**第一章 绪论**

1. 填空

⑴（**数据元素**）是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。

⑵（**数据项**）是数据的最小单位，（**数据元素**）是讨论数据结构时涉及的最小数据单位。

*数据结构指的是数据元素以及数据元素之间的关系。*

⑶ 从逻辑关系上讲，数据结构主要分为（**集合**）、（**线性结构**）、（**树结构**）和（**图结构**）。

⑷ 数据的存储结构主要有（**顺序存储结构**）和（**链式存储结构**）两种基本方法，不论哪种存储结构，都要存储两方面的内容：（**数据元素**）和（**数据元素之间的关系**）。

⑸ 算法具有5个特性，分别是（有零个或多个**输入**）、（有一个或多个**输出**）、（**有穷性**）、（**确定性**）、（**可行性**）。

⑹ 算法的描述方法通常有（**自然语言**）、（**程序设计语言**）、（**伪代码**）和（**流程图**）四种，其中，（**伪代码**）被称为算法语言。

⑺ 在一般情况下，一个算法的时间复杂度是（**问题规模**）的函数。

2. 选择题

⑴ 顺序存储结构中数据元素之间的逻辑关系是由（**C**）表示的，链接存储结构中的数据元素之间的逻辑关系是由（**D**）表示的。

A 线性结构 B 非线性结构 C 存储位置 D 指针

*顺序存储结构就是用一维数组存储数据结构中的数据元素，其逻辑关系由存储位置（即元素在数组中的下标）表示；链接存储结构中一个数据元素对应链表中的一个结点，元素之间的逻辑关系由结点中的指针表示。*

⑵ 假设有如下遗产继承规则：丈夫和妻子可以相互继承遗产；子女可以继承父亲或母亲的遗产；子女间不能相互继承。则表示该遗产继承关系的最合适的数据结构应该是（B）。

A 树 B 图 C 线性表 D 集合

⑶ 计算机所处理的数据一般具有某种内在联系，这是指（**B**）。

A数据和数据之间存在某种关系 B元素和元素之间存在某种关系

C元素内部具有某种结构 D数据项和数据项之间存在某种关系

⑷ 对于数据结构的描述，下列说法中不正确的是（A）。

A相同的逻辑结构对应的存储结构也必相同

B数据结构由逻辑结构、存储结构和基本操作三方面组成

C对数据结构基本操作的实现与存储结构有关

D数据的存储结构是数据的逻辑结构的机内实现

⑸ 可以用（**D**）定义一个完整的数据结构。

A数据元素 B数据对象 C数据关系 D抽象数据类型

（6）算法指的是（**A**）。

A 对特定问题求解步骤的一种描述，是指令的有限序列。

B 计算机程序 C 解决问题的计算方法 D 数据处理

（7）下面（**C**）不是算法所必须具备的特性。

A 有穷性 B 确切性 C 高效性 D 可行性

（8）算法分析的目的是（**C**），算法分析的两个主要方面是（**E**）。

A 找出数据结构的合理性 B 研究算法中输入和输出的关系

C 分析算法的效率以求改进 D 分析算法的易读性和文档性

E 空间性能和时间性能 F 正确性和简明性

G 可读性和文档性 H 数据复杂性和程序复杂性

3. 判断题

⑴ 算法的时间复杂度都要通过算法中的基本语句的执行次数来确定。（**×**）

***时间复杂度要通过算法中基本语句执行次数的数量级来确定。***

⑵ 每种数据结构都具备三个基本操作：插入、删除和查找。（**×**）

***如数组就没有插入和删除操作。。。***

⑶ 所谓数据的逻辑结构指的是数据之间的逻辑关系。（**×**）

***所谓数据的逻辑结构指的是数据之间逻辑关系的整体。***

⑷ 逻辑结构与数据元素本身的内容和形式无关。（**√**）

⑸ 基于某种逻辑结构之上的基本操作，其实现是唯一的。（**×**）

***基本操作的实现是基于某种存储结构设计的，因而不是唯一的。***

**第二章 线性表**

1. 填空

⑴ 在顺序表中，等概率情况下，插入和删除一个元素平均需移动（**n-i+1（表长的一半）**）个元素，具体移动元素的个数与（**MaxSize（表长）**）和（**该元素在表中的位置**）有关。

⑵ 顺序表中第一个元素的存储地址是 100，每个元素的长度为 2，则第 5 个元素的存储地址是（**108**）。

⑶ 设单链表中指针 p 指向结点 A，若要删除 A 的后继结点（假设 A 存在后继结点），则需修改指针的操作为（**p->next=(p->next)->next**）。

⑷ 单链表中设置头结点的作用是（统一空表和非空表（**为了方便运算）**）。

2. 选择题

⑴ 线性表的顺序存储结构是一种（**A**）的存储结构，线性表的链接存储结构是一种（**B**）的存储结构。

A 随机存取 B 顺序存取 C 索引存取 D 散列存取

*顺序存储结构的地址在内存中是连续的所以可以通过计算地址实现随机存取，而链式存储结构的存储地址不一定连续，只能通过某个结点的指针顺序存取。*

⑵ 线性表采用链接存储时，其地址（**D**）。

A 必须是连续的 B 部分地址必须是连续的

C 一定是不连续的 D 连续与否均可以

⑶ 链表不具有的特点是（**A**）。

A 可随机访问任一元素 B 插入、删除不需要移动元素

C 不必事先估计存储空间 D 所需空间与线性表长度成正比

（6）在一个单链表中，已知 q 所指结点是 p 所指结点的直接前驱，若在 q 和 p 之间插入 s 所指结点，则执行（**B**）操作。

A s->next=p->next; p->next=s; B q->next=s; s->next=p;

C p->next=s->next; s->next=p; D p->next=s; s->next=q;

3. 判断题

⑴ 线性表的逻辑顺序和存储顺序总是一致的。（**×**）

***顺序表的逻辑顺序和存储顺序一致，链表的逻辑顺序和存储顺序不一定一致。***

⑵ 线性表的顺序存储结构优于链接存储结构。（**×**）

⑶ 设 p，q 是指针，若 p=q，则\*p=\*q。（**×**）

***p=q只能表示p和q指向同一起始地址，而所指类型则不一定相同。***

⑷ 线性结构的基本特征是：每个元素有且仅有一个直接前驱和一个直接后继。（**×**）

***每个元素最多只有一个直接前驱和一个直接后继，第一个元素没有前驱，最后一个元素没有后继。***

⑸ 在单链表中，要取得某个元素，只要知道该元素所在结点的地址即可，因此单链表是随机存取结构。（**×**）

***要找该节点的地址，必须从头指针开始查找，所以单链表是顺序存取结构。***

4.算法设计

（1）单链表遍历算法。

**void PrintList(Node \*first){**

**Node \*p = first->next;**

**while(p!=NULL){**

**printf("%d ",p->data);**

**p = p->next;**

**}**

**printf("\n");**

**}**

1. 单链表求长度算法。

**int Length(Node \*first){**

**Node \*p = first->next;**

**int count = 0;**

**while (p != NULL){**

**count++;**

**p = p->next;**

**}**

**return count;**

**}**

1. 单链表按位查找算法。

**int Locate(Node \*first, Node \*x){**

**Node \*p = first->next;**

**int count = 1;**

**while (p != NULL){**

**if (p==x)**

**return count;**

**count++;**

**p = p->next;**

**}**

**return 0;**

**}**

1. 单链表的插入算法。

**int Insert(Node \*first, int i, DataType x){**

**Node \*s = NULL, \*p = first ;**

**int count = 0;**

**while (p != NULL && count < i - 1){**

**p = p->next;**

**count++;**

**}**

**if (p == NULL) {**

**printf("位置错误，插入失败\n");**

**return 0;**

**}**

**else {**

**s = (Node \*)malloc(sizeof(Node));**

**s->data = x;**

**s->next = p->next;**

**p->next = s;**

**return 1;**

**}**

**}**

1. 单链表的删除算法

**int Delete(Node \*first, int i, DataType \*ptr){**

**Node \*p = first, \*q = NULL;**

**int count = 0;**

**DataType x;**

**while (p != NULL && count < i - 1){**

**p = p->next;**

**count++;**

**}**

**if (p == NULL || p->next == NULL) {**

**printf("位置错误，删除失败\n ");**

**return 0;**

**}**

**else {**

**q = p->next;**

**\*ptr = q->data;**

**p->next = q->next;**

**free(q);**

**return 1;**

**}**

**}**

1. **特殊线性表**

1. 填空

⑴ 设有一个空栈，栈顶指针为 1000H，现有输入序列为 1、 2、 3、 4、 5， 经过 push， push， pop， push，pop， push， push 后，输出序列是（**23**），栈顶指针为（**1003H**）。

⑵ 栈通常采用的两种存储结构是（**顺序存储结构和链式存储结构**）；其判定栈空的条件分别是（**栈顶指针top=-1和top=NULL**），判定栈满的条件分别是（**栈顶指针top等于数组的长度（top=StackSize-1）和内存无可用空间**）。

⑶（**栈**）可作为实现递归函数调用的一种数据结构。

*递归函数的调用和返回正好符合后进先出性*

⑸ 栈和队列是两种特殊的线性表，栈的操作特性是（**后进先出**），队列的操作特性是（**先进先出**），栈和队列的主要区别在于（**插入和删除的位置不同**）。

⑹ 循环队列的引入是为了克服（**假溢出**）。

2. 选择题

⑴ 一个栈的入栈序列是 1， 2， 3， 4， 5，则栈的不可能的输出序列是（C）。

A 54321 B 45321 C 43512 D 12345

⑵ 若一个栈的输入序列是 1， 2， 3， …， n，输出序列的第一个元素是 n，则第 i 个输出元素是（**D**）。

A 不确定 B n-i C n-i-1 D n-i+1

⑶ 若一个栈的输入序列是1，2，3，…，n，其输出序列是p1，p2，…，pn，若p1=3，则p2的值（C）。

A 一定是2 B一定是1 C不可能是1 D以上都不对

⑷ 设计一个判别表达式中左右括号是否配对的算法，采用（**B**）数据结构最佳

A 顺序表 B 栈 C 队列 D 链表

*每个右括号与它前面的最后一个没有匹配的左括号配对，因此具有后进先出性。*

⑸ 在解决计算机主机与打印机之间速度不匹配问题时通常设置一个打印缓冲区，该缓冲区应该是一个（**B**）结构。

A 栈 B 队列 C 数组 D 线性表

*先进入打印缓冲区的文件先被打印，因此具有先进先出性*

⑹ 一个队列的入队顺序是 1， 2， 3， 4，则队列的输出顺序是（**B**）。

A 4321 B 1234 C 1432 D 3241

⑺ 栈和队列的主要区别在于（D）。

A 它们的逻辑结构不一样 B 它们的存储结构不一样

C 所包含的运算不一样 D 插入、删除运算的限定不一样

3. 判断题

⑴ 有 n 个元素依次进栈，则出栈序列有(n-1)/2 种。（×）

***应该有种。***

⑵ 栈可以作为实现过程调用的一种数据结构。（**√**）

⑶ 在栈满的情况下不能做进栈操作，否则将产生“上溢”。（**√**）

⑷ 在循环队列中， front 指向队头元素的前一个位置， rear 指向队尾元素的位置，则队满的条件是front=rear。（**×**）

***这是队空的判定条件，在循环队列中要将队空和队满的判定条件区别开。***

1. 解答下列问题
2. 设有一个栈，元素进栈的次序为A，B，C，D，E，能否得到如下出栈序列，若能，请写出操作序列，若不能，请说明原因。

① C，E，A，B，D

**不能，因为在C，E出栈的情况下，A一定在栈中，而且在B的下面，不可能先于B出栈。**

② C，B，A，D，E

**能，push(A)、push(B)、push(C)、pop、pop、pop、push(D)、pop、push(E)、pop**

1. 在操作序列 EnQueue(1)、 EnQueue(3)、 DeQueue、EnQueue(5)、EnQueue(7)、DeQueue、EnQueue(9)之后，队头元素和队尾元素分别是什么？（EnQueue(k)表示整数 k 入队， DeQueue 表示队头元素出队）

**对头：5，队尾：9**

1. **字符串和多维数组**

1. 填空

（1）串是一种特殊的线性表，其特殊性体现在（**数据元素是一个字符**）。

（2）两个串相等的充分必要条件是（**字符串长度相同且对应位置字符相同**）。

（3）数组通常只有两种运算：（**存取**）和（**修改**），这决定了数组通常采用（**顺序存储**）结构来实现存储。

2. 选择题

（1）设有两个串 p 和 q，求 q 在 p 中首次出现的位置的运算称作（**B**）。

A 连接 B 模式匹配 C 求子串 D 求串长

（3）将数组称为随机存取结构是因为（**B**）

A 数组元素是随机的 B 对数组任一元素的存取时间是相等的

C 随时可以对数组进行访问 D 数组的存储结构是不定

（4）下面的说法中，不正确的是（**C**）

A 数组是一种线性结构 B 数组是一种定长的线性结构

C 除了插入与删除操作外，数组的基本操作还有存取、修改、检索和排序等

D 数组的基本操作有存取、修改、检索和排序等，没有插入与删除操作

*数据属于广义线性表，数组被创建后，其维度和每维中的元素个数是确定的，所以数组通常没有插入和删除操作。*

（5）对特殊矩阵采用压缩存储的目的主要是为了（**D**）

A 表达变得简单 B 对矩阵元素的存取变得简单

C 去掉矩阵中的多余元素 D 减少不必要的存储空间

*在特殊矩阵中，有很多值相同的元素并且它们的分布有规律，没有必要为值相同的元素重复存储。*

（6）下面（**C**）不属于特殊矩阵。

A 对角矩阵 B 三角矩阵 C 稀疏矩阵 D 对称矩阵

1. 下面的说法中，不正确的是（**D**）

A 对称矩阵只须存放包括主对角线元素在内的下（或上）三角的元素即可。

B 对角矩阵只须存放非零元素即可。

C 稀疏矩阵中值为零的元素较多，因此可以采用三元组表方法存储。

D 稀疏矩阵中大量值为零的元素分布有规律，因此可以采用三元组表方法存储

*稀疏矩阵中大量值为零的元素分布没有规律，因此采用三元组表存储。如果零元素的分布有规律，就没有必要存储非零元素的行号和列号，而需要按其压缩规律找出相应的映象函数。*

3. 判断题

（1）数组是一种复杂的数据结构，数组元素之间的关系既不是线性的，也不是树形的。（**×**）

***例如二维数组可以看成是数据元素为线性表的线性表。***

（2）空串与空格串是相同的。（**×**）

1. **树和二叉树**
2. 填空

⑴ 树是 n（n≥0）结点的有限集合，在一棵非空树中，有（**1**）个根结点，其余的结点分成 m（m＞0）个（**互不相交**）的集合，每个集合都是根结点的子树。

⑵ 树中某结点的子树的个数称为该结点的（**度**），子树的根结点称为该结点的（**孩子**），该结点称为其子树根结点的（**双亲**）。

⑶ 一棵二叉树的第 i（i≥1）层最多有（**2i-1**）个结点；一棵有 n（n>0）个结点的满二叉树共有（）个叶子结点和（）个非终端结点。

⑷ 设高度为 h 的二叉树上只有度为 0 和度为 2 的结点，该二叉树的结点数可能达到的最大值是（**2h-1**），最小值是（**2h-1**）。

⑸ 深度为 k 的二叉树中，所含叶子的个数最多为（**2k-1**）。

⑹ 具有 100 个结点的完全二叉树的叶子结点数为（**50**）。

⑺ 已知一棵度为 3 的树有 2 个度为 1 的结点，3 个度为 2 的结点，4 个度为 3 的结点。则该树中有（**12**）个叶子结点。

⑻ 某二叉树的前序遍历序列是 ABCDEFG，中序遍历序列是 CBDAFGE，则其后序遍历序列是（**CDBGFEA**）。

⑼ 在具有 n 个结点的二叉链表中，共有（**2n**）个指针域，其中（**n-1**）个指针域用于指向其左右孩子，剩下的（**n+1**）个指针域则是空的。

（10） 在有 n 个叶子的哈夫曼树中，叶子结点总数为（**n**），分支结点总数为（**n-1**）。

2. 选择题

⑴ 如果结点 A 有 3 个兄弟，B 是 A 的双亲，则结点 B 的度是（**D**）。

A 1 B 2 C 3 D 4

⑵ 设二叉树有 n 个结点，则其深度为（**D**）。

A n-1 B n C+1 D不能确定

*没有指明是完全二叉树，则其深度最多是n，最少是**+1*

⑶ 二叉树的前序序列和后序序列正好相反，则该二叉树一定是（**B**）的二叉树。

A 空或只有一个结点 B 高度等于其结点数

C 任一结点无左孩子 D 任一结点无右孩子

（4） 深度为 k 的完全二叉树至少有（**B**）个结点，至多有（**C**）个结点。

A 2k-2+1 B 2k-1 C 2k -1 D 2k–1 -1

（5） 一个高度为 h 的满二叉树共有 n 个结点，其中有 m 个叶子结点，则有（**D**）成立。

A n=h+m B h+m=2n C m=h-1 D n=2m-1

（6） 任何一棵二叉树的叶子结点在前序、中序、后序遍历序列中的相对次序（**A**）。

A 肯定不发生改变 B 肯定发生改变 C 不能确定 D 有时发生变化

（7）如果 T' 是由有序树 T 转换而来的二叉树，那么 T 中结点的前序序列就是 T' 中结点的（**A**）序列，T 中结点的后序序列就是 T' 中结点的（**B**）序列。

A 前序 B 中序 C 后序 D 层序

（8）设森林中有 4 棵树，树中结点的个数依次为 n1、n2、n3、n4，则把森林转换成二叉树后，其根结点的右子树上有（**D**）个结点，根结点的左子树上有（**A**）个结点。

A n1-1 B n1 C n1+n2+n3 D n2+n3+n4

（9）讨论树、森林和二叉树的关系，目的是为了（**B**）。

A 借助二叉树上的运算方法去实现对树的一些运算

B 将树、森林按二叉树的存储方式进行存储并利用二叉树的算法解决树的有关问题

C 将树、森林转换成二叉树

D 体现一种技巧，没有什么实际意义

1. 判断题

（1） 在二叉树的前序遍历序列中，任意一个结点均处在其子女的前面。（**√**）

（2） 二叉树是度为 2 的树。（**×**）

（3）由树转换成二叉树，其根结点的右子树总是空的。（**√**）

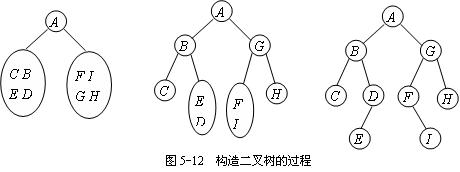
***因为根结点无兄弟结点***

1. 用一维数组存储二叉树时，总是以前序遍历存储结点。（**×**）

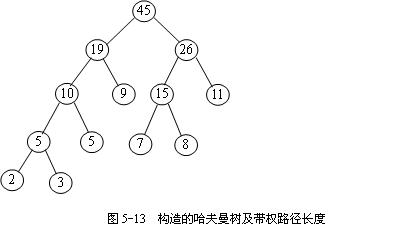
***二叉树的顺序存储结构是按层序存储的，一般适合存储完全二叉树***

1. 解答题

（1）已知二叉树的中序和后序序列分别为CBEDAFIGH和CEDBIFHGA，试构造该二叉树。



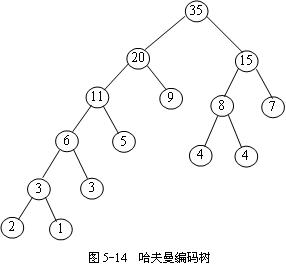
1. 对给定的一组权值W＝（5，2，9，11，8，3，7），试构造相应的哈夫曼树，并计算它的带权路径长度。



**wpl=2\*4+3\*4+5\*3+9\*2+7\*3+8\*3+11\*2=120**

（3）已知某字符串S中共有8种字符，各种字符分别出现2次、1次、4次、5次、7次、3次、4次和9次，对该字符串用[0，1]进行前缀编码，问该字符串的编码至少有多少位。

**以各字符出现的次数作为叶子结点的权值构造的哈夫曼编码树如图5-14所示。其带权路径长度=2×5+1×5+3×4+5×3+9×2+4×3+4×3+7×2=98，所以，该字符串的编码长度至少为98位。**



1. 算法设计题

⑴ 设计算法实现二叉树的前序遍历操作。（就是二叉树的前序遍历算法）

**void PreOrder(BiNode \*root){**

**if (root == NULL) return;**

**else {**

**printf("%c ",root->data);**

**PreOrder(root->lchild);**

**PreOrder(root->rchild);**

**}**

**}**

⑵ 设计算法实现二叉树的中序遍历操作。（就是二叉树的中序遍历算法）

**void InOrder(BiNode \*root){**

**if(root == NULL) return;**

**else {**

**InOrder(root->lchild);**

**printf("%c ",root->data);**

**InOrder(root->rchild);**

**}**

**}**

⑶ 设计算法实现二叉树的后序遍历操作。（就是二叉树的后序遍历算法）

**void PostOrder(BiNode \*root){**

**if(root == NULL) return;**

**else {**

**PostOrder(root->lchild);**

**PostOrder(root->rchild);**

**printf("%c ",root->data);**

**}**

**}**

⑷ 设计算法求二叉树的结点个数。

**void Count(BiNode \*root) {**

**if (root == NULL) return;**

**else {**

**Count(root->lchild);**

**count++;**

**Count(root->rchild);**

**}**

**}**

1. 设计算法按前序次序打印二叉树中的叶子结点。

**void PreOrder(BiNode \*root){**

**if (root == NULL) return;**

**else {**

**if (!root->lchild && !root->rchild)**

**printf("%c ",root->data);**

**PreOrder(root->lchild);**

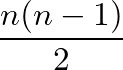
**PreOrder(root->rchild);**

**}**

**}**

1. **图**

1.填空

⑴ 设无向图 G 中顶点数为 n，则图 G 至少有（**0**）条边，至多有（）条边；若 G 为有向图，则至少有（**0**）条边，至多有（**n(n-1)**）条边。

⑵ 任何连通图的连通分量只有一个，即是（**其本身**）。

⑶ 图的存储结构主要有两种，分别是（**邻接矩阵**）和（**邻接表**）。

⑷ 已知无向图 G 的顶点数为 n，边数为 e，其邻接表表示的空间复杂度为（**O(n+e)**）。

⑸ 已知一个有向图的邻接矩阵表示，计算第 j 个顶点的入度的方法是（**求第j列的所有元素之和**）。

⑹ 有向图 G 用邻接矩阵 A[n][n]存储，其第 i 行的所有元素之和等于顶点 i 的（**出度**）。

⑺ 图的深度优先遍历类似于树的（**前序**）遍历，它所用到的数据结构是（**栈**）；图的广度优先遍历类似于树的（**层序**）遍历，它所用到的数据结构是（**队列**）。

2. 选择题

⑴ 在一个无向图中，所有顶点的度数之和等于所有边数的（**C**）倍。

A 1/2 B 1 C 2 D 4

⑶ 含 n 个顶点的连通图中的任意一条简单路径，其长度不可能超过（**C**）。

A 1 B n/2 C n-1 D n

*若超过n-1，则路径中必存在重复的顶点*

（5） 对于一个具有 n 个顶点的无向图，若采用邻接矩阵存储，则该矩阵的大小是（**D**）。

A n B (n-1)2 C n-1 D n2

（6） 图的生成树（**C**）， n 个顶点的生成树有（**F**）条边。

A 唯一 B 不唯一 C 唯一性不能确定 D n E n +1 F n-1

（7）设无向图 G=(V, E)和 G' =(V', E' )，如果 G' 是 G 的生成树，则下面的说法中错误的是（**B**）。

A G' 为 G 的子图 B G' 为 G 的连通分量

C G' 为 G 的极小连通子图且 V = V' D G' 是 G 的一个无环子图

*连通分量是无向图的极大连通子图，其中极大的含义是将依附于连通分量中顶点的所有边都加上，所以，连通分量中可能存在回路。*

（8） G 是一个非连通无向图，共有 28 条边，则该图至少有（**D**）个顶点。

A 6 B 7 C 8 D 9

（11） 最小生成树指的是（**C**） 。

A 由连通网所得到的边数最少的生成树

B 由连通网所得到的顶点数相对较少的生成树

C 连通网中所有生成树中权值之和为最小的生成树

D 连通网的极小连通子图

3.判断题

⑴ 用邻接矩阵存储图，所占用的存储空间大小只与图中顶点个数有关，而与图的边数无关。（**√**）

***邻接矩阵的空间复杂度为O(n2)，与边的个数无关。***

1. 图 G 的生成树是该图的一个极小连通子图（**×**）

***必包含全部顶点***

1. 无向图的邻接矩阵一定是对称的，有向图的邻接矩阵一定是不对称的（**×**）

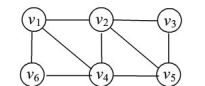
***有向图的邻接矩阵不一定对称***

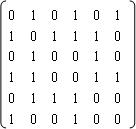
1. 对任意一个图，从某顶点出发进行一次深度优先或广度优先遍历，可访问图的所有顶点。（**×**）

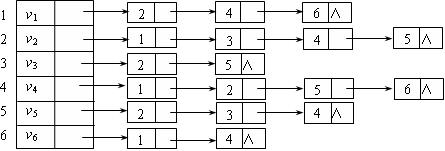
***只有连通图从某顶点出发进行一次遍历，可访问图的所有顶点。***

4.解答题



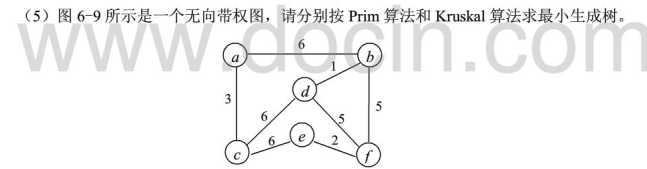


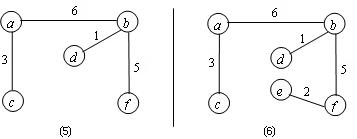
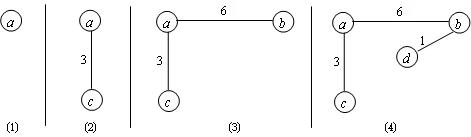
**邻接矩阵：**

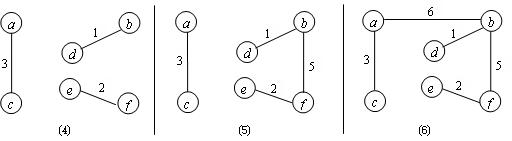
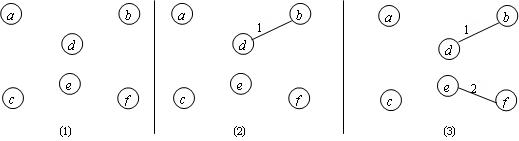
**邻接表：**

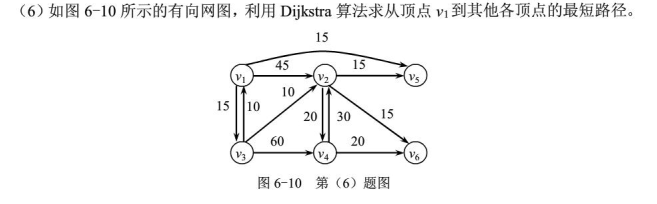
**深度优先遍历序列：v1、v2、v3、v5、v4、v6**

**广度优先遍历序列：v1、v2、v4、v6、v3、v5**



**Prime算法：**

**Kruskal算法：**



**v1->v3：v1 v3 最短路径：15**

**v1->v5：v1 v5 最短路径：15**

**v1->v2：v1 v3 v2 最短路径：25**

**v1->v6：v1 v3 v2 v6 最短路径：40**

**v1->v4：v1 v3 v2 v4 最短路径：45**

5.算法设计题

⑴设计算法，基于邻接矩阵实现图的深度优先遍历。

**void DFTraverse(MGraph \*G, int v) {**

**printf("%c ", G->vertex[v]);**

**visited[v] = 1;**

**for (int j = 0; j < G->vertexNum; j++)**

**if (G->edge[v][j] == 1 && visited[j] == 0)**

**DFTraverse(G, j);**

**}**

⑵设计算法，基于邻接矩阵实现图的广度优先遍历。

**void BFTraverse(MGraph \*G, int v){**

**int i, j, Q[MaxSize];**

**int front = rear = -1;**

**printf("%c ", G->vertex[v]);**

**visited[v] = 1; Q[++rear] = v;**

**while (front != rear){**

**i = Q[++front];**

**for (j = 0; j < G->vertexNum; j++)**

**if (G->edge[i][j] == 1 && visited[j] == 0 ) {**

**printf("%c ", G->vertex[j]);**

**visited[j] = 1;**

**Q[++rear] = j;**

**}**

**}**

**}**

⑶设计算法，基于邻接表实现图的深度优先遍历。

**void DFTraverse(ALGraph \*G, int v){**

**EdgeNode \*p = NULL;**

**int j;**

**printf("%c ", G->adjlist[v].vertex);**

**visited[v] = 1;**

**p = G->adjlist[v].first;**

**while (p != NULL){**

**j = p->adjvex;**

**if (visited[j] == 0)**

**DFTraverse(G, j);**

**p = p->next;**

**}**

**}**

1. 设计算法，基于邻接表实现图的广度优先遍历。

**void BFTraverse(ALGraph \*G, int v){**

**int Q[MaxSize],front,rear,i = 0,j=0;**

**front = rear = -1;**

**EdgeNode \*p = NULL;**

**printf("%c ", G->adjlist[j].vertex);**

**visited[v] = 1;**

**Q[++rear]=v;**

**while (front != rear){**

**v=Q[++front];**

**p = G->adjlist[i].first;**

**while (p!=NULL) {**

**j = p->adjvex;**

**if (visited[j] == 0) {**

**printf("%c ", G->adjlist[j].vertex);**

**visited[j] = 1;**

**Q[++rear] = j;**

**}**

**p=p->next;**

**}**

**}**

**printf("%c ", G->adjlist[v].vertex);**

**visited[v] = 1;**

**Q[++rear] = v;**

**}**

1. **查找技术**

1.填空

⑴ 顺序查找技术适合于存储结构为（**顺序存储和链式存储**）的线性表，而折半查找技术适用于存储结构为（**顺序存储**）的线性表，并且表中的元素必须是（**按关键码有序**）。

（2） 在散列技术中，处理冲突的两种主要方法是（**开放定址法**）和（**拉链法**）。

（3） 在各种查找方法中，平均查找长度与结点个数无关的查找方法是（**散列查找**）。

2. 选择题

⑴ 静态查找与动态查找的根本区别在于（**B**）。

A 它们的逻辑结构不一样 B 施加在其上的操作不同

C 所包含的数据元素的类型不一样 D 存储实现不一样

*静态查找不涉及插入和删除操作，而动态查找涉及插入和删除操作。*

（2） 二叉排序树中，最小值结点的（**A**）。

A 左指针一定为空 B 右指针一定为空

C 左、右指针均为空 D 左、右指针均不为空

*在二叉排列树中，值最小的结点一定是中序遍历序列中第一个被访问的结点，即二叉树最左下结点。*

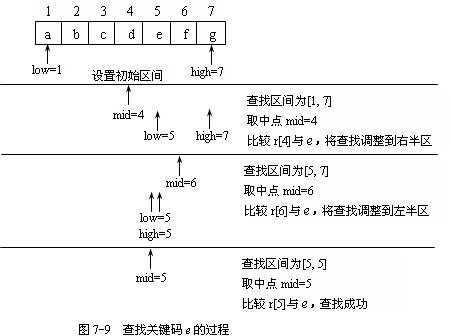
（3）散列技术中的冲突指的是（**D**）。

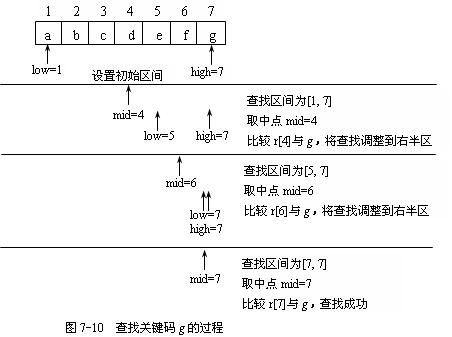
A 两个元素具有相同的序号 B 两个元素的键值不同，而其他属性相同

C 数据元素过多 D 不同键值的元素对应于相同的存储地址

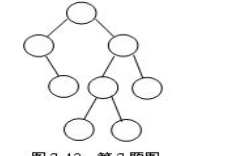
3.解答题

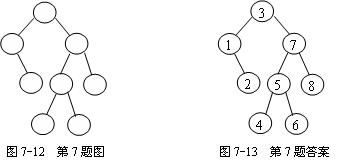
1. 分别画出在线性表（a， b， c， d， e， f， g）中进行折半查找关键码 e 和 g 的过程。



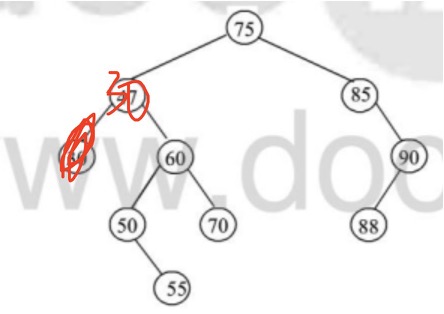
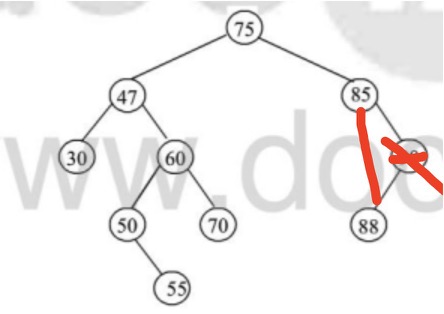
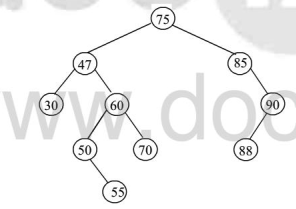


1. 一棵二叉排序树的结构如图 所示，结点的值为 1～8，请标出各结点的值。





（3）



**第八章 排序技术**

1.判断题

⑴ 如果某种排序算法是不稳定的，则该排序方法没有实际应用价值。（**×**）

***一种排序算法适合于某种特定的数据环境，有时对排序的稳定性没有要求。***

⑵ 当待排序的元素很大时，为了交换元素的位置，移动元素要占用较多的时间，这是影响时间复杂性的主要因素。（**√**）

⑶ 对 n 个记录的集合进行快速排序，所需要的附加空间是 Ο(n)。（**×**）

***最坏情况下是 Ο(n)。***

⑷ 堆排序所需的时间与待排序的记录个数无关。（**×**）

***堆排序最好最坏及平均时间均为O(nlog2n)，是待排序的记录个数n的函数。一般来说，待排序的记录个数越多，排序所消耗的时间也就越多。***

⑸ 设有键值序列（k1, k2, …, kn），当 i>n/2 时，任何一个子序列（ki, ki+1,… , kn）一定是堆。（**√**）

2.解答题

1. 已知数据序列为(12， 5， 9， 20， 6， 31， 24)，对该数据序列进行排序，写出直接插入排序、起泡排序、简单选择排序以及二路归并排序每趟的结果。



1. 判别下列序列是否为堆，如不是，按照堆排序思想把它调整为堆，用图表示建堆的过程：
2. （1， 5， 7， 25， 21， 8， 8， 42）

**是堆。**

b.（3， 9， 5， 8， 4， 17， 21， 6）。

**不是堆。**