《数据结构 A 类》（A 卷）笔试试卷参考答案

一、单项选择题（每格 1.5 分，共 24 分）

1. C　2. A　3. B　4. A　5. D　6. A　7. D　8. B　9. B

10. A　11. A　12. D　13. B　14. C　15. D　16. B

二、程序填充题（每格 1.5 分，共 24 分）

1. p!=NULL　!found　f = p;　t != p　t = NULL;

2. ptr != NULL　!stack.empty()　ptr != NULL　ptr = ptr->left;　ptr = ptr->right;

3. flag = false;　i+=2　t = a[i+1];　i+=2　t = a[i+1];　!flag

三、简答题（每题 8 分，共 24 分）

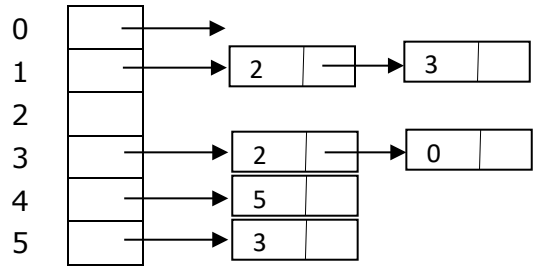
1. 应用置换选择，一共生成 3 个初始排序片段，具体如下：（2 分）

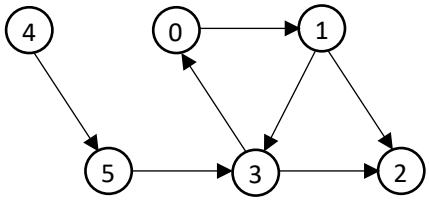
1) 2, 5, 10, 23, 34, 54（2 分）

2) 3, 4, 7, 12, 26, 33, 40（2 分）

3) 1, 11, 15, 18, 27, 35（2 分）

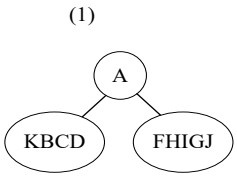
2. 该图的邻接表为：（4 分）

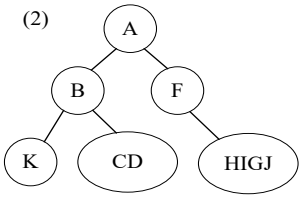


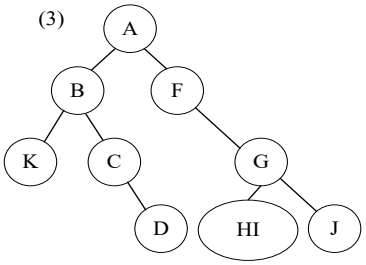


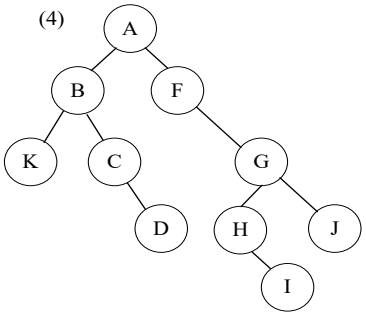
该图共有 4 个强连通分量。（4 分）

3. 该棵二叉树为：









四、分析题（20 分）

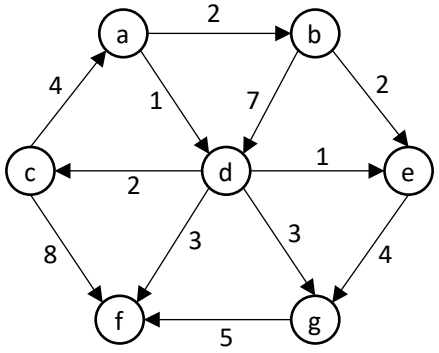
1. 在邻接矩阵类中实现 Kruskal 函数由两个步骤组成：
   1) 将所有的边加入优先级队列，这是通过扫描整个邻接矩阵完成的，所需的时间为 $O(|V|^2)$。（3 分）
   2) 执行|V|-1 次归并，这是通过处理|E|条边实现的。每次处理一条边时，先检查边的两个端点是否连通，即对两个端点分别调用 find 操作。在不相交集中，有|V|个结点，find 操作的时间复杂度是 O(log|V|)。如果两个端点不连通，则调用 Union 归并这两个子集。Union 函数的时间复杂度是 O(1)的，所以执行归并所需的时间是 O(|E|*log|V|)。（3 分）
   3) 总的时间复杂度就是 $O(|V|^2 + |E|*\log|V|)$。（2 分）

2. //寻找最短路径（1 分）
   //更新 min.node 的邻接点距离（1 分）
   //保留经过结点最少的那条路径（1 分）

   该算法的运行时间主要由两部分组成：
   1) 查找路径最短的尚未在 S 中的结点（设为 u），以及扫描 u 的邻接点，更新它们的距离。利用优先级队列存放源点到所有结点的距离，每次找出距离最短的结点所需的时间为 O(log|V|)，在整个算法的执行过程中，寻找距离最短的结点将花去 O(Vlog|V|)的时间。（2 分）
   2) 更新 V-S 中的结点的距离所需的时间是 O(|E|)，所以总的时间复杂度是 O(|E|+Vlog|V|) = O(Vlog|V|)。（2 分）

   该算法运行结果如下：（5 分）

   b to a:a-d-c-b Length:13
   b to b:b Length:0
   b to c:b-d-c Length:9
   b to d:b-d Length:7
   b to e:b-e Length:2
   b to f:b-d-f Length:10
   b to g:b-e-g Length:6



五、程序题（8 分）

对于最小化堆来讲，叶子上的结点之值大于等于内部结点之值。因此，挑选最大结点只需在叶子上挑选一个最大的即可。即：从下标为⌊n/2⌋+1 到下标为 n 的结点，进行一次比较。（2 分）

```
int max(int a[], int size) {
    int m = size/2;                          //（2 分）
    for (int k = size/2; k < size; ++k)      //（1 分）
        if (a[k] > a[m]) m = k;              //（2 分）
    return m;                                //（1 分）
}
```

六、附加题（10 分）

1) 算法思想（2 分）

定义一个大小为 n 的布尔数组，初始化为 false。如果被访问结点的元素值 a，flag[|a|]为 false，则保留此结点，并设 flag[|a|]为 true，否则删除该结点。

2) 结点的数据结构定义如下（6 分）

```
typedef struct Node {              //（1 分）
    int data;
    struct Node * next;
} Node;

bool flag[n];                      //全局数组，标志结点的绝对值的值是否出现过（1 分）

void DeleteABSEqualNode (Node * head) {
    memset (flag, false, n);               //初始化为 false
    if (head == NULL) return NULL;     //（1 分）
    Node * p = head; Node * r = head;
    while (p != NULL) {                    //（3 分）
        if (flag[abs(p->data)]) {          //如此绝对值已出现过
            r->next = p->next;
            delete p;
            p = r->next;
        }   //则删除当前结点
        else {
            flag[abs(p->data)] = true;
            r = p;
            p = p->next;
        }   //否则，将数组中对应的元素置 true，并将指针指向下一个元素
    }
    return head;
}
```

3) 只遍历一次链表，所以时间复杂度为 O(m)；因为申请大小为 n（n 为节点绝对值的最大值）的数组，所以空间复杂度为 O(n)。（2 分）