2017-2018 学年第 1 学期《算法设计与分析》试卷

参考答案和评分标准

一、填空题（评分标准：每答对 1 题计 1 分，答错不计分）

1、时间复杂度、空间复杂度　　2、相同

3、重叠子问题性质　　4、O(n)

5、概率算法　　6、约束函数、限界函数

7、分支限界法　　8、2^n

二、单选题（评分标准：每答对1题计 2 分，答错不计分）

BCADB　　AADAA　　BBCAD

三、算法应用题（评分标准：每答对1题计10分，否则酌情计分）

1、答：给定图 $G = (V, E)$

初始时，所有顶点均未被访问过，任选一个顶点 v 作为源点；

先访问源点 v，并将其标记为已访问过；

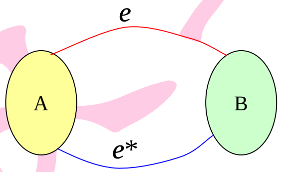
然后从 v 出发，选择 v 的一个未被访问过的邻接点（子结点）w，标记 w 为已访问，并以 w 为新的出发点，继续进行深度优先搜索；

如果 w 及其子结点均已搜索完毕，则返回到 v，再选择它的另一个未访问过的邻接点继续搜索，直到所有和源点连通的顶点均已访问过为止；

如果此时图 G 中仍存在未被访问过的顶点，则另选一个尚未访问过的顶点作为新的源点重复上述过程，直到图中所有顶点均已访问过为止。

2、证明：

假设 G 存在不止一棵最小耗费生成树，不失一般性，假设存在 T 和 T^* 均为无向图 $G = (V, E)$ 的最小耗费生成树，并且假设顶点集 A 和 B 是 V 的两个不相交子集，且 $A \cup B = V$，则至少存在边 $e \in \{E(T)\}$ 和 $e^* \in \{E(T^*)\}$ 连接顶点集 A 和 B，如图所示。



因为图 G 中并不存在权值相同的两条边，所以为不失一般性，假设边 e 的权值大于边 e^* 权值，于是可知 T 在集合 A 和 B 中的边的权值之和小于 T^* 在集合 A 和 B 中边的权值之和。

因此用边 e^* 去替换 T 中的边 e，可以构造一棵耗费更小的生成树 T'，这与 T 和 T^* 均是最小耗费生成树的假设矛盾，故假设不成立。

因此，对于没有两条边有相同权值的含权无向图而言，它具有惟一的最小生成树。

3、(1) 第 5 步执行的次数为：　　（3 分）

$$\sum_{i=1}^{\lfloor\log n\rfloor}\sum_{j=i}^{i+5}\sum_{k=1}^{i^2}1=\sum_{i=1}^{\lfloor\log n\rfloor}\sum_{j=i}^{i+5}i^2=\sum_{i=1}^{\lfloor\log n\rfloor}6i^2=\lfloor\log n\rfloor\cdot(\lfloor\log n\rfloor+1)\cdot(2\cdot\lfloor\log n\rfloor+1)$$

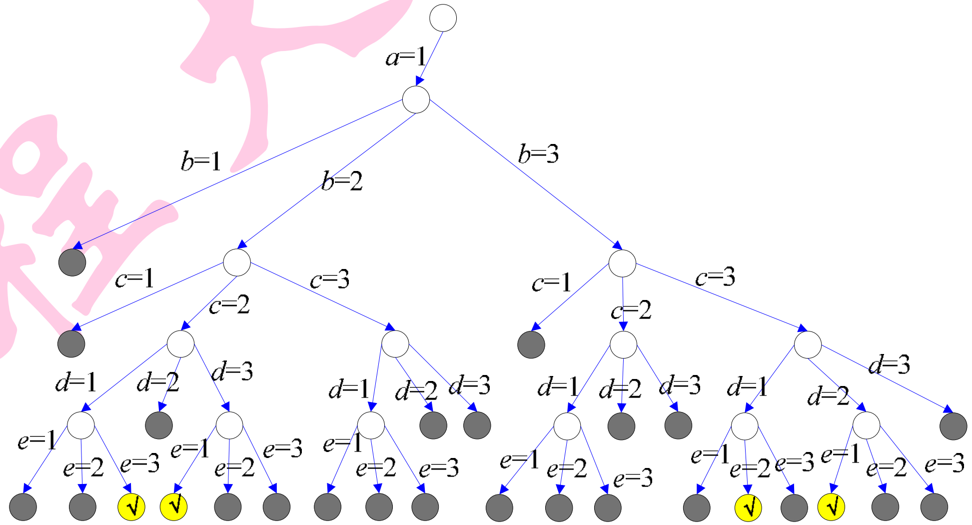
(2) 用 Θ 更合适；　　（2 分）

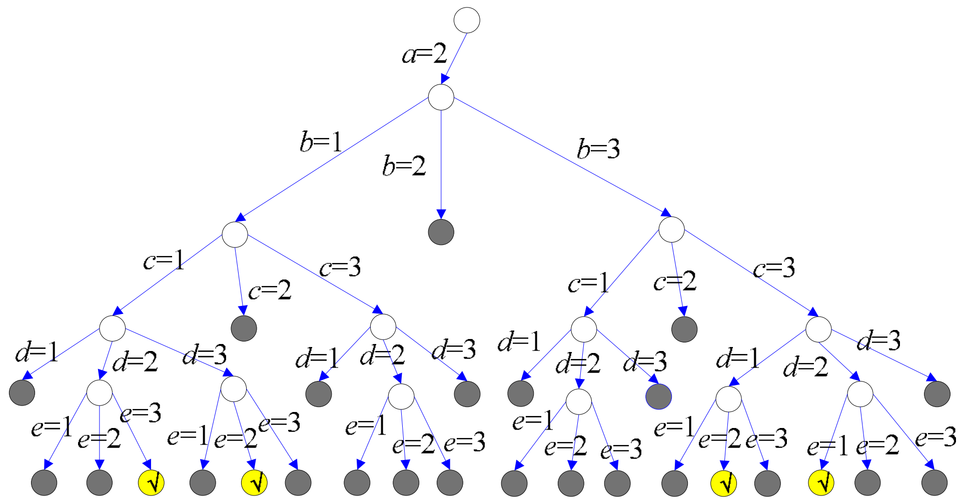
步骤 5 的执行次数与输入无关，只与输入规模有关。　　（3 分）

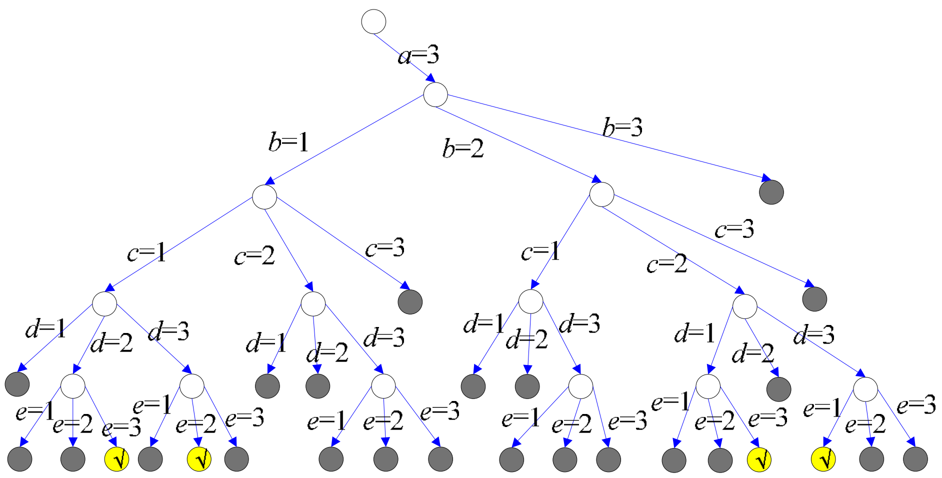
(3) 算法的时间复杂度为$\Theta(\log^3 n)$。　　（2 分）

4、解：假设三种颜色分别编号为 1, 2, 3，搜索过程如下所示：

（合法着色方案的搜索过程按根的每棵子树 2 分）







由上面三个图中搜索过程可知，所有可能的合法着色方案为：

(1, 2, 2, 1, 3), (1, 2, 2, 3, 1), (1, 3, 3, 1, 2), (1, 3, 3, 2, 1),
(2, 1, 1, 2, 3), (2, 1, 1, 3, 2), (2, 3, 3, 1, 2), (2, 3, 3, 2, 1),
(3, 1, 1, 2, 3), (3, 1, 1, 3, 2), (3, 2, 2, 1, 3), (3, 2, 2, 3, 1)。

答对一种可能的合法着色方案计 4 分

四、算法设计题（评分标准：每答对 1 题计 10 分，否则酌情计分）

1、分数背包求解：先按比值降序排序，再依次选择，直到背包满。

```
#include <iostream>
#include <algorithm>
using namespace std;
struct item
{   //  物品类型
    double value;   //  价值
    double weight;  //  重量
    double ratio;   //  比值
    double select;  //  装入的量
    int index;      //  原始编号
    bool operator < (const item &it) const      //  递减排序用
    {   return ratio > it.ratio;    }
};
bool restore(const item &it1, const item &it2)
{   //  还原原始排列用
    return it1.index < it2.index;
}
void fraction(item Set[], double c, int n)
{   //  求解分数背包
    for (int i = 0; i < n; i ++)
    {
        Set[i].ratio = Set[i].value / Set[i].weight;
        Set[i].select = 0;
        Set[i].index = i + 1;
    }
    sort(Set, Set + n);                 //  递减排序
    for (int i = 0; i < n && c > 0; i ++)
    {
        Set[i].select = c >= Set[i].weight ? Set[i].weight : c;
        c -= Set[i].weight;
    }
    sort(Set, Set + n, restore);        //  还原原始序列
    if (c > 0)
        cout << "背包太大" << endl;
}
```

2、调用格式：*CountX*(*A*, *x*, 1, *n*);

```
int CountX(int A[], int x, int begin, int end)
{   //  中点二等分治，begin 起点下标，end 终点下标
    int count = 0;
    if (end == begin)
        count += (A[begin] == x);
    else if (end > begin)
    {
        int mid = (begin + end) / 2;
        count += CountX(A, x, begin, mid) + CountX(A, x, mid + 1, end);
    }
    return count;
}
```