山东大学计算机科学与技术学院

数据结构与算法课程设计报告

学号：202000130143	姓名： 郑凯饶	班级： 20.1
上机学时： 6		日期：2022-5-15
课程设计题目：网络放大器设置问题		
软件环境：Windows 10 家庭中文版 64 位（10.0，版本 18363） Microsoft VS Code cmake version 3.23.0-rc3		

报告内容：

1. 需求描述

1.1 问题描述

问题描述

对于一个石油传输网络可由一个加权有向无环图 G 表示。该图中有一个称为源点的顶点 S，从 S 出发，石油流向图中的其他顶点，S 的入度为 0，G 中每条边上的权重为它所连接的两点间的距离。

在输送石油的过程中，需要有一定的压力才能使石油从一个顶点传输到另一个顶点，但随着石油在网络中传输，压力会损失，压力的损失量是传输距离的函数。

为了保证石油在网络的正常运输，在网络传输中必须保证在任何位置的压力都要不小于最小压力 Pmin。为了维持这个最小压力，可在 G 中的一些或全部顶点放置压力放大器，压力放大器可以将压力恢复至该网络允许的最大压力 Pmax。

设 d 为石油从压力 Pmax 降为 Pmin 所走的距离，在无压力放大器时石油可输送的距离不超过 d 的情况下，要求在 G 中**放置最少数量的放大器**，使得该传输网络从 S 出发，能够将石油输送到**图中的所有其他顶点**。

为了简化计算，可设每走一个单位长度就会降低一个单位压力且 Pmin=0，即 d=Pmax-Pmin=Pmax。起点处的压力为 Pmax。为了尽可能保证石油的运输（考虑到现实中某些管道可能损坏），首先我们保证所有通道都可以运输，即最大边权值不超过 d；且如果顶点 x 有多条入边，x 处的压力为到达 x 处的压力中的最小值。

1.2 基本要求

基本要求

1. 设计并实现加权有向无环图的 ADT。
2. 给出两种方法以解决上述问题，验证两种方法的正确性。
3. 比较两种方法的时间和空间性能，用图表显示比较结果。

程序正确性的验证要求

你需要设计程序验证你所实现的方法的正确性，本次实验我们提供了示例数据集供你验证程序的正确性。以下是关于此数据集的一些输入输出说明。

1.3 输入说明

输入界面设计

文件流输入

输入样例

8 9 56

1 2 9

1 3 54

2 4 37

2 5 53

2 7 36

2 8 44

2 4 18

4 6 32

4 6 18

1.4 输出说明

输出界面设计

文件流输出

输出样例

2

2. 分析与设计

2.1 问题分析

该问题是一个图论中树形 DP 问题的变形。问题保证图为有向无环图（DAG），根据一个贪心结论，我们 dfs 过程中递推出答案。

贪心结论如下：（与数据结构课本 P285 中结论相似，但它假设了网络为树形结构，我们的问题节点入度可能大于 1，因此重新论述）

设节点 i 到它的每一个后代都有一个压力衰减值,我们将其最大者记为 toLeaf[i]，它可以有如下公式计算：

$$\text{toLeaf[i]} = \max_{\{j\ is\ i's\ son\}} toLeaf[j] + dis[i][j]$$

为方便计算我们记节点 i 最长入边长为 fa[i]，这样判断节点是否放置放大器只需判断 toLeaf[i] + fa[i] 是否大于 d（题目给定压力初始值）。若不在该节点放置，在其祖先节点放置，它所处分支必定存在节点不满足压力要求；另一方面，若在该节点的后代进行放置，可能可以保证该节点无需放置，但会浪费部分压力，使总放大器数增大。

以上。

2.2 主程序设计

```
/*
Main() {
    f.open(path);     // 文件流读入/输出

    init();           // 变量初始化
    solve();          // 两种算法分别执行
        dfs();        // 记忆化搜索，回溯时进行求解
        topo();       // 对节点进行拓扑排序，按照拓扑序进行递推
    bfs();            // 根据算法结果进行图遍历，计算节点最终的压力值
    visualResult();   // 可视化计算结果

    f.close();        // 文件流关闭
}
*/
```

2.3　设计思路

思路主要围绕问题的解决，再得到贪心结论之后我们该如何进行 toLeaf[]数组的推导，其实我们只需保证所有孩子的 toLeaf 在其父亲之前计算出即可，我们可以采用 dfs 或者以拓扑序进行递推即可。

2.4 数据及数据类(型)定义

边的定义：

```
struct edge
{
    int v, w;
    edge(int v, int w):v(v), w(w) {}
};
```

由于是有向带权图，我们记录端点以及边长。

图的存储：

```
vector<edge> g[N], g1[N];
```

以邻接表的方式存储图以及反图。

其他数据定义：

```
vector<int> t;          // 存储拓扑序
bool vis[N];            // vis记录节点是否访问
bool vis1[N];           // vis1记录答案，是否放置网络放大器
int toLeaf[N];          // i到后代节点压力的最大衰减值
int in_deg[N];          // 节点入度，用于拓扑排序
int fa[N];              // 节点的最长入边，方便递推计算
int p[N];               // 节点压力值
```

2.5. 算法设计及分析

方法一：根据拓扑逆序进行递推

```cpp
void inferInTopo() {
    tsort();
    reverse(t.begin(), t.end());

    for (int i = 0; i < n; i++) {
        int u = t[i];
        if (toLeaf[u] + fa[u] > d) {
            ans++;
            toLeaf[u] = 0;
        }
        for (auto j : g1[u]) {
            toLeaf[j.v] = max(toLeaf[j.v], toLeaf[u] + j.w);
        }
    }
}
```

算法步骤：
(1)Tsort()计算拓扑序并存储与 vector 对象 t，我们反转 t 得到拓扑逆序。
(2)对于节点 u 我们判断若 toLeaf[u] + fa[u] > d 则在当前节点放置放大器，而且我们可以保证当前分支压力供给已经满足了，将 toLeaf[u]置为 0.
(3)同时将该节点压力信息反馈给其父亲节点.

时间复杂度分析：
计算拓扑序列的时间为 O(所有节点的入度之和)，实际为 O(m)，m 为边的数目；递推的时间为 O(n)，按照拓扑逆序进行一遍递推，因此总的时间复杂度为 O(m + n).

空间复杂度分析：
节点相关信息存储使用空间为 O(n)，邻接表存图占用 O(m)，因此总的空间复杂度为 O(m + n).

方法二：dfs 回溯

```cpp
void dfs(int u) {
    vis[u] = 1;
    for (auto i : g[u]) {
        // cout << u << "->" << i.v << '\n';
        if (!vis[i.v]) dfs(i.v);
        toLeaf[u] = max(toLeaf[u], toLeaf[i.v] + i.w);
    }
    // -> 决策点u是否放置发大器
    if (fa[u] + toLeaf[u] > d) {
        vis1[u] = 1;
        ans++;
        toLeaf[u] = 0;
    }
}
```

（1） 标记当前节点已访问，并继续深度优先搜索；
（2） 回溯时计算当前节点 toLeaf；
（3） 判断是否放置放大器。

时间复杂度分析：
实际上就是图遍历的复杂度，为 O(m)，m 为图中边的数目。

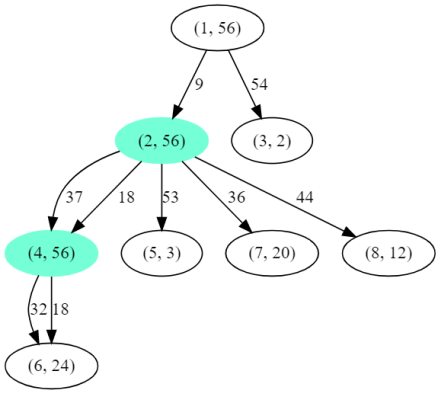
空间复杂度分析：
Dfs 最大的递归深度不超过节点数目，为 O(n)，邻接表存图占用 O(m)，因此总的空间复杂度为 O(m + n).

3. 测试
正确性测试：

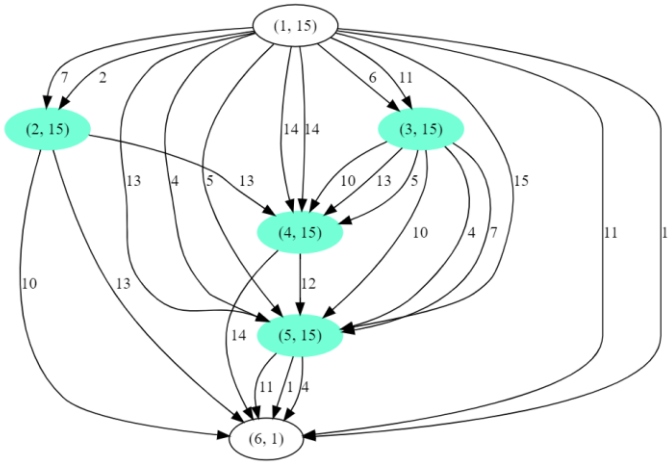
```
void c() {
    string s2 = "C:\\Users\\DELL\\Desktop\\Coder\\DSA\\4\\Test\\output\\my.out";
    f2.open(s2);
    for (int i = 1; i <= 100; i++) {
        string s1 = "C:\\Users\\DELL\\Desktop\\Coder\\DSA\\4\\Test\\input\\input" + to_string(i) + ".in";
        string path = "C:\\Users\\DELL\\Desktop\\Coder\\DSA\\4\\Test\\output\\" + to_string(i) + ".dot";
        f1.open(s1);

        init();
        solve(1);
        viResult(n, g, vis1, p, path);
        f1.close();
    }
    // chk();
}
```

全部通过助教给的测试点。

结果可视化呈现：
测试点 97：



测试点 91：



性能测试：

通过 dataGenerator.cpp 中的数据生成器生成大规模节点数的完全图进行性能碰撞，通过引入<time.h>和<psapi.h>进行时间性能和空间性能的比较：

```
ofstream f;
f.open("p.in");
srand(time(0));

int n = 7e3, m = n * (n - 1) / 2; // (n - 1), ... , 1, 0
int d = rand() % 10000;
f << n << ' ' << m << ' ' << d <<'\n';
for (int i = 1; i <= n; i++) {
    for (int j = i + 1; j <= n; j++) {
        f << i << ' ' << j << ' ' << rand() % d / 10 << '\n';
    }
}
f.close();
```

结果展示如下：

```
n = 1e4
ans: 9979
method dfs: 95630 ms
memory usage: 1202876416 b
ans: 9979
method topo: 97096 ms
memory usage: 1202077696 b

n = 5e3
ans: 4973
method dfs: 23958 ms
memory usage: 309669888 b
ans: 4973
method topo: 24275 ms
memory usage: 309272576 b
```

```
n = 1e3
ans: 980
method dfs: 987 ms
memory usage: 22339584 b
ans: 980
method topo: 1001 ms
memory usage: 22233088 b

n = 100
ans: 78
method dfs: 10 ms
memory usage: 11497472 b
ans: 78
method topo: 10 ms
memory usage: 11497472 b
```

4. 分析与探讨

这次实验解决了网络放大器问题，练习了图论问题相关的算法设计。还通过 graphviz 进行结果可视化演示，以及在性能测试方面引入了<time.h>和<psapi.h>获取程序的时空性能进行比较。

5. 附录：实现源代码

（另附）