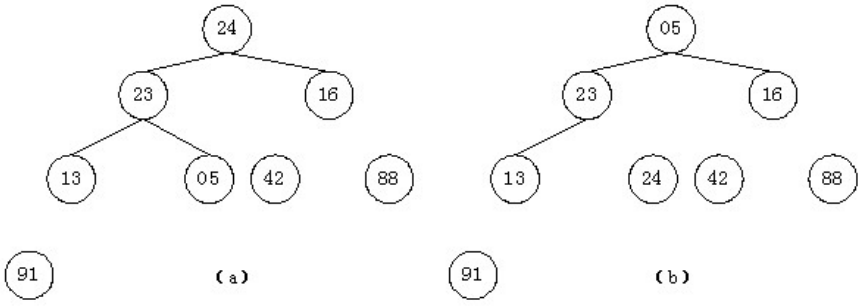


图1-20 调整堆的过程之3

⑨在图1-20（b）的基础上，因05小于其左右子结点23和16,则和23交换。交换后，05还是小于其子结点13,和13再交换。如图1-21（a）所示。

⑩图1-21（a）又是一个n–4个元素的堆，把第一个元素（23）和最后一个元素（05）交换，输出23.这时，如图1-21（b）所示。

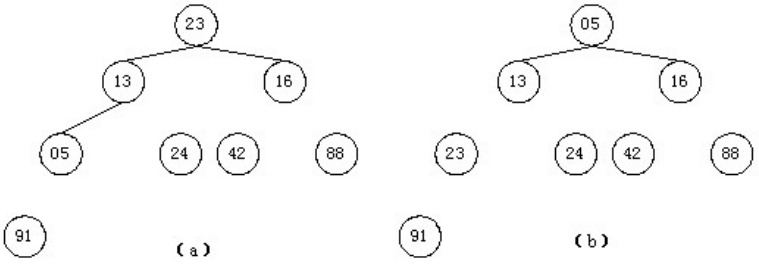


图1-21 调整堆的过程之4

在图1-21（b）的基础上，因05小于其左右子结点13和16,则和16交换。如图1-22（a）所示。

图1-22（a）又是一个n–5个元素的堆，把第一个元素（16）和最后一个元素（05）交换，输出16.这时，如图1-22（b）所示。

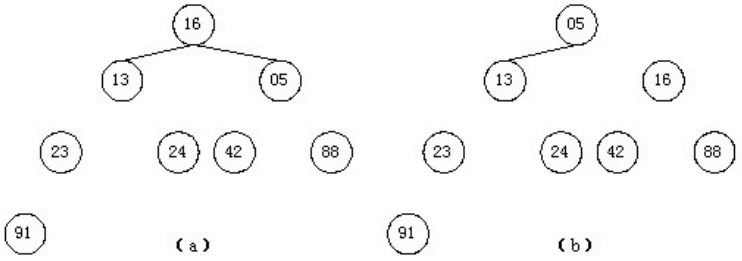


图1-22 调整堆的过程之5

在图1-22（b）的基础上，因05小于其左子结点13,则和13交换，如图1-23（a）所示。

图1-23（a）又是一个n–6个元素的堆，把第一个元素（13）和最后一个元素（05）交换，输出13.这时，如图1-23（b）所示，堆排序过程结束。

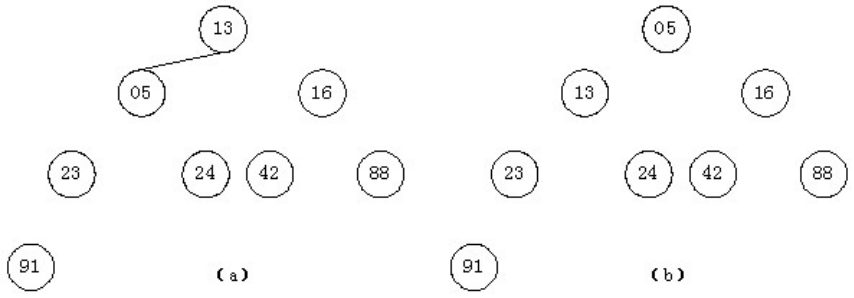


图1-23 调整堆的过程之6

堆排序的最坏时间复杂度为O（$nlog_2n$），堆排序的平均性能较接近于最坏性能。由于建初始堆所需的比较次数较多，所以堆排序不适宜于记录数较少的文件。堆排序是就地排序，辅助空间为O（1），它是不稳定的排序方法。

上一节　本书简介　下一节

作者：希赛教育软考学院　来源：希赛网　2014年03月13日

交换排序

1.4.3 交换排序

交换排序的基本思想是，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。

1.冒泡排序

冒泡排序将被排序的记录数组R[1…n]垂直排列，每个记录R[i]看成是重量为k_i的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组R,凡扫描到违反本原则的轻气泡，就使其向上"飘浮".如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

冒泡排序的具体过程如下。

第一步，先比较k_1和k_2,若$k_1>k_2$,则交换k1和k2所在的记录，否则不交换。继续对k_2和k_3重复上述过程，直到处理完kn-1和kn.这时最大的排序码记录转到了最后位置，称第一次起泡。共执行n−1次比较。

与第一步类似，从k1和k2开始比较，到kn-2和kn-1为止，共执行n−2次比较。称第二次起泡。

依此类推，共进行n−1次起泡，完成整个排序过程。

若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数为n−1次，记录移动次数为0.因此，冒泡排序最好的时间复杂度为O（n）。

若初始文件是反序的，需要进行n−1趟排序。每趟排序要进行n−i次关键字的比较（1≤i≤n-1），且每次比较都必须移动记录3次来达到交换记录位置的目的。在这种情况下，比较次数达到最大值n（n−1）/2=O（n^2），移动次数也达到最大值3n（n−1）/2=O（n^2）。因此，冒泡排序的最坏时间复杂度为O（n^2）。

虽然冒泡排序不一定要进行n−1趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。冒泡排序是就地排序，且它是稳定的。

2.快速排序

快速排序采用了一种分治的策略，通常称其为分治法。其基本思想是，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。

快速排序的具体过程如下。

第一步，在待排序的n个记录中任取一个记录，以该记录的排序码为准，将所有记录分成两组，第一组各记录的排序码都小于等于该排序码，第二组各记录的排序码都大于该排序码，并把该记录排在这两组中间。

第二步，采用同样的方法，对左边的组和右边的组进行排序，直到所有记录都排到相应的位置为止。

例如，要对关键码{7,2,5,1,9,6,8,3}进行排序，选择第一个元素为基准。第一趟排序的过程如图1-13所示。

要注意的是，在快速排序中，选定了第一个元素为基准，接着就拿最后一个元素和第一个元素比较，如果大于第一个元素，则保持不变，再拿倒数第二个元素和基准比较；如果小于基准，则进行交换。交换之后，再从前面的元素开始与基准比较，如果小于基准，则保持不变；如果大于基准，则交换。交换之后，再从后面开始比较，依此类推，前后交叉进行。

然后，再采取同样的办法对{3,2,5,1,6}和{8,9}分别进行排序，具体过程如图1-14所示。

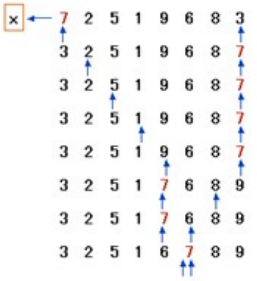


图1-24 第一趟排序过程

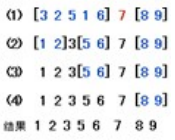


图1-25 各趟排序过程

快速排序的时间主要耗费在划分操作上，对长度为k的区间进行划分，共需k–1次关键字的比较。

最坏情况是每次划分选取的基准都是当前无序区中关键字最小（或最大）的记录，划分的结果是基准左边的子区间为空（或右边的子区间为空），而划分所得的另一个非空的子区间中记录数目，仅仅比划分前的无序区中记录个数减少一个。因此，快速排序必须进行n-1次划分，第i次划分开始时区间长度为n–i+1,所需的比较次数为n–i（1≤i≤n-1），故总的比较次数达到最大值n（n-1）/2=O（n2）。如果按上面给出的划分算法，每次取当前无序区的第1个记录为基准，那么当文件的记录已按递增次序（或递减次序）排列时，每次划分所取的基准就是当前无序区中关键字最小（或最大）的记录，则快速排序所需的比较次数反而最多。

在最好情况下，每次划分所取的基准都是当前无序区的"中值"记录，划分的结果是基准的左、右两个无序子区间的长度大致相等。总的关键字比较次数为O（$n\log_2 n$）。

因为快速排序的记录移动次数不大于比较的次数，所以快速排序的最坏时间复杂度应为O（n^2），最好时间复杂度为O（$n\log_2 n$）。

尽管快速排序的最坏时间为O（n^2），但就平均性能而言，它是基于关键字比较的内部排序算法中速度最快者，快速排序亦因此而得名。它的平均时间复杂度为O（$n\log_2 n$）。快速排序在系统

内部需要一个栈来实现递归。若每次划分较为均匀，则其递归树的高度为O（$\log_2 n$），故递归后需栈空间为O（$\log_2 n$）。在最坏的情况下，递归树的高度为O（n），所需的栈空间为O（n）。快速排序是不稳定的。

上一节　本书简介　下一节

归并排序

1.4.4 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由n个长度都为1的有序子表所组成，将它们依次两两归并得到长度为2的若干有序子表，再对它们两两合并。直到得到长度为n的有序表，排序结束。

例如，我们需要对关键码{72,28,51,17,96,62,87,33}进行排序，其归并过程如图1-26所示。

72 28 51 17 96 62 87 33
72 28 51 17 96 62 87 33
[28 72] [17 51] [62 96] [33 87]
[28 72] [17 51] [62 96] [33 87]
[17 28 51 72] [62 33 87 96]
[17 28 33 51 62 72 87 96]

图 1-26　归并排序的过程

归并排序是一种稳定的排序，可用顺序存储结构，也易于在链表上实现。对长度为n的文件，需进行$\log_2 n$趟二路归并，每趟归并的时间为O（n），故其时间复杂度无论是在最好情况下还是在最坏情况下均是O（$n\log_2 n$）。归并排序需要一个辅助向量来暂存两个有序子文件归并的结果，故其辅助空间复杂度为O（n），显然它不是就地排序。

上一节　本书简介　下一节

基数排序

1.4.5 基数排序

设单关键字的每个分量的取值范围均是$C_0 \le k_j \le C_{rd-1}$（$0 \le j < d$），可能的取值个数rd称为基数。基数的选择和关键字的分解因关键字的类型而异。

（1）若关键字是十进制整数，则按个、十等位进行分解，基数$r_d=10$,$C_0=0$,$C_9=9$,d为最长整数的位数。

（2）若关键字是小写的英文字符串，则r_d=26,C_0='a',C_{25}='z',d为字符串的最大长度。

基数排序的基本思想是：从低位到高位依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。

基数排序的具体实现过程如下：设有r个队列，队列的编号分别为0,1,2,…，r–1.首先按最低有效位的值把n个关键字分配到这r个队列中；然后从小到大将各队列中的关键字再依次收集起来；接着再按次低有效位的值把刚刚收集起来的关键字分配到r个队列中。重复上述收集过程，直至最高有效位，这样便得到一个从小到大的有序序列。为减少记录移动的次数，队列可以采用链式存储分配，称为链式基数排序。每个队列设有两个指针，分别指向队头和队尾。

例如，需要对{288,371,260,531,287,235,56,299,18,23}进行排序，因为这些数据最高位为百位，所以需要3趟分配和收集。第1趟分配和收集（按个位数）的过程如图1-16所示；第2趟分配与收集（按十位数）的过程如图1-17所示；第3趟分配与收集（按百位数）的过程如图1-18所示。

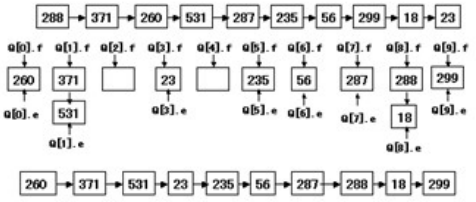


图1-27 第1趟分配与收集

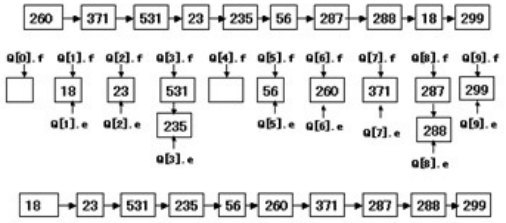


图1-28 第2趟分配与收集

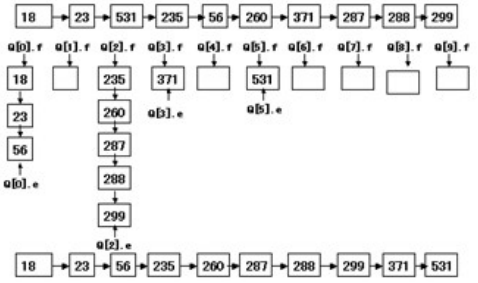


图1-29 第3趟分配与收集

基数排序的时间复杂度为O（d（r+n））,所需的辅助存储空间为O（n+r_d），基数排序是稳定的。

上一节 本书简介 下一节

作者：希赛教育软考学院 来源：希赛网 2014年01月26日

算法复杂性比较

1.4.6 算法复杂性比较

在此，我们把常用的排序算法的复杂度进行列表，如表1-3所示。

表1-3 排序算法时间复杂度表

排序方法	最好情况	平均时间	最坏情况	辅助空间	稳定性
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	√
简单选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	√
冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	√
快速排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n^2)$	$O(\log_2 n)$	×
堆排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(1)$	×
归并排序	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n\log_2 n)$	$O(n)$	√
基数排序	$O(d(n+rd))$	$O(d(n+rd))$	$O(d(n+rd))$	$O(rd)$	√

注：rd称为基数，基数的选择和关键字的分解因关键字的类型而异。

上一节 本书简介 下一节

查找

1.5 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做修改操作（如插入和删除），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效率优劣的标准。平均查找长度ASL定义为：

$$ASL = \sum_{i=1}^{n} p_i c_i$$

其中n是结点的个数，p_i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$p_1=p_2=\ldots=p_n=1/n$;c_i是找到第i个结点所需进行的比较次数。

上一节 本书简介 下一节

顺序查找

1.5.1 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与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。

成功时的顺序查找的平均查找长度如下：

$$ASL = \sum p_i c_i = \sum (n-i+1) = np_1 + (n-1)p_2 + \cdots + 2p_{n-1} + p_n$$

在等概率情况下，$p_i=1/n$（1≤i≤n），故成功的平均查找长度为（n+…+2+1）/n=（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，则需进行（n+1）次比较之后才能确定查找失败。

若事先知道表中各结点的查找概率不相等和它们的分布情况，则应将表中结点按查找概率由小到大地存放，以便提高顺序查找的效率。

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。

上一节　本书简介　下一节

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

二分法查找

1.5.2 二分法查找

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，并且要用向量作为表的存储结构。

二分法查找的基本思想是：（设R[low,…，high]是当前的查找区间）

①确定该区间的中点位置：mid=[（low+high）/2];

②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，否则需确定新的查找区间，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。

③下一次查找是针对新的查找区间进行，重复步骤（1）和（2）。

④在查找过程中，low逐步增加，而high逐步减少。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。

例如我们要在{11,13,17,23,31,36,40,47,52,58,66,73,77,82,96,99}中查找58的过程如图1-30所示（粗体表示mid位置）。在上述序列中查找35的过程如图1-31所示。

图1-30 二分法查找58

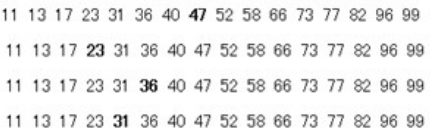


图1-31 二分法查找35

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。由此得到的二叉树，称为描述二分查找的判定树或比较树。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].key的取值无关。

设内部结点的总数为$n=2^h-1$,则判定树是深度为$h=\log_2(n+1)$的满二叉树。树中第k层上的结点个数为2^{k-1},查找它们所需的比较次数是k.因此在等概率假设下，二分法查找成功时的平均查找长度为$\log_2(n+1)-1$.二分法查找在查找失败时所需比较的关键字个数不超过判定树的深度，在最坏情况下查找成功的比较次数也不超过判定树的深度。即为$[\log_2(n+1)]$.二分法查找的最坏性能和平均性能相当接近。

虽然二分法查找的效率高，但是要将表按关键字排序。而排序本身是一种很费时的运算。即使采用高效率的排序方法也要花费$O(n\log_2 n)$的时间，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。

上一节 本书简介 下一节

第 1 章：数据结构基础 作者：希赛教育软考学院 来源：希赛网 2014年03月13日

分块查找

1.5.3 分块查找

分块查找（Blocking Search）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查找方法。二分查找表由分块有序的线性表和索引表组成。表R[1,…，n]均分为b块，前b−1块中结点个数为s=[n/b],第b块的结点数允许小于等于s;每一块中的关键字不一定有序，但前一块中的最大关键字必须小于后一块中的最小关键字，即表是分块有序的。

抽取各块中的最大关键字及其起始位置构成一个索引表ID[1,…，b],即ID[i]（$1\le i\le b$）中存放第i

块的最大关键字及该块在表R中的起始位置。由于表R是分块有序的，所以索引表是一个递增有序表。

分块查找的基本思想是索引表是有序表，可采用二分查找或顺序查找，以确定待查的结点在哪一块。

由于块内无序，只能用顺序查找。分块查找是两次查找过程。整个查找过程的平均查找长度是两次查找的平均查找长度之和。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ASL1 = $\log_2$（b+1）−1+（s+1）/2≈$\log_2$（n/s+1）+s/2;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ASL2 =（b+1）/2+（s+1）/2=（s^2+2s+n）/（2s）。

注意，当 $s=\sqrt{n}$ 时，ASL2取极小值 $\sqrt{n}+1$，即当采用顺序查找确定块时，应将各块中的结点数选定为 $\sqrt{n}$。

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，所以插入或删除比较容易，无须移动大量记录。

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。

上一节　本书简介　下一节

第 1 章：数据结构基础　作者：希赛教育软考学院　来源：希赛网　2014年03月13日

散列表

1.5.4 散列表

散列表又称杂凑表，是一种非常实用的查找技术，能在O（1）时间内完成查找。

将所有可能出现的关键字集合记为U（简称全集）。实际发生（即实际存储）的关键字集合记为K（|K|比|U|小得多）。散列方法是使用函数h将U映射到表T[0,…，m−1]的下标上（m=O（|U|））。这样以U中关键字为自变量，以h为函数的运算结果就是相应结点的存储地址。从而达到在O（1）时间内就可完成查找。

其中：

h:U→{0,1,2,…，m−1},通常称h为散列函数（Hash函数）。散列函数h的作用是压缩待处理的下标范围，使待处理的|U|个值减少到m个值，从而降低空间开销。

T为散列表（Hash Table）。

h（K_i）（K_i∈U）是关键字为Ki的结点存储地址（也称为散列值或散列地址）。

将结点按其关键字的散列地址存储到散列表中的过程称为散列（Hashing）。

1.常见的散列函数

除余法：令p是接近M的质数，设查找码为key,要求的桶号为T,计算T的散列函数为T=key %。

基数转换法：把查找码看做是某个基数制上的整数，然后将它转换成另一基数制上的数。

平方取中法：先通过求关键字的平方值扩大相近数的差别，然后根据表长度取中间的几位数作为散列函数值。又因为一个乘积的中间几位数和乘数的每一位都相关，所以由此产生的散列地址较为均匀。

折叠法：此方法将关键字分割成位数相同的几部分（最后一部分的倍数可以不同），然后取这几部分的叠加和（舍去进位）作为哈希地址。如果关键字位数很多，而且关键字中每一位上数字分布大致均匀时，可以采用折叠法得到哈希地址。

随机数法：选择一个随机函数，取关键字的随机函数值为它的散列地址，即 h（key）= random（key），其中random为伪随机函数，但要保证函数值在0到m−1之间。

2.冲突的解决

两个不同的关键字，由于散列函数值相同，因而被映射到同一表位置上。这种现象称为冲突或碰撞。发生冲突的两个关键字称为该散列函数的同义词。

冲突的频繁程度除了与h相关外，还与表的填满程度相关。设m和n分别表示表长和表中填入的结点数，则将α=n/m定义为散列表的装填因子。α越大，表越满，冲突的机会也越大，通常取α≤1.

解决冲突的方法是设法在散列表中找一个空位，通常有两类方法处理冲突，分别是开放定址法和拉链法。前者是将所有结点均存放在散列表T[0,…，m−1]中，后者通常是将互为同义词的结点链成一个单链表，而将此链表的头指针放在散列表T[0,…，m−1]中。

1）开放定址法

用开放定址法解决冲突的做法是当冲突发生时，使用某种探查（也称为探测）技术在散列表中形成一个探查序列。沿此序列逐个单元地查找，直到找到给定的关键字，或者碰到一个开放的地址（即该地址单元为空）为止。若要插入结点，在探查到开放的地址时，则可将待插入的新结点存入该地址单元。查找时探查到开放的地址则表明表中无待查的关键字，即查找失败。

根据探查方式不同，开放定址法又可分为线性探查法和双散列函数法。

（1）线性探查法

线性探查法将散列表T[0,…，m−1]看成一个循环向量，若初始探查的地址为d（即h（key）=d），则探查序列为d,d+l,d+2,…，m−1,0,1,…，d−1.即探查时从地址d开始，首先探查T[d],然后依次探查T[d+1],…，T[m−1],此后又循环到T[0],T[1],…，T[d−1].

那么，线性探查法什么时候终止呢？一般来说，线性探查法终止于下列3种情况之一。

若当前探查的单元为空，则表示查找失败（若是插入则将key写入其中）；

若当前探查的单元中含有key,则查找成功，但对于插入意味着失败；

若探查到T[d−1]时仍未发现空单元，也未找到key,则无论是查找还是插入，均意味着失败（此时表满）。

例如设记录关键码为（4,11,16,54,28,34,21），取m=11,p=7,h（key）=key%p,则采用线性探查法处理冲突的结果如图1-32所示。

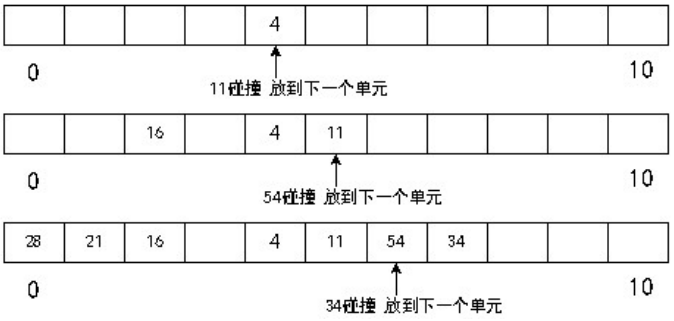


图1-32 线性探查法处理冲突

（2）双散列函数法

由于线性探查法容易产生堆积现象，会大大降低查找效率，可用双散列函数法。双散列函数法

是开放定址法中最好的方法之一，它的探查序列是：

hi=（h（key）+i*（ key%（p–1））））%p,0≤i≤m–1

该方法使用了两个散列函数h（key）和h1（key），故也称为双散列函数探查法。定义h1（key）的方法较多，但无论采用什么方法定义，都必须使h1（key）的值和m互素，才能使发生冲突的同义词地址均匀地分布在整个表中，否则可能造成同义词地址的循环计算。

例如设记录关键码为（4,11,16,54,28,34,21），取m=11,p=7,h（key）=key%p,则采用双散列函数法处理冲突的结果如图1-33所示。

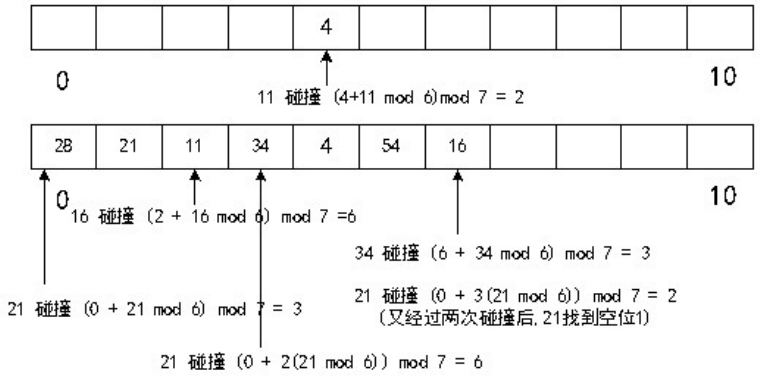


图1-33 双散列函数法处理冲突

2）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，用于链接同义词的子表，链表中的结点都是同义词。

拉链法解决冲突的做法是：将所有关键字为同义词的结点链接在同一个单链表中。若选定的散列表长度为m,则可将散列表定义为一个由m个头指针组成的指针数组T[0,…，m–1].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。T中各分量的初值均应为空指针。在拉链法中，装填因子α可以大于1,但一般均取α≤1.

例如设记录关键码为（4,11,18,54,28,34,21），取m=d=5,h（key）=key%d.采用拉链法的结果如图1-34所示。

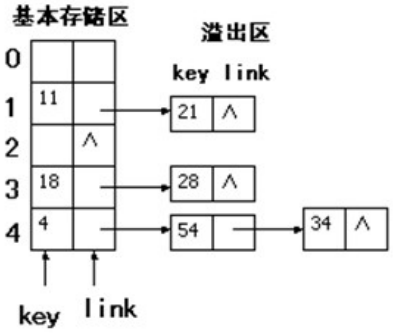


图1-34 拉链法处理冲突

上一节 本书简介 下一节

例题分析

1.6 例题分析

例题1（2011年5月试题57）

设下三角矩阵（上三角部分的元素值都为0）A[0n,0n]如图1-35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]（O≤i≤n,j≤i）存储在数组M的（57）中。

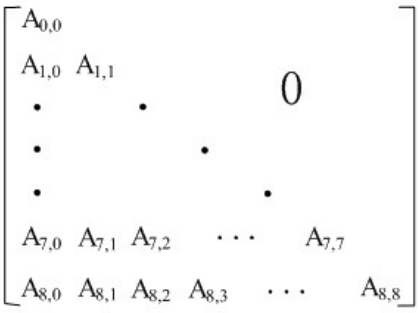


图1-35 状态转换图

（57）A. $M[\frac{i(i+1)}{2}+j+1]$　　B. $M[\frac{i(i+1)}{2}+j]$

C. $M[\frac{i(i-1)}{2}+j]$　　D. $M[\frac{i(i-1)}{2}+j+1]$

例题分析：

本题考查数据结构基础知识。

如题图所示，按行方式压缩存储时，A[i, j]之前的元素数目为（1+2+…+i+j）个，因为第i行前面的每行的元素个数分别为1,2,3,…，i.数组M的下标从1开始，因此A[i, j]的值存储在中 $M[\frac{i(i+1)}{2}+j+1]$。

例题答案：A

例题2（2011年5月试题58）

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.log2 n　　D.n2

例题分析：

本题主要考查顺序查找。

对于n个数据元素的表，若给定值key与表中第i个元素的关键字相等，则需进行n-i+1次关键字比较，即Ci=n-i+1.例如，当第n个元素的关键字为key时，需要比较1次（n-n+1=1），又如，当第1个元素为所求时，需要比较n次（n-1+1=n）。因此，查找成功时，顺序查找的平均查找长度为：

$$ASL=\sum_{i=1}^{n}P_i*(n-i+1)$$

其中Pi为每个元素的查找概率，假设所有元素的查找概率均相等，即 $P_i=\frac{1}{n}$，则在等概率情况下有：

例题答案：B

例题3（2011年5月试题59）

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

例题分析：

本题主要考查一些特殊二叉树的性质。

若二叉树中最多只有最下面两层的结点度数可以小于2,并且最下面一层的叶子结点都依次排列在该层最左边的位置上，则这样的二叉树称为完全二叉树，因此在完全二叉树中，任意一个结点的左、右子树的高度之差的绝对值不超过1。

二叉排序树的递归定义如下：二叉排序树或者是一棵空树；或者是具有下列性质的二叉树：

（1）若左子树不空，则左子树上所有结点的值均小于根结点的值；

（2）若右子树不空，则右子树上所有结点的值均大于根结点的值；

（3）左右子树也都是二叉排序树。

在n个结点的二叉树链式存储中存在n+1个空指针，造成了巨大的空间浪费，为了充分利用存储资源，可以将这些空链域存放指向结点在遍历过程中的直接前驱或直接后继的指针，这种空链域就称为线索，含有线索的二叉树就是线索二叉树。

最优二叉树即哈夫曼树。

例题答案：A

例题4（2011年5月试题60）

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

例题分析：

本题主要考查图的邻接矩阵存储结构。

设G=（V,E）是具有n个顶点的图，其中V是顶点的集合，E是边的集合，那么邻接矩阵中的每个元素的定义如下：

$$A[i,j]=\begin{cases}1 & (v_i,v_j)\in E \ \text{或} \ <v_i,v_j>\in E \\ 0 & (v_i,v_j)\notin E \ \text{或} \ <v_i,v_j>\notin E\end{cases}$$

从这个定义我们可以知道，一条边在矩阵中有两个1表示，比如顶点1和顶点2之间有一条边，那么矩阵元素A[1,2]和A[2,1]的值都是1.

在本题中，题目告诉我们有E条边，那么其邻接矩阵中的非零元素数目应该为2E。

例题答案：C

例题5（2011年5月试题61）

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

例题分析：

根据题目给出的散列函数我们可以分别计算出关键字（26,25,72,38,8,18,59）对应的散列地址分别为（0,12,7,12,8,5,7）。

开放定址处理冲突的基本思路是为发生冲突的关键字在散列表中寻找另一个尚未占用的位置，其解决冲突能力的关键取决于探测序列，在本题中，题目告诉我们采用顺序探查法，即增量为1的线性探测法，在该线性探测法中，设Hi（1≤i<m）为第i次在散列表中探测的位置，其中增量序列为{1,2,3,4,5,…，m-1}则有：

Hi = （H（Key）+i）%m

其中H（Key）为散列函数，m为散列表长度，i为增量序列。而本题中m=13.因此本题的散列表构造过程如下：

（1）关键字26,25,72由散列函数H（key）得到没有冲突的散列地址而直接存入散列表中。

（2）计算关键38的散列地址为12,发生冲突（与关键字25冲突），其第一次线性探测地址为（12+1）%13=0,但仍然发生冲突（与关键字26冲突），因此需要进行第二次线性探测，其地址为（12+2）%13=1,这时没有发生冲突，即将38存入地址为1的空间。

（3）接着将关键字8,18计算其散列地址，由于没有冲突，即分别存入散列地址为8和5的空间中。

（4）计算关键59的散列地址为7,发生冲突（与关键字72冲突），其第一次线性探测地址（7+1）%13=8,但仍然发生冲突（与关键字8冲突），因此需要进行第二次线性探测，其地址为（7+2）%13=9,这时没有发生冲突，即将59存入地址为9的存储空间。

因此本题的答案选D。

例题答案：D

例题6（2011年5月试题65）

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

例题分析：

插入排序的基本思想是逐个将待排序元素插入到已排序的有序表中。假设n个待排序元素存储在数组R[n+1]中（R[0]预留），则：

（1）初始时数组R[11]中只包含元素R[1],则数组R[11]必定有序；

（2）从i=2到n,执行步骤3;

（3）此时，数组R被划分成两个子区间，分别是有序区间R[1i-1]和无序区间R[in],将当前无序区间的第1个记录R[i]插入到有序区间R[1i]中适当的位置上，使R[1i]变为新的有序区间。

在实现的过程中，设置监视哨R[0],并从R[i-1]到R[0]查找元素R[i]的插入位置

那么用插入排序对数组<3,1,4,1,5,9,6,5>进行排序的过程为：

原元素序列：　监视哨　（3），1,4,1,5,9,6,5

第一趟排序：　3　　（1,3），4,1,5,9,6,5　　3插入时与1比较1次

第二趟排序：　4　　（1,3,4），1,5,9,6,5　　4插入时与3比较1次

第三趟排序：　1　　（1,1,3,4），5,9,6,5　　1插入时比较3次

第四趟排序：　5　　（1,1,3,4,5），9,6,5　　5插入时与4比较1次

第五趟排序：　9　　（1,1,3,4,5,9），6,5　　9插入时与5比较1次

第六趟排序：　6　　（1,1,3,4,5,6,9），5　　6插入时与9和5分别比较1次

第七趟排序：　5　　（1,1,3,4,5,5,6,9）　　5插入时与9,6,5分别比较1次

那么整个排序过程需要比较的次数为12次。

归并排序的思想是将两个相邻的有序子序列归并为一个有序序列，然后再将新产生的相邻序列进行归并，当只剩下一个有序序列时算法结束。其基本步骤如下：

（1）将n个元素的待排序序列中每个元素看成有序子序列，对相邻子序列两两合并，则将生成

$\lceil n/2 \rceil$个子有序序列，这些子序列中除了最后一个子序列长度可能小于2外，其他的序列长度都等于2；

（2）对上述$\lceil n/2 \rceil$个长度为2的子序列再进行相邻子序列的两两合并，则产生$\lceil n/4 \rceil$个子有序序列，同理，只有最后一个子序列的长度可能小于4;

（3）第i趟归并排序为，对上述$\lceil n/i \rceil$个长度为i的子序列两两合并，产生$\lceil n/2i \rceil$个长度为2i的子有序序列；

（4）重复执行此步骤，直到生成长度为n的序列为止。

那么用归并排序对数组<3,1,4,1,5,9,6,5>进行排序的过程为：

原元素序列：　3,1,4,1,5,9,6,5

第一趟排序：　[1,3],[1,4],[5,9],[5,6]　比较4次

第二趟排序：　[1,1,3,4], [5,5,6,9]　前半部分比较3次，后半部分比较3次

第三趟排序：　[1,1,3,4,5,5,6,9]　5分别与1,1,3,4比较一次

所以整个排序过程需要比较的次数为12次。

例题答案：D

例题7（2011年5月试题20~21）

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

例题分析：

逆波兰式也叫后缀表达式，即将运算符写在操作数之后的表达式，它不需使用括号，在将算术表达式转换为逆波兰式表示时，需要分配2个栈，一个作为临时存储运算符的栈S1（含一个结束符号），一个作为输入逆波兰式的栈S2（空栈）。

而逆波兰式ab-cd+*转换为中缀表达式的过程为：ab-cd+* =（ab-）*（cd+）=（a-b）*（cd+）=（a-b）*（c+d）。因此本题答案选C.

例题答案：（20）B（21）C

上一节　本书简介　下一节

第 2 章：程序语言基础知识　作者：希赛教育软考学院　来源：希赛网　2014年01月26日

汇编系统基本原理

第2章　程序语言基础知识

根据考试大纲，本章要求考生掌握以下知识点：

汇编、编译、解释系统的基础知识和基本工作原理；

程序设计语言的基本成分：数据、运算、控制和传输，过程（函数）调用；

各类程序设计语言的主要特点和适用情况。

程序语言是表达编程思想、描述计算过程的规范性语言。一般来说，程序语言可以分为低级语