

2015 年哈工大计算机科学与技术专业期 末题解析

一、选择题

- 1。A。顺序入栈,第一个出栈的为 n(最后入栈的数字),故后续出栈顺序也一定是按 n-1、n-2,,,3,2,1
- 2。A。详情见王道书中森林与二叉树的转换。
- 3。B、D。参考四个选项的定义
- 4。C。观察每个选项,看当可以两端入队时,按照 abcde 的入队顺序能否得到 选项中的序列
- 5。A。根据图的邻接表定义易得
- 6。C。题目描述的其实就是满二叉树
- 7。B。书中原话
- 8。B。初始堆是一个堆,堆顶元素必为最大(或最小),然后按照构造初始堆的办法,自己将这个初始堆构造出来。方法在书中有。
- 9。D。冒泡排序有一个 flag 标志一轮冒泡中是否发生交换, 初始已有序时, 第一次冒泡完毕即停止排序。对于快速排序, 初始序列已经基本有序或者基本逆序时, 为最坏情况。直接插入排序的效率也与初始序列的状态有关。选择排序, 元素间比较次数与序列初始状态无关。
- 10。B。根据 B 树定义。至少含有多少个非叶节点,则要求每个结点的子树尽可能多(三阶 b 树,每个结点最多三颗子树)

二、填空题

- 1。克鲁斯卡尔。克鲁斯卡尔适用于顶点多,边少的图,即稀疏图。
- 2。10;38。问至少有多少个顶点,则考虑每个顶点之间两两都有边连接的情况。对于9个顶点的图,两两都有边时,有9*(9-1)/2=36条边,题目说37条,故必须再多一个顶点。第二个空,考察生成树的性质
- 3。(n+1)n/2; n*n。采用压缩矩阵时,只存上半部分和对角线元素
- 4。4、15。最少情况:每次分支时再编一个且仅编一个码。0110分支了四次,故最少还可以给4个字符编码。最多的情况,即哈夫曼树是一颗满二叉树。高度为5的满二叉树,叶节点个数为2的(5-1)次方=16,去掉已经编码的一个结点,还剩15个。
- 5。高度;阶数
- 6。9。二次探测再散列。d=0, 1, -1, 4, -4.。。

三、简答题

1.此题定义模糊。给出思路

无序数组,插入时在队尾插入。删除时找优先级最大的元素,删除,然后后续 元素前移

无序单向链表。插入时在链表头插入。删除时从表头找最大元素,然后删除 有序数组,让优先级最大的在数组尾部(方便删除),数组头部优先级最小, 插入时,从数组头部找位置,插入后后续元素后移。删除时直接删除数组尾部 元素。 有序单向链表, 让链表首结点为最大优先级, 插入时从首结点向后查找插入位置。删除时直接删除首结点。

最大堆。插入时,插入堆尾部,进行一次堆的插入操作(书中有时间复杂度),删除时进行堆的删除操作(即先将堆的最后一个元素与堆顶元素交换,再从根节点进行向下调整操作,书中有时间复杂度)。

2.王道书中有完整详细的置换-选择排序方法介绍

3.常规题,必考,王道书中有完整详细的知识介绍

四、算法设计

1.最简单思路:从头开始,依次比较相邻元素,当第一次出现前面的元素<后面的元素时,前面那个元素即为最小。

高级思路:运用二分法,当二分点处的值小于左右两点值时,继续二分。 2.要求不用栈和递归,则栈的作用由结点增加的父链域和标志域完成。 可参考王道课后题中,编写后序非递归遍历二叉树的方法。

```
算法思想:后序非递归遍历二叉树的顺序是先访问左子树,再访问右子树,最后访问根结点。
5用堆栈来存储结点时,必须分清返回根结点时是从左子柯返回的还是从右子树返回的。所以,
使用辅助指针 r. 其指向最近访问过的结点。也可在结点中增加一个标志域,记录是否已被访问。
    void PostOrder (BiTree T) [
       InitStack(S);
       p=T;
       r=NULL;
       while (p) | ! IsEmpty(S)) {
                                   //走到最左边
          if (p) {
             push (S,p);
              p=p->lchild;
                                   //向右
          else
                                   //取栈顶结点
             GetTop(S,p);
```

```
if(p->rchild%%p->rchild!=r)( //若右子網存在。且未被访问过
        p*p->rchild:
                             //转向右
          push (S.p);
                             //压入栈
         p=p->lchild:
                             //再走到最左
                             //否则, 弹出结点并访问
         pop(S,p);
                             //将结点弹出
         visit(p->data);
                             //访问该结点
         r=p;
                             //记录最近访问过的结点
         p=NULL;
                             //结点访问完后, 重置 p.指针
  1//else
1//while
```

访问一个结点*p 时,栈中结点恰好是*p 结点的所有祖先。从栈底到栈顶结点再加上*p 结点。刚好构成从根结点到*p 结点的一条路径。在很多算法设计中都利用了这一特性求解。如求根结点到某结点的路径、求两个结点的最近公共祖先等,都可以利用这个思路来实现。