# 2019 华中科技大学 887 数据结构与算法分析

## 【满神劝退卷】 卷一

## 仅供测试之用

**一、名词解释（25 分）**

1.DP

【解析】动态规划(dynamic programming)是运筹学的一个分支，是求解决策过程(decision process)最优化的数学方法。动态规划把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解。

2. 单源最短路径

【解析】给定一个带权有向图 G=（V,E），其中每条边的权是一个实数。另外，还给定 V 中的一个顶点，称为源。现在要计算从源到其他所有各顶点的最短路径长度。这里的长度就是指路上各边权之和。这个问题通常称为单源最短路径问题。

3.红黑树

【解析】红黑树是一种自平衡二叉查找树，是在计算机科学中用到的一种数据结构，典型的用途是实现关联数组。

红黑树是每个节点都带有颜色属性的二叉查找树，颜色或红色或黑色。在二叉查找树强制一般要求以外，对于任何有效的红黑树增加了如下的额外要求:

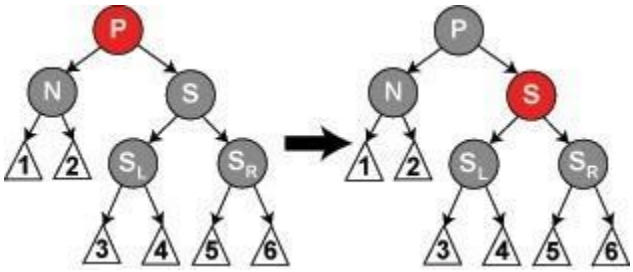
性质 1. 节点是红色或黑色。

性质 2. 根节点是黑色。

性质 3 每个叶节点是黑色的。

性质 4 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

性质 5. 从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。




4. BM

【解析】BM 算法是一种精确字符串匹配算法（区别于模糊匹配）。采用从右向左比较的方法，同时应用到了两种启发式规则，即坏字符规则和好后缀规则，来决定向右跳跃的距离。

5. 胜者树与败者树

【解析】胜者树和败者树都是完全二叉树，是树形选择排序的一种变型。每个叶子结点相当于一个选手，每个中间结点相当于一场比赛，每一层相当于一轮比赛。

不同的是，胜者树的中间结点记录的是胜者的标号；而败者树的中间结点记录的败者的标号。

## 二、选择题（25 分）

2.1 下列函数的增长率正确的是（ C ）。

A.. $2^N < 2^{1000} < N\log N$　　B.. $N^3 < 1000N < N\log N$

C. $1000\mathrm{N}\log\log n < N\log N < N^2$　　D. $N^{1.5} < N < N\log N$

2.2 假设栈和队列是已提供的非透明数据类型，仅实现了元素增加、元素删除、以及是否为 空的测试。一个程序员要计算一个栈或者队列 C 的元素个数，而 C 当前的状态是 t 并且 只能使用一个辅助的栈或队列 D。C 和 D 可以被任何合理的方式使用，但计数之后 C 必 须恢复到原来的状态 t。下面哪些选项可以实现上述的计数操作？（ A ）。

I. C 是队列并且 D 是队列

II. C 是栈并且 D 是栈

III. C 是队列并且 D 是栈

A. Ⅰ 和 II　　B. Ⅰ 和 III　　C. II 和 III　　D. Ⅰ，II 和 III

2.3 Kruskal 算法和 Prim 算法是计算图中最小生成树的两个经典算法，下列哪些项是肯定正确的？(A )。

①Kruskal 算法是一种贪心算法

②Kruskal 算法是一种动态规划算法

③Prim 算法是一种贪心算法

④Prim 算法是一种分治算法

A. ①③　　B. ①④　　C. ②③　　D. ②④

2.4 一个具有 2019 个节点的 7 阶 B 树，若根节点常驻内存，则一次查找最少进行（ D ）次 I/O 操作。

A. 7　　B. 6　　C. 5　　D. 4

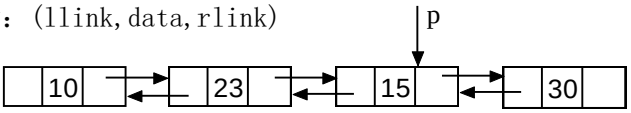
2.5 对一个集合连续的做两个操作：首先删除一个元素，然后马上将刚才被删除 的元素再插入回去。在两个操作进行之前和之后，集合的逻辑结构没有任何改变。若对三种数据结构查找树（即二叉排序树、散列表和 B+树）分别执行以上操作， 则其中操作之前和之后必定完全相同的数据结构有（ B ）种。

A.0　　B. 1　　C. 2　　D. 3

## 三、应用题（60 分）

3.1 写出下图双链表中对换值为 23 和 15 的两个结点相互位置时修改指针的有关语句。

结点结构为：(llink,data,rlink)




**【解析】**

设 q=p->llink; 则

```
q->rlink=p->rlink;
p->rlink->llink=q;
p->llink=q->llink;
q->llink->rlink=p;
p->rlink=q;
q->llink=p
```

3.2 设一棵树 T 中边的集合为{（A，B），（A，C），（A，I），（B，D），（B，E），（E，F），（C，G），（G，H），（I，J），（I，K）},按要求回答下面的问题：

（1） 请用图的形式表示该树。

（2） 请写出该树用孩子表示法时所对应的存储结构定义，并画出该树对应的存储结构。

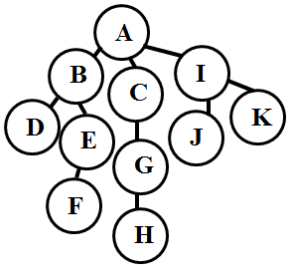
（3） 基于（2）中所定义的存储结构，对其进行深度遍历，请写出其深度遍历的结果。

（4） 该树转换成相应的二叉树，并以图的形式表示。

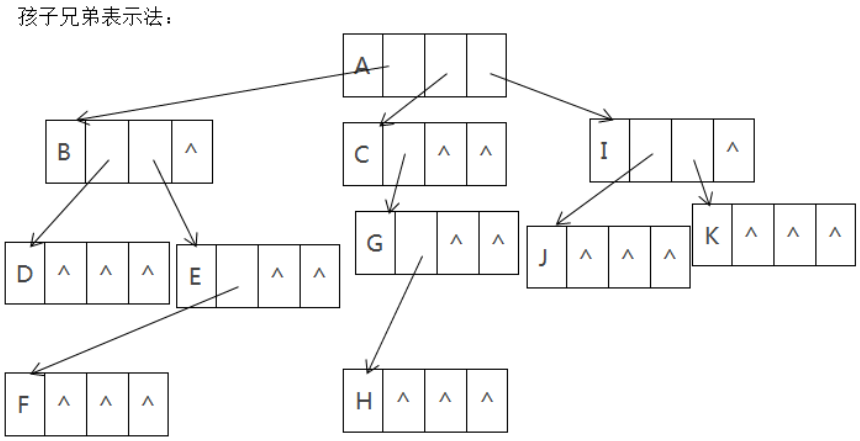
（5） 对（4）中的二叉树进行后序遍历，写出其后序遍历结果。

【解析】

（1）

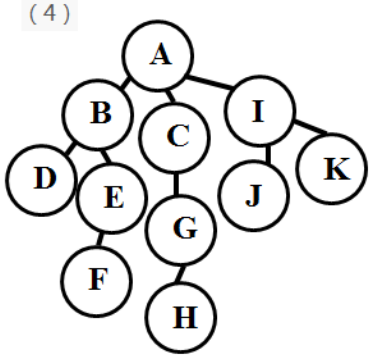


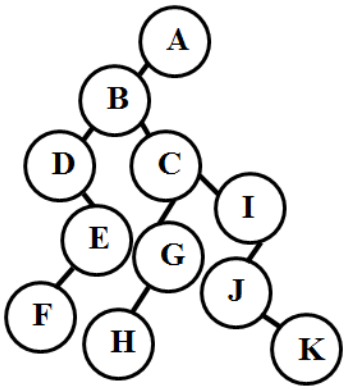

（2）定义请参考全书



（3）深度遍历：ABDEFCGHIJK

（4）



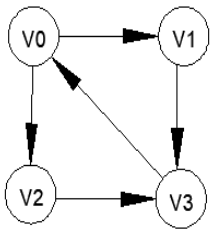

（5）后序遍历：FEDHGKJICBA

3.3 有向图 G 如图所示：

(1)求该有向图的邻接矩阵 A

(2)求 $A^2$，并说明 $A^2$ 中非零元素代表什么

(3)推广至 A 的 m 次方，说明 A 的 m 次方中非零元素代表什么




【解析】

（1）该有向图的邻接矩阵为 $A=\begin{bmatrix}0&1&1&0\\0&0&0&1\\0&0&0&1\\1&0&0&0\end{bmatrix}$

（2）$A^2=\begin{bmatrix}0&0&0&2\\1&0&0&0\\1&0&0&0\\0&1&1&0\end{bmatrix}$，其中非零元素代表当前顶点到指定顶点路径长度为 2 的路径条数，如 0 行 3 列元素代表顶点 0 到顶点 3 路径长度为 2 的路径共有 2 条

（3）$A^m$中位于 i 行 j 列的非零元素的含义为：图中从顶点 i 到顶点 j 长度为 m 的路径条数。

3.4 你有两个同样的玻璃球和一栋 100 层的高楼，已知这两个玻璃球会在 1~100 层的某一层楼会摔碎，你的任务是在最小的测量次数的前提下测试出玻璃球的摔碎楼层。给出最小的测量次数。

**解决方法：**假如只有一个球，那很显然，只有一个办法：从第一层开始投，如果没碎再试第二层、第三层.....

现在有两个球，我们应该利用第一个球缩小临界楼层所在的楼层范围，可能会想到第一个球先从 50 层开始投，如果碎了，再用最后一个球从第一层开始投，最多到 49 楼肯定可以找到临界楼层，如果没碎那就说明临界楼层在 50-100 中，此时继续尝试 75 楼....

可惜，上面的方法并非是最好的。因为最坏情况的投掷次数分布不均匀。比如，如果 50 楼碎了，此时最坏情况下需要尝试 50 次，如果 50 楼没碎，而 75 层不管是否碎了，最坏情况也只需要尝试 25 次，也就是说，50 楼投掷结果不同，所需最坏情况下的次数也不同，这就是分布不均匀的意思，那是否有办法让其分布均匀，也就是说假如在 f 层投，不管碎还是不碎，继续时最坏情况所需次数也是一样的。这样才能使最坏情况下所需投掷次数达到最小。

假设一开始从第 k 层投，如果坏了，那么第二个球可以在剩下的 k-1 层一定能用 k-1 次确定哪层坏，再加上第 k 层的那次，所以总共 k 次。如果第 k 层没有坏，那么第一个球在第 k 层的基础上再加 k-1 层（注意不是 k 层，原因见后面），所以第二次从 2k-1 层投下，在 k，k+1，k+2，……，2k-2，2k-1 中则有 k-2 层，所以用 k-2 次才能找出，再加上原来的那两次，还是共用了 k 次（为了保持总共还是 k 次，所以上面只能加 k-1 层）

依次类推.....

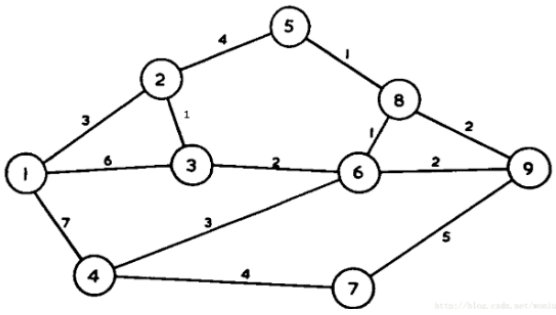
在第 2k-1 层基础上再加 k-2 层，在第 3k-3 层基础上再加 k-3 层，……，依次类推，一定能加到在第某层的基础上再加 2 层，最后再原来的基础上面还能再加 1 层，到目前为止，设为第 n-1 层，如果第 n-1 层坏了，那么就是第 n-1 层，如果没有坏，根据已知题说某一层能摔碎球，那么在第 n-1 层上面有且仅还有一层就是刚摔坏的层（此时不需要实际投，而是直接推理得出），这时用的次数还是 k 次（最不理想的结果），所以 k 次能测最多 n 层，由上面的解说，可知 n-1=k+(k+1)+(k+2)+(k+3)+……+2+1=k(k+1)/2，得 n=k(k+1)/2+1，由上面可知 n=100(层)，k 次最多能推测 k(k+1)/2+1(层)，所以 k(k+1)/2+1≥100，得 k(k+1)≥198，且 k 的最小整数为 14，所以最多 14 次就能找出题目中的那层。

**补充：**由上得知，第一次在 14 层投，第二次在第 27 层投，则依次为 14 27 39 50 60 69 77 84 90 95 99。还能得出 14 次最多能测得到 106 楼，只不过题目中给了 100 楼，如果是 107 楼，那么就需要 15 次了。

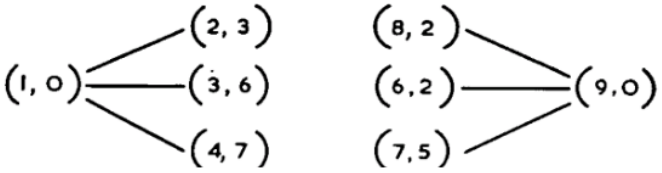
3.5 迪杰斯特拉按照离原点 s 的距离从近到远以此扩展的方式寻找最短路径。显然若 s 与 t 之间的最短路径长度为 d，则迪杰斯特拉方法需要搜索一个半径为 d 的球。

现对迪杰斯特拉算法进行改进，从起点与终点同时开始搜索，我们将其称为双向迪杰斯特拉算法。

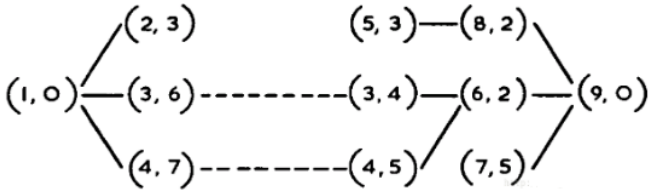
**现我们需要找到从 1 到 9 的最短路径：具体过程如下：**



**第一步：**因为到结点 1 与结点 9 已知的均为 0（相等），因此我们扩展两个分别以结点 1 和结点 9 为半径的圆，如图 2-2 所示。其中左边的（1, 0）表示从节点 1（源点 s）出发到结点 1 的距离为 0，同理（3, 6）表示结点 1 出发到结点 3 之间的距离为 6。显然有结点 1 与结点 3 之间的最短距离不是 6,而是 4。相似有（8, 2）表示结点 8 到结点 9 的距离为 2。




**第二步：**第一步求出的路条路径中结点 8 到结点 9 和结点 6 到结点 9 有最小距离 2。因此，我们扩展这两个顶点，扩展结果如下图所示。

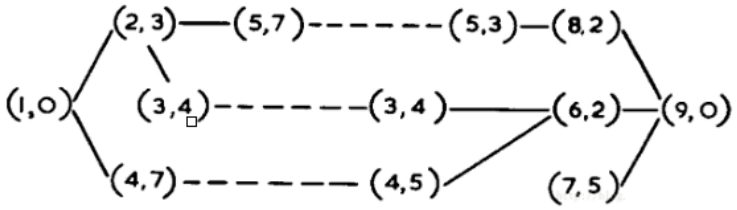



这时我们找到了路径两条路径，即 1-3-6-9、1-4-6-9，且这两条路径的距离分别为 10 与 12。但我们注意到从结点 1 出发最小值为 3，从结点 9 出发最小值也为 3，无法判断是否存在一条长度为 6 的最短路径，因此需要继续扩展。

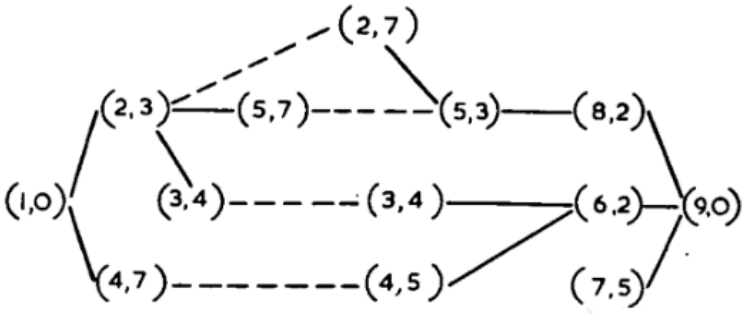
**请根据上述算法的描述补充第三步与第四步的图。**

**【解析】**

第三步




第四步



**四、编程题（40 分）**

4.1 一个树的节点的坡度定义即为，该节点左子树的结点之和和右子树结点之和的差的绝对值。空结点的的坡度是 0。整个树的坡度就是其所有节点的坡度之和。给定一个二叉树，计算整个树的坡度。

**示例:**

输入:

```
         1
       /   \
      2     3
```

输出: 1

解释:

结点的坡度 2 : 0

结点的坡度 3 : 0

结点的坡度 1 : |2-3|= 1

树的坡度 : 0 + 0 +1 = 1

注意:

任何子树的结点的和不会超过 32 位整数的范围。

坡度的值不会超过 32 位整数的范围。

**解决方法：**

```
//灰灰考研
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    int findTilt(TreeNode* root) {
    if(root == NULL) return 0;
    //abs是取绝对值
    else return abs(treesum(root->left)-treesum(root->right))+findTilt(root->left)+findTilt(root->right);
}
    int treesum(TreeNode* node){
    if(node == NULL) return 0;
    else return node->val+treesum(node->left)+treesum(node->right);
}
};
```

4.2 假设给定一个长度为 n 的数组，并且数组中每个值的位置距离排序后该值的位置不超过 k（小于或等于 k）， k<=n。比如数组[2 3 1 4 6 5 7 9 8]，每个值的位置距离其排序后的位 置不超过 2。设计一个最坏时间复杂度为 O(n log k)的排序算法，以伪代码形式给出，并解释该程序。

**错误的解决方法：**

**当我第一眼看到这个题目，我就在想这不就是一个分组排序的题目吗？再后来看到了算法导论的一个课后题，验证了我的猜想。**

**思路：将长度为 n 的数组分成 n/k 个子序列，这个子序列一定满足：**

1. **每个子序列中的元素都小于后继子序列中的元素**
2. **每个子序列中的元素都大于前驱子序列中的元素**

**如果我们分别对每个子序列进行排序，就可以得到整个序列的排序结果。**

**考虑复杂度即证明这个排序问题所需要比较的次数有一个下界值 O（nlogk）。**

```
//灰灰考研
void quickSort(int s[], int l, int r)
{
    if (l< r)
    {
        int i = l, j = r, x = s[l];
        while (i < j)
        {
            while (i < j && s[j] >= x) // 从右向左找第一个小于x的数
                j--;
            if (i < j)
                s[i++] = s[j];
            while (i < j && s[i]< x) // 从左向右找第一个大于等于x的数
                i++;
            if (i < j)
                s[j--] = s[i];
        }
        s[i] = x;
        quickSort(s, l, i - 1); // 递归调用
        quickSort(s, i + 1, r);
    }
}
int Sort(int A[], int k,int n) {
    if (n == 1)
        return 1;
    int rest = n / k;//分组个数

    for (int i = 0; rest; rest--) {

        quickSort(A, i, i+k);

        i = i + k;
    }
    return 0;

}
```

灰灰考研

**算法复杂度分析：**

**每个子序列的全排列为 k！则整个序列在上述条件下的排列数目为（k！）n/k。按照枚举决策树的分析方法，设树高为 h，则**

2^h>= (k!)^(n/k)，

得 h >= lg((k!)^(n/k)) =(n/k)lg(k!) >= (n/k)lg((k/2)^(k/2)) = (n/2)lgk

**h = O（nlgk）**

解决方法：

后来发现，解答一并不符合题意。

比如数组[2　3　1　4　6　5　7　9　8]；

当 k=2，分成 9/2=4，分成 4 组，[2,3]一组，会出现很明显的错误。

当 k=2，分成 9/（k+1）　= 3，分成 3 组，这样可以完成这个例子。

但是如果例子变成了[2，1，4，3，6，5，7，9，8]又会出现错误。

后来仔细想想

这道题考的应该是堆排序。

```
//灰灰考研
void adjust(int arr[], int len, int index)
{
    int left = 2*index + 1;
    int right = 2*index + 2;
    int maxIdx = index;
    if(left<len && arr[left] > arr[maxIdx]) maxIdx = left;
    if(right<len && arr[right] > arr[maxIdx]) maxIdx = right;  // maxIdx是3个数中最大数的下标
    if(maxIdx != index)                                       // 如果maxIdx的值有更新
    {
        swap(arr[maxIdx], arr[index]);
        adjust(arr, len, maxIdx);       // 递归调整其他不满足堆性质的部分
    }

}
void heapSort(int arr[], int size)
{
    for(int i=size/2 - 1; i >= 0; i--)  // 对每一个非叶结点进行堆调整(从最后一个非叶结点开始)
    {
        adjust(arr, size, i);
    }
    for(int i = size - 1; i >= 1; i--)
    {
        swap(arr[0], arr[i]);           // 将当前最大的放置到数组末尾
        adjust(arr, i, 0);              // 将未完成排序的部分继续进行堆排序
    }
}
```



时间复杂度为 O（nlogk）。