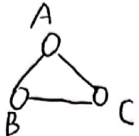


一. 选择题.

1. C

2. D

→ 无向图中每边关联两个度, 实际不在理解每个草图  , 3个顶点, 3条边 → 6个度.

3. C

→ 口诀: "快, 快些选一堆朋友玩耍"
 快速 ← 希尔 → 选择

4. C

→ 循环队列判空: $front = rear$

判满: $(rear + 1) \% max = front$

5. A (同2010-20)

6. C.

→ 见严版 P289页.

7. C

→ 这里的表结点指边表结点, 在邻接表中, 每个顶点后的链表表结点数表示该顶点的出度. 而对整个链表而言, 结点在全链表中出现次数即该结点的入度.

8. C.

→ 树这种数据结构特点就是层次, 分支化.

9. A

→ 考查最小连通子图 (即生成树)

10. A

→ 见严版 P289页.

11. A

→ 首先, 这里要有递归的另格, 树的定义本身就有递归思想. 无论是前序, 中序还是后序都是先访问左子树, 后访问右子树. 故选 A

12.

→ 这题依然有歧义, 不过按常规冒泡规则 (小的上浮, 大的下沉), 选 C.

13. D (同2012-13)

14. A.

先序遍历即变 $root = a$, 中序遍历即左子树右子树



15. C

→ 对比2011-12, 与之不同的是本题从A[0][0]开始, 把公式 $k = \begin{cases} \frac{i(i-1)}{2} + j - 1, & i \geq j \\ \frac{j(j-1)}{2} + i - 1, & i < j \end{cases}$ 改为

$$\begin{matrix} i=j+1 \\ \rightarrow \\ j=j+1 \end{matrix} k = \begin{cases} \frac{(i+1)i}{2} + j, & i \geq j \\ \frac{j(j+1)}{2} + i, & i < j \end{cases}, \text{代入 } i=8, j=5.$$

二. 填空题.

1. 三元组 十字链表.

2. 一对一 一对多

3. $2n-1$

4. 计算方便 散列地址分布均匀

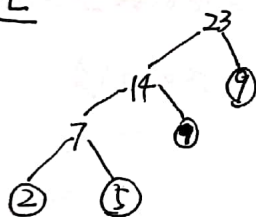
5. 前驱 后继.

6. n.

7. 60 ($10 \times 3 \times 2$)

8. $L \rightarrow \text{next} = L$

9. 44.



$$WPL = (2+5) \times 3 + 7 \times 2 + 9 = 44$$

10. 15, 40, 95, 20, 50, 70

三. 判断题.

1. f (相当于先序遍历)

2. t

3. f

4. t (这里要区分顺序查找与下址机查找, 前者原理是任关键字与一个序列的数从最后一个开始这个比较, 有点类似于线性表的遍历, 后者类似于树的查找. 本题考查了B树与B+树的一个区别: B+树有一个指针指向关键字最小的结点, 所有叶子结点链接成一个线性链表, 所以B+树可以顺序查找.)

5. f

6. f

7. f

8. t

9. f (最大的与root最近)

10. f

2013-2



扫描全能王 创建

四. 简答题

1. 解:

根据 Dijkstra 算法, 求顶点 0 到其余各顶点的最短路径如下表所示

顶点 0 先加入集合 S, 下表为其余各顶点加入后路径变化, 此时 $S = \{0\}$.

顶点	从顶点 0 到各顶点的 dist 值			
	第 1 趟	第 2 趟	第 3 趟	第 4 趟
1	1 $0 \rightarrow 1$			
2	∞	6 $0 \rightarrow 1 \rightarrow 2$	5 $0 \rightarrow 3 \rightarrow 2$	
3	3 $0 \rightarrow 3$	3 $0 \rightarrow 3$		
4	10 $0 \rightarrow 4$	10 $0 \rightarrow 4$	9 $0 \rightarrow 3 \rightarrow 4$	6 $0 \rightarrow 3 \rightarrow 2 \rightarrow 4$
集合 S	$\{0, 1\}$	$\{0, 1, 3\}$	$\{0, 1, 2, 3\}$	$\{0, 1, 2, 3, 4\}$

由上表可知, 顶点 0 到其余各顶点最短路径为

顶点 0 到顶点 1: $0 \rightarrow 1$, 长度为 1

顶点 0 到顶点 2: $0 \rightarrow 3 \rightarrow 2$, 长度为 5

顶点 0 到顶点 3: $0 \rightarrow 3$, 长度为 3

顶点 0 到顶点 4: $0 \rightarrow 3 \rightarrow 2 \rightarrow 4$, 长度为 6.

题型分析: 典型的 Dijkstra 求最短路径, 本题解法参考《王道》. 看懂这个表格前提是理解这个算法. 已加入顶点的集合 S, 未加入顶点集合 T, 初始 $S = \{0\}$, $T = \{1, 2, 3, 4\}$, 每次从 T 中挑出一个顶点放入 S 中, 这个点必须满足 S 中顶点到 T 中顶点距离最短, 如第一趟时 $S = \{0\}$ 有四种情况, 选最短的 $0 \rightarrow 1$, 把 T 中的 1 划掉, 使得 $S = \{0, 1\}$, 每次加入 1 个顶点都可能更新 S 中顶点到 T 中顶点距离.

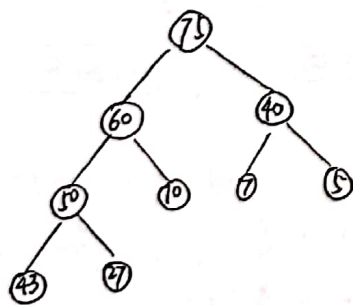
怎么由表格得出结论呢?

直接看顶点每行最右边的值, 如顶点 4, 最右边是 $0 \rightarrow 3 \rightarrow 2 \rightarrow 4$, 长度为 6, 表示经过 4 趟后顶点 4 才加入 S 中. 0 到 4 的最短路径为 $0 \rightarrow 3 \rightarrow 2 \rightarrow 4$. 也可看顶点什么时候加入 S 中, 如顶点 1 第一趟就加入 S 了, 第几趟所在列与顶点所在行的交点即为最短路径.

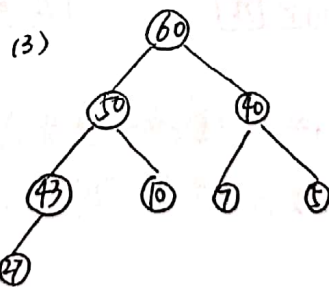
2.

解:

(1) 初始大根堆对应的完全二叉树为



(2) 75, 60, 40, 30, 10, 7, 5, 43, 27



→ 典型题“堆排序问题”, 下面会详讲这个专题

(1) 问要注意题干中是“大根堆”, 该完全二叉树要满足“堆定义”, 部分同学会写成“初始序列”的完全二叉树.

→ 堆排序拆解

首先理解大小堆堆定义 (大根堆: 父亲大, 孩子小; 小根堆: 父亲小, 孩子大)

其次理解堆排序调整是从最后一个非叶结点开始调整, 一直到根结点, 这个过程“自下而上”

最后理解每次非叶结点的调整是“从上往下”调整

→ 排序原理:

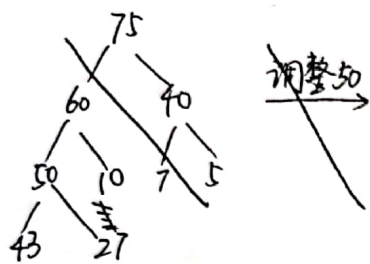
堆其实是用数组存储 (数据结构和为完全二叉树, 即完全二叉树的顺序存储, 即层次遍历序列的顺序存储)



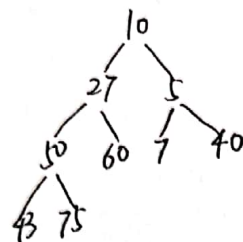
扫描全能王 创建

它的原理是每次调整完，将堆顶元素与最底层最右边元素互换（相当于数组的第一个元素与最后一个元素互换），然后再调整新产生的完全二叉树使其满足堆定义，（即重新调整数组前 $n-1$ 个元素的值，直到该数组有序之后结束（最后处理的二叉树仅有一个元素），这就是它的全流程了。下面实现教学。

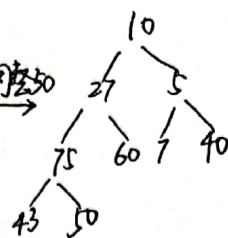
初始序列 ^{完全} 二叉树为



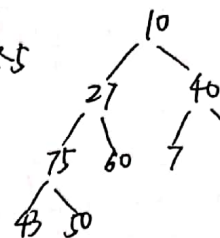
调整50



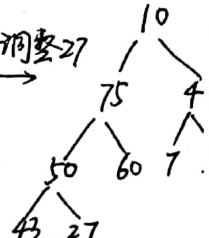
调整50



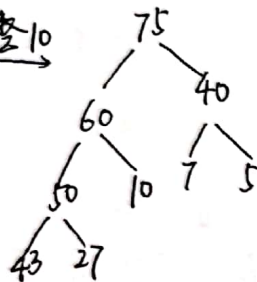
调整5



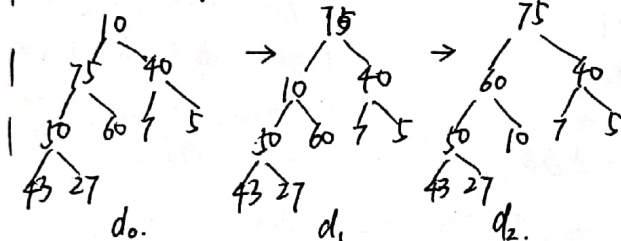
调整27



调整10

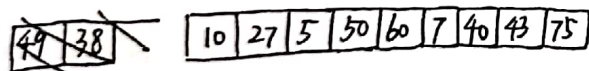


过程 a → e 是问题 (1) 所答全过程，只需注意两点，一是调整非叶结点是“自底向上”从右往左”的原则，而调整每一个非结点则是自上向下，如过程 d, e 都是经过多步调整（看作递归就好）
过程 d → e 详解：



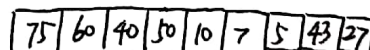
帮大家理解排序是怎么实现的。

初始序列



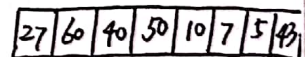
a.

初始大根堆



b.

交换 75 与 27



c.

这是一次堆排序的结果，75 跑到数组的最后面，然后我们重新将前 (27, 60, 40, 50, 10, 7, 5, 43) 作为初始序列
这是一个递归的过程，最后该数组将会有序，记住两个结论：
① 大根堆排序产生从小到大的序列
② 小根堆排序产生从大到小的序列

好的，堆排序所有考点都在这里！

→ 补充：学长将会对真题的每一种经典题型作非常详细的解析，希望我的工作能对得起大家给的表



5.

解:

算法的功能是返回二叉树中元素值为0的元素个数之和。

→ 开头数据结构是二叉树的链式结构, func函数是一个递归函数, 意思是若该结点值为0, 则返回1+该结点左子树中0个数+该结点右子树中0的个数; 若该结点不为0, 则直接返回左右子树中0的个数。

4. 解: 设集合S存储添加生成树的结点, $S_0 = \{V_1\}$, 尚未加入的结点集合 $T = \{V_2, V_3, V_4, V_5, V_6\}$

① 以 V_1 为起点, 此时候选边为 $V_1-V_2(16)$, $V_1-V_6(21)$, $V_1-V_5(19)$, 选择最小的 V_1-V_2 , $S_1 = \{V_1, V_2\}$
 $T_1 = \{V_3, V_4, V_5, V_6\}$

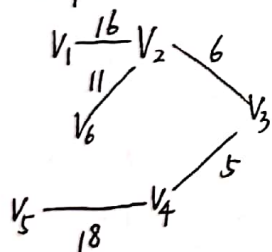
② S_1 到 T_1 候选边为 $V_1-V_5(19)$, $V_1-V_6(21)$, $V_2-V_6(11)$, $V_2-V_3(6)$, $V_2-V_4(6)$, 选择 V_2-V_3 , $S_2 = \{V_1, V_2, V_3\}$
 $T_2 = \{V_4, V_5, V_6\}$

③ S_2 到 T_2 候选边为 $V_1-V_5(19)$, $V_1-V_6(21)$, $V_2-V_6(11)$, $V_2-V_4(6)$, $V_3-V_4(5)$, 选择 V_3-V_4 , $S_3 = \{V_1, V_2, V_3, V_4\}$, $T_3 = \{V_5, V_6\}$

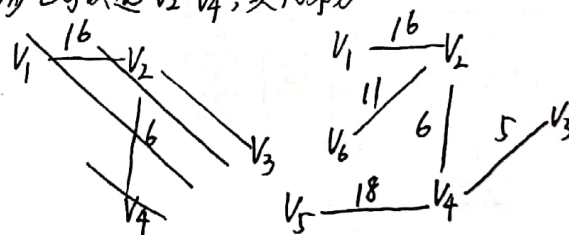
④ S_3 到 T_3 候选边为 $V_1-V_5(19)$, $V_1-V_6(21)$, $V_2-V_6(11)$, $V_4-V_6(14)$, $V_4-V_5(18)$, 选择 V_4-V_5 , $S_4 = \{V_1, V_2, V_3, V_4, V_5\}$, $T_4 = \{V_6\}$

⑤ S_4 到 T_4 候选边为 $V_1-V_6(21)$, $V_5-V_6(33)$, $V_4-V_6(18)$, 选择 V_4-V_6 , $S_5 = \{V_1, V_2, V_3, V_4, V_5, V_6\}$

综上所述, 最小生成树



其实, 第②步也可以选 V_2-V_4 , 其结果为



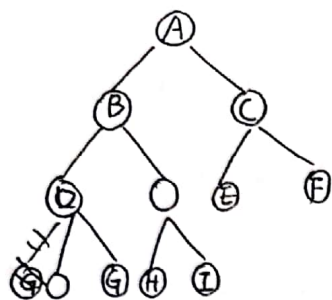
→ Kruskal

→ Prime 算法与克鲁斯卡尔算法 (Kruskal 算法) 区别在于一个只关心权值最小, 一个是已加入生成树结点到未加结点中所有路径权值最小。



5.

(1) 本题疑以为题中图有误, 按题干要求如下图



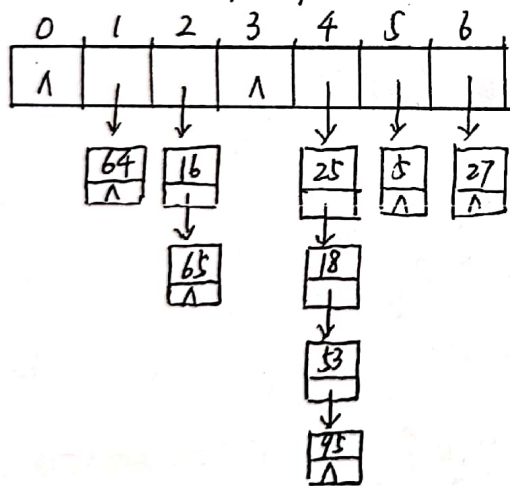
若题干图2成立的话如上图, 一个空结点上挂了H,I, 明显不合理.
→ 题干有问题, 第(2)不再讲解.

6.

(1) 由散列函数 $H(key) = key \% 7$ 得各元素的散列元素如下表

key	64	5	95	53	18	25	65	27	16
H(key)	1	5	4	4	4	4	2	6	2

采用链地址法解决冲突,得



(2) 查找 95 时, 需依次与 25, 18, 53, 95 比较

(3) $ASL_{成功} = \frac{1 \times 5 + 2 \times 2 + 3 \times 1 + 4 \times 1}{9} = \frac{16}{9}$



五. 填空题

1.

① p

② p = p->next

③ p->next = q->next

④ free(q)

→ 本算法是删除链表中值最小元素，算法前半部分在找链表中最小元素，后半部分找到最小元素的前一个元素，再 $p \rightarrow next = q \rightarrow next$ 删除。

→ 注意 ③④ 句不在 while 循环里面。

→ 这类简单应用严版书没有参考，这是代码题变题的另一种出发点，平时积累就好。

2.

① $j \leq n-1$

② $i \leq j$

③ $a[i] = a[i+1]$

④ $a[i+1] = temp$

⑤ $flag = 0$ $flag == 0$

→ 这是“冒泡排序”的全过程

① 代表排序趟数 $n-1$ 趟，写 $j < n$ 也 OK，也可以从另一个角度看，看这句代码 ($a[i].key > a[i+1].key$)，于是 i 最大为 $n-1$ ，所以②空填 $i \leq j$ 。但若①填了 $j < n$ ，②应该填 $i < j$ ，③-④就是交换。

→ $flag$ 的作用是判断是否排序结束，若结束，那么不会置 $flag = 0$ 。



六. 递归算法

1.

```
int Bsearch (int R[], int low, int high, int k)
{
    int mid;
    while (low <= high)
    {
        mid = (low + high) / 2;
        if (R[mid] == k)
            return mid;
        else if (R[mid] > k)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return 0;
}
```

→ 非常常规的题, 这种分必级拿下.

2.

```
bool Topologicalsort (Graph G) {
    InitStack(S);
    for (int i = 0; i < G.Vexnum; i++) {
        if (Indegree[i] == 0) // 入度数组.
            push(S, i);
    }
    int count = 0;
    while (!IsEmpty(S)) {
        pop(S, i); count++;
        printf("%d ", i); // 打印拓扑排序序列
        for (p = G.vertices[i].firstarc; p; p = p->nextarc) { // 遍历依附于顶点i的全部边(弧)
            // 将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈S
            v = p->adjvex; // 该弧所指向顶点位置.
            if (! (--Indegree[v]))
                push(S, v);
        } // for
    } // while
    if (count < G.Vexnum)
        return false;
    else
        return true;
}
```

