

学号：202200130039	姓名：张向荣	班级：22.3
上机学时：4 小时		实验日期：2024. 4. 29
课程设计题目： 网络放大器设置问题		
软件开发环境： Dev-c++		
报告内容： <p>1. 需求描述</p> <p>1.1 问题描述</p> <p>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。</p> <p>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。</p> <p>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壓力都要不小于最小压力 P_{min}。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 P_{max}。</p> <p>设 d 为石油从压力 P_{max} 降为 P_{min} 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。</p> <p>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 $P_{min}=0$，即 $d=P_{max}-P_{min}=P_{max}$。起点处的压力为 P_{max}。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。</p> <p>1.2 基本要求</p> <ol style="list-style-type: none"> 设计并实现加权有向无环图的 ADT。 给出两种方法以解决上述问题，验证两种方法的正确性。 比较两种方法的时间和空间性能，用图表显示比较结果。 <p>1.3 输入说明</p> <p>输入文件</p> <p>第一行包含三个整数 $[n, m, d]$，设源点 S 为 1 号顶点。</p> <ul style="list-style-type: none"> n 表示顶点个数 m 表示边的个数 		

- d 表示石油从压力 P_{\max} 降为压力 P_{\min} 所走的距离。
- 接下来 m 行，每行三个正整数[x, y, c]用来标识一条有向边。
- x 表示起点 • y 表示终点
 - c 表示边权

输入界面设计

首先输入测试样例组数，然后输入想要测试的样例的序号。

输入样例

1.4 输出说明

一行一个整数，表示最少的放大器的个数。

输出界面设计

每组测试样例都会输出所需放大器数目、两种方法所需时间、空间。

输出样例

2. 分析与设计

2.1 问题分析

要求 1 说明要定义一个无环图的类并实现相关功能。

要求 2 说明要定义两种算法求解。

要求 3 说明要计算出每种算法的时间与空间性能，并使用可视化图表展示比较结果。

2.2 主程序设计

在 main.cpp 中，首先输入想要测试的样例组数，然后定义四个数组，分别用来存储每组测试样例在两种方法下的时间与空间。然后使用可视化将其转化为图表。

2.3 设计思路

在 graph.h 中首先定义一个边结构体，存储起点、权值。定义两个数组，用来存储两种算法的放大器数量。定义一个 Graph 类，用来实现相关功能，包括初始化、两种算法、求出空间性能，可视化。

2.4 数据及数据类型定义

对于存储放大器的数组为 bool 类型，每个存储单元有为 true，没有则为 false。

遍历数组为 bool 类型的动态数组，图的数组为 Edge 结构体类型的 vector 动态数组。其余数组全部为 int 型的动态数组。

2.5 算法设计及分析

在 graph.h 中定义一个边结构体，存储起点与权值，定义一个构造函数对起点与权重赋值。定义一个类 Graph，其中私有成员变量有图的顶点、边数与最大压力。以及存储边的 vector 数组，放大器的数量，表示是否遍历的动态数组，记录每条边的出度的动态数组，以及计算空间变量。

在 Graph.cpp 中实现类的相关功能。

在 Graph() 构造函数中，打开输入文件，读取顶点数、边数与最大压力。首先将数组初始化，读取每条边的起点终点，权值，并将起点与边权存放在终点所对应的数组中，记录终点的边数+1。

在 method1() 函数中，表示第一种求放大器数量的方法。首先将没有出度的点存入队列中，并设置 vis 遍历数组为 1。当队列不为空时，开始循环，取队首元素，将其从队列中删除，遍历该元素的边数组，如果该点的压力加起点的边权超过了最大压力，说明该位置的位置需要放置放大器，使用数组记录，并将压力清 0，更新放大器的数量。重新遍历该元素的每个起点，更新起点压力的值，为原来的值与该点加边权的最大值。并且将起点的边减去与队首元素连接的边，如果起点的出度为 0 且未遍历，则将起点压入队列中，设置为遍历。循环直到队列元素为空。计算数组所用的空间。

在 method2() 函数中，使用了 0/1 背包解法，每个点都有设置与不设置 2 种情况，共 n 个点，就有 2^n 种情况。对 2^n 种情况进行遍历，将每个点存储情况存储在一个数组中。判断

存储的放大器数量，如果超过之前记录的最小数量，则该情况不符合，直接遍历下一种情况。否则，判断该情况是否符合要求，首先将出度为 0 的点存入队列，取出队首元素，如果该位置需要放置放大器，则将压力置为 0，判断队首元素连接的起点的压力，更新起点压力的值为原来的值与该点加边权的最大值，如果超过了最大压力，说明该情况不成立，直接结束循环进行下一种情况分析。否则将该边的出度从起点中删除，如果出度为 0 且该点未遍历，将该点存入队列，将该点设置为遍历。直到队列为空，判断该种情况的放大器数量，如果小于之前统计的数量，则更新放大器数量，将新的存放情况存储到存储点放置放大器情况的数组中，最后求出所有情况种使用放大器数量最少得情况。记录数组所用空间大小。

Getans1() 函数中，返回第一种方法放大器数量。

Getans2() 函数中，返回第二种方法放大器数量。

Getspace1() 函数中，返回第一种方法所用的空间。

Getspace2() 函数中，返回第二种方法所用的空间。

Visualize1() 函数中，首先打开相应序号对应的 dot 文件，通过刚刚方法一的数组记录的放大器的位置，将所在位置节点颜色设置为红色，否则设置为白色。将每条边的权值记录下来。

```
digraph {
1[label="node1",style=filled,fillcolor=white];
2[label="node2",style=filled,fillcolor=red];
3[label="node3",style=filled,fillcolor=red];
4[label="node4",style=filled,fillcolor=red];
5[label="node5",style=filled,fillcolor=red];
6[label="node6",style=filled,fillcolor=red];
7[label="node7",style=filled,fillcolor=red];
8[label="node8",style=filled,fillcolor=red];
9[label="node9",style=filled,fillcolor=red];
10[label="node10",style=filled,fillcolor=red];
11[label="node11",style=filled,fillcolor=white];
12[label="node12",style=filled,fillcolor=red];
13[label="node13",style=filled,fillcolor=red];
14[label="node14",style=filled,fillcolor=red];
15[label="node15",style=filled,fillcolor=white];
1->2[label="cost=59"];
1->2[label="cost=48"];
1->2[label="cost=67"];
1->3[label="cost=33"];
2->4[label="cost=13"];
1->5[label="cost=64"];
2->5[label="cost=5"];
3->5[label="cost=41"];
3->5[label="cost=19"];
3->5[label="cost=32"];
4->5[label="cost=52"];
4->5[label="cost=87"];
1->6[label="cost=67"];
2->6[label="cost=32"];
4->6[label="cost=21"];
5->6[label="cost=90"];
5->6[label="cost=29"];
5->6[label="cost=27"];
5->6[label="cost=88"];
4->7[label="cost=5"];
5->7[label="cost=26"];
5->7[label="cost=77"];
6->7[label="cost=10"];
6->7[label="cost=38"];
6->7[label="cost=33"];
2->8[label="cost=29"];
4->8[label="cost=92"];
7->8[label="cost=31"];
1->9[label="cost=68"];
1->9[label="cost=83"];
2->9[label="cost=85"];
6->9[label="cost=65"];
7->9[label="cost=55"];
7->9[label="cost=62"];
7->9[label="cost=71"];
1->10[label="cost=6"];
1->10[label="cost=28"];
1->10[label="cost=39"];
2->10[label="cost=48"];
3->10[label="cost=77"];
7->10[label="cost=90"];
7->10[label="cost=50"];
9->10[label="cost=41"];
9->10[label="cost=68"];
3->11[label="cost=49"];
5->11[label="cost=78"];
5->11[label="cost=14"];
5->11[label="cost=66"];
1->12[label="cost=58"];
5->12[label="cost=76"];
5->12[label="cost=17"];
5->12[label="cost=40"];
8->12[label="cost=84"];
8->12[label="cost=17"];
9->12[label="cost=29"];
9->12[label="cost=16"];
10->12[label="cost=22"];
10->12[label="cost=47"];
10->12[label="cost=47"];
11->12[label="cost=2"];
4->13[label="cost=53"];
5->13[label="cost=43"];
5->13[label="cost=89"];
6->13[label="cost=65"];
6->13[label="cost=88"];
8->13[label="cost=88"];
9->13[label="cost=26"];
9->13[label="cost=13"];
10->13[label="cost=16"];
12->13[label="cost=57"];
12->13[label="cost=59"];
12->13[label="cost=26"];
2->14[label="cost=49"];
3->14[label="cost=92"];
5->14[label="cost=2"];
5->14[label="cost=58"];
5->14[label="cost=52"];
6->14[label="cost=67"];
13->14[label="cost=84"];
2->15[label="cost=76"];
3->15[label="cost=67"];
4->15[label="cost=28"];
7->15[label="cost=85"];
8->15[label="cost=94"];
8->15[label="cost=59"];
9->15[label="cost=25"];
9->15[label="cost=40"];
10->15[label="cost=32"];
12->15[label="cost=94"];
14->15[label="cost=45"];
```

Visualize2() 函数中，打开相应序号对应的 dot 文件，通过方法二中数组记录下各点放置放大器的情况，将有放大器的节点的颜色设置为红色，否则设置为白色。将每条边的权值记录下来。

```

digraph g {
1[label="node1",style=filled, fillcolor=white];
2[label="node2",style=filled, fillcolor=red];
3[label="node3",style=filled, fillcolor=red];
4[label="node4",style=filled, fillcolor=red];
5[label="node5",style=filled, fillcolor=red];
6[label="node6",style=filled, fillcolor=red];
7[label="node7",style=filled, fillcolor=red];
8[label="node8",style=filled, fillcolor=red];
9[label="node9",style=filled, fillcolor=red];
10[label="node10",style=filled, fillcolor=red];
11[label="node11",style=filled, fillcolor=white];
12[label="node12",style=filled, fillcolor=red];
13[label="node13",style=filled, fillcolor=red];
14[label="node14",style=filled, fillcolor=red];
15[label="node15",style=filled, fillcolor=white];
1->2[label="cost=59"];
1->2[label="cost=48"];
1->2[label="cost=67"];
1->3[label="cost=33"];
2->4[label="cost=13"];
1->5[label="cost=64"];
2->5[label="cost=5"];
3->5[label="cost=41"];
3->5[label="cost=19"];
3->5[label="cost=32"];
4->5[label="cost=52"];
4->5[label="cost=87"];
1->6[label="cost=67"];
2->6[label="cost=32"];
4->6[label="cost=21"];
5->6[label="cost=90"];
5->6[label="cost=29"];
5->6[label="cost=27"];
5->6[label="cost=88"];
4->7[label="cost=5"];
5->7[label="cost=26"];
5->7[label="cost=77"];
6->7[label="cost=10"];
6->7[label="cost=38"];
6->7[label="cost=33"];
2->8[label="cost=29"];
4->8[label="cost=92"];
7->8[label="cost=31"];
1->9[label="cost=68"];
1->9[label="cost=83"];
2->9[label="cost=85"];
6->9[label="cost=65"];
7->9[label="cost=55"];
7->9[label="cost=62"];
7->9[label="cost=71"];
4->10[label="cost=6"];
1->10[label="cost=39"];
2->10[label="cost=48"];
3->10[label="cost=77"];
7->10[label="cost=90"];
7->10[label="cost=50"];
9->10[label="cost=41"];
9->10[label="cost=68"];
3->11[label="cost=49"];
5->11[label="cost=78"];
5->11[label="cost=14"];
5->11[label="cost=66"];
1->12[label="cost=58"];
5->12[label="cost=76"];
5->12[label="cost=17"];
5->12[label="cost=40"];
8->12[label="cost=84"];
8->12[label="cost=17"];
9->12[label="cost=29"];
9->12[label="cost=16"];
10->12[label="cost=22"];
10->12[label="cost=47"];
10->12[label="cost=47"];
11->12[label="cost=2"];
4->13[label="cost=53"];
5->13[label="cost=43"];
5->13[label="cost=89"];
6->13[label="cost=65"];
6->13[label="cost=88"];
8->13[label="cost=88"];
9->13[label="cost=26"];
9->13[label="cost=13"];
10->13[label="cost=16"];
12->13[label="cost=57"];
12->13[label="cost=59"];
12->13[label="cost=26"];
2->14[label="cost=49"];
3->14[label="cost=92"];
5->14[label="cost=2"];
5->14[label="cost=58"];
5->14[label="cost=52"];
6->14[label="cost=67"];
13->14[label="cost=84"];
2->15[label="cost=76"];
3->15[label="cost=67"];
4->15[label="cost=28"];
7->15[label="cost=85"];
8->15[label="cost=94"];
8->15[label="cost=59"];
9->15[label="cost=25"];
9->15[label="cost=40"];
10->15[label="cost=32"];
13->15[label="cost=94"];

```

```

1->10[label="cost=39"];
2->10[label="cost=48"];
3->10[label="cost=77"];
7->10[label="cost=90"];
7->10[label="cost=50"];
9->10[label="cost=41"];
9->10[label="cost=68"];
3->11[label="cost=49"];
5->11[label="cost=78"];
5->11[label="cost=14"];
5->11[label="cost=66"];
1->12[label="cost=58"];
5->12[label="cost=76"];
5->12[label="cost=17"];
5->12[label="cost=40"];
8->12[label="cost=84"];
8->12[label="cost=17"];
9->12[label="cost=29"];
9->12[label="cost=16"];
10->12[label="cost=22"];
10->12[label="cost=47"];
10->12[label="cost=47"];
11->12[label="cost=2"];
4->13[label="cost=53"];
5->13[label="cost=43"];
5->13[label="cost=89"];
6->13[label="cost=65"];
6->13[label="cost=88"];
8->13[label="cost=88"];
9->13[label="cost=26"];
9->13[label="cost=13"];
10->13[label="cost=16"];
12->13[label="cost=57"];
12->13[label="cost=59"];
12->13[label="cost=26"];
2->14[label="cost=49"];
3->14[label="cost=92"];
5->14[label="cost=2"];
5->14[label="cost=58"];
5->14[label="cost=52"];
6->14[label="cost=67"];
13->14[label="cost=84"];
2->15[label="cost=76"];
3->15[label="cost=67"];
4->15[label="cost=28"];
7->15[label="cost=85"];
8->15[label="cost=94"];
8->15[label="cost=59"];
9->15[label="cost=25"];
9->15[label="cost=40"];
10->15[label="cost=32"];
13->15[label="cost=94"];

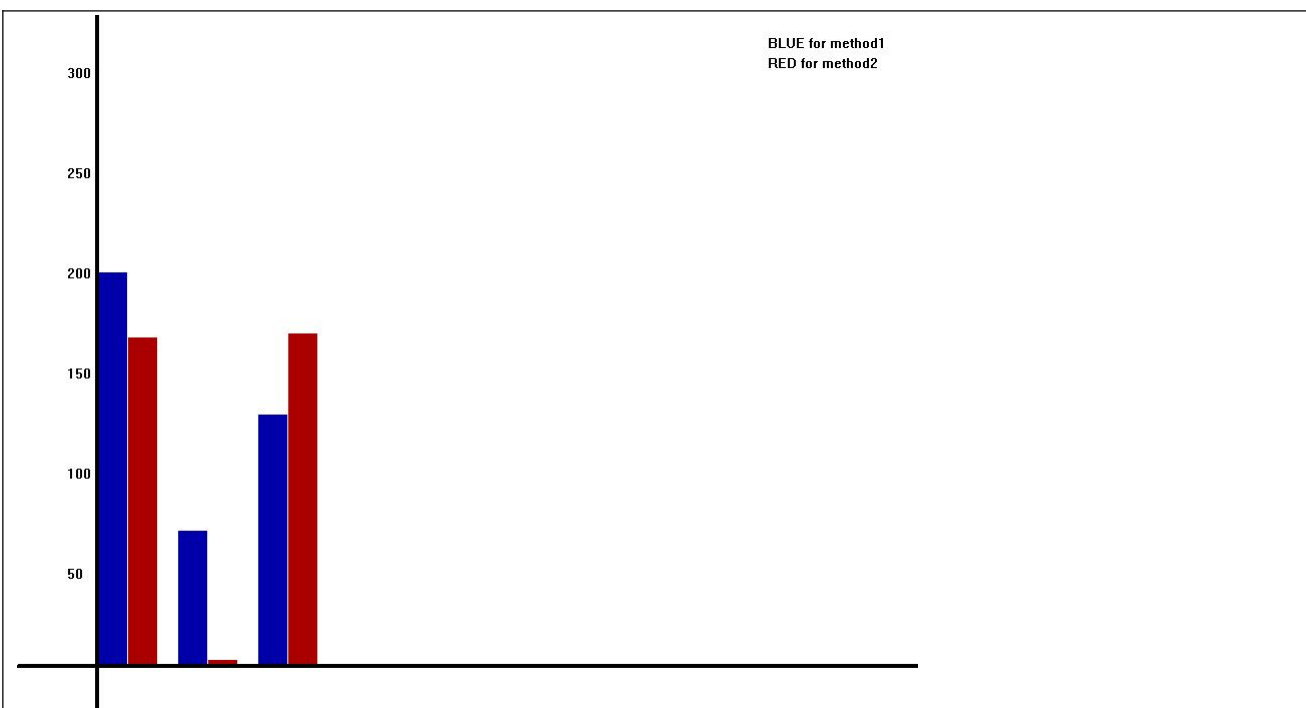
```

Compare1() 函数中，使用库 easyx.h 中的函数创建窗口，initgraph() 设置窗口大小，setbkcolor() 设置背景颜色，cleardevice() 清除当前绘图窗口的所有图形，将第一种方法每组所用的时间用蓝色填充在条形图中，第二种方法每组所用的时间用红色填充在条形图中，设置线条样式、颜色。使用 line() 函数在窗口上绘制线条，做坐标轴，并且设置坐标轴上的刻度，使用 saveimage() 函数保存图片。

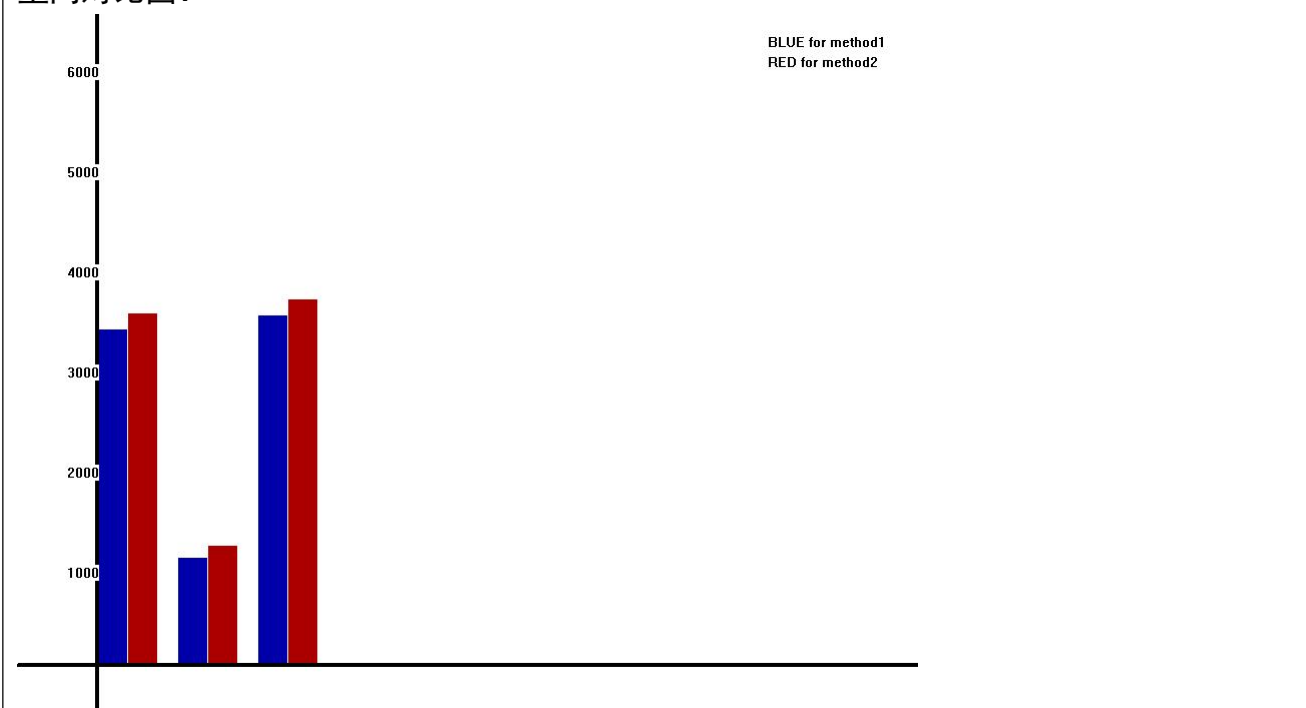
Compare2() 函数中，使用库 easyx.h 中的函数创建窗口，initgraph() 设置窗口大小，setbkcolor() 设置背景颜色，cleardevice() 清除当前绘图窗口的所有图形，将第一种方法每组所用的空间用蓝色填充在条形图中，第二种方法每组所用的空间用红色填充在条形图中，设置线条样式、颜色。使用 line() 函数在窗口上绘制线条，做坐标轴，并且设置坐标轴上的刻度，使用 saveimage() 函数保存图片。

以三组测试样例为例：

时间对比图：



空间对比图：



3. 测试

提交详情(网络放大器)

代码

运行

3 总结

评测信息

结论	测试通过
编号	7ef6896d0003a65
提交时间	2024-04-26 20:12:00
评测时间	2024-04-26 20:12:07
评测模板	C++17

关联信息

提交用户	202200130039 (ID: 10990)
题目信息	网络放大器(1)

关闭

测试结果通过样例。

4. 分析与探讨

在方法一中，使用了 bfs 和拓扑排序，时间复杂度为 $O(n+m)$ 。

方法二中，使用了 0/1 背包的算法和 bfs，时间复杂度为 $O(2^n (n+m))$ 。

5. 附录：实现源代码

```
#include<bits/stdc++.h>
#include<Windows.h>
#include<graphics.h>
#include<conio.h>
#include<easyx.h>
#include"graph.h"

Graph::Graph(string str)
{
    ifstream fin(str);
    if(fin.is_open())
    {
        cout<<"文件"<<str<<"打开成功!"<<endl;
    }
    else
    {
        cout<<"打开失败！ "<<endl;
    }
    fin>>n>>m>>Pmax;
    vis = new bool[n+1];
    indeg = new int[n+1];
    degL = new int[n+1];
    for (int i=1;i<=n;i++)//初始化
```

```

{
    vis[i]=false;
    indeg[i]=0;
    degL[i]=0;
}

gra=new vector<Edge>[n+1];//边数组

for(int i=1;i<=m;i++)
{
    int u,v,w;
    fin>>u>>v>>w;//起点，终点，边权
    Edge tp(u,w);
    gra[v].push_back(tp);//终点存放其起点与边权
    indeg[u]++;//起点边数++
}
}

```

void Graph::method1()

{//方法一

```

    ans1 = 0;
    memset(nums1,0,sizeof nums1);//初始化
    int* tpindeg=new int[n+1];
    for(int i=1;i<=n;i++)
    {
        tpindeg[i]=indeg[i]; //存储边数
        vis[i]=0;
        degL[i]=0;
    }
    queue<int>que;

    //首先将只有入度的顶点进队列
    for (int i=1;i<=n;i++)
    {
        if (indeg[i]==0) //起点无边离开，无出度
        {
            que.push(i);
            vis[i]=1;//设置为遍历
            degL[i]=0;
        }
    }
    while (!que.empty())
    {
        int x=que.front();
        que.pop();
        for(auto& i: gra[x])

```

```

    {
        if(degL[x]+i.w>Pmax)
        {
            nums1[x]=true;//超压
            ans1++;//放大器++
            degL[x]=0;//设置放大器后，压力清 0
            break;
        }
    }
    for(auto& i : gra[x])
    {
        //设置起点的压力值
        degL[i.to]=max(degL[i.to],i.w+degL[x]);
        //i 的起点的点，
        if (--tpindeg[i.to]==0&&!vis[i.to]) //如果除了这条边没有别的边从起点离开
        {
            que.push(i.to);//起点进队列
            vis[i.to] = 1;//此顶点设置为遍历
        }
    }
}

space1=sizeof(gra)*m+sizeof(tpindeg)*n+sizeof(vis)*n+sizeof(degL)*n;//计算空间
delete[]tpindeg;
}

```

void Graph::method2()

```

{
    ans2 = INT_MAX;
    memset(nums2,0,sizeof nums2);//压力放大器清 0
    int* tpnums=new int[n+1];
    memset(tpnums,0,sizeof tpnums);//初始化

    for (int i=0;i<pow(2,n);i++)//0/1 背包
    {
        int idx=0,temp=i,tpans=0;//

        while(idx<n) //遍历二进制 n
        {
            if(temp%2==1) tpans++;
            //放大器数量
            tpnums[++idx]=temp%2;
            //存二进制
            temp/=2;
        }
        //
        if(tpans>=ans2) continue;
        //放大器数量超过前面求过的情况
        //的数量
        int* tpindeg=new int[n+1];
        queue<int>que;
        for (int j=1;j<=n;j++)

```

```

{
    tpindeg[j]=indeg[j];//记录起点的出度
    degL[j]=0;
    vis[j]=0;
    if(tpindeg[j]==0)//出度为 0 的点进队列
    {
        que.push(j);
        vis[j]=true;//设置为遍历
    }
}
while (!que.empty())
{
    int x = que.front();
    que.pop();
    if (tpnums[x]==1) degL[x]=0;//表示应该放置
    for (auto& e : gra[x])
    {
        degL[e.to]=max(degL[e.to],degL[x]+e.w);//压力
        if (degL[e.to]>Pmax)
        {
            tpans=INT_MAX;//假如 x 点没有放置，导致 e.to 超压，说明这种情况
            break;
        }
        if (--tpindeg[e.to]==0&&!vis[e.to])//除去这条边起点没有别的出度边，起点
        {
            que.push(e.to);//起点压入队列
            vis[e.to]=1;//设置为遍历
        }
    }
    if(tpans==INT_MAX) break;
}
if(tpans<ans2)
{
    //求最小放大器数量
    ans2 = tpans;
    for (int i=1;i<=n;i++) nums2[i]=tpnums[i];//存储当前二进制数组
}
}
space2
sizeof(indeg)*n+sizeof(gra)*m+sizeof(vis)*n+sizeof(degL)*n+sizeof(tpnums)*n;//空间计算
delete[]tpnums;
}

```

不可取

未遍历

int Graph::getans1()

{

```

    return ans1;
}

int Graph::getans2()
{
    return ans2;
}

int Graph::getspace1()
{
    return space1;
}

int Graph::getspace2()
{
    return space2;
}

void Graph::visualize1(int i)
{
    //打开文件
    string name = to_string(i)+"-m1.dot";
    ofstream out(name);
    out << "digraph g {\n";
    for(int i=1;i<=n;i++)
    {
        out<<i;
        if (nums1[i])//放置网络放大器
            out<<"[label=\"node\"<<i<<"\",style=filled, fillcolor=red];\n";
        else
            out<<"[label=\"node\"<<i<<"\",style=filled, fillcolor=white];\n";
    }
    for(auto i=1;i<=n;++i)
    {
        for(auto& edge:gra[i])
        {
            out<<edge.to<<"->"<<i<<"[label=\"cost=\"<<edge.w<<"\"";\n";
        }
    }
    out << "}\n";
    out.close();
    string str1=to_string(i)+"-m1.dot";
    string str2=to_string(i)+"-m1.png";
    string order = "dot -Tpng "+str1+" -o "+str2;
    system(order.c_str());
}

```

```

void Graph::visualize2(int i)
{
    string name=to_string(i)+"-m2.dot";
    ofstream out(name);
    out << "digraph g {\n";
    for (int i = 1; i <= n; i++)
    {
        out<<i;
        if (nums2[i])//放置网络放大器
            out<<"[label=\"node\"<<i<<"\",style=filled, fillcolor=red];\n";
        else
            out<<"[label=\"node\"<<i<<"\",style=filled, fillcolor=white];\n";
    }
    for(auto i=1;i<=n;++i)
    {
        for(auto& edge : gra[i])
        {
            out<<edge.to<< "->" <<i<<"[label=\"cost=" <<edge.w<<""];\n";
        }
    }
    out<<"}\n";
    out.close();
    string str1=to_string(i)+"-m2.dot";
    string str2=to_string(i)+"-m2.png";
    string order="dot -Tpng "+str1 +" -o "+str2;
    system(order.c_str());
}

```