西安交通大学实验报告

课程名称：算法设计与分析　　实 验 名 称：分治与递归

学　　院：电子与信息学部　　实 验 日 期：2023 年 11 月 20 日

姓　　名：林圣翔　　班级：计试 2201　　学号：2223312202

诚信承诺：我保证本实验报告中的程序和本实验报告是我自己编写，没有抄袭。

一、实验目的

1.深入了解贪心算法的内容及应用

2.运用贪心算法的思想，用编程实现 T/S 的 d 森林问题。

二、实验环境

系统类型：64 位操作系统, 基于 x64 的处理器

处理器：12th Gen Intel(R) Core(TM) i7-1260P　　2.10 GHz

软件：Dev-C++ 5.9.2 TDM-GCC 4.8.1 64-bit Debug

三、问题描述

编程实现 T/S 的 d 森林问题。

设 T 为一带权树，树中的每个边的权都为整数。又设 S 为 T 的一个顶点的子集，从 T 中删除 S 中的所有结点，则得到一个森林，记为 T/S。如果 T/S 中所有树从根到叶子节点的路径长度都不超过 d，则称 T/S 是一个 d 森林。设计一个算法求 T 的最小顶点集合 S，使 T/S 为一个 d 森林。

输入格式：多组输入，第一行输入树包含的顶点个数 n 和路径长度 d。其中，树包含的顶点我们将其从 0 到 n-1 进行编号，0 为根节点。接下来 n 行分别对应树的第 0 个顶点到 n-1 个顶点的孩子节点的信息。具体的，每一行的第一个元素 k 表示一共包含 k 个孩子节点，接下来包含 2*k 个元素，两两配对分别表示孩子的编号 id 和边的权值 w。（除过第一行外，后面的第 i 行表示第 i-1 个节点的孩子节点个数 k 和对应的孩子节点编号和权值）

四、问题分析

我们需要求的是 T 的最小顶点集合 S，即我们要从 T 中拿走尽可能少的点，使其符合题意。我们可以转变一下思路，从原先由 T 中拿走尽可能少的点变为从 T 中选取尽可能多的点，不难看出，这两种表述是等价的。该问题具有最优子结构性质，我们可以作如下分析。设 p 为一节点，s[1],s[2],s[3]……s[n]表示其子节点，我们用 maxsonroad[i]记录 s[i]到其叶子结点的最大路径长度,dis[i]为 p 到 s[i]的路径长度。在这里，我们假设以 s[1],s[2],s[3]……s[n]的节点所表示的树均满足题意，那么我们需要考虑的是加入节点 p 之后，组成的新树能否满足题意。我们可以一一检验，若发现 maxsonroad[i]+dis[i]<=d（1≤i≤n），这说明加入节点 p 之后，组成的新树满足题意，这时 p 点可以加入；若

发现有点不符合 maxsonroad[i]+dis[i]<=d，那我们需要从这 n+1 个点中选取尽可能多的点，我们可以选取 n 个点，即不选节点 p。故综上所述，产生不满足题意的情况时，不选高度更大的节点，符合贪心性质，可以得到最优解。

五、算法设计

根据题目要求，我们在定义一个 node 类型的结构体，用来存储节点信息。其中，father 表示该节点的父节点（若无父节点，计为-1），dis 表示该节点到其父节点边的权重，degree 记录该节点子节点的数量，maxsonroad 表示该节点到其叶子结点的最大距离（初值为 0），flag 表示该点是否取（1 表示拿走，0 表示留下）。此外，我们用 nn 来记录点的总数，即题目中的 n，dd 来记录路径长度不超过的最小值，即题目中的 d，sum 表示应该删去的节点的个数，即题目所求的集合 s 大小的最小值。

首先根据题目输入，第 i 个节点及其信息用 node 类型的 tree[i]来记录，从而建立树 T。然后我们根据问题分析中提到的贪心算法的思想得出的结论，产生不满足题意的情况时，不选高度更大的节点，即尽可能选取高度低的节点，故从叶子结点向上遍历。我们在这里用一个队列 que 来存储可能可以选择的点。我们从叶子结点开始，若该点到其父节点的距离不大于 dd，说明该点是可以选择的点，故加入，否则，跳过。然后我们从子节点进行遍历。我们可以发现，当 que 为空时，说明我们已经对所有可能的点都遍历过了一遍，此时的 sum 即为所求，输出即可。当 que 中存在元素时，我们按照从前往后的顺序，每次选取 que 中存在的第一个元素即节点位置，记为 p，对其父节点进行判断，其父节点记为 fa，如果其父节点本身已经标记了不选即 tree[fa].flag=1，那就跳过，否则若 tree[p].maxsonroad+tree[p].dis>dd,说明 fa 不可取，故 tree[fa].flag=1。当然，这里还需要注意一点，我们在判断 fa 是否可取的时候，可以顺便更新 tree[fa].maxsonroad，即若 tree[fa].flag==0 且当前 tree[p].maxsonroad+tree[p].dis ≤ dd 时，让 tree[fa].maxsonroad=max(tree[fa].maxsonroad,tree[p].maxsonroad+tree[p].dis)。最后，每做完这样一次，我们对当前 tree[fa].degree-1,当我们发现当前的 tree[fa].degree==0 时，说明该父节点的子节点已经全部检查完，这是，fa 可以看作是新的子节点，加入 que 队列中。

六、算法实现

```
#include<bits/stdc++.h>
using namespace std;
struct node
{
    int father,dis,degree,maxsonroad,flag;
};
class dTree
{
    private:
        node tree[10010];
        deque<int> que;
        int sum=0,nn,dd;
    public:
        dTree(int n, int d)
        {
            nn=n;
            dd=d;
            tree[0].father=-1;
            tree[0].dis=0;
            for(int i=0; i<=n-1; i++)
```

```
        {
            cin>>tree[i].degree;
            tree[i].maxsonroad=tree[i].flag=0;
            for(int j=1; j<=tree[i].degree; j++)
            {
                int x,y;
                cin>>x>>y;
                tree[x].father=i;
                tree[x].dis=y;
            }
        }
    }
    void solution()
    {
        for(int i=nn-1; i>=0; i--)
        {
            if(tree[i].degree==0) que.push_back(i);
        }
        while(!que.empty())
        {
            int p=que.front();
            que.pop_front();
            int dis=tree[p].dis;
            int fa=tree[p].father;
            if(tree[fa].flag==0&&tree[p].maxsonroad+dis>dd)
            {
                tree[fa].flag=1;
                fa=tree[fa].father;
                sum++;
            }
            else if(tree[fa].flag==0&&tree[fa].maxsonroad<tree[p].maxsonroad+dis)
            {
                tree[fa].maxsonroad=tree[p].maxsonroad+dis;
            }
            tree[fa].degree--;
```

```
            if(tree[fa].degree==0) que.push_back(fa);
        }
        cout<<sum;
    }
};
int main()
{
    int n, d;
    cin >> n >> d;
    dTree dt(n, d);
    dt.solution();
    return 0;
}
```

七、运行结果

以下是部分数据输入与输出

第一组

程序输入：

```
5 5
3 1 4 2 9 3 2
0
1 4 5
0
0
```

程序输出：

```
1
```

答案为 1，程序运行正确

第二组

程序输入：

```
5 3
3 1 4 2 9 3 2
0
1 4 5
0
0
```

程序输出：

```
2
```

答案为 2，程序运行正确

第三组

程序输入：

```
5 1
3 1 4 2 9 3 2
0
1 4 5
0
0
```

程序输出：

```
2
```

答案为 2，程序运行正确

……