# 目录

**【注】确认收货后评价+带 3 图以上联系客服加 VIP 群 圆梦西工大**

# 期末试题部分

# 西北工业大学 2000-2001 学年第二学期期末考试

## 一.简述题

**1.**数据结构是相互之间存在一种或多种特定关系的数据元素的集合。算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。

两者之间的关系：数据结构只是静态地描述了数据元素之间的关系，而算法侧重于对问题的建模。程序设计=数据结构+算法，高效的程序需要在数据结构的基础上设计和选择算法。数据结构是算法实现的基础，算法总是要依赖于某种数据结构来实现的。

**2.**线性表是具有相同数据类型的 n(n≥0)个数据元素的有限序列。除第一个元素外，每个元素有且仅有一个直接前驱。除最后一个元素外，每个元素有且仅有一个直接后继。线性表是从逻辑结构的角度来定义的；

数组是从物理结构的角度来定义的，当线性表采用顺序存储结构时，可用数组来实现。

广义表是一种非线性的数据结构，是线性表的一种推广。即广义表中放松对表元素的原子限制，容许它们具有其自身结构。

**3.**线索二叉树：在二叉链表表示的二叉树中存在大量的空指针，若利用这些空链域存放指向其直接前驱或后继的指针，称之为线索。加上线索的二叉树称为线索二叉树。

在中序线索二叉树查找直接前驱：

(1)若左标志为 1，则左链域指示其前驱；

(2)若左标志为 0，则遍历左子树时最后访问的一个结点(左子树最右下的结点)为其前驱。

在中序线索二叉树查找直接后继：

(1)若右标志为 1，则右链域指示其后继；

(2)若右标志为 0，则遍历右子树时访问的第一个结点(右子树最左下的结点)为其后继。

**4.**数据结构物理描述的基本方式：

(1)顺序存储：把逻辑上相邻的元素存储在物理位置上也相邻的存储单元里，元素之间的关系由存储单元的邻接关系来体现；

(2)链式存储：不要求逻辑上相邻的元素在物理位置上也相邻，借助指示元素存储地址的指针表示元素之间的逻辑关系；

(3)索引存储：建立附加的索引表，索引项的一般形式是：(关键字，地址)。

(4)散列存储：根据元素的关键字直接计算出该元素的存储地址。

**5.**哈希(散列)查找：通过计算数据元素的存储地址进行查找的一种方法。【略】

**6.**外部排序的基本思想：主要采用归并排序方法。首先，根据内存缓冲区的大小，将外存上含 n 个记录的文件分成若干长度为 h 的子文件，依次读入内存并利用有效的内部排序算法对它们进行排序，并将排序后得到的有序子文件(归并段)重新写回外存；然后，对这些归并段进行逐趟归并，使其逐渐由小变大，直到得到整个有序文件为止。

## 二.算法思想

**1.**拓扑排序的算法思想：(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出；

(2)从图中删除该顶点和所有以它为起点的有向边；

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。

所用数据结构：邻接矩阵或邻接表，栈或队列

**2.**一般树转化为二叉树：

(1)将树的根节点直接作为二叉树的根节点。

(2)将树的根节点的第一个孩子节点作为二叉树根节点的左指针，若该孩子节点存在兄弟节点，则将该孩子节点的第一个兄弟节点(方向从左往右)作为该子节点的右指针。

(3)树中的剩余节点按照上一步的方式(左孩子，右兄弟)，依序添加到二叉树中。直到树中所有的节点都在二叉树中。

所用数据结构：树和二叉树都采用二叉链表存储

**3.**最小生成树(任选一种)：【注】最小生成树的题目要先写算法思想再画步骤。

(1)Prim 算法：

初始化：向空树 $T=(V_T,E_T)$中添加图 $G=(V,E)$的任一顶点$\mu_0$，使 $V_T=\{\mu_0\}$，$E_T=\varnothing$ 。

循环(重复下列操作至 $V_T=V$):从图 G 中选择满足$\{(\mu, v)|\mu\in V_T, v\in V-V_T\}$且具有最小权值的边(μ, v)，并置 $V_T=V_T\cup\{v\},E_T=E_T\cup\{(\mu, v)\}$。

所用数据结构：邻接矩阵

(2)Kruskal 算法：

初始化：$V_T=V,E_T=\varnothing$ 。即每个顶点构成一棵独立的树，T 此时是一个仅含|V|个顶点的森林；

循环(重复下列操作至 T 是一棵树)：按 G 的边的权值递增顺序依次从 $E-E_T$ 中选择一条边，如果这条边加入 T 后不构成回路，则将其加入 $E_T$,否则舍弃，直至 $E_T$ 中含有 n-1 条边。

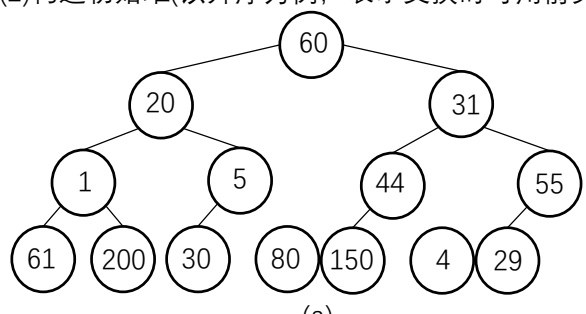
所用数据结构：并查集

## 三.算法应用

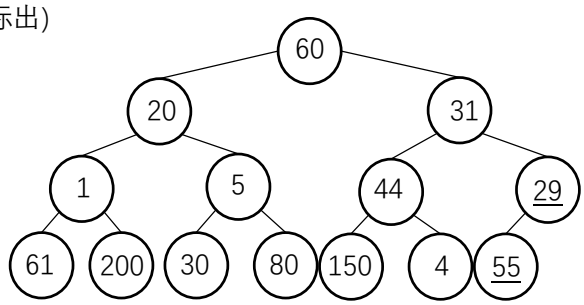
**1.**(1)一趟快速排序(以升序为例)：

| | |
|---|---|
| 60 20 31 1 5 44 55 61 200 30 80 150 4 29 | 29 20 31 1 5 44 55 4 200 30 80 150 60 61 |
| 29 20 31 1 5 44 55 61 200 30 80 150 4 60 | 29 20 31 1 5 44 55 4 60 30 80 150 200 61 |
| 29 20 31 1 5 44 55 60 200 30 80 150 4 61 | 29 20 31 1 5 44 55 4 30 60 80 150 200 61 |

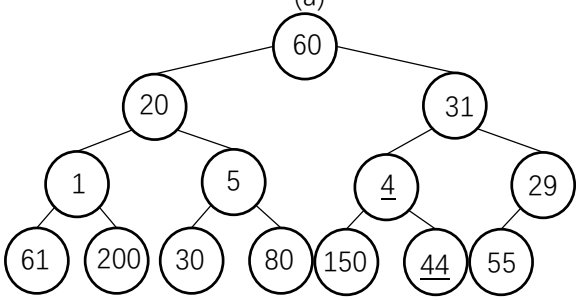
(2)构造初始堆(以升序为例，表示交换时可用箭头标出)



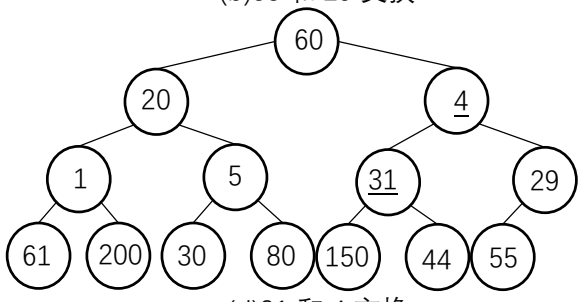
(a)



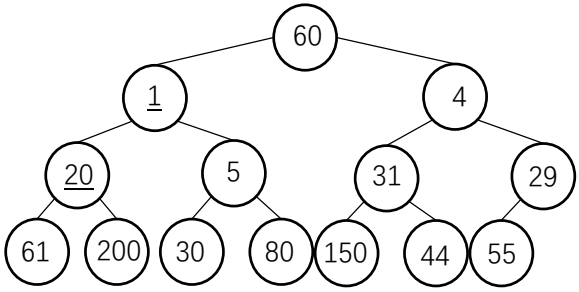
(b)55 和 29 交换



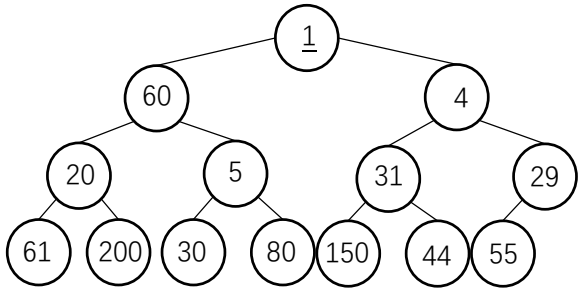
(c)44 和 4 交换



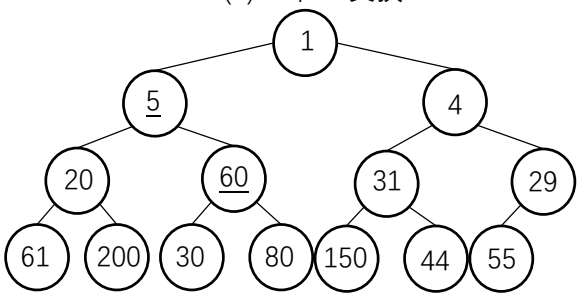
(d)31 和 4 交换



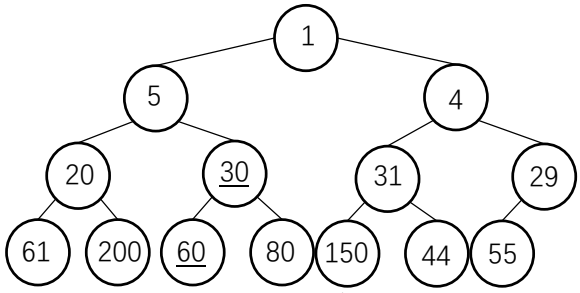

(e)20 和 1 交换




(f)60 和 1 交换



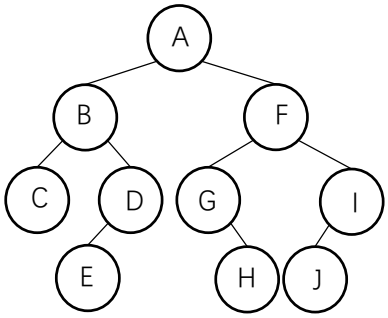

(g)60 和 5 交换



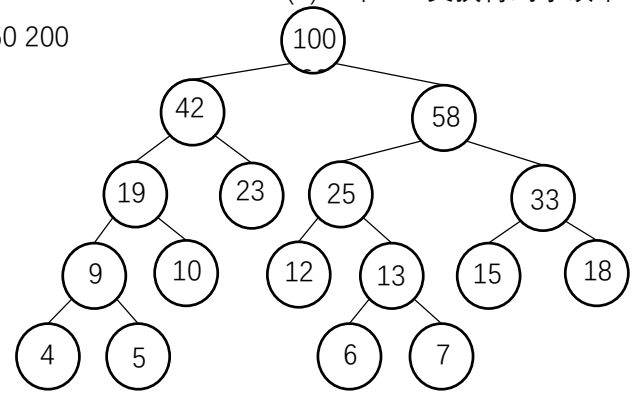

(h)60 和 30 交换得到小顶堆

排序结果：1 4 5 20 29 30 31 44 55 60 61 80 150 200

**2.**



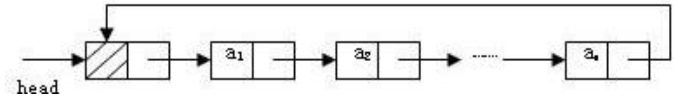

**3.**




带权路径长度：(4+5+6+7)×4+(10+12+15+18)×3+23×2=299

**四.用 C 语言写一个程序**

**1.**带头结点的单循环链表：




代码：

```
int judge(LNode* L){
    if(L->next == L)   //只有头结点
        return 1;
    LNode* pre = L->next;
    LNode* q = pre->next;
    while(q!=L){
    if(abs(pre->data-q->data)>2)
        return 0;
        pre = q;
        q = q->next;
    }
    if(abs(pre->data-L->next->data)>2)
        return 0;
    return 1;
}
```

**2.**采用递归的方法，代码如下：

```
int Degree(BiTree * t){
    if(!t) //根节点为空
        return 0;
    //只有根节点
    else if (t->lchild == NULL && t->rchild == NULL)
        return 1+ Degree(t->lchild);
//有右孩子没有左孩子
else if (t->lchild == NULL && t->rchild != NULL)
    return 1 + Degree(t->rchild);
//左右孩子都有
```

```
        return 0;
    //有左孩子没有右孩子
    else if (t->lchild != NULL && t->rchild == NULL)
        return Degree(t->lchild) + Degree(t->rchild);
}
```

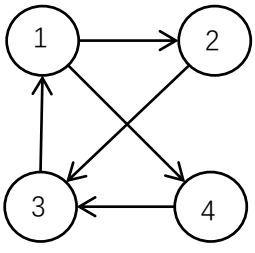
# 西北工业大学 2003-2004 学年第二学期期末考试

## 一.名词解释(16 分)

**1.**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它的根结点的值;
(2)若右子树不空，则右子树上所有结点的值均大于它的根结点的值;
(3)左、右子树也分别为二叉排序树; (4)没有键值相等的节点。
**2.**在 AOE 网中，从源点到汇点的所有路径中，具有最大路径长度的路径称为关键路径。
**3.**数据结构是相互之间存在一种或多种特定关系的数据元素的集合。
**4.**希尔排序是把记录按下标的一定增量分组，对每组使用直接插入排序算法排序。随着增量逐渐减少，每组包含的关键词越来越多，当增量减至 1 时，所有恰被分成一组，对全体记录进行一次直接插入排序。

## 二.简答题(24 分)

**1.**略　**2.**【解析】见《西北工业大学 2000-2001 学年第二学期期末考试》一.简述题 3 小题
**3.**主要是为了克服“假溢出”情况，头指针用于在队列头部读出元素，尾指针用于在队列尾部插入新元素，先进去队列的元素位于队列的头部。对于使用顺序队列情况，随着队列尾部不断插入新元素，尾指针最终会指向分配给队列的最后的内存地址，当再有新元素要插入时，此时队列尾部已经无法插入新元素了。而头指针由于有元素出队列，队列内存空间的前面一部分其实还是空的，因此就造成了“假溢出”这种情况。循环队列就是为解决该问题的。
判断队满/空：牺牲一个单元来区分(也可记录队列元素个数或增设 tag 标志)
队满：(Q.rear+1)%MaxSize=Q.front；队空：Q.front==Q.rear
**4.**不会改变。根据三种遍历的次序和特点：前序是根左右、中序是左根右、后序是左右根,因此相对次序发生变化的都是子树的根,也就是分支结点(非叶子结点)。
**5.**

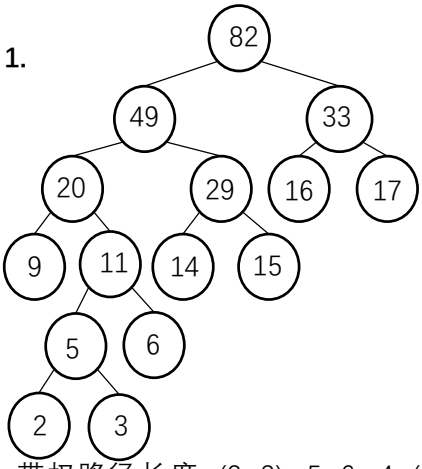



不存在拓扑序列；还需增加 7 条边。

**6.**$p(x,y,z)=x^{10}y^5z^3+3x^6y^3z^3+2x^2y^2z^2+6xyz+yz+10$
$=(x^{10}y^5+3x^6y^3)z^3+2x^2y^2z^2+(6xy+y)z+10$
=z((A,3),(B,2),(C,1),(10,0))
其中：　A=y((D,5),(E,3))　B=y((F,2))　C=y((G,1) ,(1,1))　或 C=y((G,1))
D=x((1,10))　F=x((2,2))　G=x((6,1))　G=x((6,1),(1,0))
E=x((3,6))

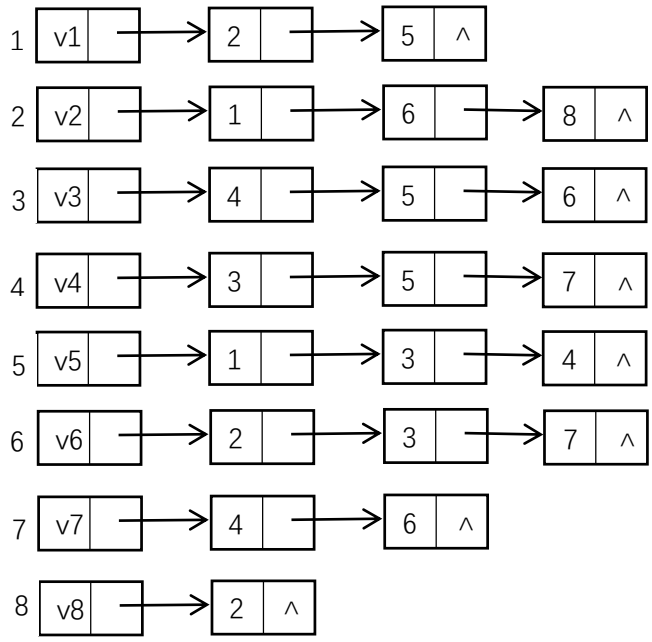
三.计算题(40 分)

**1.**

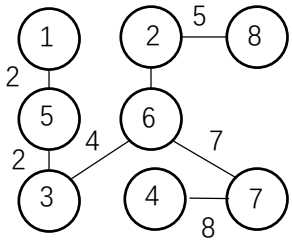


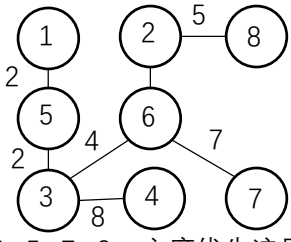

带权路径长度=(2+3)×5+6×4+(9+14+15)×3+(16+17)×2=229

**2.**(1)邻接表




(2)最小生成树有两种



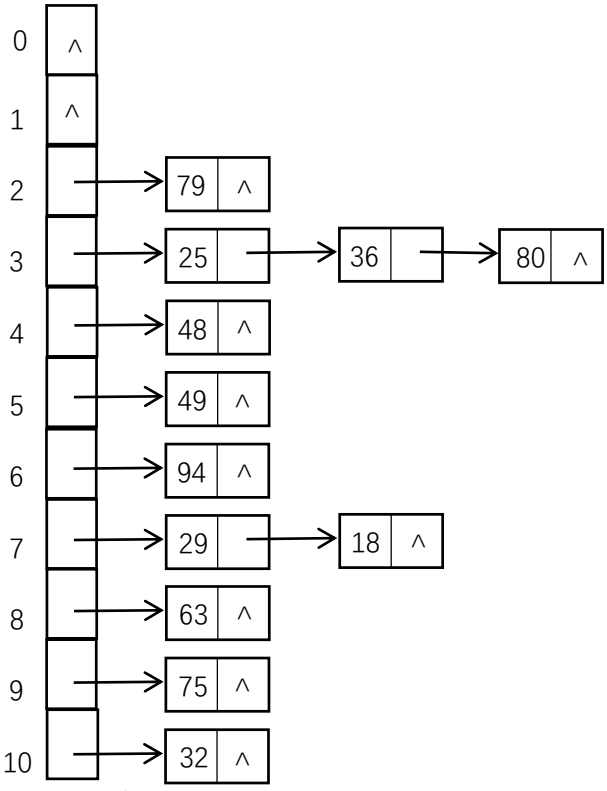




【注】下标从 0 还是 1，没有统一规定。一般字母写在框内，下标写在外边，之后的同理，不再说明

(3)深度优先遍历：v1 v2 v6 v3 v4 v5 v7 v8；广度优先遍历：v1 v2 v5 v6 v8 v3 v4 v7

(4)v1 到 v2 的最短路径：v1 v5 v3 v6 v2

**3.**




平均查找长度=(9×1+2×2+3×1)/12=1.33

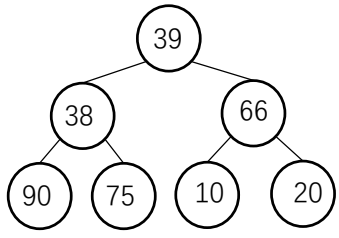
**4.**以升序为例：

(1)第一趟：{20 38 10} 39 {75 90 66};

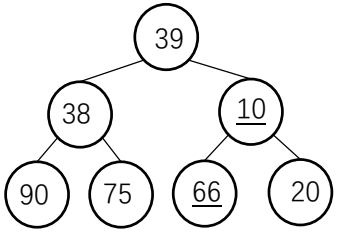
第二趟：{10} 20 {38} 39 {66} 75 {90}

【注】快速排序第一趟无疑问，将 key 放在 k 位置上，左端都小于 key，右端都大于 key；第二趟就是将 key 左右两边分别进行一次快速排序，第三趟则是将每个区间再进行一次快速排序，以此类推，直到每个区间剩一个元素。

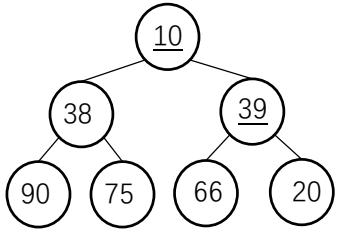
(2) 构造小顶堆：



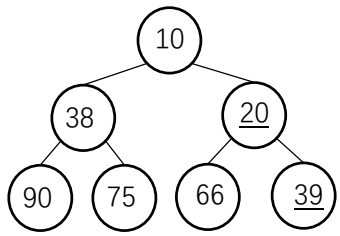

(a)



(b)交换 10 和 66



(c)交换 39 和 10



(d)交换 39 和 20 得到小顶堆

(3)堆排序。

只选前两个最大的元素，不需要所有元素整体有序，使用堆排序建立大根堆，时间复杂度为 O(n),输出堆顶元素后，向下调整一次的时间复杂度为 O(h)。

**四.算法分析题(10 分)**

功能：二叉树的中序遍历

【注】原题中一开始没有给 top 和 p 赋值，top 的作用相当于一个栈顶指针，暂且认为其初始值为-1，p=BT(指向根结点的指针)。

算法执行过程：s[i]相当于栈的作用,首先 top=-1，p=根节点指针。将 A 压入栈中，p 指向 A 的左儿子 B;再将 B 压入栈中，p 指向 B 的左儿子 C；再将 C 入栈，p 指向 C 的左儿子(空指针)。C 出栈，输出 C，p 指向 C 的右儿子(空指针)；B 出栈，输出 B，p 指向 B 的右儿子 D；将 D 压入栈，p 指向 D 的左儿子(空指针)；D 出栈，输出 D，p 指向 D 的右儿子(空指针)；A 出栈，输出 A，p 指向 A 的右儿子 E；E 入栈，p 指向 E 的左儿子(空指针)；E 出栈，输出 E，p 指向 E 的右儿子(空指针)；判断，结束。

| top | -1 | 0 | 1 | 2 | | 1 | | | 0 | | 1 | | 0 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| s[i] | | A | B | C | | | | | | | D | | | |
| p | A | B | C | NULL | C | C | NULL | B | B | D | NULL | D | D | NULL |
| 输出 | | | | | | C | | | B | | | | D | |

| top | | -1 | | 0 | | -1 | |
|---|---|---|---|---|---|---|---|
| s[i] | | | | E | | | |
| p | A | A | E | NULL | E | E | NULL |
| 输出 | | A | | | | E | |

**五. (10 分)以含有头结点的单链表为例，逐个元素遍历即可。**

```
void Delete(LNode* &L){
    LNode* p = L->next, *q, *pre;
    while(p) {
        q = p->next;
        pre = p;
        while(q){
            if(p->data == q->data){
                pre->next = q->next;
                free(q);
                q = pre->next;
            }
            Else {
            pre = q;
            q = q->next;
            }
        }
        p = p->next;
    }
}
```

# 西北工业大学 2004-2005 学年第二学期期末考试

## 一.简述题

**1.**数据结构的逻辑描述(应该是逻辑结构)：指数据元素之间的逻辑关系，即从逻辑关系上描述数据。它与数据的存储无关，是独立于计算机的，分为线性结构和非线性结构。

数据结构的物理描述(存储结构)：指数据结构在计算机中的表示，主要分为顺序存储、链接存储、索引存储和散列存储。

**2.**哈希查找：通过散列函数计算出数据元素的存储地址进行查找。

冲突：散列函数可能会把两个或两个以上的不同关键字映射到同一地址。

解决冲突的基本方法：

| | (1)开放定址法 | (2)再哈希法 | (3)链地址法 |
|---|---|---|---|
| 优点 | 计算简单 | 不易产生聚集 | 处理冲突简单，无聚集现象，平均查找长度较短 |
| 缺点 | 二次聚集(在处理同义词的冲突过程中又添加了非同义词的冲突) | 增加了计算的时间 | 链接指针占用空间 |

**3.**线索二叉树：在二叉链表表示的二叉树中存在大量的空指针，若利用这些空链域存放指向其直接前驱或后继的指针，称之为线索。加上线索的二叉树称为线索二叉树。

先序线索二叉树的直接前驱：

如果有左线索，则左线索指向其前驱；否则：

(1)若结点 x 是双亲左孩子或是双亲右孩子且双亲无左子树，则前驱是双亲；

(2)如果结点 x 是双亲右孩子且双亲有左子树，则前驱是左子树按先序遍历的最后一个结点。

先序线索二叉树的直接后继：

如果有右线索，则右线索指向其后继；否则：

(1)如果该结点有左孩子，则后继为左孩子；

(2)如果该结点无左孩子而有右孩子，则后继为其右孩子。

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

算法特征：(1)有穷性，(2)确定性，(3)可行性，(4)输入，(5)输出

方法：(1)自然语言描述，(2)流程图描述，(3)伪代码描述

**5.**拓扑序列：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，若边(u,v)∈E(G)，则 u 在线性序列中出现在 v 之前。将这样的线性序列称为拓扑序列。

拓扑排序的算法思想：

(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出；

(2)从图中删除该顶点和所有以它为起点的有向边；

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。

作用：判断一个有向图是否存在回路

**二.**采用递归的思想求解，若 T1 和 T2 都是空树，则同构；若有一个为空而另一个不空，则不同构；否则递归地比较它们的左、右子树是否同构。

```
typedef struct BiTNode{
    ElemType data;
    struct BiTNode *lchild, *rchild;
```

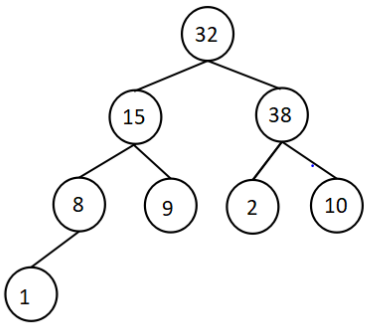
```
else if(T1 == NULL || T2 == NULL)
    return 0;
else{
```

```
}BiTNode,*BiTree;
int similar(BiTree T1, BiTree T2){
    int l, r;
    if(T1 == NULL && T2 == NULL)
        return 1;
    l = similar(T1->lchild, T2->lchild);
    r = similar(T1->rchild, T2->rchild);
    return l&&r;
}
}
```

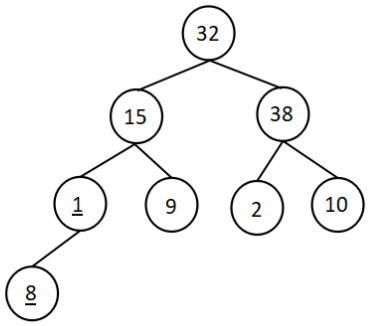
【注】本题题目为考察同构，此题答案实为判断二叉树相似算法。因课本上无同构定义，又此题出题时间较早，因此建议大家掌握判断相似的算法，同构的算法比较长，复杂，可以稍微看下。附:同构算法参考 https://blog.csdn.net/Phenixfate/article/details/48999439

三.应用题(20 分)

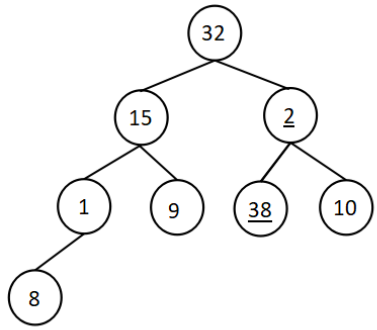
1.(题中未说明构造小顶堆或大顶堆，这里以小顶堆为例，画时可用箭头标注交换)



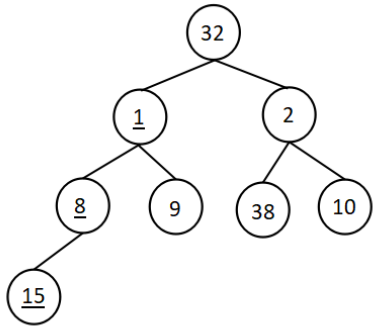

（1）初始状态



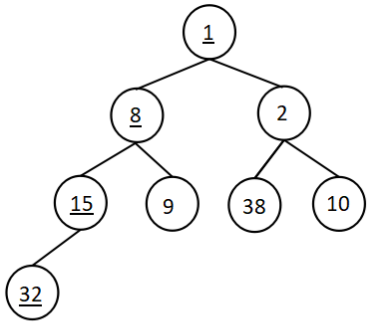

（2）8 和 1 交换



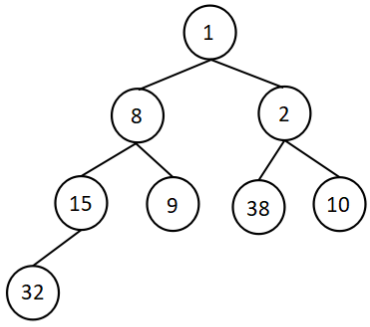

（3）38 和 2 交换




（4）15 和 1 交换，15 和 8 交换

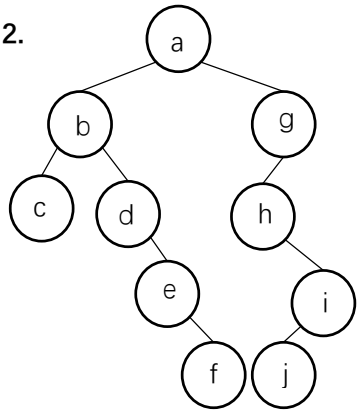



（5）32 和 1 交换，32 和 8 交换，32 和 15 交换



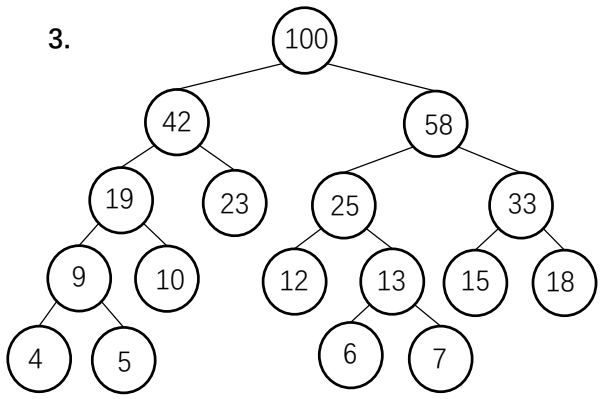

（6）初始堆构造完成

2.




后序序列：cfedbjihga

3.

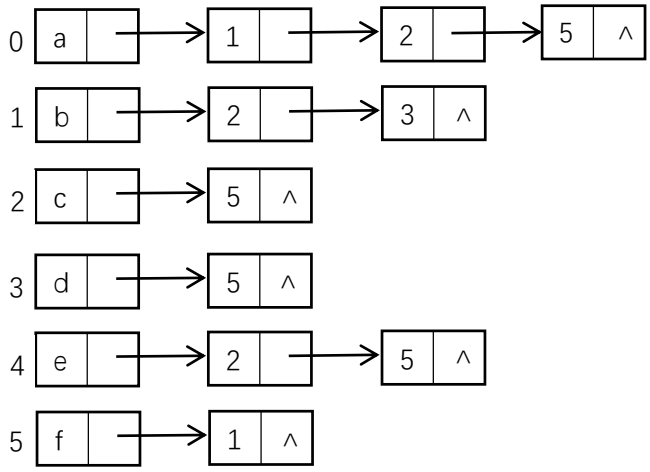



带权路径长度=(4+5+6+7)×4+(10+12+15+18)×3+23×2=299

**四．**(1)邻接矩阵：

$$\begin{bmatrix} \infty & 3 & 5 & \infty & \infty & 4 \\ \infty & \infty & 1 & 1 & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & \infty & \infty & \infty & 2 \\ \infty & \infty & 2 & \infty & \infty & 3 \\ \infty & 3 & \infty & \infty & \infty & \infty \end{bmatrix}$$

(2)邻接表




求最短路径：

| 顶点 | 第一趟 | 第二趟 | 第三趟 | 第四趟 |
|---|---|---|---|---|
| b | 3<br>a→b | | | |
| c | 5<br>a→c | 4<br>a→b→c | | |
| d | ∞ | 4<br>a→b→d | 4<br>a→b→d | |
| e | ∞ | ∞ | ∞ | ∞ |
| f | 4<br>a→f | 4<br>a→f | 4<br>a→f | 4<br>a→f |
| 集合 S | {a，b} | {a,b,c} | {a,b,c,d} | {a,b,c,d,f} |

**五.算法思想：**采用队列

(1)根结点入队列，即放入列尾；(2)从队列头部取出一个结点；

(3)将取出来的结点的左右儿子交换，然后依次放入队列尾部；

(4)如果队列不为空，循环执行第 2、3 步。

```
void   SwapNode(BTree* pTree){
    if(pTree == NULL)
        return;
    queue <BTree*>   qu;
    qu.push(pTree);
    BTree* pTree2 = NULL;
    while(!qu.empty()){
        pTree2 = qu.front();
        qu.pop();
        BTree* pTemp = pTree2-> left;
        pTree2->left = pTree2-> right;
        pTree2->right = pTemp;
        if(pTree2-> left!=null)
            qu.push(pTree2-> left);
        if(pTree2-> right!=null)
            qu.push(pTree2-> right);
    }
}
```

六.程序如下

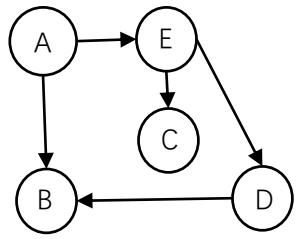
```
bool visited[vexnum];
void DFS(Graph G, int v){
    int w;
    initstack(s);
            push(s, w);
            visited[w] = true;
        }
    }
```

```
        push(s, v);
        visited[v] = true;
        while(!IsEmpty(s)){
            k = pop(s);
            visit(k);
            for(p = G.vertices[k].firstarc; p; p = p->nextarc){
                w = p->adjvex;
                if(!visited[w]) {
```

```
        }
    }
    void DFSTraverse(Graph G){
        for(i = 0;i < G.vexnum;i ++)
            visited[i] = false;
        for(i = 0;i < G.vexnum;i ++)
            if(!visited[i])
                DFS(G, i);
    }
```

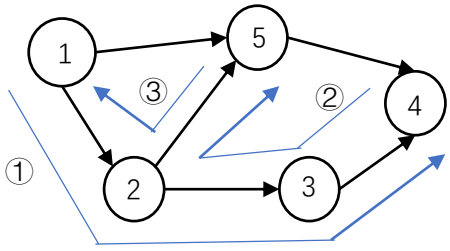
【注】还有 09-10 期末三.1、02 真题六、附录 A1 卷五 2 等题涉及到深度优先非递归程序。

上述程序与《王道》思想一致，只读取栈顶元素并不出栈，读取以后按邻接表的顺序找到它邻接点中第一个未被标记的结点，然后访问、标记、入栈，这样就可以一直往下走，直到走不动也就是当前栈顶结点的所有邻接点都已被访问或者没有邻接点的时候，才会把栈顶元素出栈，然后再去找下一个栈顶结点的第一个没访问的邻接点。这样虽然会多次访问同一个顶点的邻接表，但是避免重复入栈的问题。




按照这个思路改一句也可以实现，只是有重复入栈的问题，空间复杂度高。将 if(!visited[w]) 判定去掉，也就是不管是否已经入过栈，直接把当前访问点的所有邻接点都入栈，然后在出栈的那里加一个判定，如果出栈的结点时已经访问过的，就不再访问，直接跳过继续出栈，这样是可以实现的。也就是说相当于如果一个结点重复入栈多次，只有最后一次入栈有效，自动把前面的忽略。

对于如右图所示的深度优先遍历，按照上述算法输出序列为 AEDCB，但这不是正确的深度遍历序列。




《天勤》上关于 DFS 的非递归不会出现上述错误，我们引用《天勤》上关于这个知识点的讲解供大家参考：

本题关键在于自己写一个栈来代替系统栈，除此之外还要知道栈在 DFS 算法中到底起了什么作用。为了说明这点，可以看右图所示的例子，图中反映了在以顶点 1 为起点的 DFS 算法中，游历图中各顶点的过程，从顶点 1 开始，沿着路线①一直走到顶点 4，然后沿着路线②走到顶点 5，最后沿着路线走到③返回起点。在这个过程中，每当来到一个新顶点的时候需要做两件事：第一，对这个顶点访问（如输出顶点等操作），并且将顶点标记为已访问；第二，看看有没有可以通往下一个顶点的边（这里指的是所指的顶点没有被访问过的边），如果有则找出一条，沿着这条边走向下一个顶点，如果没有则原路返回。

如果当前被访问的顶点还有其他边可以通往其他顶点，这就说明即便是已经沿着选出的边走向另一个顶点，对于当前顶点的处理依然没有完成。如图中顶点 2，即便已经离开了顶点 2 走向了顶点 3，但是对于顶点 2 的另一个邻接点-顶点 5 还没有访问。所以必须记下当前顶点的位置，待之后的顶点处理完后再返回来处理，这就要用到栈。

**当问题执行到一个状态，以现有的条件无法完全解决时，必须先记下当前状态，然后继续往下进行，等条件成熟后在返回来解决**，这一类问题可以用栈来解决。以下代码供大家参考

```
void DFS1(AGraph *g,int v){
    ArcNode *p;
    int stack[maxSize],top=-1;          //定义一个栈来记录访问过程中的顶点
    int i,k;
    int visit[maxSize];                 //顶点访问标记数组
    for(i=0;;i<g->n;++i)
        visit[i]=0;
    visti(v);                           //假设此函数已经声明，包含对顶点访问的各种操作
```

```
    visit[v]=1;                    //标记起始顶点已被访问
    stack[++top]=v;                //起始顶点入栈
    /*以下是本算法的关键部分*/
    while(top!=-1){
        k=stack[top];              //取栈顶元素
        p=g->adjlist[k].firstarc;  //p 指向该顶点的第一条边
        /*下面这个循环是 p 沿着边行走并将图中经过的顶点入栈的过程*/
        while(p!=NULL&&visit[p->adjvex]==1)
            p=p->nextarc;          //找到当前顶点第一个没访问过的邻接顶点或者 p 走到当前链表尾部
时，while 循环停止
        if(p==NULL) //如果 p 到达当前链表尾部，则说明当前顶点的所有点都访问完毕，当前顶点出栈
            --top;
        else{                      //否则访问当前邻接点并入栈
            visit(p->adjvex);
            visit[p->adjvex]=1;
            stack[++top]=p->adjvex;
        }
    }
}
```

# 西北工业大学 2006-2007 学年第一学期期末考试

## 一.简述题

**1.**数据结构的物理描述(即存储结构或物理结构)指数据结构在计算机中的表示，它包括数据元素的表示和关系的表示。

【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 4 小题

**2.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 2 小题

**3.**线索二叉树：在二叉链表表示的二叉树中存在大量的空指针，若利用这些空链域存放指向其直接前驱或后继的指针，称之为线索。加上线索的二叉树称为线索二叉树。

后序线索二叉树直接前驱：

如果有左线索，则左线索指向其前驱；否则：

(1)若结点 x 有右孩子，则前驱为其右孩子；

(2)若结点 x 有左孩子而无右孩子，则前驱为其左孩子。

后序线索二叉树直接后继：

如果有右线索，则右线索指向其后继；否则：

(1)若结点 x 为根，则无后继；

(2)若结点 x 为其双亲的右孩子或为其双亲的左孩子且双亲无右子女，则其后继为其双亲；

(3)若结点 x 为其双亲的左孩子，且双亲有右子女，则后继是其双亲的右子树中按后序遍历的第一个结点。

**4.**【注】参考《西北工业大学 2003-2004 学年第二学期期末考试》二简答题 3 小题

**5.**【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 2 小题

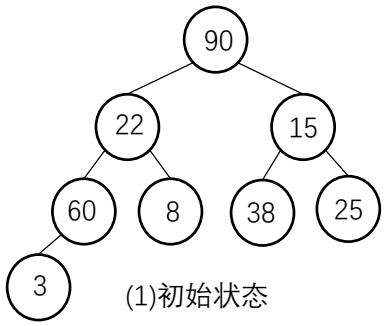
二.请求解决下面问题

**1.** (15 分)堆的定义：n 个关键字序列 $L[1...n]$ 称为堆，当且仅当该序列满足：

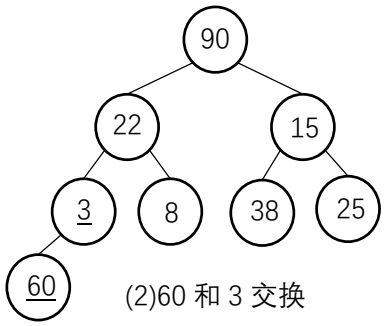
① $L(i) \le L(2i)$ 且 $L(i) \le L(2i+1)$ 或② $L(i) \ge L(2i)$ 且 $L(i) \ge L(2i+1)$ $(1 \le i \le \lfloor n/2 \rfloor)$

满足第①种情况的堆称为小顶堆，满足第②种情况的堆称为大顶堆，

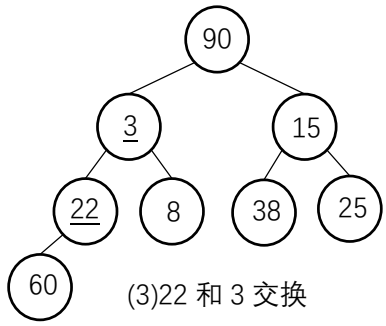
(1)构造初始小顶堆



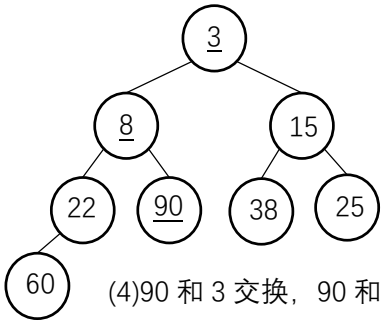

(1)初始状态



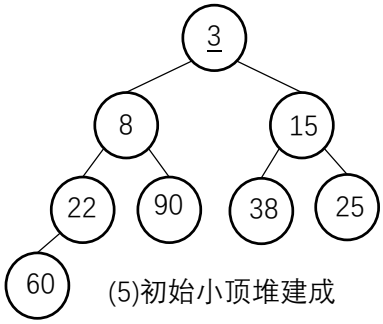

(2)60 和 3 交换




(3)22 和 3 交换




(4)90 和 3 交换，90 和 8 交换




(5)初始小顶堆建成

(2)快速排序

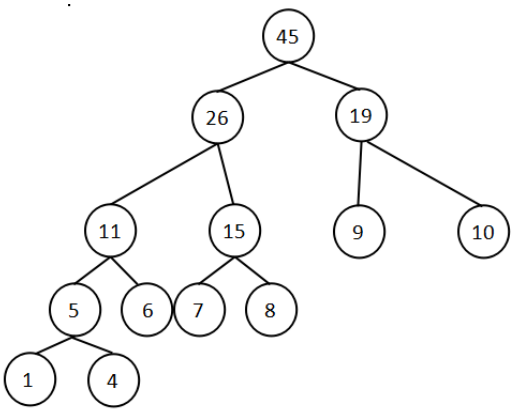
第一趟：{3 22 15 60 8 38 25} 90　　第二趟：3 {22 15 60 8 38 25} 90

第三趟：3 {8 15} 22 {60 38 25} 90　第四趟：3 8 {15} 22 {25 38} 60 90

第五趟：3 8 15 22 25 {38} 60 90

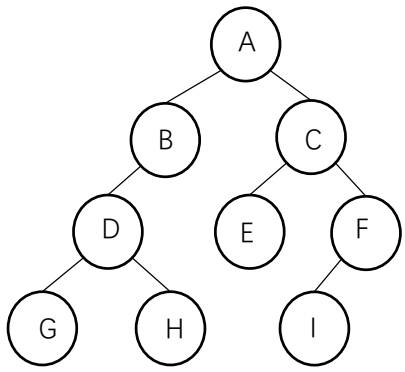
【注】有同学问为什么最后两趟一样？答：这是为了让最后 ij 指针碰到一块

**2.** (10 分)构造哈夫曼树：



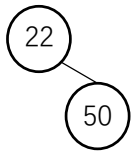

带权路径长度=(1+4)×4+(6+7+8)×3+(9+10)×2=121

**3.** (10 分)后序序列：GHDBEIFCA

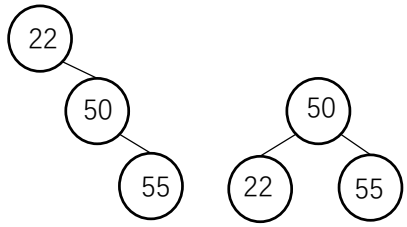


**4.** (10 分)22 50 55 70 35 25 66 15 10

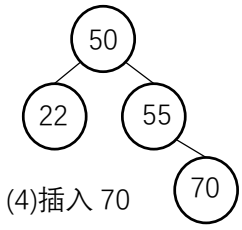

(1)插入 22



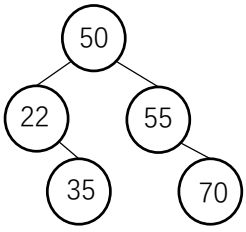

(2)插入 50



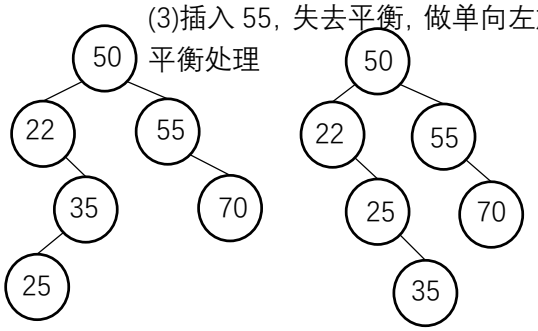

(3)插入 55，失去平衡，做单向左旋平衡处理

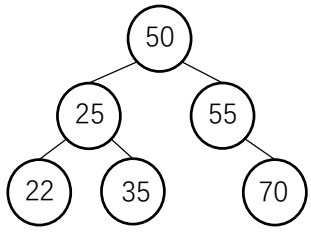



(4)插入 70

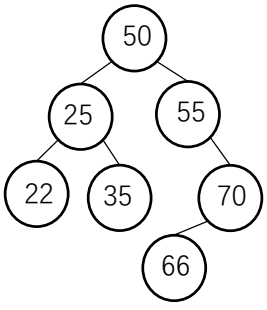


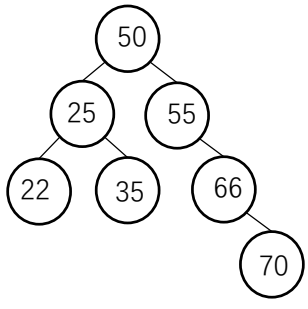

(5)插入 35

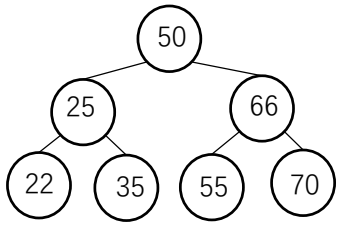


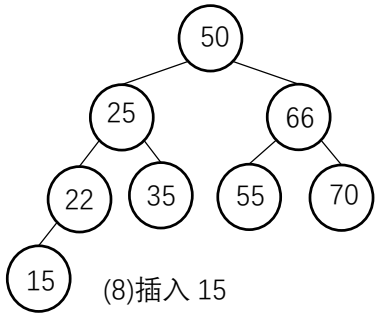




(6)插入 25，失去平衡，做先右后左双向平衡处理



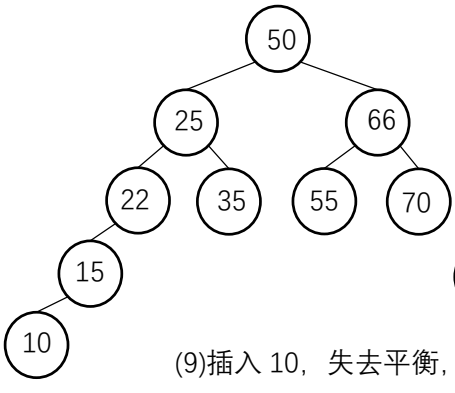



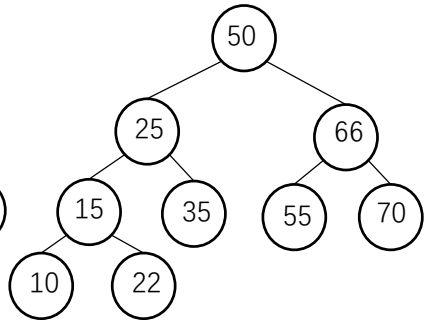



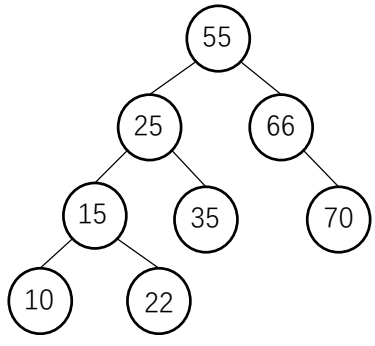

(8)插入 15

(7)插入 66，失去平衡，做先右后左双向平衡处理




(9)插入 10，失去平衡，做单向右旋平衡处理




删除键值为 50 的元素，将结点 50 与结点 55 交换，再删除。

三．(15 分)拓扑排序的算法思想:

(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出;

(2)从图中删除该顶点和所有以它为起点的有向边;

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。所用数据结构：邻接矩阵或邻接表，栈或队列

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈，存储入度为 0 的顶点
    for(i = 0; i < G.vexnum;i ++)
        v = p->adjvex;
        if(!(--indegree[v]))
            Push(S, v);
```

```
        if(indegree[i] == 0)
            Push(S, i);
    int count = 0; //记录当前已经输出的顶点数
    while(!IsEmpty(S)){
        Pop(S, i);
        print[count ++] = i;
        for(p=G.vertices[i].firstarc;p; p=p->nextarc){
            }
        }
        if(count < G.vexnum)
            return false;
        else
            return true;
    }
```

**四．**(10 分)算法思想：两个链表 HA 和 HB 已经按元素值递增次序排序，将其合并时，均从第一个结点起进行比较，将小的结点链入链表中，同时后移工作指针。要求结果链表按元素递减次序排列，故新链表的建立应采用头插法。比较结束后，可能会有一个链表非空，此时用头插法将剩下的结点依次插入新链表即可。

```
void MergeList(LNode* &HA, LNode* &HB){
    LNode * r, *pa=HA->next, *pb=HB->next;
    HA->next = NULL; //新链表
    while(pa != NULL && pb != NULL){
        if(pa->data <= pb->data){
            r = pa->next;
            pa->next = HA->next;
            HA->next = pa;
            pa = r;
        }
        else{
            r = pb->next;
            pb->next = HA->next;
            HA->next = pb;
            pb = r;
        }
    }
    if(pa != NULL)
        pb = pa;
    while(pb != NULL){
        r = pb->next;
        pb->next = HA->next;
        HA->next = pb;
        pb = r;
    }
    free(HB);
}
```

# 西北工业大学 2009-2010 学年第一学期期末考试

一.简述题(30 分，每小题 10 分)

**1.**连通图的生成树是包含图中全部顶点的一个极小连通子图。存在条件：连通图。无向图中的极大连通子图称为连通分量。

【补充】生成树存在的条件：图是连通的。连通分量不需要存在条件

**2.**拓扑序列：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，若边(u,v)∈E(G)，则 u 在线性序列中出现在 v 之前。将这样的线性序列称为拓扑序列。拓扑排序的算法思想：

(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出；

(2)从图中删除该顶点和所有以它为起点的有向边；

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。

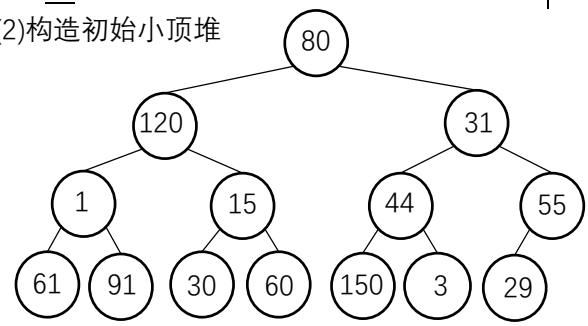
**3.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 3 小题

二.请求解以下问题(40 分，每小题 10 分)

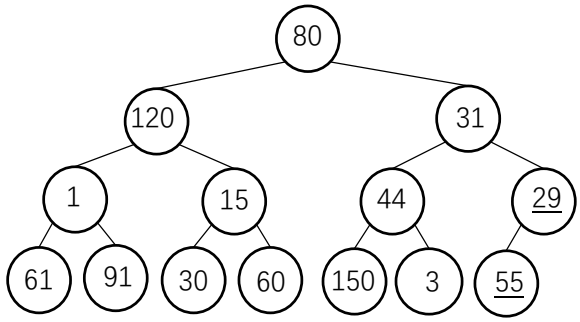
**1.**(1)一趟快速排序

| | |
|---|---|
| 80 120 31 1 15 44 55 61 91 30 60 150 3 29 | 29 3 31 1 15 44 55 61 91 30 60 150 80 120 |
| 29 120 31 1 15 44 55 61 91 30 60 150 3 80 | 29 3 31 1 15 44 55 61 80 30 60 150 91 120 |
| 29 80 31 1 15 44 55 61 91 30 60 150 3 120 | 29 3 31 1 15 44 55 61 60 30 80 150 91 120 |

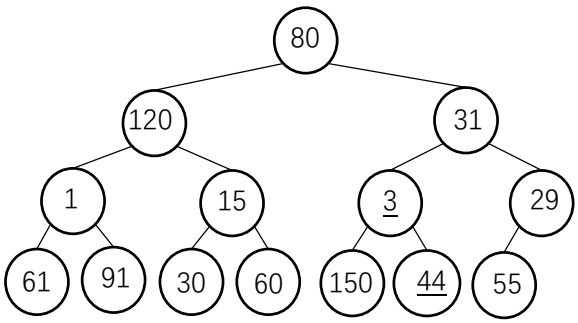
(2)构造初始小顶堆



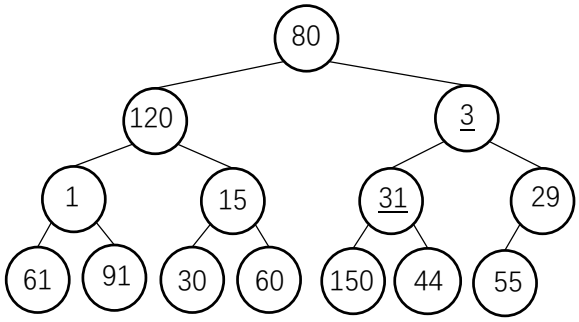

(1)初始状态



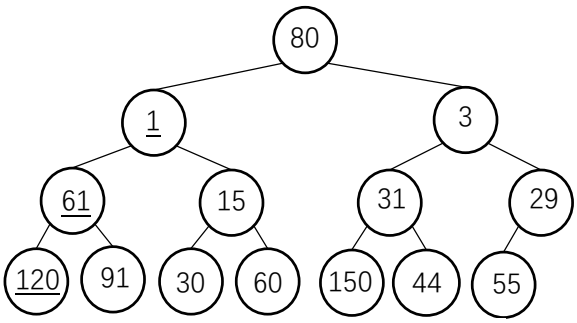

(2)交换 55 和 29



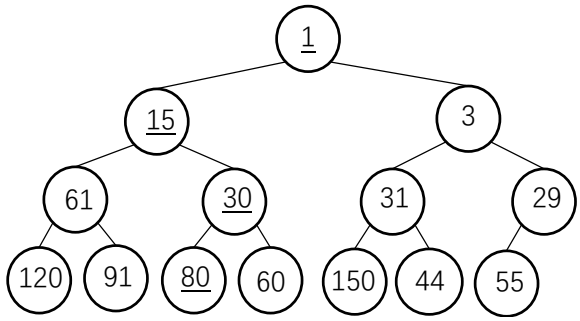

(3)交换 44 和 3




(4)交换 31 和 3

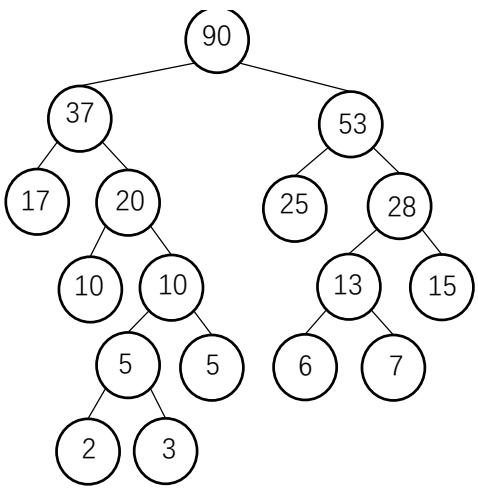



(5)120 和 1 交换，120 和 61 交换

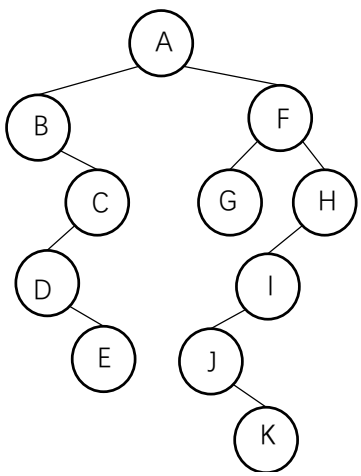



(6)80 和 1 交换，80 和 15 交换，80 和 30 交换

**2.**




**3.**

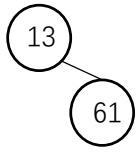




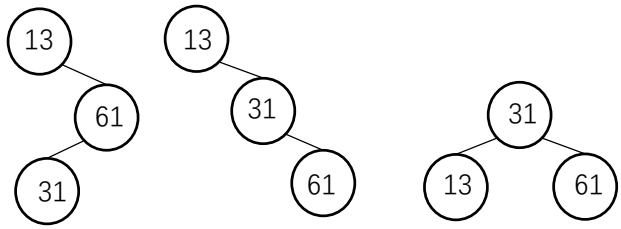

后序序列：EDCBGKJIHFA

**4.**13,61,31,15,38，10,58,91，20,9,50,3

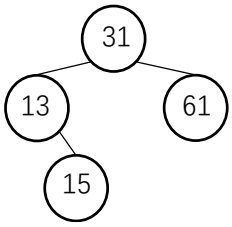

(1)插入 13



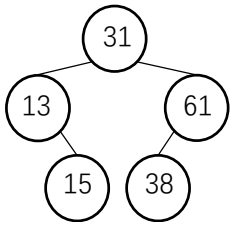

(2)插入 61



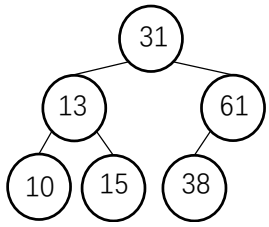

(3)插入 31，失去平衡，做先右后左双向旋



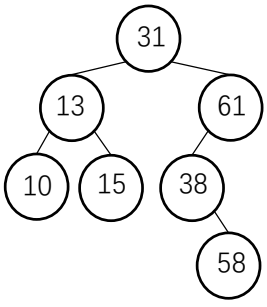

(4)插入 15

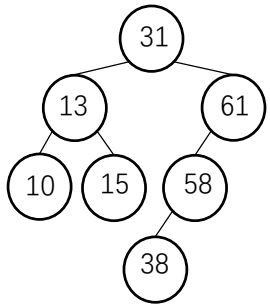


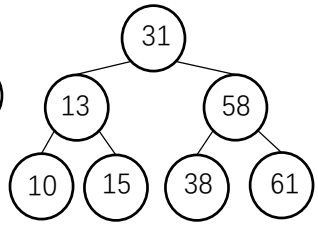

(5)插入 38

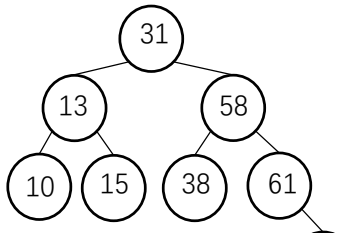



(6)插入 10



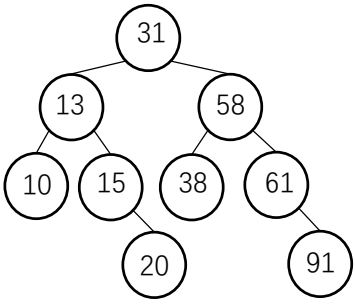



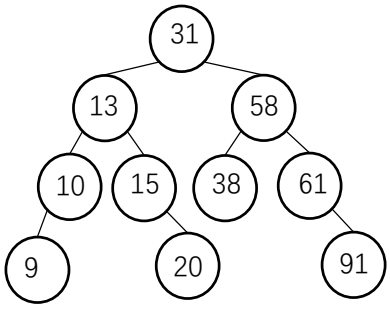




(8)插入 91

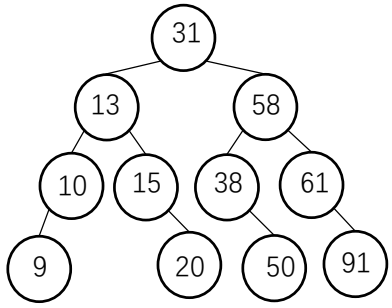

(7)插入 58，失去平衡，做先左后右双向旋转



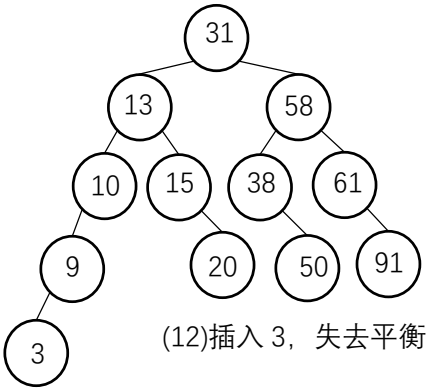

(9)插入 20

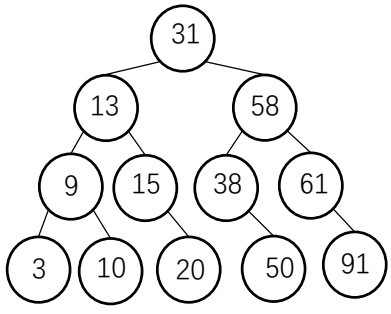



(10)插入 9




(11)插入 50




(12)插入 3，失去平衡，做单向右旋平衡处理

三.算法描述(20 分)

**1.**所用中间数据结构：栈

```
bool visited[vexnum];
```

```
visited[w] = true;
```

```
void DFS(Graph G, int v){
    int w;
    initstack(s);
    Push(s, v);
    visited[v] = true;
    while(!IsEmpty(s)){
        k = Pop(s);
        visit(k);
        for(w = 0; w < G.vexnum; w ++){
            if(G.Edge[k][w] == 1 && !visited[w]) {
                Push(s, w);
            }
        }
    }
}
void DFSTraverse(Graph G){
    for(i = 0;i < G.vexnum;i ++)
        visited[i] = false;
    for(i = 0;i < G.vexnum;i ++)
        if(!visited[i])
            DFS(G, v);
}
```

**2.**所用中间数据结构：队列

根据完全二叉树的定义，具有 n 个结点的完全二叉树与满二叉树中编号 1 到 n 的结点一一对应。算法思想：采用层次遍历的算法，将所有结点加入队列(包括空结点)。当遇到空结点时，查看其后是否有非空结点。若有，则不是完全二叉树。

```
bool IsComplete(BiTree T){
    InitQueue(Q);
    if(!T)
        return 1;
    EnQueue(Q, T);
    while(!IsEmpty(Q)){
        DeQueue(Q, p);
        if(p){
            EnQueue(Q, p->lchild);
            EnQueue(Q, p->rchild);
        }
        else{
            while(!IsEmpty(Q)) {
                DeQueue(Q, p);
                if(p)
                    return 0;
            }
        }
    }
    return 1;
}
```

**四．**题目中未明确告诉，我们认为 H1 和 H2 所存储的长整数是从低位到高位逆序存储的，这样有利于后边计算。将两个链表中每个结点的数据逐个相加(注意进位)。

```
LNode *Add(LNode* H1, LNode* H2){
    LNode* p1 = H1->next, *p2 = H2->next, *p = NULL;
    LNode* H3 = (LNode*)malloc(sizeof(LNode));
    LNode *r = H3, *s;
    int mod = 0,sum = 0;
    while(p1 && p2) {
        sum = p1->data + p2->data + mod;
        mod = sum/10;     //进位
        sum = sum%10;
        s = (LNode*)malloc(sizeof(LNode));
        s->data = sum;
        r->next = s;
        r = s;
        p1=p1->next;
        p2=p2->next;
    }
    while(p) {
        sum = p->data + mod;
        mod = sum/10;
        sum = sum%10;
        s = (LNode*)malloc(sizeof(LNode));
        s->data = sum;
        r->next = s;
        r = s;
        p = p->next;
    }
    if(mod != 0) {
        s = (LNode*)malloc(sizeof(LNode));
        s->data = mod;
        r->next = s;
        r = s;
    }
```

```
    if(p1)
        p = p1;
    else if(p2)
        p = p2;
    r->next = NULL;
    return H3;
}
```

# 西北工业大学 2010-2011 学年第一学期期中考试

一.选择题(每小题 2 分，共 20 分)

**答案速查：CBDBC　DACDA**

**2.**B【解析】(0+1+…+125)/126=62.5

**3.**D【注】head 和 tail 是取广义表表首或表尾的函数，建议好好理解《王道》上的概念。有同学提议可在 B 站搜索青岛大学王卓老师讲的数据结构来加深理解。

**9.**D【解析】A[0..9,0...19]这个写法不常见，相当于 A[10][20]，因为是列优先，所以表示的是 10 列 20 行，每行元素有 10 个，因此 A[i][j]=A[6][6]=A[0][0]+(i*n+j)*d=100+(10*6+6)*2=232。

【补充】二维数组 A[0..9,0...10]采用行优先的存储方法，若每个元素各占 3 个存储单元且 A[0,0]的地址为 200，则 A[6,9]地址为_____

【解析】由题可知二维数组 A[m][n]即 A[10][11]，d=3，采用行优先存储算法，地址 A[i][j]=A[6][9]=A[0][0]+(i*n+j)*d=200+(6*11+9)*3=200+225=425。

二.填空题(每小题 2 分，共 20 分)

**1.**n(n+1)/2　**2.** S->link = H; H= S;　**3.**$2n_0-1$

**4.**front->link->link==NULL; 或 front->link->link==rear;　**5.**O(1);O(n)

**6.**(n+1)/2;(n-1)/2　**7.**66【解析】每个元素要用行号,列号,元素值来表示,在用三元组表示稀疏矩阵，还要三个成员来记住矩阵的行数、列数、总的元素数,所以所需的字节数是 10*（1+1+1）*2+3*2=66 ，故 11*3*2=66

**8.**中序　**9.**8　**10.**56

三.判断题(每题 1 分，共 10 分)

**1.** ×　**2.** ×　**3.** ×　**4.** √　**5.** ×

**6.** √【解析】三元组中的元素个数就等于矩阵中非零元素个数，有些同学说第 0 行存储非零元素以及行列信息，所以认为错，但这些矩阵信息不应该计算在内，请看例题 1、2。

【例题 1】对稀疏矩阵进行压缩存储，可采用三元组表，一个 10 行 8 列的稀疏矩阵 A 共有 73 个零元素，其对应的三元组表共有（　）个元素。

A.8　B.80　C.7　D.10

【解析 1】10 行 8 列共有 80 个元素，其中 73 个零元素，则有 7 个非零元素，因此三元组表有 7 个元素，选 C。

【例题 2】对稀疏矩阵进行压缩存储，可采用三元组表，一个 10 行 8 列的稀疏矩阵 A，其对应的三元组表共有 6 个元素，矩阵 A 共有（　）个零元素。

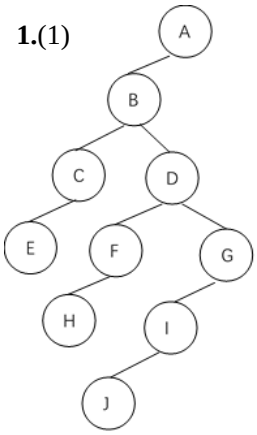
A.8　B.72　C.74　D.10

【解析 2】10 行 8 列共有 80 个元素，其中 6 个非零元素，则有 74 个非零元素，因此选 C。

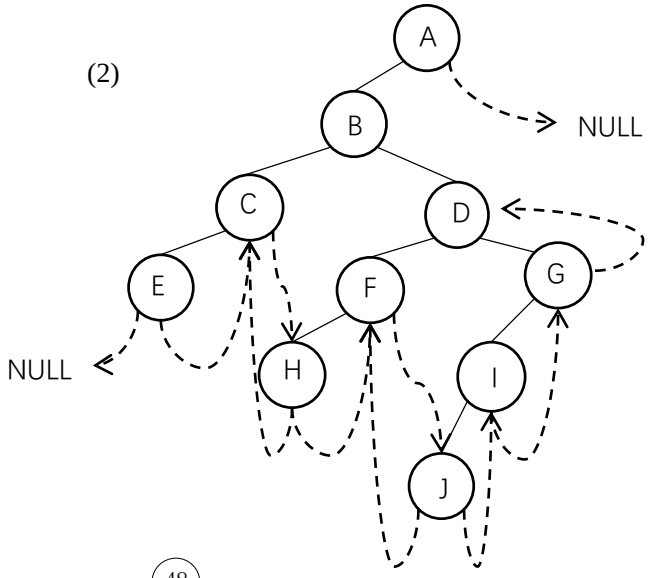
**7.** √ **8.** √ **9.** √【解析】只有一个根节点的话，那他既是根节点也是叶子结点，没有孩子的就是叶子节点，这个需要注意。 **10.** √

## 四.简答题(每题 5 分，共 20 分)

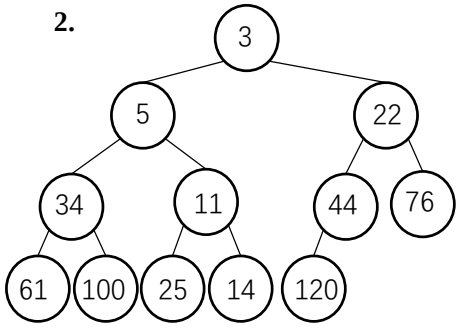
**1.**(1)



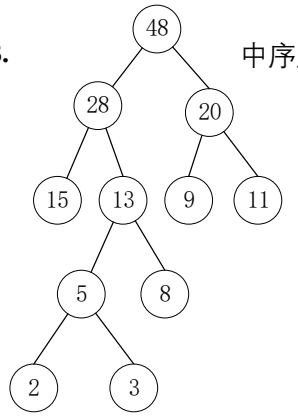

(2)




**2.**




**3.**




中序序列为：ECHFJIGDBA

**4.**XSXXSSXXSXXSXXSSSS

## 五.算法设计题(每题 10 分，共 30 分)

**1.**

```
LinkNode * tmp, * pre;
tmp = L-> link;
pre =L;
while (tmp->link != NULL && tmp->link != p){
      pre = tmp;
      tmp = tmp -> link;
}
```

```
if (tmp->link == NULL)
      return;
else {
      pre->link =p;
      delete tmp;
}
```

**2.**A: i<j; B: A[i] % 2 C: !(A[j] % 2) D: A[i] +=A[j]; A[j]= A[i]- A[j]; A[i]= A[i]- A[j];

**3.**【解析】出现设计算法的题目，按照 408 的三段式来完整的答题可以保证不出错，即设计算法、代码及算法思想。这样可以保证不丢分。

算法思路：题目要求输出叶子结点及高度，可采用递归的方法。当左右孩子都为空时，即是叶子结点。否则，继续向下遍历。(叶子结点的 data 可以是任意类型，这里以字符为例）调用： solve(T, 0);

```
void solve(BiTNode* T, int h){
```

```
if(T->lchild != NULL)
```

```
    if(T == NULL)
        return;
    if(T->lchild == NULL && T->rchild == NULL){
        printf("叶子结点：%c   路径长度：%d\n",T->data, h);
     }
    else{
                solve(T->lchild, h+1);
            if(T->rchild != NULL)
                solve(T->rchild, h+1);
        }
        return;
}
```

# 历年真题部分

# 西北工业大学 1998 年研究生入学考试

一.简答(20 分)

**1.**【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 1 小题

**2.**【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 2 小题

**3.**二叉树是树的一种，其特点是每个结点至多有两颗子树，并且子树有左右之分，其次序不能颠倒。

二叉树和度为 2 的有序树的区别:

(1)度为 2 的树至少有 3 个结点，而二叉树可以为空;

(2)度为 2 的有序树的孩子结点的左右次序是相对于另一孩子结点而言的,如果某个结点只有一个孩子结点,这个孩子结点就无须区分左右次序，而二叉树无论其孩子数是否为 2，均需确定其左右次序。

**4.**后序线索二叉树直接后继:

如果有右线索，则右线索指向其后继；否则:

(1)若结点 x 为根，则无后继;

(2)若结点 x 为其双亲的右孩子或为其双亲的左孩子且双亲无右子女，则其后继为其双亲;

(3)若结点 x 为其双亲的左孩子，且双亲有右子女，则后继是其双亲的右子树中按后序遍历的第一个结点。

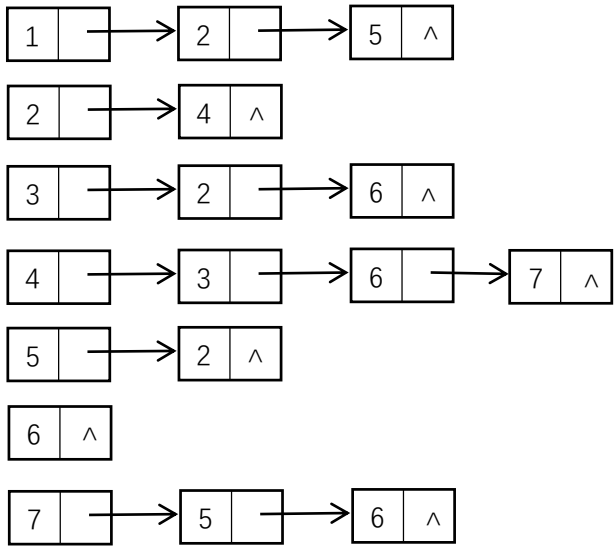
中序线索二叉树直接前驱:

如果有左线索，则左线索指向其前驱；否则中序遍历左子树时最后访问的一个结点(左子树中最右下的结点)为其前驱。

**5.**(1)邻接矩阵

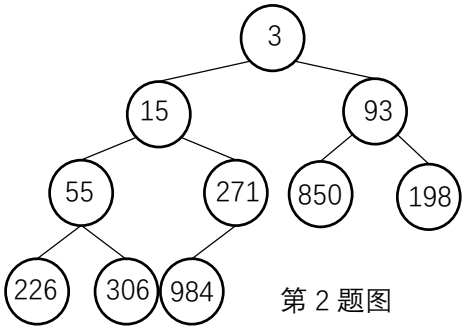
$$\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

(2)邻接表



## 二.算法应用(20 分)

**1.**略 **2.**(1)初始堆(以小顶堆为例):




第 2 题图

排序结果：3 15 55 93 198 226 271 306 850 884

(2)求解关键路径

| | ve(i) | vl(i) |
|---|---|---|
| v1 | 0 | 0 |
| v2 | 4 | 9 |
| v3 | 14 | 22 |
| v4 | 8 | 8 |
| v5 | 20 | 20 |
| v6 | 16 | 16 |
| v7 | 30 | 30 |
| v8 | 45 | 45 |

| | e(i) | l(i) | l-e |
|---|---|---|---|
| a1 | 0 | 5 | 5 |
| a2 | 4 | 12 | 8 |
| a3 | 0 | 0 | 0 |
| a4 | 4 | 9 | 5 |
| a5 | 8 | 8 | 0 |
| a6 | 8 | 8 | 0 |
| a7 | 14 | 22 | 8 |
| a8 | 20 | 20 | 0 |
| a9 | 16 | 16 | 0 |
| a10 | 30 | 30 | 0 |

A.关键路径：(v1，v4，v5，v7，v8)或(v1，v4，v6，v7，v8)

B.加快那些包括在所有关键路径上的关键活动才能缩短工期，如 a3，a10。

C.邻接表

## 三.证明(10 分)

**1.**在二叉树中，结点总数 $n=n_0+n_1+n2$；除根结点外，其余结点都有一个分支进入，设 b 为分支总数，则 $n=b+1$。由于这些分支是由度为 1 或 2 的结点发出的，故有 $b=n_1+2*n_2$。于是有 $n_0+n_1+n_2=n_1+2*n_2+1$,所以 $n_2=n_0-1$，又因为 $n_1=0$，所以 $b=2*n_2=2*(n_0-1)$。

**2.**设 m 行 n 列稀疏矩阵中非零元素个数为 t，当 $3*(t+1)<m*n$ 时，利用三元组存储可节省存储空间。

## 四.算法思想(20 分)

1. 【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》二算法思想题 1 小题
2. 【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》二算法思想题 3 小题

## 五.算法实现(30 分)

1.算法思想：找到表达式中最后使用的操作符，作为二叉树根节点，左侧右侧递归生成的二叉树分别为左孩子和右孩子。

```
typedef struct BtreeNode{//二叉树结点结构体定义
    char data;
    struct BtreeNode *lchild;
    struct BtreeNode *rchild;
}BtreeNode;
/*  afa 为指向表达式字符串的指针
    s 为要转化的表达式字符串的起始位置
    e  为要转化的表达式字符串的结束位置的后一个  */
BtreeNode* afaToBtree(char *afa,int s,int e){
    //如果只有一个数那就是叶子结点了
    if(e-s==1){
        BtreeNode* bn=(struct BtreeNode*)malloc(sizeof(struct BtreeNode));
        bn->data=afa[s];
        bn->lchild=NULL;
        bn->rchild=NULL;
        return bn;
    }
    /*local_r 记录当前要转化的表达式生成二叉树的根节点操作符的位置;
    flag 记录是否当前搜索在括号里面;
    pos 记录当前表达式中括号外面最右边的^位置;
    m_m_p 记录当前表达式中括号外面最右边的*、/位置;
    a_s_p 记录当前表达式中括号外面最右边的+、-位置*/
    int local_r=0,flag=0;
    int m_m_p=0,a_s_p=0,pos=0;
    for(int i=s;i<e;i++){
        if(afa[i]=='(') flag++;
        else if(afa[i]==')') flag--;
        if(flag==0){
            if(afa[i]=='*'||afa[i]=='/')
                m_m_p=i;
            else if(afa[i]=='+'||afa[i]=='-')
                a_s_p=i;
            else if(afa[i]=='^')
                pos=i;
        }
    }
    if((m_m_p==0)&&(a_s_p==0) &&(pos==0))
    //如果式子整个有括号如(5-2*3+7)，即括号外面没有操作符，则去掉括号找二叉树
        afaToBtree(afa,s+1,e-1);
    else{ //如果有+或者-，则根节点为最右边的+或-，否则是最右边的*或/
        if(a_s_p>0)    local_r=a_s_p;
        else if(m_m_p>0)    local_r=m_m_p;
        else if(pos>0)   local_r=pos;
        //确定根节点和根节点的左孩子和右孩子
        BtreeNode* b=(struct BtreeNode*)malloc(sizeof(struct BtreeNode));
        b->data=afa[local_r];
        b->lchild=afaToBtree(afa,s,local_r);
        b->rchild=afaToBtree(afa,local_r+1,e);
        return b;
    }
}
```

2.略

# 西北工业大学 1999 年研究生入学考试

一.概念解释(15 分)

**1.**广义表是一种非线性的数据结构，是线性表的一种推广。即广义表中放松对表元素的原子限制，容许它们具有其自身结构。

**2.**在二叉树中，某结点的左子树与右子树的高度(深度)差即为该结点的平衡因子

**3.**平均查找长度是所有查找过程中进行关键字的比较次数的平均值。其数学定义为： $ASL = \sum_{i=1}^{n} P_i C_i$，其中$\gamma_i$是查找表的长度，$P_i$是查找第 i 个数据元素的概率，$C_i$是找到第 i 个数据元素所需进行的比较次数。

**4.**(找不到伙伴空间的解释，这里对伙伴系统作一解释)伙伴系统是操作系统中一种动态存储管理方法，在用户

提出申请时，分配一块大小“恰当”的内存区给用户；反之，在用户释放内存区时即回收。无论是占用块或空闲块，其大小均为 2 的 k 次幂(k 为某个正整数)。

**5.**AOE 网中，从源点到汇点的所有路径中，具有最大路径长度的路径称为关键路径。

**二. (8 分)**

| 排序算法 | 基本思想 | 时间复杂度 | 稳定性 |
|---|---|---|---|
| 直接插入排序 | 每次将一个待排序的记录，按其关键字大小插入到前面已经排好序的子序列中，直到全部记录插入完成。 | $O(n^2)$ | 稳定 |
| 简单选择排序 | 假设排序表为 L[1…n],第 i 趟排序即从 L[i…n]中选择关键字最小(大)的元素与 L(i)交换，经过 n-1 趟排序可使排序表有序。 | $O(n^2)$ | 不稳定 |
| 2-路归并排序 | 假设待排序表含有 n 个记录，则可以看成 n 个有序的长度为 1 的子表，然后两两归并，得到$\lceil n/2 \rceil$个长度为 2 或 1 的有序表；再两两归并……；如此重复，直到合并成一个长度为 n 的有序表。 | $O(n\log_2 n)$ | 稳定 |

**三. (8 分)**队满：(Q.rear+1)%MaxSize=Q.front;　队空：Q.front==Q.rear

牺牲一个单元来区分队空和队满，入队时少用一个队列单元。

改进：增设表示元素个数的数据成员。队空条件：Q.size==0；队满条件：Q.size==MaxSize。

**四.(10 分)** (1)顺序文件只能顺序查找,优点是批量检索速度快,不适于单个记录的检索。顺 序文件不能像顺序表那样插入、删除和修改,因文件中的记录不能象向量空间中的元素那样“移动”,只能通过复制整个文件实现上述操作。

(2)索引非顺序文件适合随机存取,不适合顺序存取,因主关键字未排序,若顺序存取会引起磁头频繁移动。索引顺序文件是最常用的文件组织,因主文件有序，既可顺序存取也可随机存取。索引非顺序文件是稠密索引,可以“预查找”,索引顺序文件是稀疏索引,不能“预查找”,但由于索引占空间较少,管理要求低,提高了索引的查找速度。

(3)散列文件也称直接存取文件,根据关键字的哈希函数值和处理冲突的方法，将记录散列到外存上。这种文件组织只适用于像磁盘那样的直接存取设备,其优点是文件随机存放,记录不必排序,插入、删除方便,存取速度快,无需索引区 ，节省存储空间。缺点是散列文件不能顺序存取,且只限于简单查询。经多次插入、删除后,文件结构不合理,需重组文件,这很费时。

**五.(10 分)** (1) $\mathbf{K^{h-1}}$ (h 为层数)

(2)因为该树上每层均有 $\mathbf{K^{h-1}}$ 个结点，从根开始编号为 1，则结点 i 的从右向左数第 2 个孩子的结点编号为 **K*i**。设 n 为结点 i 的子女，则关系式**(i-1)*K+2≤n≤i*K+1** 成立，因 i 是整数，故结点 n 的双亲 i 的编号为 $\left\lfloor \frac{n-2}{K} \right\rfloor + 1$(或$\left\lceil \frac{n-1}{K} \right\rceil$)。

(3)结点(n>1)的前一结点编号为 n-1(其最右边子女编号是**(n-1)*K+1**)，故结点 n 的第 i 个孩子的编号是 **(n-1)*K+1+i**。

(4)结点 n 有右兄弟的条件是：它不是双亲从右边的第一个子女，即**(n-1)%K!=0**，其右兄弟的编号为 n+1。

**六.**略 **七.(10 分)**图的存储结构主要有邻接矩阵、邻接表、十字链表(有向图)和邻接多重表(无向图)。图的 BFS 和 DFS 遍历，采用邻接矩阵时间复杂度为 **O(|V|$^2$)**，邻接表为 **O(|V|+|E|)**。图的拓扑排序，采用邻接表的时间复杂度为 **O(|V|+|E|)**。

借助于邻接矩阵容易判断任意两个顶点是否有边/弧相连，并容易求出顶点的度。对于无向图，顶点 $\mathbf{V_i}$ 的度为矩阵第 i 行或第 i 列的不为零的元素个数之和；对于有向图，顶点 $\mathbf{V_i}$ 的出度为矩阵第 i 行的不为零元素的个数之和，顶点 $\mathbf{V_i}$ 的入度为矩阵第 i 列的不为零元素的个数之和。

在无向图的邻接表中，第 i 个链表中表结点个数为 $\mathbf{V_i}$ 的度，而在有向图中，第 i 个链表中表结点个数为 $\mathbf{V_i}$ 的出度。

利用十字链表容易求得入度和出度。

邻接多重表适合于对边进行操作。

**八.(15 分)(原版真题中后序序列缺少一个字母)**逻辑结构即是根据中序和后序序列画出二叉树。存储结构可以是顺序存储结构和链式存储结构(二叉链表)。证明如下。

当 n=1 时，只有一个根结点，由中序序列和后序序列可以确定这棵二叉树。

设当 n=m-1 时结论成立，现证明当 n=m 时结论成立。

设中序序列为 S1,S2,…,Sm,后序序列是 P1,P2,…,Pm。因后序序列最后一个元素 Pm 是根，则在中序序列中可找到与 Pm 相等的结点(设二叉树中各结点互不相同)Si(1≤i≤m)，因中序序列是由中序遍历而得，所以 Si 是根结点，S1,S2,…,Si-1 是左子树的中序序列，而 Si+1,Si+2,…,Sm 是右子树的中序序列。

若 i=1，则 S1 是根，这时二叉树的左子树为空，右子树的结点数是 m-1,则{S2,S3,…,Sm}和{P1,P2,…,Pm-1}可以唯一确定右子树，从而也确定了二叉树。

若 i=m，则 Sm 是根，这时二叉树的右子树为空，左子树的结点数是 m-1，则{S1,S2,…,Sm-1}和{P1,P2,…,Pm-1}唯一确定左子树，从而也确定了二叉树。

最后，当 1<i<m 时，Si 把中序序列分成{S1,S2,…,Si-1}和{Si+1,Si+2,…,Sm}。由于后序遍历是"左子树-右子树-根结点"，所以{P1,P2,…,Pi-1}和{Pi,Pi+1,…Pm-1}是二叉树的左子树和右子树的后序遍历序列。因而由{S1,S2,…,Si-1}和{P1,P2,…,Pi-1}可唯一确定二叉树的左子树，由{Si+1,Si+2,…,Sm}和{Pi,Pi+1,…,Pm-1}可唯一确定二叉树的右子树。算法如下。

```
void in_post_to_bt(char *INO,char *POST,int len,bt &T) {
    int k;   //k 为根结点在中序序列中的下标
    if(len<=0) {
        T=NULL;
        return;
    }
    for(char *temp=INO;temp<INO+len;temp++)//在中序序列 in 中找到根节点所在的位置
        if(*(POST+len-1)==*temp) {
            k=temp-INO;
            T=(bt)malloc(sizeof(struct tnode));
            T->data =*temp;
            break;
        }
     in_post_to_bt(INO,POST,k,T->lchild ); /*建立左子树*/
     in_post_to_bt(INO+k+1,POST+k,len-k-1,T->rchild ); /*建立右子树*/
}
```

调用：in_post_to_bt(INO+1,POST+1,strlen(INO)-1,T); //INO+1 代表 INO[1]

**九.**略　**十.**略

# 西北工业大学 2000 年研究生入学考试

一.简述题(24 分)

1. 【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 1 小题
2. 【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 2 小题

3.【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 3 小题

4.【注】略　5.【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 5 小题

6.外部排序指的是大文件的排序，即待排序的记录存储在外存上，待排序的文件无法一次装入内存，需要在内存和外存之间进行多次数据交换，以达到排序整个文件的目的。

参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 6 小题

**二.算法思想(10 分)**

1.【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》二算法思想题 1 小题

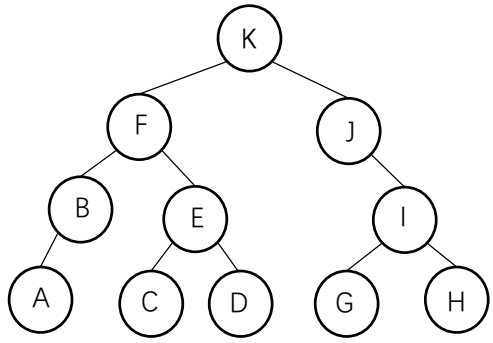
2.【注】参考《西北工业大学 2000-2001 学年第二学期期末考试》二算法思想题 2 小题

**三.算法应用(12 分)**

**1.**以升序为例，初始序列：23 6 16 8 7 3 10 3

**2.**一趟结果：3 6 16 8 7 3 10 23

**四.**(12 分)【解析】对二叉排序树来说，其中序遍历序列为一个递增有序序列。因此，对给定的二叉树进行中序遍历，如果始终能保持前一个值比后一个值小，则说明该二叉树是一棵二叉排序树。




先序遍历：KFBAECDJIGH

```
int predt = -99999999;
int JudgeBST(BiTree bt){
    int b1, b2;
    if(bt == NULL)
        return 1;
    else {
      b1 = JudgeBST(bt->lchild);
      if(b1 == 0 || predt >= bt->data)
          return 0;
      predt = bt->data;
      b2 = JudgeBST(bt->rchild);
      return b2;
    }
}
```

**五.**(12 分)【注】参考《西北工业大学 1998 年研究生入学考试》五算法实现　　**六、七.【略】**

# 西北工业大学 2001 年研究生入学考试

**一.(10 分)**【解析】参考《西北工业大学 2000-2001 学年第二学期期末考试》一简述题 2 小题和《西北工业大学 2017 年研究生入学考试》一简述题 1 小题

**二.(10 分)**【解析】参考《西北工业大学 1999 年研究生入学考试》二大题

**三.(12 分)**【证明】若让每个结点中的关键字个数达到最少，则容纳同样多关键字的 B 树的高度可达到最大。由 B 树定义：第一层至少有 1 个结点；第二层至少有 2 个结点；除根结点以外的每个非终端结点至少有$\lceil m/2 \rceil$棵子树，则第三层至少有$2\lceil m/2 \rceil$个结点……第 L+1 层至少有$2\lceil m/2 \rceil^{L-1}$个结点，注意到第 L+1 层是不包含任何信息的叶结点。对于关键字个数为 N 的 B 树，叶结点即查找不成功的结点为 N+1,由此有$N+1 \geq 2(\lceil m/2 \rceil)^{L-1}$，

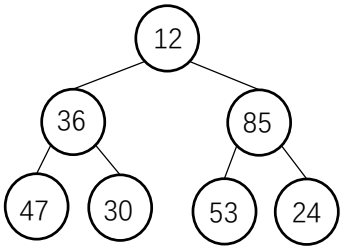
即$L \leq \log_{\lceil m/2 \rceil}\left(\frac{N+1}{2}\right)+1$。

**四. (15 分)**1.堆排序算法如下：

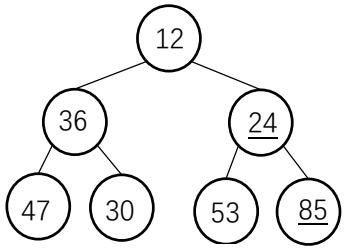
```
#include<iostream>
using namespace std;
    mySwap(arr, max, index);
    HeadAdjust(arr, max, len);
```

```
void PrintArray(int arr[], int len){
    for (int i = 0; i < len; i++)
        cout << arr[i] << " ";
    cout << endl;
}
void mySwap(int arr[],int a,int b){
    int tmp = arr[a];
    arr[a] = arr[b];
    arr[b] = tmp;
}
//index 待调整的结点下标
void HeadAdjust(int arr[], int index, int len){
    int max=index;
    int lchild = index * 2 + 1;
    int rchild = index * 2 + 2;
    if (lchild < len&&arr[lchild]>arr[max]){
        max = lchild;
    }
    if (rchild<len&&arr[rchild]>arr[max]){
        max = rchild;
    }
    if (max != index){
    }
}
//堆排序
void HeapSort(int arr[], int len){
    //初始化堆
    for (int i = len / 2 - 1; i >= 0; i--){
        HeadAdjust(arr, i, len);
    }
    for (int i = len - 1; i >= 0; i--){
        mySwap(arr, 0, i);
        HeadAdjust(arr, 0, i);
    }
}
int main( ){
    int myarr[] = { 12,36,85,47,30,53,24 };
    int len = sizeof(myarr) / myarr[0];
    PrintArray(myarr, len);
    HeapSort(myarr, len);
    PrintArray(myarr, len);
    return 0;
}
```

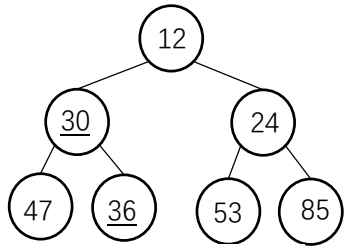
堆排序过程：先构造小顶堆




(1)初始状态

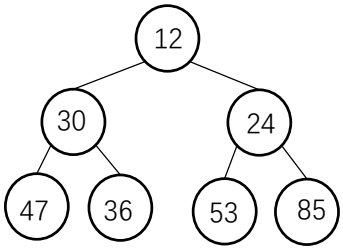



(2)85 和 24 交换

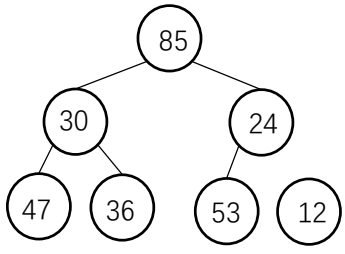



(3)36 和 30 交换

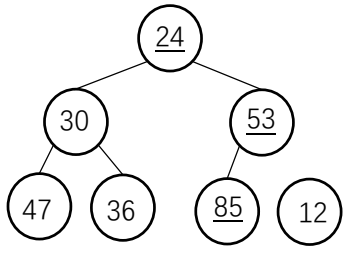
2.输出元素进行堆排序



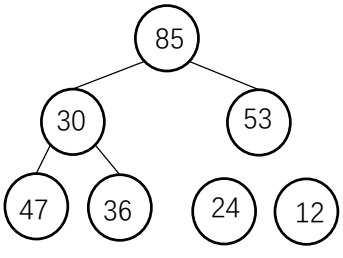

(4)小顶堆构造完成



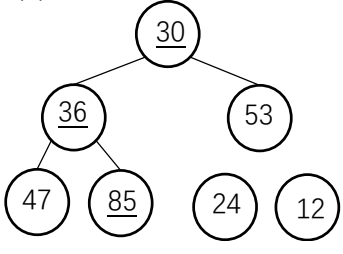

(1)输出 12，12 和 85 交换



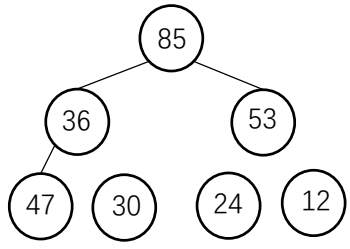

(2)向下调整，85 和 24 交换，85 和 53 交换



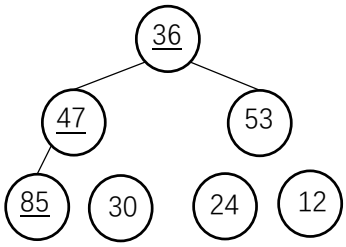

(3)输出 24，24 和 85 交换



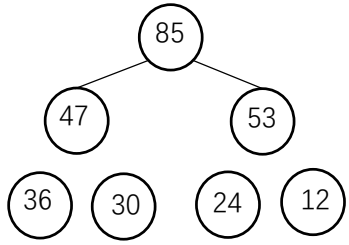

(4)向下调整，85 和 30 交



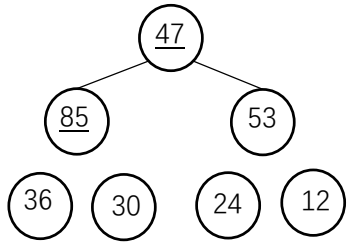

(5)输出 30，30 和 85 交换



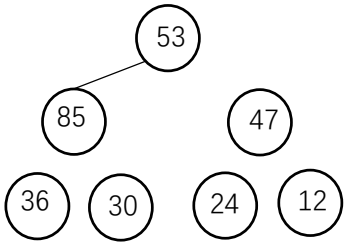

(6)向下调整，85 和 36 交换，85 和 47 交换



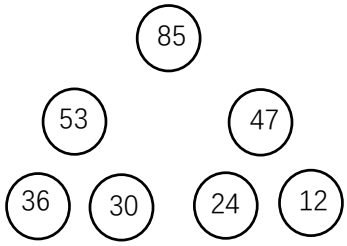

(7)输出 36，36 和 85 交换



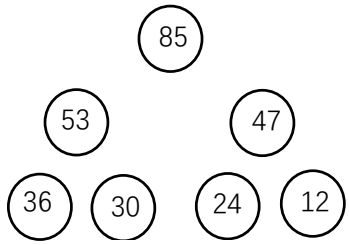

(8)向下调整, 85 和 47 交换




(9)输出 47，47 和 53 交换




(10)输出 53，53 和 85 交换




(11)输出 85，完成

**五.(18 分)** (1)p->lchild = lc->rchild;　(2)e.key < T->data.key　(3)T->bf = lc->bf = 0;
(4)T->bf = -1;　(5)L_Rotate(T->lchild);　(6)R_Rotate(T);

【注】bf 为 balance factor，平衡因子

**六.**略(题意不明确，可参考旅行商问题)

**七.(20)(1)**存储结构

```
struct PolyNode{
    int coef;   //系数
    int expon; //指数
    PolyNode* next; //指针域
};
```

**(2)**多项式输入函数：按指数降序输入，采用尾插法插入

```
void ScanfPoly(PolyNode* &L){
    int n, c, e;
    L = (PolyNode*)malloc(sizeof(PolyNode));
    PolyNode *s, *r = L;
    scanf("%d", &n); //多项式项数
    while(n --){
        scanf("%d %d", &c, &e);
        s = (PolyNode*)malloc(sizeof(PolyNode));
        s->coef = c;
        s->expon = e;
        r->next = s;
        r = s;
    }
    r->next = NULL;
}
```

**(3)**多项式相乘函数：将 L1 的各项与 L2 的各项逐项相乘，再插入结果链表中

```
PolyNode* Multiply(PolyNode* L1, PolyNode* L2){
    PolyNode *t1, *t2, *t, *rear, *L;
    int c, e;
    L = (PolyNode*)malloc(sizeof(PolyNode));
    L->next = NULL;
    t1 = L1->next;
    t2 = L2->next;
    rear = L;
    if(!t1 || !t2)//如果有一个多项式为空，则乘法结果为空
        return NULL;
    while(t1){
```

```
            while(t2) {
                c = t1->coef * t2->coef;
                e = t1->expon + t2->expon;
               //将 rear 指向指数与 e 相等或比 e 小的节点之前一个位置
                while(rear->next && rear->next->expon > e)
                    rear = rear->next;
                //如果 rear 不为空且 rear 之后的节点指数和 e 相等，则进行相加
                if(rear->next && rear->next->expon == e) {
                    if(c + rear->next->coef != 0) {   //如果相加不为 0
                        rear->next->coef += c;
                    }
                    else {   //相加结果为 0 ,删除 rear 之后的节点
                        t = rear->next;
                        rear->next = t->next;
                        free(t);
                    }
                }
                else { //构造一个新节点插入到 rear 之后
                    t = (PolyNode*)malloc(sizeof(PolyNode));
                    t->coef = c;
                    t->expon = e;
                    t->next = rear->next;
                    rear->next = t;
                }
                t2 = t2->next;
                rear = L;
            }
            t1 = t1->next;
            t2 = L2->next;
        }
        return L;
}
```

(4)多项式输出函数

```
void PrintPoly(PolyNode* L) {
    PolyNode* p = L->next;
    if(!p) {
        printf("0 0\n");
        return;
    }
    while(p) {
        printf("%d %d ",p->coef, p->expon);
        p = p->next;
    }
    printf("\n");
}
```

(5)主函数

```
int main(){
    PolyNode *L1, *L2, *L3;
    ScanfPoly(L1);
    ScanfPoly(L2);
    L3 = Multiply(L1, L2);
    PrintPoly(L3);
    return 0;
}
```

# 西北工业大学 2002 年研究生入学考试

**一．**(30 分)**1.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 1 小题。

**2.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 2 小题。

**3.**线索二叉树：在二叉链表表示的二叉树中存在大量的空指针，若利用这些空链域存放指向其直接前驱或后继的指针，称之为线索。加上线索的二叉树称为线索二叉树。

先序线索二叉树直接前驱：

(1)如果有左线索，则左线索指向其前驱；

(2)若结点 x 是双亲左孩子或是双亲右孩子且双亲无左子树，则前驱是双亲；

(3)如果结点 x 是双亲右孩子且双亲有左子树，则前驱是左子树按先序遍历的最后一个结点。

后序线索二叉树直接后继：

(1)如果有右线索，则右线索指向其后继；

(2)若结点 x 为根，则无后继；

(3)若结点 x 为其双亲的右孩子或为其双亲的左孩子且双亲无右子女，则其后继为其双亲；

(4)若结点 x 为其双亲的左孩子，且双亲有右子女，则后继是其双亲的右子树中按后序遍历的第一个结点。

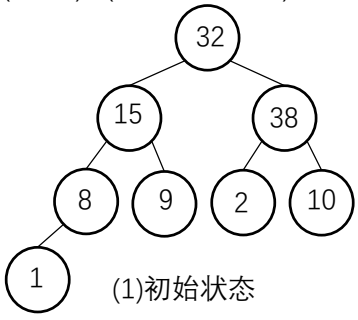
**4.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 4 小题

**5.**【注】参考《西北工业大学 2004-2005 学年第二学期期末考试》一简述题 5 小题

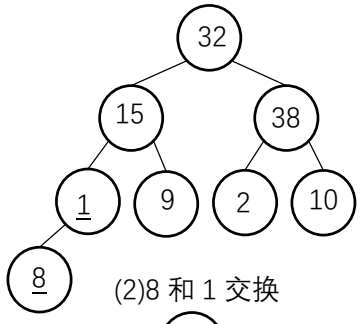
**二．**(10 分)证明：满 k 叉树一共有 $1+k+k^2+k^3+\cdots\cdots+k^x$($k^x$就是所求的叶结点数)=$(1-k^{(x+1)})/(1-k)$个结点，其中有 N 个分支节点，剩下的都是叶结点，也就是最外边的 $k^x$，所以$(1-k^{(x+1)})/(1-k)-N=k^x$。解方程得：$k^x=N(k-1)+1$，所以有 N(k-1)+1 个叶结点。

**三.** (10 分)参考《西北工业大学 2004-2005 学年第二学期期末考试》第二题

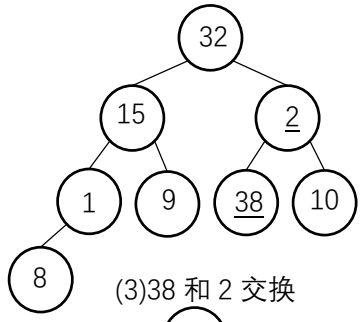
**四.** (20 分)**1.**(以小根堆为例)



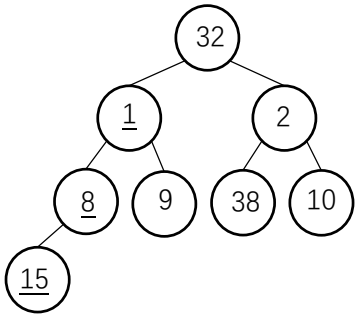

(1)初始状态



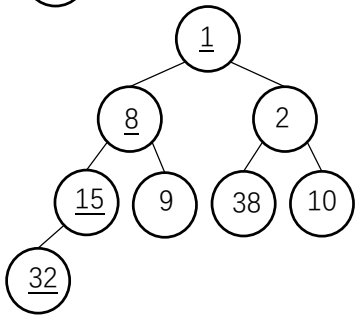

(2)8 和 1 交换




(3)38 和 2 交换

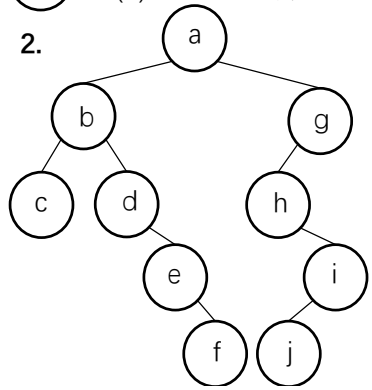



(4)15 和 1 交换，15 和 8 交换




(5)32 和 1 交换，32 和 8 交换，
32 和 15 交换，初始堆构造完成

**2.**




后序序列：cfedbjihga

**五.**根据完全二叉树的定义，具有 n 个结点的完全二叉树与满二叉树中编号 1 到 n 的结点一一对应。算法思想：采用层次遍历的算法，将所有结点加入队列(包括空结点)。当遇到空结点时，查看其后是否有非空结点。若有，则不是完全二叉树。

```
bool IsComplete(BiTree T){
    InitQueue(Q);
    if(!T)
        return 1;
    EnQueue(Q, T);
    while(!IsEmpty(Q)){
        DeQueue(Q, p);
        if(p) {
            EnQueue(Q, p->lchild);
            EnQueue(Q, p->rchild);
        }
        else{
            while(!IsEmpty(Q)){
                DeQueue(Q, p);
                if(p)
                    return 0;
            }
        }
    }
    return 1;
}
```

**六．**使用栈 s 记忆下一步可能访问的顶点，同时使用了一个访问标记数组 visited[i]，在 visited[i]中记忆第 i 个顶点是否在栈内或曾经在栈内。若是，以后它不能再进栈。

```
bool visited[vexnum];
void DFS(Graph G, int v){
    int w;
    initstack(s);
    push(s, v);
    visited[v] = true;
    while(!IsEmpty(s)){
        k = Pop(s);
        visit(k);
        for(w = FirstNeighbor(G,k);w >= 0;w =
NextNeighor(G, k, w)){
            if(!visited[w]){
                Push(s, w);
                visited[w] = true;
            }
        }
    }
}
void DFSTraverse(Graph G){
    for(i = 0;i < G.vexnum;i ++)
        visited[i] = false;
    for(i = 0;i < G.vexnum;i ++)
        if(!visited[i])
            DFS(G, i);
}
```

# 西北工业大学 2004 年研究生入学考试

## 一.简述题

**1.**线性数据结构是 n 个数据元素的有序(次序)集合,指的是数据元素之间存在着“一对一”的线性关系的数据结构。

常用的线性结构有：线性表，栈，队列

相同点：栈和队列都是操作受限制的线性表，它们和线性表一样，数据元素之间都存在“一对一”的关系。

不同之处：栈是只允许在一段进行插入或删除操作的线性表，其特点是“后进先出”，队列是只允许在一端进行插入，另一端进行删除操作的线性表，其特点是“先进后出”。

**2.**先序线索二叉树直接前驱：(1)如果有左线索，则左线索指向其前驱；

(2)若结点 x 是双亲左孩子或是双亲右孩子且双亲无左子树，则前驱是双亲；

(3)如果结点 x 是双亲右孩子且双亲有左子树，则前驱是左子树按先序遍历的最后一个结点。

后序线索二叉树直接后继：(1)如果有右线索，则右线索指向其后继；

(2)若结点 x 为根，则无后继；

(3)若结点 x 为其双亲的右孩子或为其双亲的左孩子且双亲无右子女，则其后继为其双亲；

(4)若结点 x 为其双亲的左孩子，且双亲有右子女，则后继是其双亲的右子树中按后序遍历的第一个结点。

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

算法特征：(1)有穷性，(2)确定性，(3)可行性，(4)输入，(5)输出

算法复杂度分为时间复杂度和空间复杂度。时间复杂度是指执行算法所需要的计算工作量，它定性描述了该算法的运行时间；而空间复杂度是指执行这个算法所需要的内存空间。

算法与数据结构的关系：数据结构只是静态地描述了数据元素之间的关系，而算法侧重于对问题的建模。程序设计=数据结构+算法，高效的程序需要在数据结构的基础上设计和选择算法。数据结构是算法实现的基础，算法总是要依赖于某种数据结构来实现的。

**4.**拓扑序列：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，若边(u,v)∈E(G)，则 u 在线性序列中出现在 v 之前。将这样的线性序列称为拓扑序列。

拓扑排序的算法思想：

(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出；

(2)从图中删除该顶点和所有以它为起点的有向边；

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。

**5.**数据结构的存储方式：

(1)顺序存储：把逻辑上相邻的元素存储在物理位置上也相邻的存储单元里，元素之间的关系由存储单元的邻接关系来体现；

(2)链式存储：不要求逻辑上相邻的元素在物理位置上也相邻，借助指示元素存储地址的指针表示元素之间的逻辑关系；

(3)索引存储：建立附加的索引表，索引项的一般形式是：(关键字，地址)。

(4)散列存储：根据元素的关键字直接计算出该元素的存储地址。

## 二.算法应用

**1.**(1)快速排序(可参考教材，对排序过程标出箭头和 i，j 的值)

第一趟：<u>310</u> 8 27 132 6 95 18 47

{47 8 27 132 6 95 18} <u>310</u>

第二趟：<u>47</u> 8 27 132 6 95 18 310

18 8 27 132 6 95 <u>47</u> 310

18 8 27 <u>47</u> 6 95 132 310

{18 8 27 6} <u>47</u> {95 132} 310

第三趟：<u>18</u> 8 27 6 47 <u>95</u> 132 310

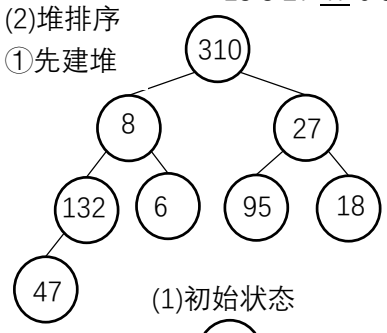
6 8 27 <u>18</u> 47 <u>95</u> 132 310

{6 8} <u>18</u> {27} 47 <u>95</u> {132} 310

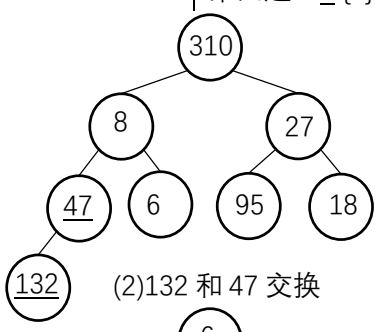
第四趟：<u>6</u> {8} 18 27 47 95 132 310

(2)堆排序

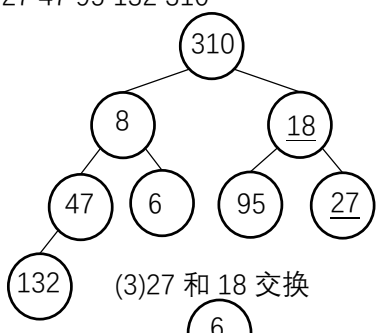
①先建堆



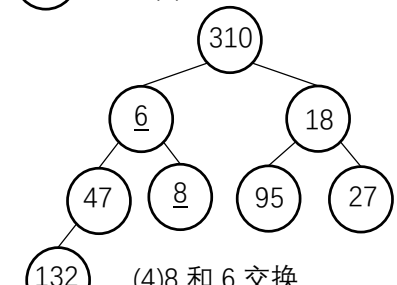

(1)初始状态



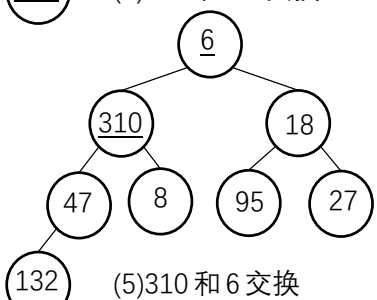

(2)132 和 47 交换



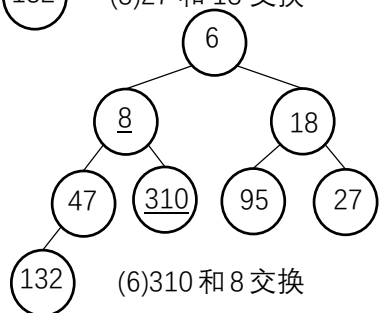

(3)27 和 18 交换




(4)8 和 6 交换

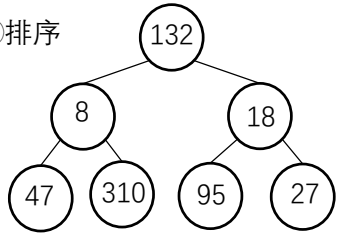



(5)310 和 6 交换

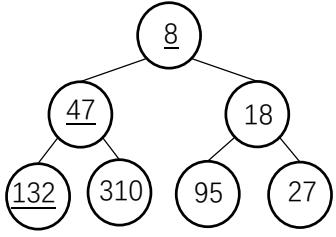



(6)310 和 8 交换

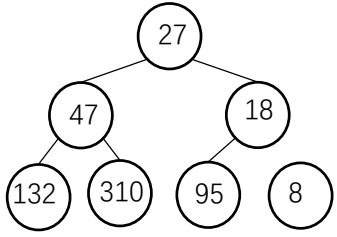
②排序



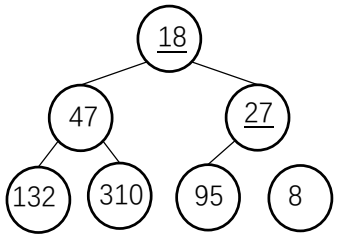

6 (1)输出 6，6 和 132 交换



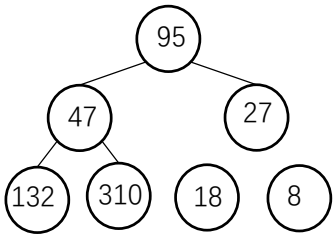

6 (2)向下调整，132 和 8 交换，132 和 47 交换



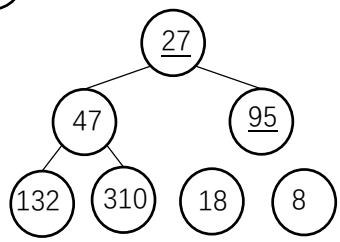

6 (3)输出 8，8 和 27 交换



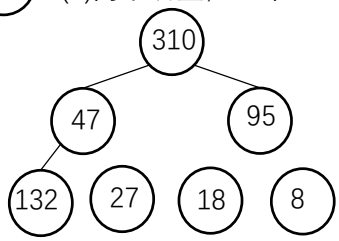

6 (4)向下调整，27 和 18 交换



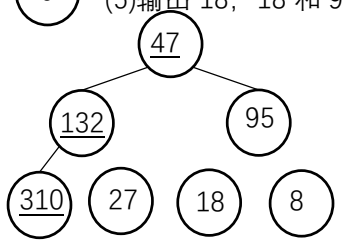

6 (5)输出 18，18 和 95 交换



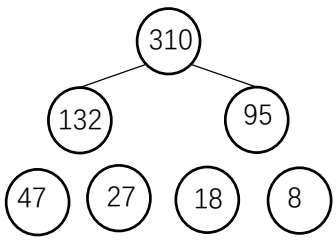

6 (6)向下调整，95 和 27 交换



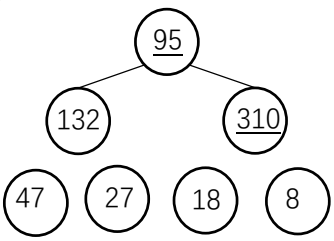

6 (7)输出 27，27 和 310 交换



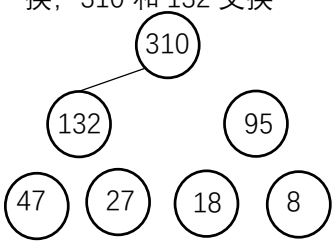

6 (8)向下调整，310 和 47 交换，310 和 132 交换



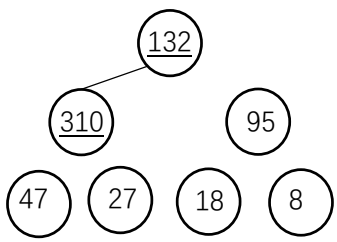

6 (9)输出 47，47 和 310 交换



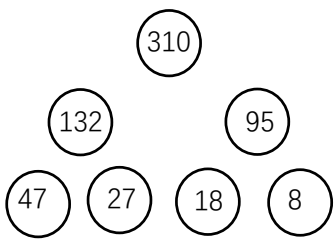

6 (10)向下调整，310 和 95 交换



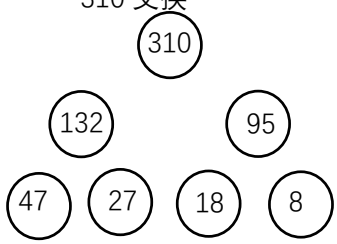

6 (11)输出 95，95 和 310 交换




6 (12)向下调整，310 和 132 交换




6 (13)输出 132，132 和 310 交换




6 (14)输出 310，结束

**2.**(见图 04-1)先序序列：ABCDEFGH

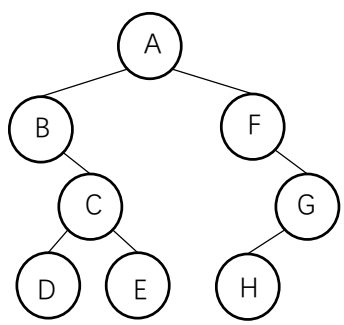



图 04-1

## 三.设计合理的数据结构并给出算法

**1.**基于回溯法寻找哈密顿回路：

在包含问题的所有解的解空间树中，按照深度优先搜索的策略，从根结点出发深度探索解空间树。当探索到某一结点时，要先判断该结点是否包含问题的解，如果包含，就从该结点出发继续探索下去，如果该结点不包含问题的解，则逐层向其祖先结点回溯(其实回溯法就是对隐式图的深度优先搜索算法)。若只需要求任一个解时，只要搜索到问题的一个解就可以结束。

```
void print(int path[], int V){
    cout << "存在哈密顿回路" << endl;
    for (int i = 0; i < V; i++) cout << path[i] << " ";
    cout << path[0] << endl;
}
//path 记录路径，visited 记录顶点是否访问过，len 记录当前路径的长度
bool hamCycle(int graph[][MAX_V], int V, int path[], bool visited[], int current) {
    if (current == V) { //访问到最后一个顶点
        if (graph[path[current - 1]][0] == 1)   return true;//有到 0 点的边
        else return false;
    }
    //遍历起点外其它顶点
    for (int v = 1; v < V; v++) {
        //如果没访问过，并且有边相连
        if (!visited[v] && graph[path[current - 1]][v] == 1) {
            visited[v] = true;
            path[current] = v;
            //当本次递归的 child 也为 true 时返回 true
            if (hamCycle(graph, V, path, visited, current + 1)) return true;
            //当本条递归线路失败时恢复原图
            path[current] = -1;
            visited[v] = false;
        }
    }
    return false;
}
//从起点开始引导
bool hamCycleStart(int graph[][MAX_V], int V) {
    int path[MAX_V];
    memset(path, -1, sizeof(path));
    bool visited[MAX_V] = { 0 };
    path[0] = 0;
    visited[V] = true; //把起点标记为访问过
    //起点已确定，current 从 1 开始
    if (hamCycle(graph, V, path, visited, 1) == false) {
        cout << "哈密顿回路不存在" << endl;
        return false;
    }
    print(path, V);
    return true;
```

}

**2.**借助队列实现，首先将给定根节点 pRoot1 和 pRoot2 都入队

(1)当两个队列未空时，分别获取两个树的当前层次中节点总数(即当前队列中节点个数)，先比较节点个数是否相同，如果不相同，则两个树自然不同；如果节点个数相同，需要出队进行比较。如果有一个队列未空，则退出比较。

(2)如果有一个队列未空，则清空队列并返回不同。

(3)如果结构相同，再比较对应结点的元素是否相等。

```
bool    BTreeCompare(BTreeNode_t *pRoot1, BTreeNode_t *pRoot2){
      if( pRoot1 == NULL && pRoot2 == NULL )
            return false;
      queue <BTreeNode_t *> que1;
  queue <BTreeNode_t *> que2;
      que1.push(pRoot1);
      que2.push(pRoot2);
      int curLevelNodeTotal1 = 0;
      int curLevelNodeTotal2 = 0;
      bool flag = true; //作为比较不一致时跳出标识
      while( ( !que1.empty()) && ( !que2.empty())){
      //当两个队列均不为空时，才进行比较
            curLevelNodeTotal1 = que1.size(); //获取树 1 的当前层节点总数
            curLevelNodeTotal2 = que2.size(); //获取树 2 的当前层节点总数
            if( curLevelNodeTotal1 != curLevelNodeTotal2){
                  flag = false;
                  //当前层节点总数都不一致，不需要比较了，直接跳出
                  break;
            }
            int cnt1 = 0;//遍历本层节点时的计数器
            int cnt2 = 0;
            while( cnt1 < curLevelNodeTotal1    && cnt2 < curLevelNodeTotal2){
                  ++cnt1;
                  ++cnt2;
                  pRoot1 = que1.front();
                  que1.pop();
                  pRoot2 = que2.front();
                  que2.pop();
                  //比较当前节点中数据是否一致
                  if( pRoot1->m_pElemt != pRoot2->m_pElemt ){
                        flag = false;
                        break;
                  }
                  //判断 pRoot1 和 pRoot2 左右节点结构是否相同
                  if( ( pRoot1->m_pLeft != NULL && pRoot2->m_pLeft == NULL )||
                        ( pRoot1->m_pLeft == NULL && pRoot2->m_pLeft != NULL )||
                        ( pRoot1->m_pRight != NULL && pRoot2->m_pRight == NULL )||
```

```
                ( pRoot1->m_pRight == NULL && pRoot2->m_pRight != NULL )){
                    flag = false;
                    break;
                }
                //将左右节点入队
                if( pRoot1->m_pLeft != NULL )
                    que1.push( pRoot1->m_pLeft);
                if( pRoot1->m_pRight != NULL )
                    que1.push( pRoot1->m_pRight);
                if( pRoot2->m_pLeft != NULL )
                    que2.push( pRoot2->m_pLeft);
                if( pRoot2->m_pRight != NULL )
                    que2.push( pRoot2->m_pRight);
            }
            if( flag == false )
                break;
        }
            //如果比较标志为 false，则不相同
            if( flag == false ){
                while( !que1.empty() )
                    que1.pop();
                while( !que2.empty())
                    que2.pop();
                return false;
            }
            return true;
}
```

# 西北工业大学 2004 年研究生入学考试(814)

## 一.单项选择题(每空 2 分，共 20 分)

**答案速查：DA　BBEC　BCCA**

**2.**B【解析】因为 KMP 算法涉及到 next 数组的存储，且 next 数组是基于模式串长度计算的。

**3.**B【解析】非循环链表和带头指针的循环链表查找尾节点的时间效率是 O(n)，而带尾指针的循环链表查找首尾节点的时间效率都是 O(1)，查找和更改时只需修改指针，无需遍历，队列的操作实际上就是对链表首尾节点的操作。

**5.**C【解析】链地址法不会产生“聚集”现象

**7.**C【解析】A[8][5]在矩阵 A 的第 9 行第 6 列，1+2+3+…+8+6-1=41。

**8.**C【解析】每一个非终端节点，它的所有孩子节点在转化之后，最后一个孩子的右指针也为空，所以为 n。森林中的最后一棵树成为右子树，其根结点也没有右孩子，所以为 n+1。

二.判断题(每小题 1 分，共 7 分)

**答案速查：×××√×　××**

**1.**×【解析】数据元素，此题常见考查形式为选择题，如下例题。

【例题】数据的基本单位是（　）

A.数据项　　B.数据类型　　C.数据元素　　D.数据变量

**2.**×【解析】因为可能出现重复的权值，所以树不唯一；如果所有权值都不唯一，那么最小生成树唯一。

【举一反三】一个带权的无向连通图的最小生成树的权值之和是唯一的（√）

【解析】树不唯一，权值和唯一。

**3.**×【解析】数组的数据元素之间的逻辑结构是一对一的线性结构。

【举一反三】多维数组元素之间的关系既不是线性的也不是树形的。（√）

**4.**×【解析】只考虑了 2-路归并情况而未考虑多路归并情况，O(nlogN)更为准确。

【注】算法复杂度一般用 logN 而非 $\log_2 n$ 是因为假如有 logab (a 为底数) 由换底公式可得 $\log_a b=\log_c b/\log_c a$，$\log_c a$（c 为底数）为常数，由运算法则“O(C×f(N))=O(f(N))，其中 C 是一个正的常数”得 $O(\log_a b)=O(\log_c b)$可知算法的时间复杂度与不同底数只有常数的关系，均可以省略。故可用 logN 代替

**5.**×【解析】与图的顶点数有关

**6.**×【解析】频率相同时，编码也不可能相同；任何两个不同字符编码都不同

**7.**×【解析】链表可以不一致

三.填空题(每空 2 分，共 8 分)

**1.**出　入　　**2.**有　无

四.简要回答问题(共 8 分)

**1.**$N=N_0+N_2=N_2+1+N_2=2*N_2+1=39,N_2=19$。

**2.**数据结构概念包括:

（1）数据的逻辑结构：数据的逻辑结构只抽象地反映数据元素之间的逻辑关系，它与数据的存储无关，是独立于计算机的；（2）数据的存储结构：数据的存储结构是数据的逻辑结构在计算机存储器里的实现（亦称为映象）；（3）数据的运算：数据的运算定义在数据的逻辑结构之上，每种逻辑结构都有一个运算的集合。常用的运算有:查找、插入、删除、更新、排序等。

抽象数据类型 ADT：指一个数学模型以及定义在此数学模型上的一组操作。抽象数据类型需要通过固有数据类型（高级编程语言中已实现的数据类型）来实现。抽象数据类型是与表示无关的数据类型，是一个数据模型及定义在该模型上的一组运算。对一个抽象数据类型进行定义时，必须给出它的名字及各运算的运算符名，即函数名，并且规定这些函数的参数性质。一旦定义了一个抽象数据类型及具体实现，程序设计中就可以像使用基本数据类型那样，十分方便地使用抽象数据类型。

五.综合算法题(共 16 分)

| | |
|---|---|
| **（1）** void InsertSort(int A[], int m, int &n, int x) {<br>　int p;<br>for (i=0;i<n;i++) { | **（2）**第一个与最后一个交换，第二个与倒数第二个交换，以此类推。<br>void reverse(int A[ ], int n) { |

```
        //判断新数据在原数组中的位置
        if (x<A[i]){
            p=i;
            break;
        }
    }
    for(i=n-1;i>=p;i--)
        //在 p 位置之后的数据全部往后挪一位
        A[i+1]=A[i];
    //把新数据放入 p 位置
    A[p]=x;
    n=n+1;
}
```

```
    int i, t;
    for(i=0;i<n/2;i++){
        t=A[i];
        A[i]=A[n-i-1];
        A[n-i-1]=t;
    }
}
```

**(3)** 利用双重循环，将每个值依次与其后面的值相比较，如果有相同的则删除该元素即可。删除时，可以使用将后面元素依次向前移动一位，同时总长度减一的方式。

```
void delDuplicate(int A[], int &n){
    int i, j, k;
    for(i = 0; i < n; i ++) {
        for(j = i+1; j < n; j ++) {          //对后面每个元素比较，去重。
            if(A[j] == A[i]) {               //发现重复元素。
                for(k = j+1; k < n; k ++)    //依次前移一位。
                    A[k-1] = A[k];
                n--;                         //总长度减一。
            }
        }
    }
}
```

六.综合算法题(16 分)

**(1)** 有序二叉树的中序遍历是从小到大的序列，但题意却是要从大到小输出，故需要采用右根左的遍历方式。（这里给出非递归的方法）

```
void FindX(BinarySearchTree* BST , int x){
    stack<BinarySearchTree*> stack;                    //初始化栈
    BinarySearchTree* binary_tree_curr = BST;          //保存当前结点
        while(binary_tree_curr || !stack.empty()){
            if(binary_tree_curr->rchild){              //右孩子非空
                stack.push(binary_tree_curr);          //当前结点入栈
                binary_tree_curr = binary_tree_curr->rchild;   //遍历右子树
            }
            else{          //右孩子为空，则打印当前结点并遍历左子树
                if(binary_tree_curr->data >= x){
                    cout<<binary_tree_curr->data<<" ";
                }
```

```
                binary_tree_curr = binary_tree_curr->lchild;
                //如果为空，且栈不空，则将栈顶节点出栈，并输出该节点，
                //同时将它的左孩子设为当前节点，继续判断，直到当前节点不为空
                while(!binary_tree_curr && !stack.empty()){
                        binary_tree_curr = stack.top();
                        if(binary_tree_curr->data >= x){
                           cout<<binary_tree_curr->data<<" ";
                        }
                        stack.pop();
                        binary_tree_curr = binary_tree_curr->lchild;
                }
            }
        }
}
```

| 【补充】递归算法：<br>InOrder(BiTree T, int x) {<br>if(!T) {<br>InOrder(T->rc);//先右<br>if(T->data>=x) | printf(T->data);<br>else return 0;<br>InOrder(T->lc);//后左<br>}<br>} |
|---|---|

**（2）**判断除 V0 以外的其它顶点是否有与 V0 相连的长度不超过 k 的简单路径，采用递归的方法，path 记录路径（也可以没有 path），visited 记录是否访问过。

```
int Search(MGraph G, int x, int y,int k,int visited[],int path[], int d){
    int n, i;
    ArcNode *p;
    visited[x] = l;
    d ++;
    path[d] = x;
    if(x == y && d <= k) return 1;
    p=G.vertices[x].firstarc;
    while(p !=NULL) {
        n=p->adjvex ;
        if(visited[n]==0){
            if ((j = Search(G, n, y, k, visited, path, d))==1)
                return j;
        }
        p = p->nextarc ;
    }
  visited[x]=0;
    d --;
    return 0;
}
int main(){
    int visited[G.vexnum];
    int path[G.vexnum];
    for(int i = 1;i < G.vexnum,;i ++){
        memset(visited, 0, sizeof(visited));
        if(Search(G, 0, i, k, visited, path, -1)) //k 是要判断的长度
            cout << G.getValue(i) <<" ";
    }
    return 0;
}
```

# 西北工业大学 2007 年研究生入学考试

一.选择题(共 20 分)

**答案速查：ABDAC　CDBCB**

**2.**B【解析】前序是 NLR，后序是 LRN，如果前序和后序刚好相反，即 NLR=NRL，说明这个树①没有左孩子或②没有右孩子，是单支树，所以树的高度等于结点数目。

**9.**C【解析】归并排序，每一次归并假设两个需要归并的序列长度是 n，如果已经有序，那比较次数就是 n；如果无序，最差情况下需要比较 2n-1 次，因此比较次数与初始序列有关。比较次数不受初始序列影响的是简单选择排序和折半插入排序，因此实际本题无正确选项。

【注】本题在网上查到的解析是选择排序在最好、最坏、平均情况下的时间性能都是 $O(n^2)$，归并排序在最好、最坏、平均情况下的时间性能均为 O(nlogn)。也就是说比较次数是按照时间复杂度来进行概念替换的，但实际上时间复杂度是比较次数的宏观体现，只能说明比较次数在哪一个数量级，但无法提现具体的微观次数。

【总结】各排序算法总结如下：①各排序复杂度：

| 排序算法 | 平均时间复杂度 | 最好情况 | 最坏情况 | 空间复杂度 | 排序方式 | 稳定性 |
|---|---|---|---|---|---|---|
| 冒泡排序 | $O(n^2)$ | O(n) | $O(n^2)$ | O(1) | In-place | 稳定 |
| 选择排序 | $O(n^2)$ | $O(n^2)$ | $O(n^2)$ | O(1) | In-place | 不稳定 |
| 插入排序 | $O(n^2)$ | O(n) | $O(n^2)$ | O(1) | In-place | 稳定 |
| 希尔排序 | O(nlogn) | $O(n\log^2 n)$ | $O(n\log^2 n)$ | O(1) | In-place | 不稳定 |
| 归并排序 | O(nlogn) | O(nlogn) | O(nlogn) | O(n) | Out-place | 稳定 |
| 快速排序 | O(nlogn) | O(nlogn) | $O(n^2)$ | O(nlogn) | In-place | 不稳定 |
| 堆排序 | O(nlogn) | O(nlogn) | O(nlogn) | O(1) | In-place | 不稳定 |
| 计数排序 | O(n+k) | O(n+k) | O(n+k) | O(k) | Out-place | 稳定 |
| 桶排序 | O(n+k) | O(n+k) | $O(n^2)$ | O(n+k) | Out-place | 稳定 |
| 基数排序 | O(n×k) | O(n×k) | O(n×k) | O(n+k) | Out-place | 稳定 |

②算法复杂度与初始状态无关：选择排序、归并排序、堆排序基数排序；

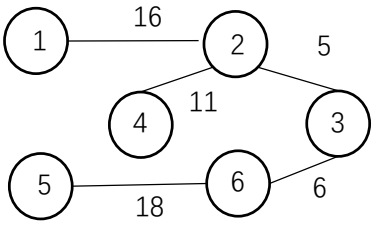
③总排序趟数与初始状态无关：除了快速排序、优化过后的冒泡排序，其他的排序算法都满足；

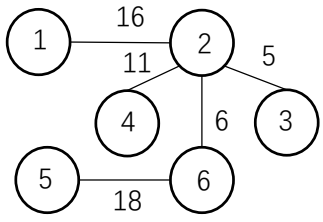
④元素总比较次数与初始状态无关：基数排序、选择排序；

⑤元素总移动次数与初始状态无关：基数排序、归并排序。

二.计算题(15 分)

**1.**求最小生成树，有两种选择



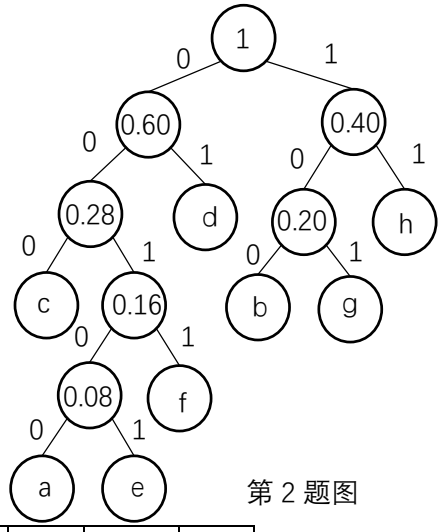
【注】注意理解原理，最小生成树和最短路径的区别，生成树的图一般是无向图，最短路径是有向，Floyd 和 Dijkstra 是最短路径，一般用 Dijkstra，很少用 Floyd。

**2.**构造哈夫曼树

哈夫曼编码：a:00100 b:100 c:000 d:01 e: 00101 f:0011 g:101 h:11

平均编码长度 =5 × (0.03+0.05)+4 × 0.08+3 × (0.09+0.12+0.11)+2 × (0.32+0.20)= 2.72

若采用三位固定编码：等长编码长度= 3，缩短了(3-2.72)/3×100%=9.33%



第 2 题图

三.分析题(25 分)

**1.**

| A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| O | P | Q | R | S | T | U | V | W | X | Y | Z | | |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | | |

**(1)线性探测法**

H(Jan)=10/2=5;
H(Feb)=6/2=3;
H(Mar)=13/2=6;
H(Apr)=1/2=0;
H(May)=13/2=6;冲突;H1=6+1=7;
H(June)=10/2=5;冲突;H1=5+1=6;冲突;H2=7;H3=8;
H(July)=5;H1=6;H2=7;H3=8;H4=9
H(Aug)=0;H1=1;
H(Sep)=9;H1=10;
H(Oct)=7;H1=8;H2=9;H3=10;H4=11;
H(Nov)=7;H1=8;H2=9;H3=10;H4=11;H5=12
H(Dec)=2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| Apr | Aug | Dec | Feb | | Jan | Mar | May | June |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | |
| July | Sep | Oct | Nov | | | | | |

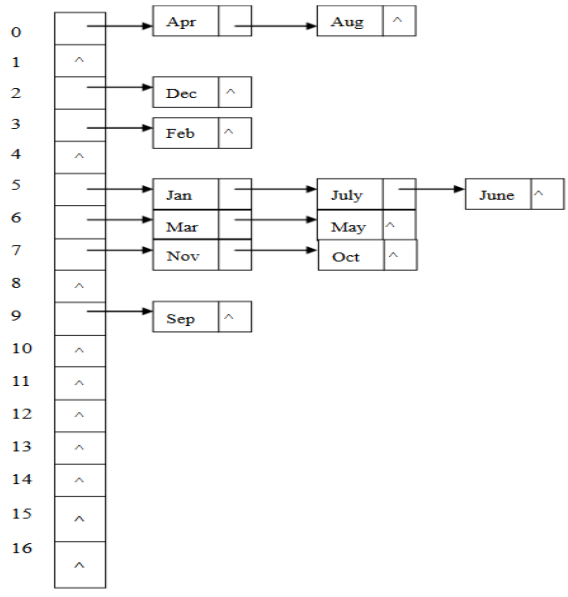
ASL 成功=(5*1+3*2+1*4+2*5+1*6)/12=31/12

ASL 失败=(5+4+3+2+1+9+8+7+6+5+4+3+2+1)/14=60/14

**(2)链地址法**(见右图)

ASL 成功=(7*1+4*2+3*1)/12=1.5

ASL 失败=(1*3+2*3+3*1)/14=6/7




【注】/14 而不是/16 的原因是散列函数的值域，26/2=13，0~13 共 14 个。也就是说最大可能找到的位置是 14，一共有 14 个位置是可能被算出来的。

【注 2】哈希表到 16 的原因地址空间是 0-16，不代表一定能存储。链地址的 0-13 就够了，线性探测 14-17 就有可能用到了。

【补充】目前求查找失败的平均查找长度有两种观点：①认为比较到空节点才算失败，比较次数等于冲突次数加 1；②认为只有与关键字的比较才算比较次数。

一般采用第一种，这是拉链法（链地址法），不算空指

针；与开放定址法不同，开放定址法要算空。这是因为：用链地址法的时候哈希表每个格子里存的是对应链表的头指针，没有值的地方存的是空指针，关键字比较之前会先判断是否是空指针，不是空才进入比较，所以不计算比较次数；开放定址的时候表里每个格子都用来存放一个值，没有值的地方就是 0 或者某个标记，当然需要进行一次关键字比较才知道这里是不是空。

【注 3】类似题目请参考《2010 年研究生入学考试计算机统考 408》二综合应用题的 41 小题及答案中的扩展部分。

**2.**堆排序；比较次数：34 次，(初始建堆：比较 20 次，得到 6； 第一次调整：比较 5 次，得到 7； 第二次调整：比较 4 次，得到 9； 第三次调整：比较 5 次，得到 11)

**四．(10 分)**可以采用快速排序的算法思想，以 0 为基准进行一趟快速排序。

```
void move(SeqList *L){
    int i, j;
    int temp;
    for (i = 0, j = L->length -1; i < j; ){
       while (i < j && L->data[i] < 0)
            i ++;
       while (i < j && L->data[j] > 0)
            j --;
       if (i < j){
          temp = L->data[i];
          L->data[i] = L->data[j];
          L->data[j] = temp;
       }
    }
}
```

# 2009 年研究生入学考试计算机统考 408

## 一.单项选择题：每小题 2 分

**答案速查：BCDBC　　BADAB**

1．B【解析】考查栈和队列的特点及应用。

C 和 D 直接排除，缓冲区的特点需要先进先出，若用栈，先进入缓冲区的数据则要排 队到最后才能打印，不符题意，故选 B。

2．C【解析】考查栈的最大递归深度。

时刻注意栈的特点是先进后出。出入栈的详细过程见下表。

| 序号 | 说明 | 栈内 | 栈外 | 序号 | 说明 | 栈内 | 栈外 |
|---|---|---|---|---|---|---|---|
| 1 | a 入栈 | a | | 8 | e 入栈 | ae | bdc |
| 2 | b 入栈 | ab | | 9 | f 入栈 | aef | bdc |
| 3 | b 出栈 | a | b | 10 | f 出栈 | ae | bdcf |
| 4 | c 入栈 | ac | b | 11 | e 出栈 | a | bdcfe |
| 5 | d 入栈 | acd | b | 12 | a 出栈 | | bdcfea |
| 6 | d 出栈 | ac | bd | 13 | g 入栈 | g | bdcfea |
| 7 | c 出栈 | a | bdc | 14 | g 栈 | | bdcfeag |

栈内的最大深度为 3，故栈 S 的容量至少是 3。

3．**D**【解析】考查二叉树的特殊遍历。

分析遍历后的结点序列，可以看出根结点是在中间被访问的，而右子树结点在左子树之 前，得遍历的方法是 RNL。本题考查的遍历方法并不是二叉树的三种基本遍历方法，对于 考生而言，重要的是要掌握遍历的思想。

4．**B**【解析】考查平衡二叉树的定义。

根据平衡二叉树的定义有，任意结点的左、右子树高度差的绝对值不超过 1。而其余三个答案均可以找到不符合的结点。

5．**C**【解析】考查完全二叉树的特点。

完全二叉树比满二叉树只是在最下面一层的右边缺少了部分叶结点，而最后一层之上是个满二叉树，并且只有最后两层有叶结点。第 6 层有叶结点则完全二叉树的高度可能为 6 或 7，显然树高为 7 时结点更多。若第 6 层上有 8 个叶结点，则前六层为满二叉树，而第 7 层缺失了 8×2=16 个叶结点，故完全二叉树的结点个数最多为 27-1-16=111 个结点。

6．**B**【解析】考查森林和二叉树的转换。

森林与二叉树的转换规则为“左孩子右兄弟”。在最后生成的二叉树中，父子关系在对应森林关系中可能是兄弟关系或原本就是父子关系。

情形Ⅰ：若结点 v 是结点 u 的第二个孩子结点，在转换时，结点 v 就变成结点 u 第一个 孩子的右孩子，符合要求。

情形Ⅱ：结点 u 和 v 是兄弟结点的关系，但二者之中还有一个兄弟结点 k，则转换后， 结点 v 就变为结点 k 的右孩子，而结点 k 则是结点 u 的右孩子，符合要求。

情形Ⅲ：结点 v 的父结点要么是原先的父结点或兄弟结点。若结点 u 的父结点与 v 的父 结点是兄弟关系，则转换之后，不可能出现结点 u 是结点 v 的父结点的父结点。

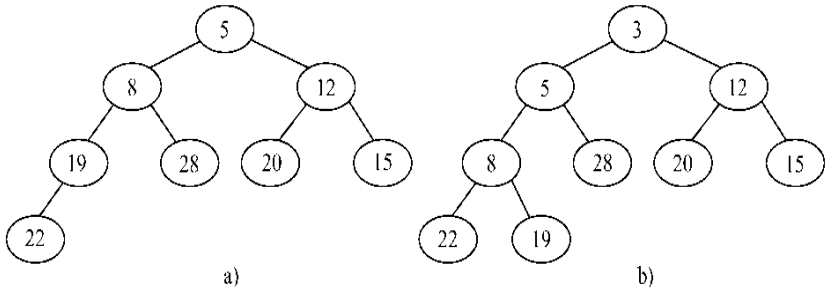
7．**A**【解析】考查无向连通图的特性。

每条边都连接了两个结点，则在计算顶点的度之和时，这条边都被计算了两次，故所有顶点的度之和为边数的两倍，显然必为偶数。而Ⅱ和Ⅲ则不一定正确，如对顶点数 N≥1 无 向完全图不存在一个顶点的度为 1，并且边数与顶点数的差要大于 1。

8．**D**【解析】考查 m 阶 B-树的定义。

选项 A、B 和 C 都是 B-树的特点，而选项 D 则是 B+树的特点。注意区别 B-树和 B+树 各自的特点。

9．**A**【解析】考查小根堆的调整操作。小根堆在逻辑上可以用完全二叉树来表示，根据关键序列得到的小顶堆的二叉树形式如下图 a 所示。




插入关键字 3 时，先将其放在小顶堆的末端，再将该关键字向上进行调整，得到的结果如图 b 所示。所以，调整后的小顶堆序列为 3，5，12，8，28，20，15，22，19。

10．**B**【解析】考查各排序算法的特点。

解答本题之前要对不同排序算法的特点极为清楚。对于冒泡排序和选择排序而言，每趟过后都能确定一个元素的最终位置，而由题目中所说，前两个元素和后两个元素均不是最小 或最大的两个元素并按序排列。答案 D 中的二路归并排序，第一趟排序结束都可以得到若 干个有序子序列，而此时的序列中并没有两两元素有序排列。插入排序在每趟排序结束后能 保证前面的若干元素是有序的，而此时第二趟排序后，序列的前三个元素是有序的，符合其 特点。

## 二.综合应用题

41.【解答】

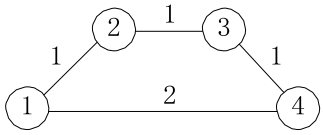



图 A-5

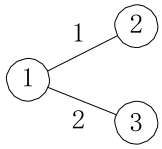



图 A-6

该方法不一定能（或不能）求得最短路径。（4 分） 举例 说明：（6 分）

图 A-5 中，设初始顶点为 1，目标顶点为 4，欲求从顶点 1 到顶点 4 之间的最短路径， 显然这两点之间的最短路径长度为 2。利用给定方法求得的路径长度为 3，但这条路径并不 是这两点之间的最短路径。

图 A-6 中，设初始顶点为 1，目标顶点为 3，欲求从顶点 1 到顶点 3 之间的最短路径。 利用给定的方法，无法求出顶点 1 到顶点 3 的路径。

【评分说明】

①若考生回答“能求得最短路径”，无论给出何种证明，均不给分。

②考生只要举出类似上述的一个反例说明“不能求得最短路径”或答案中体现了“局部最优不等于全局最优”的思想，均可给 6 分；若举例说明不完全正确，可酌情给分。

42.【解答】（1）算法的基本设计思想（5 分）： 问题的关键是设计一个尽可能高效的算法，通过链表的一趟遍历，找到倒数第 k 个结点的位置。算法的基本设计思想：定义两个指针变量 p 和 q，初始时均指向头结点的下一个结 点（链表的第一个结点）。p 指针沿链表移动，当 p 指针移动到第 k 个结点时，q 指针开始与 p 指针同步移动；当 p 指针移动到最后一个结点时，q 指针所指示结点为倒数第 k 个结点。 以上过程对链表仅进行一遍扫描。

（2）算法的详细实现步骤（5 分）：

① count=0，p 和 q 指向链表表头结点的下一个结点；

② 若 p 为空，转⑤；

③ 若 count 等于 k，则 q 指向下一个结点；否则，count=count+1；

④ p 指向下一个结点，转②；

⑤ 若 count 等于 k，则查找成功，输出该结点的 data 域的值，返回 1；否则，说明 k 值 超过了线性表的长度，查找失败，返回 0；

⑥ 算法结束。

（3）算法实现（5 分）：

```
typedef int ElemType;  //链表数据的类型定义
typedef struct LNode{  //链表结点的结构定义 ElemType    data;     //结点数据
struct Lnode *link; //结点链接指针
} *LinkList;
int   Search_k(LinkList list,int k){
//查找链表 list 倒数第 k 个结点，并输出该结点 data 域的值
LinkList p=list->link,q=list->link; //指针 p、q 指示第一个结点
int count=0;
while(p!=NULL){   //遍历链表直到最后一个结点 if(count<k) count++;     //计数，若 count<k 只移动 p
else q=q->link;p=p->link;  //之后让 p、q 同步移动
} //while if(count<k)
return 0; //查找失败返回 0
else {     //否则打印并返回 1
printf("%d",q->data); return 1;
}
```

} //Search_k

**【提示】**算法程序题，如果能够写出数据结构类型定义、正确的算法思想都会至少给一半 以上分数，如果能用伪代码写出自然更好，比较复杂的地方可以直接用文字表达。

【评分说明】

①若所给出的算法采用一遍扫描方式就能得到正确结果，可给满分 15 分；若采用两遍或多遍扫描才能得到正确结果的，最高给 10 分；若采用递归算法得到正确结果的，最高给 10 分；若实现算法的空间复杂度过高（使用了大小与 k 有关的辅助数组），但结果正确，最高给 10 分；若实现的算法的空间复杂度过高（使用了大小与 k 有关的辅助数组），但结果正 确，最高给 10 分。

②参考答案中只给出了使用 C 语言的版本，使用 C++/JAVA 语言正确实现的算法同样给 分。

③若在算法基本思想描述和算法步骤描述中因文字表达没有非常清晰地反映出算法的 思路，但在算法实现中能够清晰看出算法思想和步骤且正确，按照①的标准给分。

④若考生的答案中算法基本思想描述、算法步骤描述或算法实现中部分正确，可酌情给分。

# 2010 年研究生入学考试计算机统考 408

## 一.单项选择题

**答案速查：DCDCB ACBBD A**

1．**D**【解析】考查限定条件的出栈序列。

A 可由 in，in，in，in，out，out，in，out，out，in，out，out 得到； B 可由 in，in，in，out，out，in，out，out，in，out，in，out 得到； C 可由 in，in，out，in，out，out，in，in，out，in，out，out 得到；D 可由 in，out，in，in，in，in，in，out，out，out，out，out 得到，但题意要求不允许 连续三次退栈操作，故 D 错。

2．**C**【解析】考查受限的双端队列的出队序列。

A 可由左入，左入，右入，右入，右入得到；B 可由左入，左入，右入，左入，右入得 到；D 可由左入，左入，左入，右入，左入得到。所以不可能得到 C。

3．**D**【解析】考查线索二叉树的基本概念和构造。

题中所给二叉树的后序序列为 dbca。结点 d 无前驱和左子树，左链域空，无右子树， 右链域指向其后继结点 b；结点 b 无左子树，左链域指向其前驱结点 d；结点 c 无左子树， 左链域指向其前驱结点 b，无右子树，右链域指向其后继结点 a。

4．**C**【解析】考查平衡二叉树的插入算法。

插入 48 以后，该二叉树根结点的平衡因子由-1 变为-2，失去平衡，需进行两次旋转（先 右旋后左旋）操作。

5．**B**【解析】考查树结点数的特性。

设树中度为 i（i=0，1，2，3，4）的结点数分别为 Ni，树中结点总数为 N，则树中各结 点的度之和等于 N-1，即 N=1+N1+2N2+3N3+4N4=N0+N1+N2+N3+N4，根据题设中的数据， 即可得到 N0=82，即树 T 的叶结点的个数是 82。

6．**A**【解析】考查赫夫曼树的特性。

赫夫曼树为带权路径长度最小的二叉树，不一定是完全二叉树。赫夫曼树中没有度为 1 的结点，B 正确；构造赫夫曼树时，最先选取两个权值最小的结点作为左、右子树构造一棵 新的二叉树，C 正确；赫夫曼树中任一非叶结点 P 的权值为其左、右子树根结点权值之和， 其权值不小于其左、右子树根结点的权值，在与结点 P 的左、右子树根结点处于同一层的 结点中，若存在权值大于结点 P 权值的结点 Q，那么结点 Q 的兄

弟结点中权值较小的一个 应该与结点 P 作为左、右子树构造新的二叉树。综上可知，赫夫曼树中任一非叶结点的权 值一定不小于下一层任一结点的权值。

7．**C**【解析】考查图的连通性。

要保证无向图 G 在任何情况下都是连通的，即任意变动图 G 中的边，G 始终保持连通， 首先需要 G 的任意 6 个结点构成完全连通子图 G1，需 15 条边，然后再添一条边将第 7 个 结点与 G1 连接起来，共需 16 条边。

8．**B**【解析】考查拓扑排序序列。

图中有 3 个不同的拓扑排序序列，分别为 abced、abecd、aebcd。

9．**B**【解析】考查折半查找的过程。

具有 n 个结点的判定树的高度为 log2n +1，长度为 16，高度为 5，所以最多比较 5 次。

10．**D**【解析】考查快速排序。

递归次数与各元素的初始排列有关。如果每一次划分后分区比较平衡，则递归次数少；如果划分后分区不平衡，则递归次数多。递归次数与处理顺序无关。

11．**A**【解析】考查各种排序算法的过程。

看第一趟可知仅有 88 被移到最后。如果是希尔排序，则 12，88，10 应变为 10，12，88。因此排除希尔排序。 如果是归并排序，则长度为 2 的子序列是有序的。因此可排除归并排序。 如果是基数排序，则 16，5，10 应变为 10，5，16。因此排除基数排序。 可以看到，每一趟都有一个元素移到其最终位置，符合冒泡排序特点。

## 二.综合应用题

**41.**【解答】（1）由装载因子为 0.7，数据总数为 7，得一维数组大小为 7/0.7=10，数组下标为 0～9。 所构造的散列函数值见下表所示。

| key | 7 | 8 | 30 | 11 | 18 | 9 | 14 |
|---|---|---|---|---|---|---|---|
| H(key) | 0 | 3 | 6 | 5 | 5 | 6 | 0 |

采用线性探测再散列法处理冲突，所构造的散列表见下表所示。

| 地址 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 关键字 | 7 | 14 | | 8 | | 11 | 30 | 18 | 9 | |

（2）查找成功时，是根据每个元素查找次数来计算平均长度的，在等概率的情况下，各关键字的查找次数见下表所示。

| key | 7 | 8 | 30 | 11 | 18 | 9 | 14 |
|---|---|---|---|---|---|---|---|
| 次数 | 1 | 1 | 1 | 1 | 3 | 3 | 2 |

故 $ASL_{成功}$=查找次数/元素个数=(1+2+1+1+1+3+3)/7=12/7。 这里要特别防止惯性思维。查找失败时，是根据查找失败位置计算平均次数，根据散列函数 MOD 7，初始只可能在 0～6 的位置。等概率情况下，查找 0～6 位置查找失败的次数见下表所示。

| H(key) | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 次数 | 3 | 2 | 1 | 2 | 1 | 5 | 4 |

故，$ASL_{失败}$=查找次数/散列后的地址个数=(3+2+1+2+1+5+4)/7=18/7。

【扩展】已知一个线性序列{38,25,74,63,52,48}，假定采用散列函数 Hash(key)=key%7 计算散列地址，散列存储在表 A[10]中。如果采用线性探测法解决冲突，且各元素的查找概率相等，则在该散列表上查找不成功的平均查找长度为_____（中国科学院大学 2016）

A.2.60　　B.3.14　　C.3.71　　D.4.33

【解析】这道题也是为数不多的计算“查找不成功且线性序列个数≠key%7 的 7≠表 A[10]的 10”的题，对该方

面考查更为深刻，且翻阅西工大历年真题，仅考察过查找成功的概率，对于查找不成功的概率还未涉及，建议大家把这个知识点巩固，以防出反套路题而失分。

首先我们还是按照散列函数分别计算散列序列中的散列地址，并依次存入表 A 中，当出现 key 值相同的情况用线性探测法解决冲突（请注意，解决冲突的公式 $H_i=(H(key)+d_i)MOD\ m$，其中 H(key)为哈希函数，m 为**哈希表表长**，此时 m=10）。

Hash(38)=38%7=3，存入 A[3]中；Hash(25)=25%7=4，存入 A[4]中；Hash(74)=74%7=4，冲突，线性探测法得(4+1)MOD 10=5，存入 A[5]中；Hash(63)=63%7=0，存入 A[0]中；Hash(52)=52%7=3，冲突，线性探测法得(3+1)MOD 10=4，又冲突，…，直到线性探测法得(5+1)MOD 10=6，存入 A[6]；Hash(48)=48%7=6，冲突，线性探测法得(6+1)MOD 10=7，存入 A[7]。

因此查找成功的概率=(1+1+1+2+4+2)/6=1.83；而对于查找失败的次数，就是找到该数到下一个空白位置的次数（如果该位置为空，则次数为 1），根据散列函数 MOD7，我们只计算 A[0]~A[6]这 7 个数。比如我们与 0 号位置比较，不是我们要找的数，然后与 1 号位置相比（因为是否为空需要进入该结构，与该位置比较后才知道是否为空，除非散列之后空的位置被标记），1 号为空，证明我们查找失败了，因此查找失败次数为 2，依次类推，$ASL_{不成功}$=查找次数/散列后的地址个数=(2+1+1+6+5+4+3)/7=3.14。结果如下表所示。因此本题选 B。

| 表 A | A[0] | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] | A[9] |
|---|---|---|---|---|---|---|---|---|---|---|
| 存储元素 | 63 | | | 38 | 25 | 74 | 52 | 48 | | |
| 查找成功 | 1 | | | 1 | 1 | 2 | 4 | 2 | | |
| 查找失败 | 2 | 1 | 1 | 6 | 5 | 4 | 3 | | | |

**42.【解答】**（1）算法的基本设计思想：

可以将这个问题看作是把数组 ab 转换成数组 ba（a 代表数组的前 p 个元素，b 代表数组中余下的 n-p 个元素），先将 a 逆置得到 $a^{-1}b$，再将 b 逆置得到 $a^{-1}b^{-1}$，最后将整个 $a^{-1}b^{-1}$ 逆置得到 $(a^{-1}b^{-1})^{-1}=ba$。设 Reverse 函数执行将数组元素逆置的操作，对 abcdefgh 向左循环移动 3（p=3）个位置的过程如下：

Reverse(0,p-1)得到 cbadefgh； Reverse(p,n-1)得到 cbahgfed； Reverse(0,n-1)得到 defghabc。

注：Reverse 中，两个参数分别表示数组中待转换元素的始末位置。

（2）使用 C 语言描述算法如下：

```
void Reverse(int R[],int from,int to) {
    int i,temp;
    for(i=0;i<(to-from+1)/2;i++){
        temp=R[from+i];R[from+i]=R[to-i];R[to-i]=temp;}
}//Reverse
```

```
void Converse(int R[],int n,int p){
    Reverse(R,0,p-1);
    Reverse(R,p,n-1);
    Reverse(R,0,n-1);
}
```

（3）上述算法中 3 个 Reverse 函数的时间复杂度分别为 O(p/2)、O((n-p)/2)和 O(n/2)，故所设计的算法的时间复杂度为 O(n)，空间复杂度为 O(1)。

**【另解】**借助辅助数组来实现。

算法思想：创建大小为 p 的辅助数组 S，将 R 中前 p 个整数依次暂存在 S 中，同时将 R 中后 n-p 个整数左移，然后将 S 中暂存的 p 个数依次放回到 R 中的后续单元。时间复杂度为 O(n)，空间复杂度为 O(p)。

# 2011 年研究生入学考试计算机统考 408

## 一.单项选择题

**答案速查：ABBCC DACDA B**

1．**A**【解析】考查时间复杂度的计算。

在程序中，执行频率最高的语句为“x=2*x”。设该语句共执行了 t 次，则 $2^{t+1}=n/2$，故 t=log2(n/2)-1=log2n-2，得 T(n)=O(log2n)。

2．**B**【解析】考查出栈序列。

出栈顺序必为 d_c_b_a_，e 的顺序不定，在任一个“_”上都有可能。

3．**B**【解析】考查循环队列的性质。

入队时由于要执行(rear+1)%n 操作，所以如果入队后指针指向 0，则 rear 初值为 n-1，而由于第一个元素在 A[0]中，插入操作只改变 rear 指针，所以 front 为 0 不变。

4．**C**【解析】考查完全二叉树的性质。

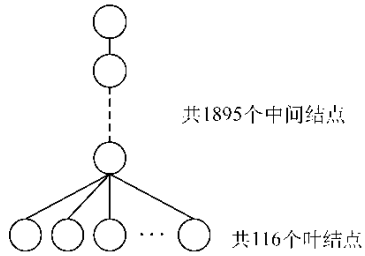
根据完全二叉树的性质，最后一个分支结点的序号为 768/2=384，故叶子结点的个数为 768-384=384。

5．**C**【解析】考查二叉树的遍历算法。

前序序列为 LRN，后序序列为 NLR，由于前序序列和后序序列刚好相反，故不可能存在一个结点同时存在左右孩子，即二叉树的高度为 4。1 为根结点，由于根结点只能有左孩子（或右孩子），因此，在中序序列中，1 或在序列首或在序列尾，ABCD 皆满足要求。仅考虑以 1 的孩子结点 2 为根结点的子树，它也只能有左孩子（或右孩子），因此，在中序序列中，2 或在序列首或序列尾，ABD 皆满足要求。

6．**D**【解析】考查树和二叉树的转换。

树转换为二叉树时，树中每一个分支结点的所有子结点中的最右子结点无右孩子，根结点转换后也没有右孩子， 因此，对应的二叉树中无右孩子的结点个数=分支结点数+1=2011-116+1=1896。通常本题应采用特殊法解，设题意中的树是如下图所示的结构，则对应的二叉树中仅有前 115 个叶结点有右孩子，故无右孩子的结点个数=2011-115=1896。




7．**A**【解析】考查二叉排序树的查找过程。

在二叉排序树中，左子树结点值小于根结点，右子树结点值大于根结点。在选项 A 中，当查找到 91 后再向 24 查找，说明这一条路径（左子树）之后查找的数都要比 91 小，而后面却查找到了 94，因此错误。

8．**C**【解析】考查图的基本概念。

回路对应于路径，简单回路对应于简单路径，故 I 错误；稀疏图是边比较少的情况，此时用邻接矩阵必将浪费大量的空间，应该选用邻接表，故 II 错误。存在回路的图不存在拓扑 序列，故III正确。

9．**D**【解析】考查散列表的性质。

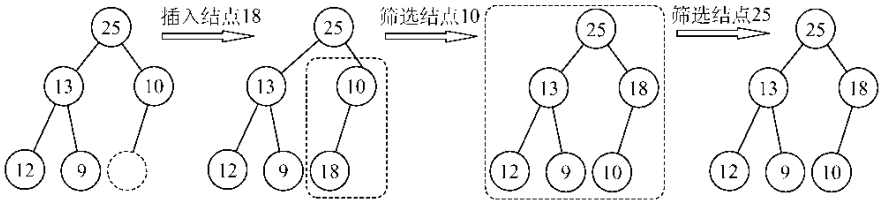
Hash 表的查找效率取决于：哈希函数、处理冲突的方法和装填因子。显然，冲突的产生概率与装填因子（即表中记录数与表长之比）的大小成正比， I 错误。冲突是不可避免的，但处理冲突的方法应避免非同义词之间地址的争夺，III正确。

10．**A**【解析】考查排序的基本特点。

对绝大部分内部排序而言，只适用于顺序存储结构。快速排序在排序的过程中，既要从后向前查找，也要从前向后查找，因此宜采用顺序存储。

11．**B**【解析】考查堆的调整。

首先 18 与 10 比较，交换位置，再与 25 比较，不交换位置。共比较了 2 次，调整的过程如下图所示。



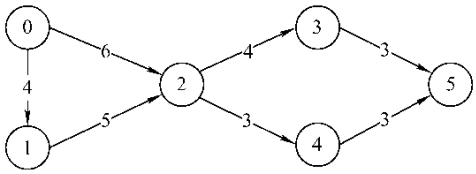

## 二.综合应用题

41.【解答】(1) 用“平移”的思想，将前 5 个、后 4 个、后 3 个、后 2 个、后 1 个元素，分别移动到矩阵对

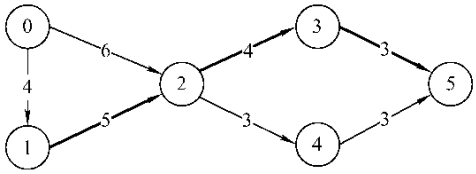
角线（“0”）右边的行上。图 G 的邻接矩阵 ***A*** 如下所示。

$$A=\begin{pmatrix} 0 & 4 & 6 & \infty & \infty & \infty \\ \infty & 0 & 5 & \infty & \infty & \infty \\ \infty & \infty & 0 & 4 & 3 & \infty \\ \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{pmatrix}$$

（2）根据上面的邻接矩阵，画出有向带权图 G，如下图所示。




3）即寻找从 0 到 5 的最长路径。得到关键路径为 0→1→2→3→5（如下图所示粗线表示），长度为 4+5+4+3=16。




42.【解答】（1）算法的基本设计思想如下。

分别求出序列 A 和 B 的中位数，设为 a 和 b，求序列 A 和 B 的中位数过程如下：

① 若 a=b，则 a 或 b 即为所求中位数，算法结束。

② 若 a<b，则舍弃序列 A 中较小的一半，同时舍弃序列 B 中较大的一半，要求舍弃的长度相等。

③ 若 a>b，则舍弃序列 A 中较大的一半，同时舍弃序列 B 中较小的一半，要求舍弃的长度相等。

在保留的两个升序序列中，重复过程 1)、2)、3)，直到两个序列中只含一个元素时为止，较小者即为所求的中位数。

（2）算法的实现如下：

```
int M_Search(int A[],int B[],int n){
    //分别表示序列 A 和 B 的首位数、末位数和中位数
    int s1=0,d1=n-1,m1,s2=0,d2=n-1,m2;
    while(s1!=d1||s2!=d2){
      m1=(s1+d1)/2;
      m2=(s2+d2)/2;
      if(A[m1]==B[m2])
        return A[m1];    //满足条件 1）
      if(A[m1]<B[m2]){   //满足条件 2）
        if((s1+d1)%2==0){ //若元素个数为奇数
          s1=m1;     //舍弃 A 中间点以前的部分，且保留中间点
          d2=m2;     //舍弃 B 中间点以后的部分，且保留中间点
        }
        else{//元素个数为偶数
          s1=m1+1;  //舍弃 A 中间点及中间点以前部分
          d2=m2;     //舍弃 B 中间点以后部分且保留中间点
        }
```

```
            }
            else{   //满足条件 3）
              if((s1+d1)%2==0) {//若元素个数为奇数
                 d1=m1;     //舍弃 A 中间点以后的部分，且保留中间点
                 s2=m2;     //舍弃 B 中间点以前的部分，且保留中间点
              }
              else{//元素个数为偶数
                 d1=m1;     //舍弃 A 中间点以后部分，且保留中间点
                 s2=m2+1;  //舍弃 B 中间点及中间点以前部分
              }
            }
        }
    return A[s1]<B[s2]? A[s1]:B[s2];
    }
```

（3）算法的时间复杂度为 $O(\log_2 n)$，空间复杂度为 O(1)。

# 西北工业大学 2014 年研究生入学考试

## 一.简述题(20 分)

**1.**相同点：n 个同类数据元素的有限序列称为线性表。线性表的特点是数据元素之间存在“一对一”的关系，栈和队列都是操作受限制的线性表，它们和线性表一样，数据元素之间都存在“一对一”的关系。

不同点：栈是只允许在一端进行插入或删除操作的线性表，其特点是“后进先出”，队列是只允许在一端进行插入，另一端进行删除操作的线性表，其特点是“先进先出”。

**2.**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。

存在条件：图 G 为连通图

**3.**(题意不是很明确，暂且理解为写出一个稳定的排序算法)

直接插入排序算法、冒泡排序算法、归并排序算法、基数排序算法等

## 二.应用题(20 分)

**1.**【注】参考《西北工业大学 2003-2004 学年第二学期期末考试》三计算题 3 小题

**2.**【注】参考《西北工业大学 2000 研究生入学考试》三算法应用 2 小题

## 三.算法描述题(20 分)

**1.**拓扑排序的算法思想：

(1)从 DAG 图(有向无环图)中选择一个没有前驱的顶点并输出；

(2)从图中删除该顶点和所有以它为起点的有向边；

(3)重复(1)和(2)直到当前 DAG 图为空或当前图中不存在无前驱的顶点为止。而后一种情况则说明有向图中必然存在环。

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈，存储入度为 0 的顶点
    for(i = 0; i < G.vexnum;i ++)
        if(indegree[i] == 0)
            Push(S, i);
    int count = 0; //记录当前已经输出的顶点数
    while(!IsEmpty(S)){
        Pop(S, i);
        print[count ++] = i;
        for(p = G.vertices[i].firstarc; p; p=p->nextarc) {
            v = p->adjvex;
            if(!(--indegree[v]))
                Push(S, v);
        }
    }
    if(count < G.vexnum)
        return false;
    else
        return true;
}
```

**2.【注】参考《西北工业大学 2002 年研究生入学考试》五题**

**四.程序设计题(15 分)**

算法用 C 语言描述如下（其中 ET 为数据元素类型）:

```
 struct  node  {                          /*  定义线性单链表结点类型  */
    ET  d ;                              /*  定义线性单链表结点数据类型  */
    struct  node  * next ;                    /*  结点指针  */
 } ;
# include  " stdio.h "
void  mglst ( struct  node *ah ,  struct  node *bh , struct  node ** ch ) {
    struct  node  * i , * j , * k , * p ;
    i = ah ;  j = bh; *ch = NULL ;  k = NULL ;
   while ( ( i != NULL )  &&  ( j != NULL ) )
    { p = ( struct  node * ) malloc ( sizeof ( struct  node ) ) ;  /*  取得一个新结点  */
      if ( j->d >= i->d ) { p->d = i->d ;  i = i->next ; }
      else  { p->d = j->d ;  j = j->next ; }
      if ( *ch = = NULL ) *ch = p ;
      else  k->next = p ;
      k = p ;
    }
if ( j = = NULL )
    while ( i != NULL )
     { p = ( struct  node * ) malloc ( sizeof ( struct  node ) ) ;  /*  取得一个新结点  */
       p->d = i->d ;  i = i->next ;
       if ( *ch = = NULL ) *ch = p ;
      else  k->next = p ;
      k = p ;
    }
 else
 while ( j != NULL )
     { p = ( struct  node * ) malloc ( sizeof ( struct  node ) ) ;  /*  取得一个新结点  */
       p->d = j->d ;  j = j->next ;
 if ( *ch = = NULL ) *ch = p ;
```

```
      else   k->next = p ;
      k = p ;
    }
 if ( k != NULL ) k->next = NULL ;
 return ;
}
```

# 西北工业大学 2015 年研究生入学考试

## 一.单项选择题(共 10 小题，每小题 2 分，本题满分共 20 分)

**答案速查：DBAAC BCADA**

**4.**A【解析】先序遍历序列为 ABEKLFCMDHI

**5.**C【解析】设叶子结点有 m 个，则总结点为 m+n 个，而森林转化成树是根据左孩子右兄弟存储原理转化过来的，当转成树时，每个结点有两个指针域，分别是左孩子域和右兄弟域，所以总的左、右指针域分别有 m+n 个。并且有下面的等量关系：

空的右指针域个数+不为空的右指针域个数=总的右指针域个数=m+n；不为空的右指针域个数+不为空的左指针域个数=总的不为空的指针域个数=m+n-1(因为总结点为 m+n 个)；

而原来森林当中有 n 个非终端结点，所以有且仅有 n 个结点有孩子，因此转化成树时有 n 个不为空的左指针域个数，代入上面的第二条式子得不为空的右指针域个数为 m-1，再代入第一条式子，得：空的右指针域个数为 n+1。【注】也可以直接画图举例

**7.**C【解析】在某中，若删除顶点 V 以及 V 相关的边后，图的一个连通分量分割为两个或两个以上的连通分量，则称顶点 V 为该图的一个关节点。

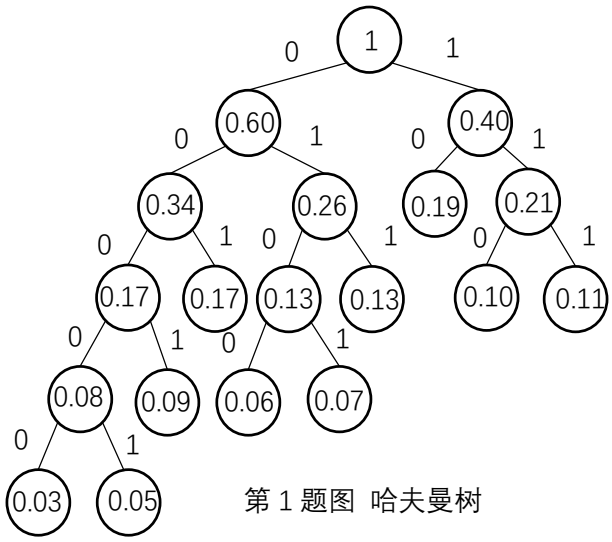
## 二.综合应用题(4 小题，每小题 10 分，本题满分 40 分)

**1.**哈夫曼编码(见右图)

0.03：00000　0.05：00001　.06：0100　0.07：0101

0.09：0001　0.10：110　0.11：111　0.13：011

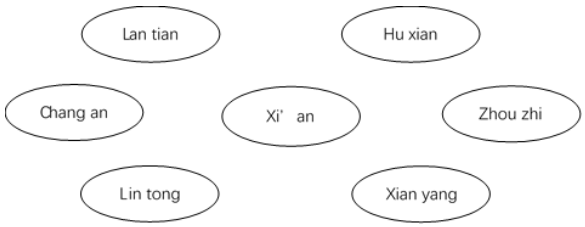
0.17：001　0.19：10



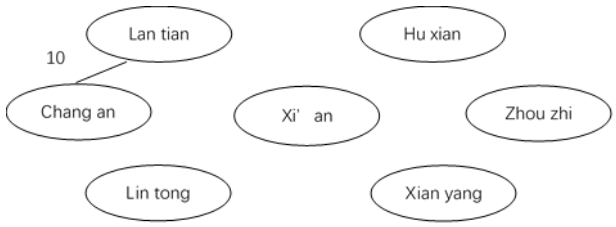

第 1 题图 哈夫曼树

**2.**这是一个最小生成树问题，可用 Prim 算法或 Kruskal 算法解决。

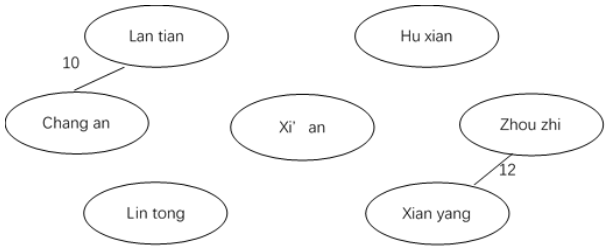
以 Kruskal 算法为例：先对所有的边按从小到大的顺序排序，再依次将边加入图中，若加入后不构成环路，则将该边作为构成最小生成树的边。如此反复，直到图中有 6 条边为止(见下图)。



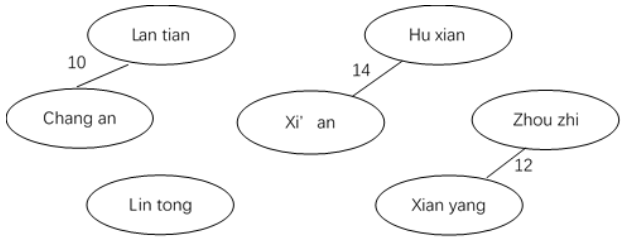

(1)



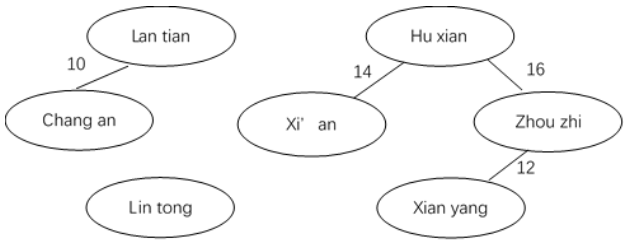

(2)



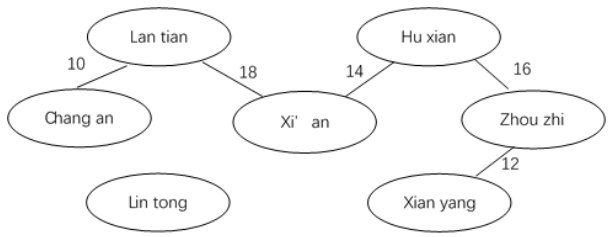

(3)



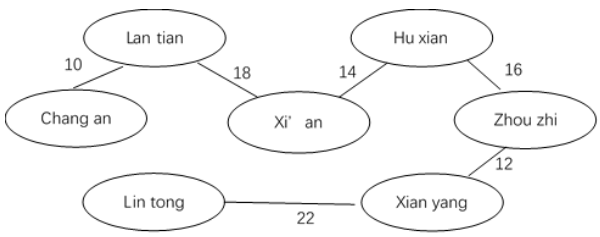

(4)




(5)



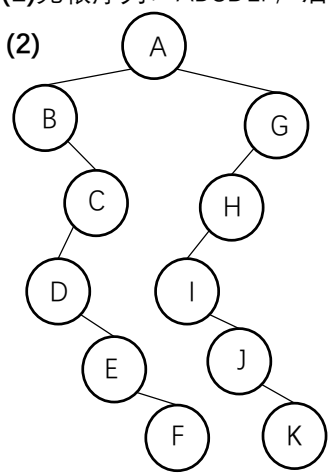

(6)



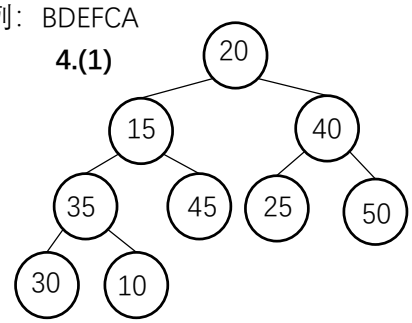

(7)

**3.(1)**先根序列：ABCDEF；后根序列：BDEFCA

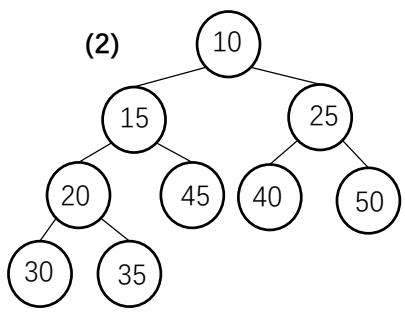
**(2)**



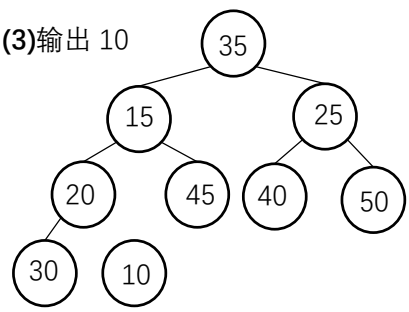

**4.(1)**



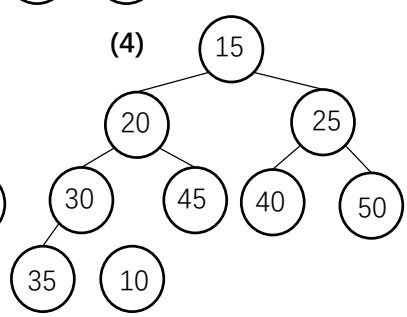

**(2)**




**(3)**输出 10




**(4)**



三.算法设计与分析题(本题满分 15 分)

算法思想：两个链表已经按元素值递增次序排序，将其合并时，均从第一个结点起进行比较，将小的结点链入链表中，同时后移工作指针。要求结果链表按元素递减次序排列，故新链表的建立应采用头插法。比较结束后，可能会有一个链表非空，此时用头插法将剩下的结点依次插入新链表即可。

```
void MergeList(LNode* &list1, LNode* &list2) {
    LNode * r, *pa=list1->next, *pb=list2->next;
    list1->next = list1; //新链表
    while(pa != list1 && pb != list2) { //单循环链表
    //while 语句中的判断条件理解为：任意一个指针回到头部，终止循环
        if(pa->data <= pb->data){
            r = pa->next;
            pa->next = list1->next;
            list1->next = pa;
            pa = r;
        }
        else {
            r = pb->next;
            pb->next = list1->next;
            list1->next = pb;
            pb = r;
        }
    }
    if(pa != list1)
        pb = pa;
    while(pb != list1 && pb != list2) {
        r = pb->next;
        pb->next = list1->next;
        list1->next = pb;
        pb = r;
    }
    free(list2);
}
```

时间复杂度 O(m+n)，m 和 n 分别为链表长度；空间复杂度 O(1)。

# 西北工业大学 2016 年研究生入学考试

一.简答题(每题 5 分，共 25 分)

**1.**用穷举的方法

```
for(i = 0;i <= n/a;i ++){
    for(j = 0;j <= n/b; j++){
        k = n-i-j;
        if( k%c == 0 && i*a+j*b+k/c == n) {
            printf("%d %d %d\n", i,j,k);
        }
    }
}
```

时间复杂度 $O(n^2/(ab))$

**2.**在升序链表 h 中插入值为 x 的新结点，先在 h 中找到 x 应插入的位置，即前一个结点的值小于等于 x，后一个大于 x，再将新结点插入该位置。

```
void Insert(LNode* &h, int x){
    LNode* p = h->next, *pre = h;
    while(p && p->data <= x) {
        pre = p;
        p = p->next;
    }
    LNode* s = (LNode*)malloc(sizeof(LNode));
    s->data=x;
    pre->next = s;
    s->next = p;
    return;
}
```

**3.**采用三元组存储稀疏矩阵，用 i 和 j 两个变量扫描两个三元组 a 和 b，按行序优先的原则进行处理，将结果存放于 c 中。当 a 的当前元素和 b 的当前元素的行号和列号均相等时，将它们的值相加，只有在相加值不为 0 时，才在 c 中添加一个新的元素。

```
typedef struct{
    int r;                          //行号
    int c;                          //列号
    int d;                          //元素值
} TupNode;                           //三元组定义
typedef struct{
    int rows;                       //行数
    int cols;                       //列数
    int nums;                       //非零元素个数
    TupNode data[MaxSize];
} TSMatrix;                         //三元组顺序表定义
bool MatAdd(TSMatrix a,TSMatrix b,TSMatrix &c){
    int i=0,j=0,k=0;
    int v;
    if (a.rows!=b.rows || a.cols!=b.cols)
        return 0;                               //行数或列数不等时不能进行相加运算
    c.rows=a.rows;
    c.cols=a.cols;                              //c 的行列数与 a 的相同
    while (i<a.nums && j<b.nums) {              //处理 a 和 b 中的每个元素
        if (a.data[i].r==b.data[j].r) {         //行号相等时
            if(a.data[i].c<b.data[j].c) {       //a 元素的列号小于 b 元素的列号
                c.data[k].r=a.data[i].r;        //将 a 元素添加到 c 中
                c.data[k].c=a.data[i].c;
                c.data[k].d=a.data[i].d;
                k++;
                i++;
            }
            else if (a.data[i].c>b.data[j].c) {     //a 元素的列号大于 b 元素的列号
                c.data[k].r=b.data[j].r;            //将 b 元素添加到 c 中
                c.data[k].c=b.data[j].c;
                c.data[k].d=b.data[j].d;
                k++;
                j++;
            }
            else {                                  //a 元素的列号等于 b 元素的列号
                v=a.data[i].d+b.data[j].d;
                if (v!=0) {                         //只将不为 0 的结果添加到 c 中
                    c.data[k].r=a.data[i].r;
                    c.data[k].c=a.data[i].c;
                    c.data[k].d=v;
                    k++;
                }
                i++;
                j++;
```

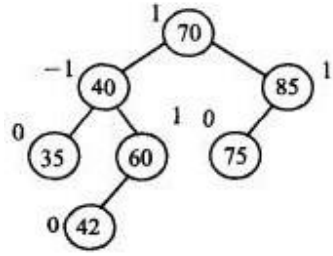
```
            }
        }
        else if (a.data[i].r<b.data[j].r) {          //a 元素的行号小于 b 元素的行号
            c.data[k].r=a.data[i].r;                 //将 a 元素添加到 c 中
            c.data[k].c=a.data[i].c;
            c.data[k].d=a.data[i].d;
            k++;
            i++;
        }
        else {                                       //a 元素的行号大于 b 元素的行号
            c.data[k].r=b.data[j].r;                 //将 b 元素添加到 c 中
            c.data[k].c=b.data[j].c;
            c.data[k].d=b.data[j].d;
            k++;
            j++;
        }
        c.nums=k;
    }
    return true;
}
```

**4.**广度优先搜索：

首先访问起始顶点 v，接着由 v 出发，依次访问 v 的各个未访问过的邻接顶点 $w_1,w_2,...,w_i$，然后再依次访问 $w_1,w_2,...,w_i$ 的所有未被访问过的邻接顶点；再从这些未访问过的顶点出发，再访问它们所有未被访问的邻接顶点……直到图中所有顶点都被访问过为止。算法借助一个辅助队列，以记忆正在访问的顶点的下一层顶点。

```
bool visited[vexnum];  //访问标记数组
void BFSTraverse(Graph G){
//对图 G 进行广度优先遍历，设访问函数为 visit()
    for(i = 0;i < G.vexnum;i ++)
        visited[i] = false;  //访问标记数组初始化
    InitQueue(Q);                    //初始化辅助队列 Q
    for(i = 0;i < G.vexnum;i ++)     //从 0 号顶点开始遍历
        if(!visited[i])          //对每个连通分量调用一次 BFS
            BFS(G, i);
}
void BFS(Graph G, int v){
    visit(v);
    visited[v] = true;
    EnQueue(Q, v);         //入队
    while(!isEmpty(Q)){
        DeQueue(Q, v);     //出队
        for(w = FirstNeighbor(G,v);w >= 0;w = NextNeighbor(G, v, w)) {
            if(!visited[w]){
                visit(w);
                visited[w] = true;
```

```
                EnQueue(Q, w);
            }
        }
    }
}
```




**5.**平衡二叉树是一棵空树或它的左右两个子树的高度差的绝对值不超过 1，并且左右两个子树都是一棵平衡二叉树。举例(见右图):

**二.应用题(每题 15 分，共 30 分)**

**1.**将中序表达式转换为逆波兰式:

**(1)**首先，需要分配 2 个栈，栈 s1 用于临时存储运算符(含一个结束符号)，此运算符在栈内遵循越往栈顶优先级越高的原则；栈 s2 用于输入逆波兰式，为方便起见，栈 s1 需先放入一个优先级最低的运算符，在这里假定为'#';

**(2)**从中缀式的左端开始逐个读取字符 x，逐序进行如下步骤:

若 x 是操作数(字母)，将 x 直接压入栈 s2;

若 x 是运算符，则分情况讨论:

①若 x 是'('，则直接压入栈 s1;

②若 x 是')'，则将距离栈 s1 栈顶的最近的'('之间的运算符，逐个出栈，依次压入栈 s2，此时抛弃'(';

③若 x 是除'('和')'外的运算符，则再分如下情况讨论:

若当前栈 s1 的栈顶元素为'('，则将 x 直接压入栈 s1;

若当前栈 s1 的栈顶元素不为'('，则将 x 与栈 s1 的栈顶元素比较，若 x 的优先级大于栈 s1 栈顶运算符优先级，则将 x 直接压入栈 s1。否者，将栈 s1 的栈顶运算符弹出，压入栈 s2 中，直到栈 s1 的栈顶运算符优先级别低于(不包括等于)x 的优先级，或栈 s2 的栈顶运算符为'('，此时再则将 x 压入栈 s1;

**(3)**在进行完(2)后，检查栈 s1 是否为空，若不为空，则将栈中元素依次弹出并压入栈 s2 中(不包括'#');

**(4)**完成上述步骤后，栈 s2 便为逆波兰式输出结果。但是栈 s2 应做一下逆序处理，因为此时表达式的首字符位于栈底。

```
char *RPExpression(char *e){
 /* 返回表达式 e 的逆波兰式 */
    //栈 s1 用于存放运算符，栈 s2 用于存放逆波兰式
      Stack s1,s2;
      InitStack(s1);
      InitStack(s2);
      Push(s1,'#');  //假设字符'#'是运算级别最低的运算符，并压入栈 s1 中
      //p 指针用于遍历传入的字符串，ch 用于临时存放字符,length 用于计算字符串长度
      char *p=e,ch;
      int length=0;
      for(;*p!='\0';p++) {              //逐个字符访问
          switch(*p)
          {
              case '(': //遇'('则直接入栈 s1
                  Push(s1,*p);
                  break;
//遇')'则将距离栈 s1 栈顶的最近的'('之间的运算符，逐个出栈，依次送入栈 s2,此时抛弃'('
```

```
            case ')':
                while(Top(s1)!='(') {
                    Pop(s1,ch);
                    Push(s2,ch);
                }
                Pop(s1,ch);
                break;
            //遇下列运算符，则分情况讨论：
            //1.若当前栈 s1 的栈顶元素是'('，则当前运算符直接压入栈 s1；
            //2.否则，将当前运算符与栈 s1 的栈顶元素比较，若优先级较栈顶元素大，则直接压入栈 s1 中；
            //  否则将 s1 栈顶元素弹出，并压入栈 s2 中，直到栈顶运算符的优先级别
低于当前运算符，然后再将当前运算符压入栈 s1 中
            case '+':
            case '-':
                for(ch=Top(s1);ch!='#';ch=Top(s1)) {
                    if(ch=='(') {
                        break;
                    }
                    else {
                        pop(s1,ch);
                        push(s2,ch);
                    }
                }
                push(s1,*p);
                length++;
                break;
            case '*':
            case '/':
                for(ch=Top(s1);ch!='#'&&ch!='+'&&ch!='-';ch=Top(s1)) {
                    if(ch=='(') {
                        break;
                    }
                    else {
                        Pop(s1,ch);
                        Push(s2,ch);
                    }
                }
                Push(s1,*p);
                length++;
                break;
            default: //遇操作数则直接压入栈 s2 中
                Push(s2,*p);
                length++;
        }
```

```
        }
        //若栈 s1 非空，则将栈中元素依次弹出并压入栈 s2 中
          while(!StackEmpty(s1)&&Top(s1)!='#') {
                Pop(s1,ch);
                Push(s2,ch);
        }
        //最后将栈 s2 输出，逆序排列成字符串;
        char *result;
        result=(char *)malloc(sizeof(char)*(length+1));
        result+=length;
        *result='\0';
        result--;
        for(;!StackEmpty(s2);result--) {
                Pop(s2,ch);
                *result=ch;
        }
        ++result;
        return result;
 }
```

**2.(1)**中序遍历二叉树

```
void InOrder(BiTNode* T) {
    if(T==NULL) return ;
    InOrder(T->lchild);
    visit(T);
    InOrder(T->rchild);
}
```

**(2)**遍历中序线索二叉树

```
BiThrNode* Firstnode(BiThrNode* p) { //求中序线索二叉树中序序列下的第一个结点
    while(p->Ltag == 0)
        p = p->lchild;   //最左下结点
    return p;
}
//求中序线索二叉树中结点 p 在中序序列下的后继结点
BiThrNode* Nextnode(BiThrNode* p) {
    if(p->Rtag == 0)
        return Firstnode(p->rchild);
    else
        return p->rchild;
}
void Inorder(BiThrNode* T) { //中序遍历算法
    for(BiThrNode* p = Firstnode(T);p != NULL; p = Nextnode(p))
        visit(p);
}
```

三.算法设计与分析题(20 分)

**(1)**堆的定义

n 个元素的序列{k1,k2,ki,…,kn}当且仅当满足下关系时，称之为堆。

($k_i <= k_{2i}, k_i <= k_{2i+1}$)或者($k_i >= k_{2i}, k_i >= k_{2i+1}$), (i = 1,2,3,4...n/2)

若将和此次序列对应的一维数组(即以一维数组作此序列的存储结构)看成是一个完全二叉树，则堆的含义表明，完全二叉树中所有非终端结点的值均不大于(或不小于)其左、右孩子结点的值。

**(2)**堆的插入

每次插入都是将先将新数据放在数组最后，由于从这个新数据的父结点到根结点必然为一个有序的序列，现在的任务是将这个新数据插入到这个有序序列中(自下向上调整)。

```
void HeapInsert(int *heap, int n, int num) {
    int i, j;
    heap[n+1] = num;//num 插入堆尾
    i = n+1;
    j = (n+1) / 2;//j 指向 i 的父结点
    while (j > 0) {
        if (heap[j] <= num)
            break;
        heap[i] = heap[j];
        i = j;
        j = i/2;
    }
    heap[i] = num;
    return;
}
```

**(2)**堆的删除

堆中每次都只能删除堆顶元素。为了便于重建堆，实际的操作是将最后一个数据的值赋给根结点，然后再从根结点开始进行一次从上向下的调整。调整时先在左右子结点中找最小的，如果父结点比这个最小的子结点还小说明不需要调整了，反之将父结点和它交换后再考虑后面的结点。

```
int HeapDelete(int *heap, int n) {
    //使用堆尾元素直接覆盖堆顶元素。
    heap[1] = heap[n];
    //从堆顶到堆尾(此时堆中只有 n-1 个元素)进行堆调整。
    HeapAdjust(heap, 1, n - 1);
    return 0;
}
void HeapAdjust(int *heap, int top, int n) {
    int j = 2 * top;      //左孩子结点
    int temp = heap[top];
    while (j <= n) {
        if (j + 1 <= n&&heap[j + 1] < heap[j])
            //使 j 指向左右孩子中较小的结点。
            j++;
        if (heap[j] >= temp)
            break;
        heap[top] = heap[j];
        top = j;
        j = 2 * top;
    }
    heap[top] = temp;
    return;
}
```

**(3)**建堆

```
void CreatHeap(int *heap, int n) {
    int i;
```

```
    for (i = n/2; i > 0; i--) {
        HeapAdjust(heap, i, n);
    }
    return;
}
```

**(4)**堆排序

在输出堆顶的最小值之后，以堆中最后一个元素替代之，此时根结点的左右子树均为堆，则仅需进行一次从上到下的调整即可重建一个堆。堆顶元素即是 n 个元素中的次小值。如此反复执行，便能得到一个有序序列。

```
void HeapSort(int *heap, int n){
    CreatHeap(heap, n);
    int i;
    int temp;
    for (i = n; i > 1; i--) {
        //将堆顶元素和未排序的最后一个元素交换。
        temp = heap[1];
        heap[1] = heap[i];
        heap[i] = temp;
        //交换之后进行堆调整
        HeapAdjust(heap, 1, i-1);
    }
    return;
}
```

建堆时间复杂度 O(n)；向下调整时间复杂度 O(h)，h 为树高；堆的插入和删除元素的复杂度也为 O(h)；堆排序时间复杂度为$O(n\log_2 n)$。

# 西北工业大学 2017 年研究生入学考试

## 一.简述题(15 分，每小题 5 分)

**1.**相同点：n 个同类数据元素的有限序列称为线性表。线性表的特点是数据元素之间存在“一对一”的关系，栈和队列都是操作受限制的线性表，它们和线性表一样，数据元素之间都存在“一对一”的关系。

不同之处：栈是只允许在一段进行插入或删除操作的线性表，其特点是“后进先出”，队列是只允许在一端进行插入，另一端进行删除操作的线性表，其特点是“先进后出”。

**2.**拓扑序列：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，若边(u,v)∈E(G)，则 u 在线性序列中出现在 v 之前。将这样的线性序列称为拓扑序列。

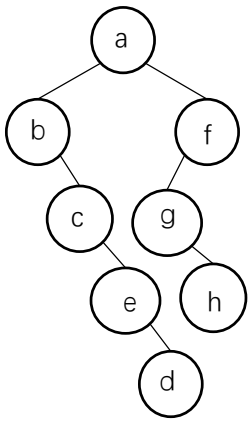
存在条件：图 G 为有向无环图。

**3.**满二叉树的结点特征：满二叉树高度为 h，含有$2^h-1$个结点的二叉树，即树中的每一都含有最多的结点。叶子结点都集中在满二叉树的最下一层，并且除了叶子结点之外的每个结点度数为 2。

深度：$\lceil\log_2(n+1)\rceil$或$\lfloor\log_2 n\rfloor+1$。

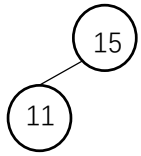
二.应用题(共 30 分，每小题 10 分)

1.

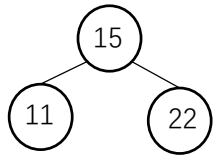



后序序列：decbhgfa

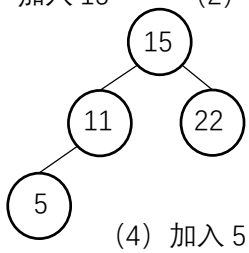
**2.** 15 11 22 5 66 58 36 10 38

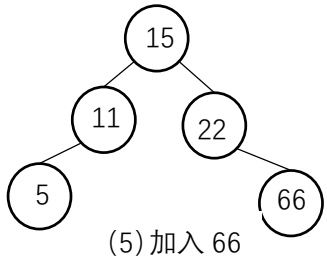

（1）加入 15　（2）加入 11



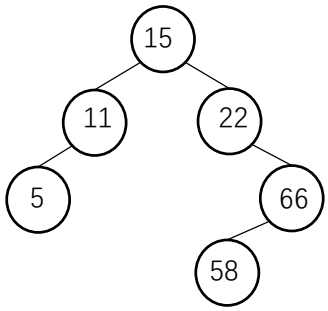

（3）加入 22

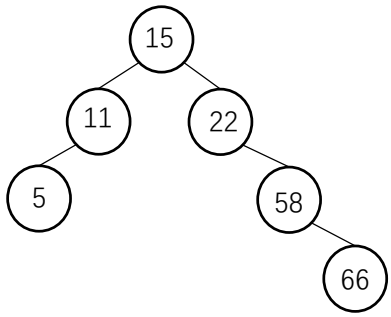


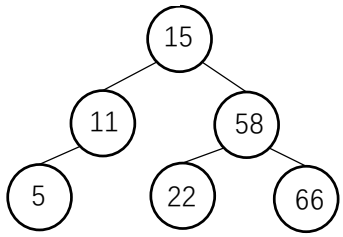

（4）加入 5



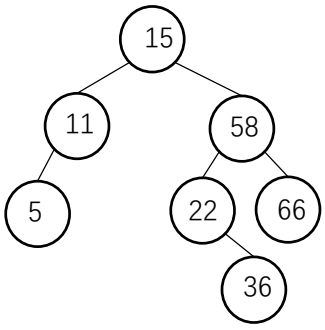

(5) 加入 66



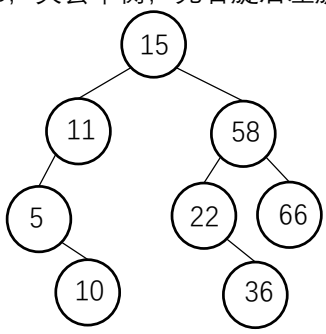



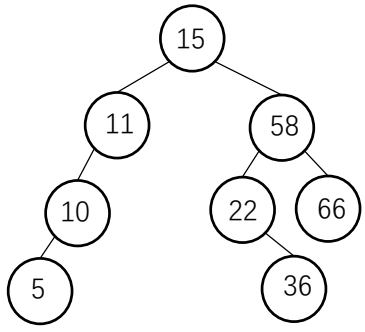




（6）加入 58，失去平衡，先右旋后左旋

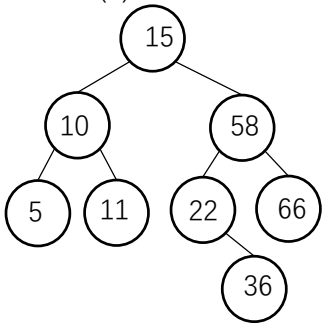



(7)加入 36

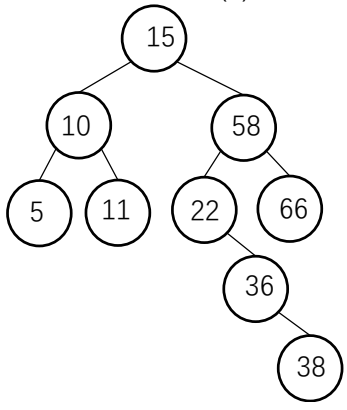


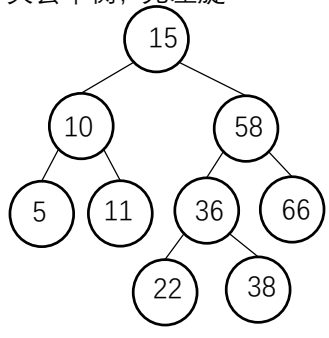




(8)加入 10，失去平衡，先左旋



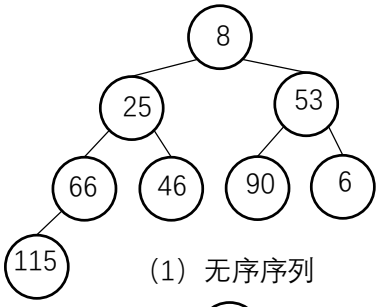

(8)后右旋



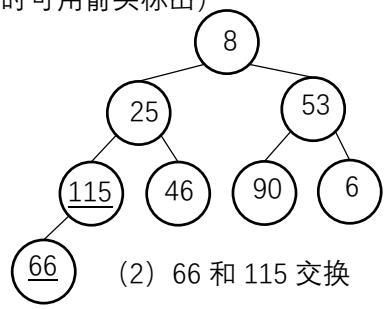




(9)加入 38，失去平衡，先左旋后右旋

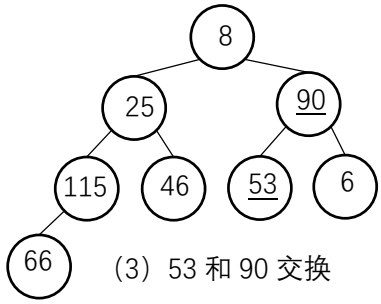
**3.** 建初始大顶堆过程（注：元素交换时可用箭头标出）



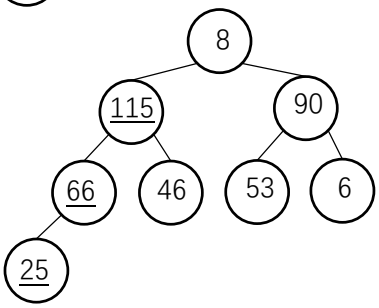

（1）无序序列



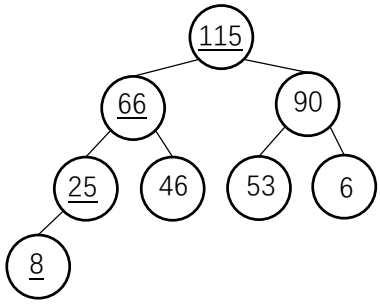

（2）66 和 115 交换




（3）53 和 90 交换




（4）25 和 115 交换,25 和 66 交换




（5）8 和 115 交换，8 和 66 交换，8 和 25 交换，大顶堆构造完成

三.算法描述图(20 分，每小题 10 分)

**1.**从图 G 中任一点开始进行深度优先搜索，若图中所有点都可遍历到，即只有一个连通分量，则这个图连通。

```
bool visited[vexnum];   //访问标记数组
void DFS(Graph G, int v){
      visit(v);
      visited[v]=true;
      for(w = 0; w < G.vexnum;w ++){
            if(G.edge[v][w] == 1 && !visited[w]) {
                  DFS(G,w);
            }
      }
}
```

```
bool Judge(Graph G){
      for(v = 0;v < G.vexnum; v ++)
            visited[v] = false;
      dfs(G,0); //从任一点遍历
      for(v = 0;v < G.vexnum; v ++){
            if(!visited[v])
                  return false;
      }
      return true;
}
```

**2.**采用递归的方法

```
int Degree(BiTree   t){
      if(!t)   //根节点为空
            return 0;
      else if (t->lchild == NULL && t->rchild == NULL)//只有根节点
            return 0;
      else if (t->lchild != NULL   &&   t->rchild == NULL) {   //有左孩子没有右孩子
            printf("%c "',t);
            return 1+ Degree(t->lchild);
      }
      else if (t->lchild == NULL && t->rchild != NULL) {   //有右孩子没有左孩子
            printf("%c ",t);
            return 1 + Degree(t->rchild);
```

```
    }
    else if (t->lchild != NULL && t->rchild != NULL)     //左右孩子都有
        return    Degree(t->lchild) + Degree(t->rchild);
}
```

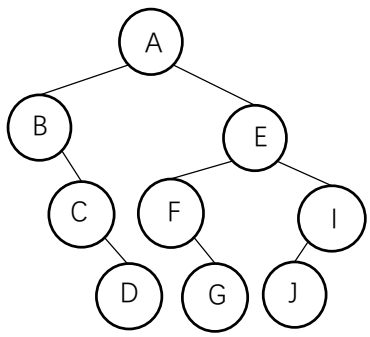
四.设计题(10 分)

题目中未明确告诉，我们认为 H1 和 H2 所存储的长整数是从低位到高位逆序存储的，这样有利于后边计算。将两个链表中每个结点的数据逐个相加(注意进位)。

```
LNode *Add(LNode* H1, LNode* H2){
    LNode* p1 = H1->next, *p2 = H2->next, *p = NULL;
    LNode* H3 = (LNode*)malloc(sizeof(LNode));
    LNode *r = H3, *s;
    int mod = 0,sum = 0;
    while(p1 && p2) {
        sum = p1->data + p2->data + mod;
        mod = sum/10;     //进位
        sum = sum%10;
        s = (LNode*)malloc(sizeof(LNode));
        s->data = sum;
        r->next = s;
        r = s;
        p1=p1->next;
        p2=p2->next;
    }
    if(p1)
        p = p1;
    else if(p2)
        p = p2;
    while(p) {
        sum = p->data + mod;
        mod = sum/10;
        sum = sum%10;
        s = (LNode*)malloc(sizeof(LNode));
        s->data = sum;
        r->next = s;
        r = s;
        p = p->next;
    }
    if(mod != 0) {
        s = (LNode*)malloc(sizeof(LNode));
        s->data = mod;
        r->next = s;
        r = s;
    }
    r->next = NULL;
    return H3;
}
```

# 西北工业大学 2018 年研究生入学考试

一.选择题(每题 3 分，共 15 分)



图二-1

**1.**D【解析】若进行快速排序的 n 个元素按关键字有序或基本有序时，快速排序将退化为起泡排序，时间复杂度为 $O(n^2)$。

**2.**C【解析】 i 的左孩子是 2i,右孩子是 2i+1.所以 45 的右孩子编号为 91

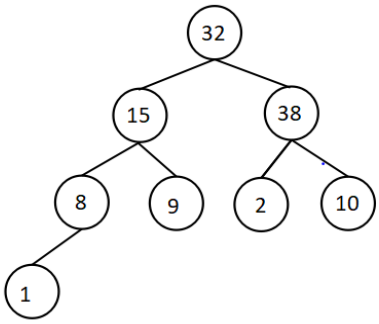
**3.**D　**5.**【解析】由于题目不全，大家记住非空广义表的概念：

任何一个非空广义表 LS=($\alpha_1$,$\alpha_2$，...,$\alpha_n$)均可分解为：

**表头** Head(LS)=$\alpha_1$ 和 **表尾** Tail(LS)=($\alpha_2$,...,$\alpha_n$)两部分

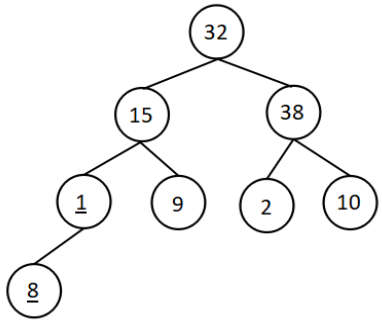
二.简答题(每题 10 分，共 60 分)

**1.**如图二-1 所示。

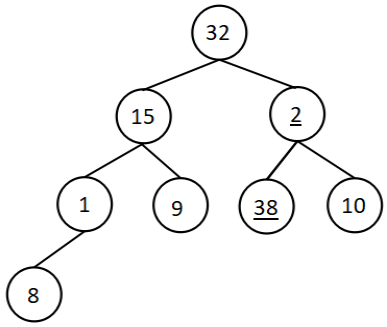



（1）初始状态

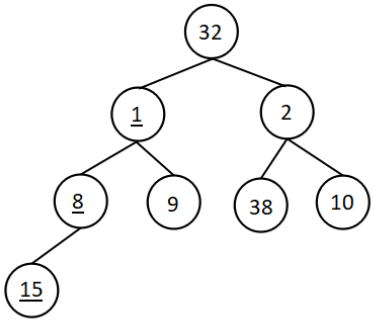
**2.**如下图所示。



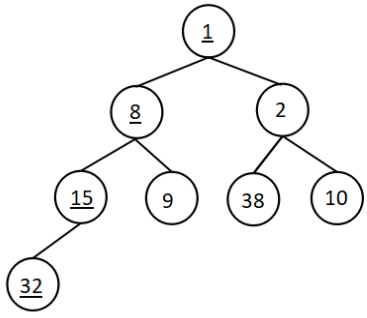

（2）8 和 1 交换



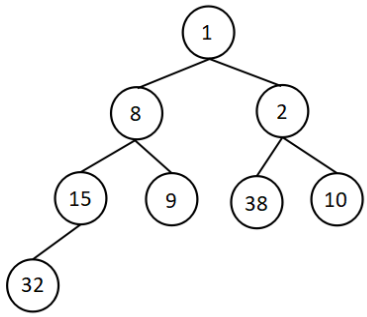

（3）38 和 2 交换




（4）15 和 1 交换，15 和 8 交换




（5）32 和 1 交换，32 和 8 交换，32 和 15 交换



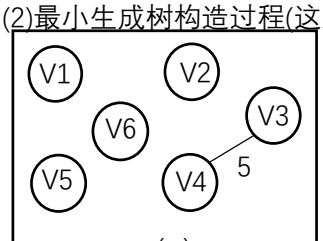

（6）初始堆构造完成

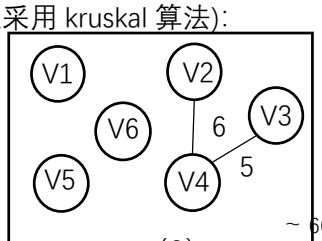
**3.**广度优先遍历序列：ABCDFE。广度优先遍历算法：借助一个辅助队列来记忆正在访问的顶点的下一层顶点。(1)访问起始顶点，并入队；(2)每次从队首取出一个顶点 v，访问与 v 邻接的所有未被访问的顶点，并入队。重复(2)，直到队空。

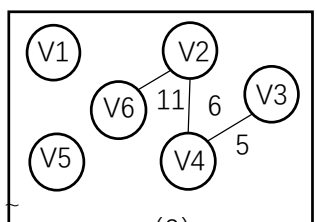
```
bool visited[vexnum];   //访问标记数组
void BFSTraverse(Graph G) {
//对图 G 进行广度优先遍历，设访问函数为 visit()
    for(i = 0;i < G.vexnum;i ++)
        visited[i] = false;   //访问标记数组初始化
    InitQueue(Q);   //初始化辅助队列 Q
    for(i = 0;i < G.vexnum;i ++)   //从 0 号顶点开始遍历
        if(!visited[i])   //对每个连通分量调用一次 BFS
            BFS(G, i);
}
void BFS(Graph G, int v) {
    visit(v);
    visited[v] = true;
    EnQueue(Q, v);        //入队
    while(!isEmpty(Q)) {
        DeQueue(Q, v);    //出队
        for(w = 0; w < G.vexnum; w ++) {
            if(G.edge[v][w]        ==        1
&& !visited[w]){
                visit(w);
                visited[w] = true;
                EnQueue(Q, w);
            }
        }
    }
}
```

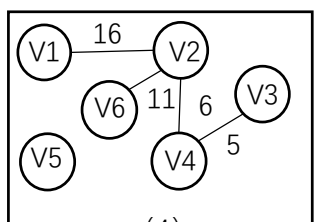
**4.**(1)先描述算法：参考《西北工业大学 2000-2001 学年第二学期期末考试》二简答题 3 小题

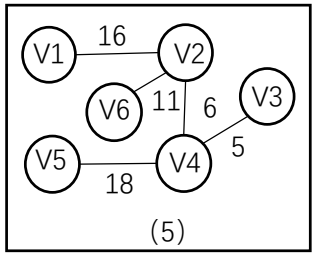
(2)最小生成树构造过程(这里采用 kruskal 算法)：



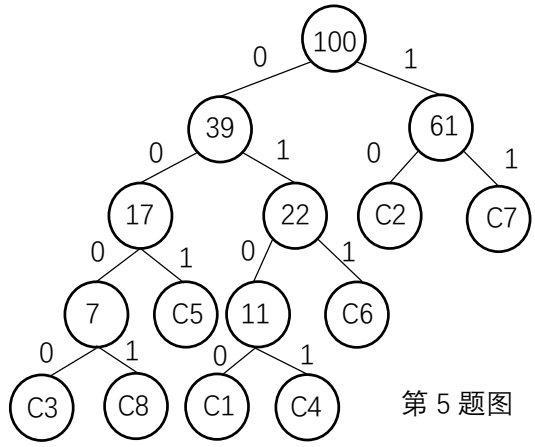






**5.**(见右图)C3:0000 C8:0001 C1:0100 C4:0101 C5:001 C6:011 C2:10 C7:11

总码数=(3+4+5+6)×4+(10+11)×3+(25+36)×2=257

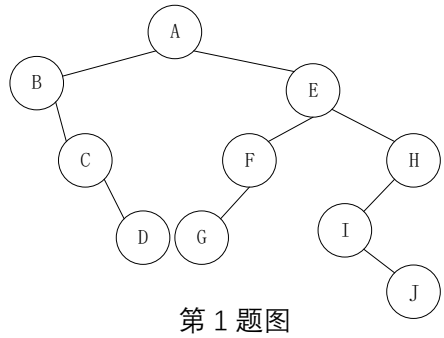



第 5 题图

**6.**首先用子问题定义状态：即 f[i][j]表示前 i 件物品恰放入一个容量为 j 的背包可以获得的最大价值。则其状态转移方程便是：f[i][j]=max{f[i-1][j],f[i-1][j-w[i]]+p[i]}若只考虑第 i 件物品的策略(放或不放)，那么就可以转化为一个只牵扯前 i-1 件物品的问题。如果不放第 i 件物品，那么问题就转化为“前 i-1 件物品放入容量为 j 的背包中”，价值为 f[i-1][j]；如果放第 i 件物品，那么问题就转化为“前 i-1 件物品放入剩下的容量为 j-w[i]的背包中”，此时能获得的最大价值就是 f[i-1][j-w[i]]再加上通过放入第 i 件物品获得的价值 p[i]。代码如下。

```
for (i = 1; i <= n; i++)
   for (j = c; j >= w[i]; j--)
      f[i][j] = max(f[i - 1][j], f[i - 1][j - w[i]] + p[i]);
```

# 西北工业大学 2019 年研究生入学考试




第 1 题图

**1.**(1)将森林化成二叉树原则上是“左孩子，右兄弟”。即将它的孩子结点放在左子树上，兄弟结点放在右子树上。

(2)前序：ABCDEFGHIJ；中序：BCDAGFEIJH；后序：DCBGFJIHEA

(3)算法基本思想：利用队列，从树的根结点开始，依次将它的左孩子和右孩子入队，然后每出队一个结点出队，都将其左孩子和右孩子入队，直到所有结点都出队，出队结点先后顺序就是遍历结果。层次遍历结果：ABECFHDGIJ

**2.**以最左侧元素作为基准值，初始：{12,15,23,8,6,20,16}

第一次交换：{6,15,23,8,12,20,16}//从后往前找比 12 小的元素，然后将其放入

第二次交换：{6,12,23,8,15,20,16}//再从前往后找比 12 大的元素，并将其放入，然后重复这两个步骤

第三次交换：{6,8,23,12,15,20,16}

完成一趟排序：{6,8},12,{23,15,20,16}//最后将存在 pivot 里的元素放入，再对两边分别进行快速排序，重复上述过程，结果如下所示。

结果：第一趟：{6 8} 12 {23 15 20 16}　　第二趟：6 {8} 12 {16 15 20} 23　　第三趟：6 8 12 {15} 16 {20} 23

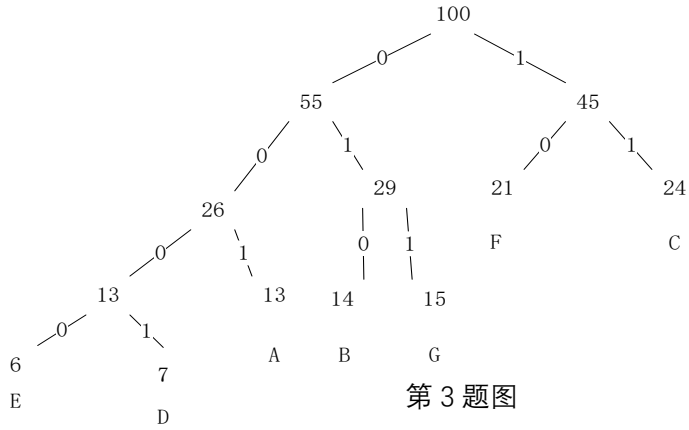
**参考代码：**

```
void QuickSort(int arr[],int l,int r){
    if(l < r) {
        int i=l,j=r;
        int pivot=arr[l];
        while(i<j){
            while(i<j && arr[j]>=pivot)
                j--;
            if(i<j)
                arr[i++]=arr[j];
            while(i<j&&arr[i]<=pivot)
                i++;
            if(i<j)
                arr[j--]=arr[i];
        }
        arr[i]=pivot;
        QuickSort(arr,l,i-1);
        QuickSort(arr,i+1,r);
    }
}
```

**3.**A:001 B:010 C:11 D:0001 E:0000 F:10 G:011

【注】可以拓展去看下哈夫曼编码比固定位数编码节省的比例

**4.**【注】本题因为考试过程中数据出现遗漏，我当时选择的是在本题开始前先写上自己的情况，以告知改卷老师，在写答案时，用 x 代替数值（如同时有几条路径值不确定），可用 min{}来表示



第 3 题图

**4.**

| | 第一次 | 第二次 | 第三次 | 第四次 |
|---|---|---|---|---|
| 1 | 4(0->1) | 4(0->1) | 3(0->2->3->1) | 3(0->2->3->1) |
| 2 | 1(0->2) | 1(0->2) | 1(0->2) | 1(0->2) |
| 3 | ∞ | 2(0->2->3) | 2(0->2->3) | 2(0->2->3) |
| 4 | ∞ | 6(0->2->4) | 6(0->2->4) | 6(0->2->4) |
| 5 | ∞ | ∞ | x(0->2->4->5) | x(0->2->4->5) |
| 6 | ∞ | 5(0->2->6) | 5(0->2->6) | 5(0->2->6) |

**5.(1)**

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | ∞ | 5 | ∞ | 9 |
| 1 | 1 | 0 | 3 | 7 | ∞ | ∞ |
| 2 | ∞ | 3 | 0 | 8 | 9 | 10 |
| 3 | 5 | 7 | 8 | 0 | 2 | ∞ |
| 4 | ∞ | ∞ | 9 | 2 | 0 | 7 |
| 5 | 9 | ∞ | 10 | ∞ | 7 | 0 |

$$\mathrm{A[i,j]}\begin{cases} w_{ij} \ \ 若(v_i,v_j)或<v_i,v_j>\epsilon E(G) \\ 0\ 或\infty \ \ 若(v_i,v_j)或<v_i,v_j>\notin E(G) \end{cases}$$

邻接矩阵定义图

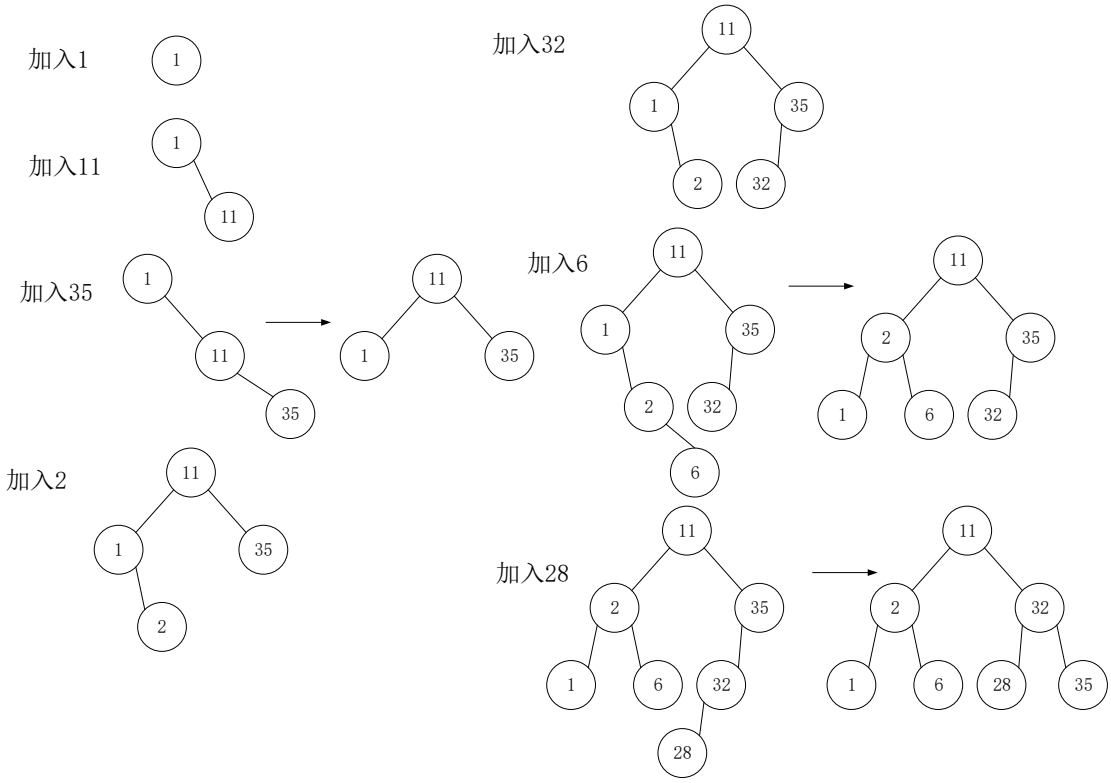
**(2)**①Prim 算法："保持连通，加最小边"，即选择包含子图中已存在结点的最小边加入子图。

②Kruskal 算法：从边集 E 中选取一条权值最小的边，若该条边的两个顶点分属不同的"树"，则将其加入子图。

破圈法："见圈破圈"，若看到图中有一个圈，则将这个圈的边去掉权值最大一条边，直至图中再无一圈为止。

【注】最好写出两种算法和对应的生成树。

**6.**【解析】




**7.**中序遍历二叉树，每遍历一个比 key 值小的数，计数器 count=count+1，遍历到大于 key 值元素时，返回 count 值。

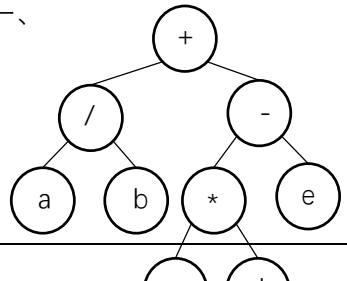
**参考代码：**

```
int InOrder(BinTreeNode *root,int key){
    stack<BinTreeNode*> st;
    BinTreeNode *p=root;
    int count=0;
    while(p!=NULL||!st.empty()) {
        while(p!=NULL) {
            st.push(p);
            p=p->leftchild;
        }
        if(!st.empty()) {
            p=st.top();
            if(p->data<=key)
                count++;
            else
                return count;
            st.pop();
            p=p->rightchild;
        }
    }
    return count;
}
```

【注】else return count;该句体现了二叉排序树的优越性，不必做后面无谓的比较，节省了时间和空间，提高了效率。是本题指明二叉排序树的原因所在。

# 西北工业大学 2020 年研究生入学考试

一、



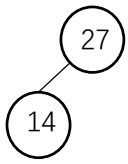

```
void PreOrder(Node* head)
{
    if (head == nullptr)
        return;
    PreOrder(head->left);
    PreOrder(head->right);
```

```
void layer_traver(BinTree BT){
    queue<BinTree> q;
    BinTree p=BT;
    if(p!=NULL){
        q.push(p);
}
```

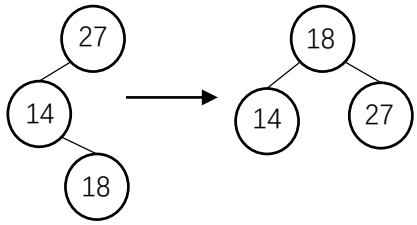
| | | |
|---|---|---|
| | }<br>先序遍历结果：+/ab-*cde | **while**(!q.empty()){<br>p = q.front();<br>q.pop();<br>cout<<p->data<<endl;<br>**if**(p->lchild!=NULL)<br>q.push(p->lchild);<br>**if**(p->rchild!=NULL){<br>q.push(p->rchild);<br>}<br>}<br>}<br>层次遍历结果：+/-ab*ecd |

二.(1)

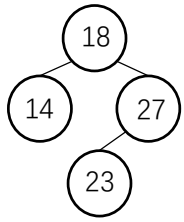

(1)插入 27



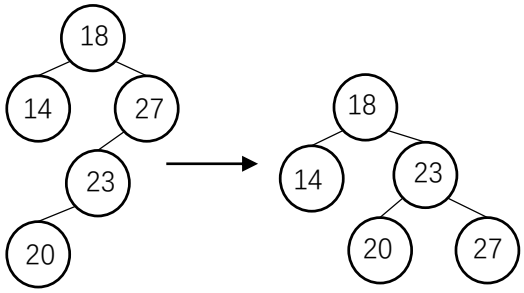

(2)插入 14



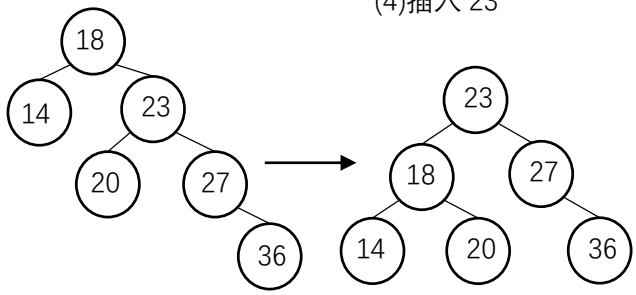

(3)插入 18，失去平衡，做旋转



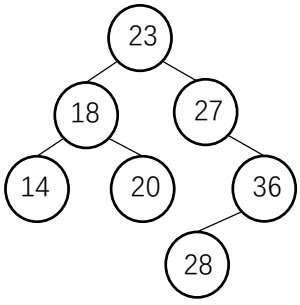

(4)插入 23



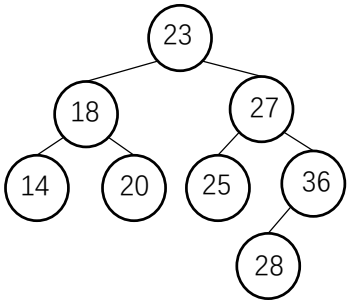

(5)插入 20，失去平衡，做单向右旋



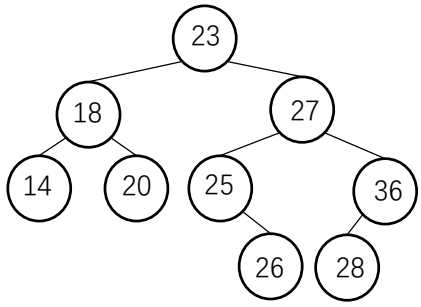

(6)插入 36，失去平衡，做单向左旋




(7)插入 28




(8)插入 25




(9)插入 26

(2)
```
/* 查找特定值 */
void SearchData(int targ, BSTree *nod) {
    if (nod != NULL)  {
        if (nod->data == targ)    {
            printf("查找值存在，值为%d\n", nod->data);
        }
        else if (nod->data > targ)  {
```

```
            SearchData(targ, nod->left);   //递归查找左子树
        }
        else if (nod->data < targ) {
            SearchData(targ, nod->right); //递归查找右子树
        }
    }
    else if (nod == NULL)  {
        printf("查找值不存在\n");
    }
}
```

三.A->B:10 为最短路径;A->C:A->D->C=30+20 为最短路径;A->D:30 为最短路径;A->E:A->D->C->E=60 为最短路径; A->F: 不可达

四.(1) 优先队列: 使用了二叉堆来实现。

二叉堆: 实际上是一颗被完全填满的二叉树。它具有堆序性。而最小值在根的位置。并且使用的是数组实现,一个 I 元素左儿子在 2I 上,右儿子在 2I+1 上。父亲则在 I/2 上。

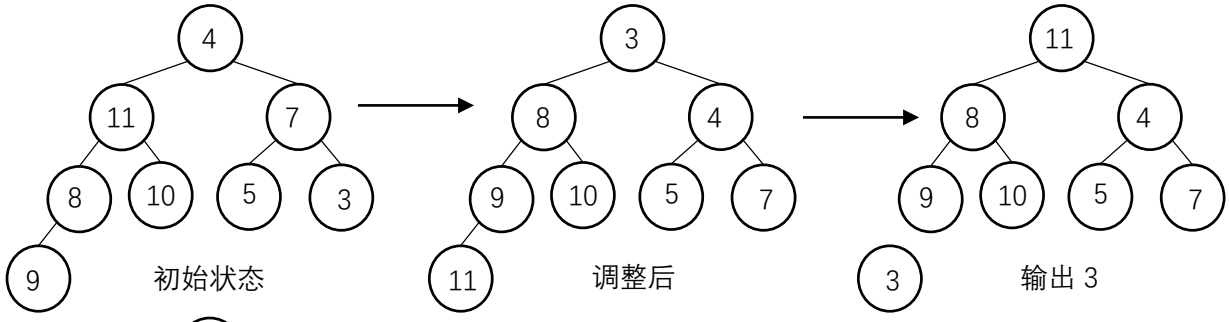
为了保证堆序性,我们在插入和删除操作上要注意。在插入时我们就要调整整个堆保证每个子树的根节点最小。删除时也一样。

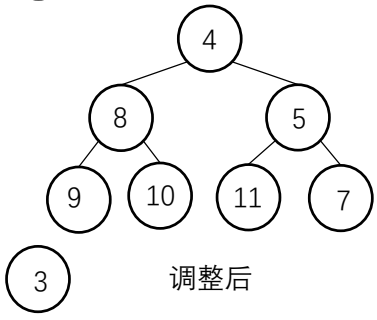
```
#include<new>
#include<iostream>
template <typename T>struct HeapNode{
    int Capacity;
    int Size;
    T * Elements;
};
template <typename T> class PriorityQueue{
    private:
        typedef struct HeapNode<T> * Priorityqueue;
        Priorityqueue H;
    public:
        Priorityqueue Initialize( int MaxElements ) {
            H = new HeapNode<T>;
            H->Elements = new T[MaxElements];
            H->Capacity = MaxElements;
            H->Size = 0;
            H->Elements[0] = 0;//最小标志
            return H;
        }
        void Insert( T X ) {
            int i;
            if( IsFull( ) ) {
                return ;
            }
            for( i= ++H->Size ; H->Elements[i/2] > 
            for( i = 1 ; i * 2 <= H->Size ; i = Child ){
//现在根节点为空依次上滤其他节点,并找到插入最后一个节点的位置 I。
                Child = i*2;
                if( Child != H->Size && H->Elements[ Child + 1] < H->Elements[ Child ]);
                    Child++;

if(LastElement>H->Elements[Child] )

H->Elements[i]=H->Elements[Child];
                else
                    break;
            }
            H->Elements[i] = LastElement;//把原本的最有一个元素放入 I 位置上。
            return MinElement;
        }
        T FindMin( ){
            return H->Elements[1];
        }
        bool IsEmpty( ) {
            if( H->Size == 0 )
                return true;
            else
                return false;
        }
```

```
X ; i /= 2 )
//从最后一个节点开始找到比插入值小的节点然后插入。
                H->Elements[i]=H->Elements[i/2];
                //父节点值给到子节点
                H->Elements[i] = X;
        }
        T DeleteMin( ) {
            int i, Child;
            T MinElement ,LastElement;
            if( IsEmpty() ) {
                return H->Elements[0];
             }
             MinElement = H->Elements[1];//记录最小节点
             LastElement=H->Elements[ H->Size-- ];//节点数减 1 并且保存在 LastElement 中
```

```
        bool IsFull( ) {
            if( H->Size == H->Capacity )
                return true;
            else
                return false;
        }
        PriorityQueue(T a[], int number) {
            H = Initialize( number * 2 );
            for(int i=0;i<number;i++) {
                Insert( a[i] );
            }
        }
        ~PriorityQueue() {
            delete []H->Elements;
            delete H;
        }
};
```

(2)堆排序的详细过程见《西北工业大学 2004 年研究生入学考试》二算法应用题 1 小题

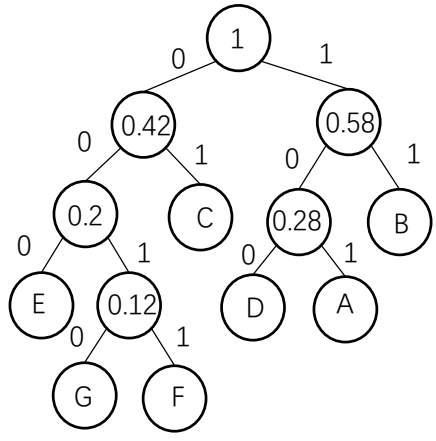



五(1)A-101；B-11；C-01；D-100；E-000；F-0011；G-0010

(2)

```
void HuffmanCode(codeType code[], HuffmanTree tree[]) {
    for (int i = codeNum - 2; i >= 0; i--) {
        // 先把 data 搞到手
        if (tree[i].data != '0') {
            code[i].data = tree[i].data;
        }
 // 从上到下进行 Huffman 编码，所以此时的 Huffman 码是逆序的
        for (int j = i, k = 0; tree[j].parent != -1; k++) {
            if (j == tree[tree[j].parent].lChild) {
```




第五题图

```
                code[i].bits[k] = '0';
            } else if (j == tree[tree[j].parent].rChild) {
                code[i].bits[k] = '1';
            }
            j = tree[j].parent;
            code[i].bitLen++;
        }
    }
    // 逆序得到的 Huffman 编码
    for (int i = 0; i < codeNum; i++) {
        reverse(code[i]);
    }
    // 把带 data 的放在前面，方便打印
    sort(code, code + codeNum, cmpByCodeData);
}
(3)void ShowHuffmanTree(HuffmanTree tree[]) {
    cout << endl << "==== The Huffman huffmanTree is as follows ====" << endl;
    cout << "Index\tlChild\tdata\tweight\trChild\tparent" << endl;
    for (int i = 0; i < codeNum; i++) {
        cout << internal << setw(3) << tree[i].index << "\t"
            << right << setw(4) << tree[i].lChild << "\t"
            << internal << setw(2) << tree[i].data << "\t"
            << setprecision(2) << tree[i].weight << "\t"
            << right << setw(4) << tree[i].rChild << "\t"
            << right << setw(3) << tree[i].parent << endl;
    }
    cout << endl;
}
```

六. 前 K 个数中的最大 K 个数是一个退化的情况，所有 K 个数就是最大的 K 个数。如果考虑第 K+1 个数 X 呢？如果 X 比最大的 K 个数中的最小的数 Y 小，那么最大的 K 个数还是保持不变。如果 X 比 Y 大，那么最大的 K 个数应该去掉 Y，而包含 X。如果用一个数组来存储最大的 K 个数，每新加入一个数 X，就扫描一遍数组，得到数组中最小的数 Y。用 X 替代 Y，或者保持原数组不变。这样的方法，所耗费的时间为 O（N * K）。

进一步，**可以用容量为 K 的最小堆来存储最大的 K 个数**。最小堆的堆顶元素就是最大 K 个数中最小的一个。每次新考虑一个数 X，如果 X 比堆顶的元素 Y 小，则不需要改变原来的堆，因为这个元素比最大的 K 个数小。如果 X 比堆顶元素大，那么用 X 替换堆顶的元素 Y。在 X 替换堆顶元素 Y 之后，X 可能破坏最小堆的结构（每个结点都比它的父亲结点大），需要更新堆来维持堆的性质。**更新过程花费的时间复杂度为 O（log2K）**。

# 附录

## 西北工业大学未命名期末试题

一.选择题(10 分，每小题 1 分)

**答案速查：**DDABA　CABCD
6.【解析】见《西北工业大学 2007 年研究生入学考试》一选择题 9 小题
10.【解析】n=m+n1+n2,n2=m-1,n1=0,故 n=2*m-1

二.判断题(10 分，每小题 1 分)

**答案速查：××√×√　　√××√√**
1.×【解析】算法的时间复杂度是通过算法中基本语句执行次数的数量级来确定的。
2.×【解析】单链表是一种顺序存取的存储结构。
5.√【解析】不会存在环路，参考本套题第四题解析。
8.×【解析】B-树是一种平衡的多路查找树，只有指向根节点的指针，所以只能进行随机查找，不能进行顺序查找。B+树有指向关键字最小的叶子结点的指针，可以进行顺序查找，还有一个指向根节点的指针，可以进行随机查找。
9.√【解析】当矩阵中非零元素个数*3<矩阵的行*列时，才会节省存储空间。

三.简答题(10 分)

**(1)**参考《西北工业大学 2014 年研究生入学考试》第一题 1 小题
**(2)**抽象数据类型包含一般数据类型的概念，但含义比一般数据类型更广、更抽象。一般数据类型由具体语言系统内部定义，直接提供给编程者定义用户数据，因此称它们为预定义数据类型。抽象数据类型通常由编程者定义，包括定义它所使用的数据和在这些数据上所进行的操作。在定义抽象数据类型中的数据部分和操作部分时，要求只定义到数据的逻辑结构和操作说明，不考虑数据的存储结构和操作的具体实现，这样抽象层次更高，更能为其他用户提供良好的使用接口。

四.证明题

**证明：**任意 n 个结点的有向无环图都可以得到一个拓扑序列。设拓扑序列为 $v_0v_1v_2...v_{n-1}$，我们来证明此时的邻接矩阵 A 为上三角矩阵。证明采用反证法。

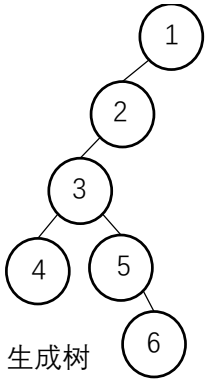
假设此时的邻接矩阵不是上三角矩阵，那么，存在下标 i 和 j（i>j），使得 A[i][j]不等于零，即图中存在从 $v_i$ 到 $v_j$ 的一条有向边。由拓扑序列的定义可知，在任意拓扑序列中，$v_i$ 的位置一定在 $v_j$ 之前，而在上述拓扑序列 $v_0v_1v_2...v_{n-1}$ 中，由于 i>j，即 $v_i$ 的位置在 $v_j$ 之后，导致矛盾。因此命题正确。

五.计算和分析题

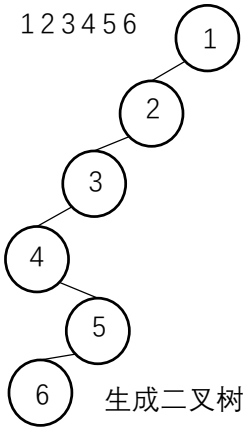
**1.**略。多次出现，类似题型请参考历年真题和期末题。

**2.**参考《西北工业大学 2007 年研究生入学考试》三分析题 1 小题

**3.**(1)深度优先遍历序列： 1 2 3 4 5 6

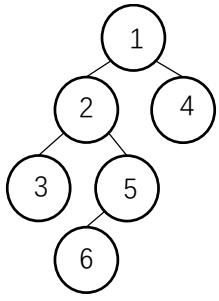



生成树

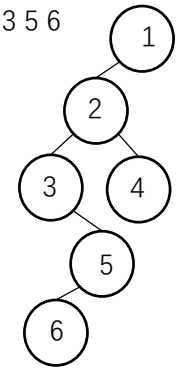



生成二叉树

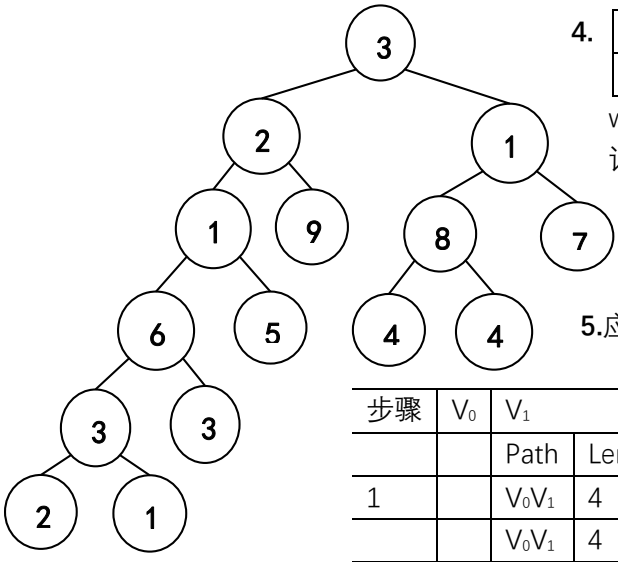
(2)广度优先遍历序列：1 2 4 3 5 6




生成树




生成二叉树




**4.**

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| 00000 | 00001 | 100 | 001 | 11 | 0001 | 101 | 01 |

wpl=5×2+5×1+4×3+3×5+2×9+3×4+3×4+2×7 = 98

该字符串编码长度至少为 98 位。

**5.**应用 Dijkstra 算法求从顶点 $V_0$ 到其他各顶点的最短路径 Path 和最短路径长度 Len 的步骤如下:

| 步骤 | $V_0$ | $V_1$ | | $V_2$ | | $V_3$ | | $V_4$ | | 动作 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Path | Len | Path | Len | Path | Len | Path | Len | |
| 1 | | $V_0V_1$ | 4 | — | ∞ | $V_0V_3$ | 7 | — | ∞ | 选 $V_0V_1$ |
| | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | — | ∞ | 参照 $V_1$ 调整 |
| 2 | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | — | ∞ | 选 $V_0V_3$ |
| | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | $V_0V_3V_4$ | 12 | 参照 $V_3$ 调整 |
| 3 | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | $V_0V_3V_4$ | 12 | 选 $V_0V_1V_2$ |
| | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | $V_0V_1V_2V_4$ | 10 | 参照 $V_2$ 调整 |
| 4 | | $V_0V_1$ | 4 | $V_0V_1V_2$ | 8 | $V_0V_3$ | 7 | $V_0V_1V_2V_4$ | 10 | 选 $V_0V_1V_2V_4$ |

**6.**略。参考《西北工业大学 2006-2007 学年第一学期期末考试》二请求解决下面问题 4 小题。

# 西北工业大学期末考试试题(A1 卷)

一.填空题(每小题 2 分，本题满分 20 分)

(1) 2140+2*(30*20-1) = 3338(3338,3339)　(2) O(m+n)　(3) _n*lg n_　(4) 2 (A[7],A[8])　2 (A[0],A[1])

(5)4　4　(6)3　(7) top[0]+1==top[1]或者 top[0] = =top[1]+1　(8)4　(9)2*i+1　(10)p->rightchild->rightchild

二.选择题(每小题 2 分，本题满分 20 分)

**答案速查：**(1)BC (2)A (3)B (4)C (5)D (6)B (7)A (8)D (9)B (10)AC

(3)B【解析】广度优先遍历不能判断有环

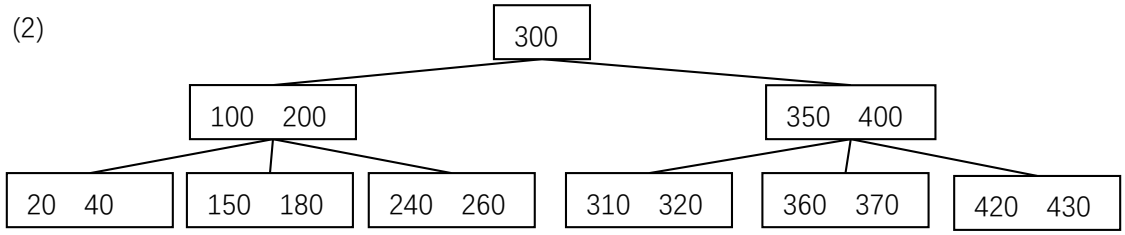
(4)C【解析】N 个结点的线索二叉树（没有头结点）上含有的线索数是 n + 1

三.简答题(每小题 5 分，本题满分 20 分)

(1)

| 顶点编号 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 最短路径 | 1 | 1-2 | 1-3 | 1-3-4 | 1-3-5 | 1-3-6 |
| 最短路径长度 | 0 | 11 | 7 | 12 | 14 | 15 |

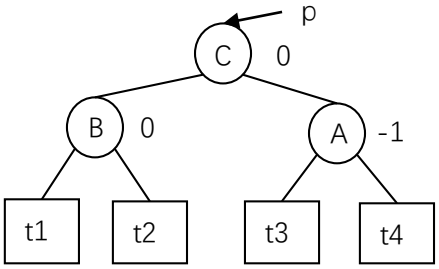
(2)



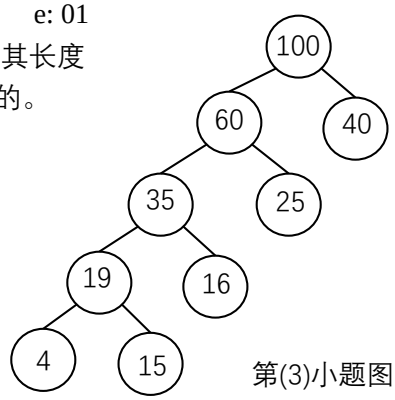

(3)Huffman 编码(见右图)：a: 001 b: 1 c: 0001 d: 0000 e: 01

【注意】因为左右子树不同，所以编码可以有多种，但是只要其长度分别是 3，1，4，4，2；且相互之间不形成前缀关系就是正确的。

(4)




第(4)小题图




第(3)小题图

【解析】某结点的平衡因子就等于左子树的高度减去右子树的高度，我们从原图中进行推断，假设 t3 高度为 n，因为 C 点的平衡因子为 1，则 t2 的高度为 n+1；由于 B 点的平衡因子为-1，且右子树的高度为 n+2，则 t1 的高度为 n+1,；由于 A 点的平衡因子为 2，可以推断出 t4 的高度为 n+1。

**四.(1)**散列函数为：f(x) = x % 12 允许其它的散列函数

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 60/1 | 31/7 | | 15/1 | | | 90/1 | 78/2 | 20/1 | 42/4 | 10/1 | 35/1 |

(2) 7 (7-->8->9->10->11->0->1 (成功)) (3) (1+7+1+1+2+1+4+1+1) / 9 = 19/9

五.程序设计题(每小题 15 分，本题满分 30 分)

**1.解：**

```
TreeNode * tree(char *preorder, *midorder) {
    return
        TreeRecursive(preorder, 0, strlen(preorder)-1, midorder, 0, strlen(midorder)-1);
    }
    TreeNode * TreeREcursive(char *pre, int preSt, int preEnd, char* mid, int midSt, int midEnd){
        if (preEnd < preSt)
            return NULL;
```

```
        char   rt = pre[preSt];
        for(int j = midSt; j<=midEnd; j++)
            if(mid[j] == rt) break;
        if(j>midEnd){cout<<"Wrong input";    return NULL;}
        TreeNode root = new binTreeNode( );
        root->data = rt;
        int lLen = j - midSt;
        root->leftChild = treeRecursive(pre, preSt+1, preSt+lLen - 1, mid, midSt,midSt+lLen-1);
        root->rightChild=treeRecursive(pre,preSt+lLen+1,preEnd, mid, midSt+lLen+1, midEnd);
        return root;
    }
```

要点在于根据 preOrder 的第一个字符，在 midOrder 中找到左右子树的分界；然后递归调用生成左右子树；做到这一点，就可以给出大部分分数。

可能有很多细节上不一样的地方，比如这里的 preEnd 指向的字符在相应的树中；但是有些人可能是 preEnd 指向下一个位置。或者传递参数时直接使用字符串传递。都是可以的。

【注】如果使用别的办法，参考其效率，酌情给分。只要效率过得去，也可以得满分。但是纯粹枚举则得分不高。

**2. 解：**

```
Graph : :DFS(int v){ //从 v 开始搜索；
    bool        visited[MAXVert];
    Edge        nextEdge[MAXVert];
    stack[0] = v;
    nextEdge[0] = graph.Nodetable[v].adj;
    top = 0;
    visited(stack[top]);
    visited[stack[top]] = true;
    while(top >= 0) {
        while(nextEdge[top] && visited[nextEdge[top]->dest])
        //寻找下一个尚未访问的邻接节点
        nextEdge[top] = nextEdge[top]->link;
        if(nextEdge[top] != NULL) {
            int    nextVert = nextEdge[top]->dest;
          visite(nextVert);   //访问下一个邻接结点；保证了被压入栈中的顶点都被访问；
            visited[nextVert] = true;
            stack[top+1] = nextVert; //压栈，进入下一个结点；
            nextEdge[top+1] = Nodetable[nextVert].adj;
            nextEdge[top] = nextEdge[top]->link;
            top ++;
        }
        else  top --;
    }
}
```