

## 一. 选择题

1. B

解析: 完全有向图:  $n(n-1)$ , 无向完全图  $n(n-1)/2$ 

2. A

队列: 先进先出 栈: 先进后出

3. C

解析: 参考严版P47页

4. A

解析: 这题的意思是查找含 $n$ 个结点的有序表中一个结点的时间复杂度, 折半查找时间复杂度为 $O(\log n)$ 

5. D

→ 仅仅是存取的话, 顺序表最合适

6. A

→ 线性表中链表优势是插入与删除方便, 而顺序表则是查找存取

7. B

→ 首先明确生成树概念: 图的极小连通子图, 这里区分一下

极大连通子图: 连通图再加一边或一点就破坏	{	极小连通子图: 连通图再加一条边就形成环

由上述定义可知,  $n$ 个顶点要是连通图的话至少 $n-1$ 条边

8. A

→ 将第 $i$ 个元素后面元素前移即可, 即 $(n-i)$ 个, 这里教大家一个技巧, 即直接代入数, 如 $n=5, i=3$ 

9. C

→ 栈是特性: 先进后出, 由于输出的第一个元素是 $n$ , 那么第2个为 $n-1$ , 第3个为 $n-2$ , ..., 第 $i$ 个为 $n-i+1$

同理可用小技巧, 代入 $n=5, i=3$

→ 扩展: 如果题目改为输出前 $i$ 个元素是 $i$  ( $1 \leq i \leq n$ ) 的话, 那么选D

10. C

→ 考虑到满二叉树性质 → 第 $i$ 层至多有 $2^{i-1}$ , 前 $k$ 层结点个数为 $2^k - 1$ 

11. B

→ A.C.D为线性表皆有的性质

12. C

压缩存储方法	{	顺序	{	三元组表示法
				伪地址表示法
	{	链式	{	邻接表表示法
				十字链表表示法

13. A

→ 考查关键路径

{	① 最长: 图中最长路径
	② 最短: 工期完成最短时间

14. C



15. B

→  $ltag=0$  表示指针,  $ltag=1$  表示线索, 没有左子树说明  $T \rightarrow ltag=1$ , C改为小写t.

16. D\*

→ 有向图中所有顶点入度 = 所有顶点出度, 不难理解, 每一条边一个入度, 一个出度.

17. D

→ 快排在数据基本有序情况下类似于插入排序, 时间复杂度为  $O(n^2)$

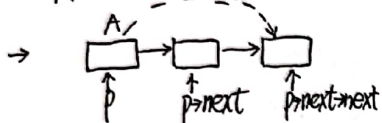
18. D

判断有向图是否存在回路 { ① 拓扑排序  
② 深度优先遍历

19. A

→ 满足题干的有: ① 堆排序 ② 冒泡排序 ③ 简单选择排序

20. A



## 二. 填空题.

1. 归并排序

→ 见严版书籍 P289页

2. 计算方便 散列地址分布均匀

3. 一端 先进后出

4. 顺序  $\frac{n+1}{2}$

5. 从任意结点出发能访问所有结点. (从任意结点出发可以遍历整个链表)

6. 时间 空间

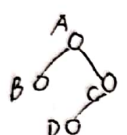
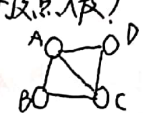
7.  $n-1$   $\frac{n(n-1)}{2}$  (前者为已排好序, 后者为已排好序的逆序)

8.  $2n-1$

9.  $n$   $n$



### 三. 判断题.

1. f
2. t
3. f (非连通图需要多次遍历)
4. t ( $O(n^2)$ )
5. f
6. f (如果该活动是多条关键路径的公共边的话, 有可能缩短工期)
7. f (先序+中序, 中序+后序  $\rightarrow$  可以构造)
8. t
9. f
10. t
11. f
12. f (   $\rightarrow$  中序: BADC, 先序: ABCD )
13. f (邻接表表示每个顶点出度, 逆邻接表表示每个顶点入度)
14. f (当有多个环时, 可能为非连通图, 如  (五条边, 5个顶点) )
15. f (平衡树除了平衡因子要满足外, 它还有一个条件: 二叉排序树, 即平衡树是二叉排序树)

### 四. 应用题.

1. 答: 逻辑结构: ① 数据元素之间逻辑关系.  
② 与数据存储结构无关

物理结构: ① 数据元素在计算机内的存储方式

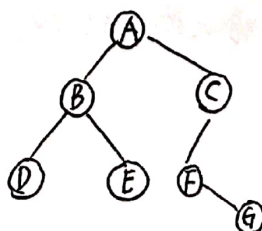
(注: 这种题没有标准答案, 考试时言之有理即可, 多写一些没有坏处, 但重要的写前面)

2. 答:  
拓扑序列如下:

① a, b, c, e, d, f

② a, d, b, c, e, f

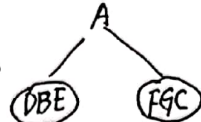
3. 答:  
由前序遍历序列与中序遍历序列可得二叉树如下:






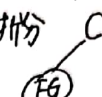
解析：下面笔者将细致地讲解这一经典题型（由前中后，中后后前）

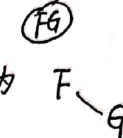
首先，解题核心是理解“前/后序中得到根结点，由该根结点在中序序列中划分左右子树”，再有就是“递归”思想，树的定义就有递归的身影，就是把分解出的左右子树看成一棵完整的树重复上述“找根”，“划左右子树”的操作。下面回到本题：

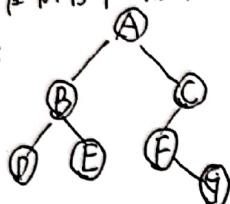
前序：ABDECFG → 找出  $root=A$  → 中序 DBEAFGC  $\xrightarrow{\text{划分}}$  

→ 将 DBE 和 FGC 看成完整的树

→ 前序：DBE 的前序中排列为 BDE → 找出  $root=B$  → DBE 在中序中排列为 DBE → 划分 

→ FGC 在前序中排列为 CFG → 找出  $root=C$  → FGC 在中序中排列为 FGC → 划分 

→ FG 在前序中排列为 FG → 找出  $root=F$  → FG 在中序中排列为 FG → 划分为 F-G 

→ 整合：

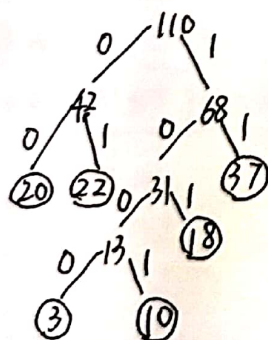
→ 笔者估计这是这种题型最细致地讲解了，熟练之后1分钟就可以写出答案，中+后→前序形类似，不累赘

4.  $t = \text{These are book}$ ,  $v = nmn$ ,  $u = mcmcmC$

→ 见书版书 P11-12页.

5. 答：

哈夫曼树如下：



这6个字母设计的哈夫曼编码分别为 11, 101, 01, 1000, 00, 1001

(或这样写：

字母出现频率	37	18	22	3	20	10
哈夫曼编码	11	101	01	1000	00	1001

2020-4



扫描全能王 创建

分析：求哈夫曼树与哈夫曼编码的过程

→ 所以哈夫曼树唯一

首先，哈夫曼树构造有个习惯：左小右大，“一般答案都遵循这个“潜规则”，除非题目有明确要求，不然就遵守“左小右大”，其次，哈夫曼编码一般“左0右1”，若题目有特殊要求，按题目来下面更题析：


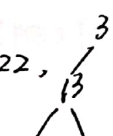
哈夫曼树构造过程：

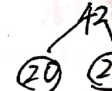

- ① 将几个权值分别看作只有根结点的  $n$  棵二叉树，这些二叉树构成集合记为  $F$ 。
- ② 从  $F$  中选出两棵根结点的权值最小的树作为左右子树，构造一棵新的二叉树，新二叉树根结点权值为左右子树根结点权值之和。
- ③ 从  $F$  中删除  $a, b$ ，加入新构造的  $C$ 。
- ④ 重复②③两步，直到  $F$  中只剩一棵树为止，该树即为哈夫曼树。


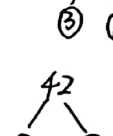
→ 由题可知

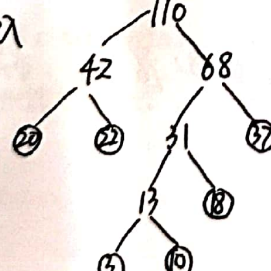
初始： $F: 3, 10, 18, 20, 22, 37$

第一步：删 3, 10，加入 ， $F$  更新为 ， $18, 20, 22, 37$ （按根结点排序）

第二步：删 13, 18，加入 ， $F$  更新为 ， $20, 22, 37$

第三步：删 20, 22，加入 ， $F$  更新为 ， $31, 37, 42$

第四步：删 31, 37，加入 ， $F$  更新为 ， $42, 68$

第五步：删 42, 68，加入 

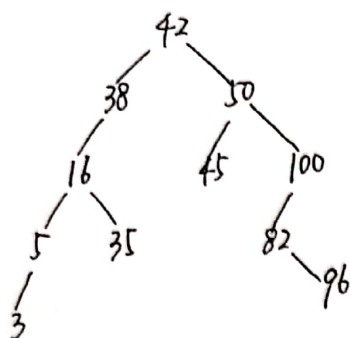
→ 上述就是完整过程，以绝对“学生视角”来分析！



6.

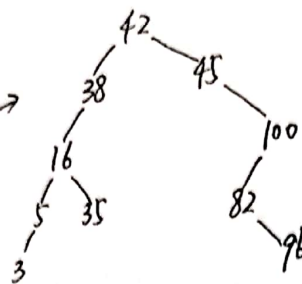
解:

二叉排序树为

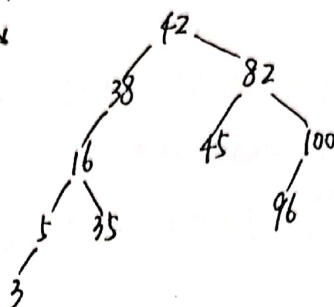


删除 50

方法一



方法二



排序

→ 解析: 二叉树删除结点P的情况如下:

① P为叶子结点, 直接删

② P只有左子树或右子树, 将P删掉, 然后将P的子树直接连在原来P与其双亲连接的指针上

③ P皆有左右子树

方法一: 找直接前驱r → 将r的值给P结点, 删除r结点

方法二: 找直接后继r (注: 基于中序遍历)

→ 找前驱或后继 { 为叶子 → ①  
为有左或右子树 → ②  
不可能出现情况③

→ 本题二叉排序树的中序遍历为 3, 5, 16, 35, 38, 42, 45, 50, 82, 96, 100, 可知50的前驱为45, 后继为82, 就对应方法一与方法二。

→ 注: 为什么要用中序遍历?

→ 二叉排序树的特性(中序遍历是递增的)不随二叉排序树的插入删除发生改变



扫描全能王 创建



## 五. 算法设计题 (30分)

### 1. 算法填空 (10分)

①  $&(L.elem[i-1])$  //  $p$  为被删除元素位置

②  $L.length-1$  // 表尾元素位置

③  $++p$

④  $*p$  // 被删除元素之后的元素左移

⑤  $L.length-1$  // 表长减一

→ 素材来自严版《数据结构》P24的算法2.4, 就最后一空稍微改编, 基本上暨大830真题算法填空皆来自于严版《数据结构》, 大家理解加记忆即可。送分题!

### 2.

```
void Createlist_L (LinkList &L, int n) {  
    // 逆序输入  $n$  个元素的值, 建立带尾头结点的单链线性表  $L$   
    L = (LinkList) malloc (sizeof (LNode))  
    L->next = NULL // 先建立一个带头结点的单链表  
    for (i = n; i > 0; --i) {  
        p = (LinkList) malloc (sizeof (LNode)); // 生成新结点  
        scanf ("%d", &p->data); // 输入新元素值  
        p->next = L->next; L->next = p // 头插法  
    }  
}
```

→ 本题来自严版《数据结构》P30的算法2.10

→ 本题的“逆位序”是指输入  $1, 2, \dots, n$  得到的链表打印出来是  $n, n-1, \dots, 1$ , 因为本代码用了头插法, 当然也可以用尾插法, 至于没有具体限制。



3.

- (1) 用有向图的邻接矩阵描述  $n$  个城市之间的公路网，矩阵为  $n$  阶方阵 ( $n \times n$ )  
同一城市在矩阵中的值为 0 ( $A \rightarrow A$  距离为 0)，若城市  $A$  到城市  $B$  有道路，则矩阵中  
相当位置值为两城市之间乘车费用。

(2)

```

void ShortestPath_DIJ (MGraph G, int V0, int dist[], int path[])
{
    int set [maxSize]; // 标记数组, set[u]=0 表示 u 在 T 中, 尚未加入 S 中 (S 是已加入最短路径顶点的集合)
    int min, i, j, u;
    // 初始化开始
    for (i=0; i<G.n; ++i)
    {
        dist[i] = G.edges[V0][i]; // dist[V0][i] 表示当前已找到的从 V0 到每一个顶点 Vi 的最短路径长度
        set[i] = 0;
        if (G.edges[V0][i] < INF) // path[V0][i] 保存了从 V0 到 Vi 最短路径上 Vi 的前一个顶点, u 是起
            path[i] = V0;
        else
            path[i] = -1;
    }
    set[V0] = 1; path[V0] = -1;
    // 核心算法开始
    for (i=0; i<G.n-1; ++i)
    {
        min = INF;
        // 这个循环每次从剩余顶点中选出一个顶点, 通往这个顶点的路径在通往所有剩余
        // 顶点的路径中是长度最短的
        for (j=0; j<G.n; ++j)
            if (set[j] == 0 && dist[j] < min)
            {
                u = j;
                min = dist[j];
            }
        set[u] = 1;
        for (j=0; j<G.n; ++j)
        {
            // 判断顶点 u 的加入是否会出现通往顶点 j 的更短的路径, 如果出现, 则改变
            // 原来路径及其长度, 否则什么也不做
            if (set[j] == 0 && dist[u] + G.edges[u][j] < dist[j])
            {
                dist[j] = dist[u] + G.edges[u][j];
                path[j] = u;
            }
        }
    }
}

```

总结:

本题参考《数据结构》P189, 有出算法课本的  
可能, 本答案来自 19 版《天勤》, 简单易懂  
代码稍长, 学长建议大家记代码时, 如代码  
多种, 选自己可以理解的那种来记, 千万别死  
因为“死记”部分到考场上一紧张就大脑空白  
总之, “理解”万岁!!!



扫描全能王 创建