山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

学号：202200130048	姓名： 陈静雯	班级： 6
实验题目：网络放大器设置问题		
实验学时：4	实验日期： 5.7	

实验目的：

针对网络设计问题考虑使用两种方法解决，并比较两种方法的时间性能，用图表显示比较结果。

软件开发工具：

Vscode

1. 实验内容

一个汽油传送网络可由加权有向无环图 G 表示。图中有一个称为源点的顶点 S。从 S 出发，汽油被输送到图中的其他顶点。S 的入度为 0，每一条边上的权给出了它所连接的两点间的距离。通过网络输送汽油时，压力的损失是所走距离的函数。为了保证网络的正常运转，在网络传输中必须保证最小压力 Pmin。为了维持这个最小压力，可将压力放大器放在网络中的一些或全部顶点。压力放大器可将压力恢复至最大可允许的量级 Pmax。令 d 为汽油在压力由 Pmax 降为 Pmin 时所走的距离。在设置信号放大器问题中，需要放置最少数量的放大器，以便在遇到一个放大器之前汽油所走的距离不超过 d。编写一个程序来求解该问题。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

以下都是按边算的，即保证每条边都有油

（1）拓扑+贪心：对一个图（保证有一个起点，即入度为零的点），按照它的拓扑序列，首先把入度为零的点放入队列，每次 pop 队首，遍历队首节点邻接的点，根据每条边的花费，判断它的下一个点的 p 会不会小于 pmin，如果不会，更新邻接点取 p 的较大值，如果小于了，那么该点就要放一个放大器，并重新遍历该点，更新邻接点的 p 值，还是取较大的一个。

（2）dfs 分支定界（一次剪枝）：首先用 ans 维护一个最优解，用 dfs 遍历所有可能的方案，对于该点放还是不放有两种可能，共 2 的 n 次种情况，dfs 深搜时，如果当前方案的数量已经大于最优解，那么直接递归，如果没有，判断该种情况能否确保每个点的 p 值是否大于 pmin，如果是则方案可行，更新 ans。

（3）回溯（剪枝又剪枝）：

①最开始的活结点是根节点，之后求活结点的子节点进行判断其能否作为扩展结点。

②求出子节点压力，如果子节点有<pmin 的，则该节点作为扩展结点，放一个放大器，cnt++，否则作为活结点，直到 level == n-1 结束（最后一个一定不需要放放大器）

③用一个子集树节点存储每个扩展结点的父节点，即从哪来的，press，level，是否放放大器，bstnum,该路上已经放了多少个 booster,以此 max_to_cost，用来减少复杂度限界使用。

之后进行求解，使用优先队列存储子集树节点，按照 bstnum 排序，最后求出第一个 level = n-1 的解，即最优解。

3. 测试结果（测试输入，测试输出）

输入：

```
cpppp > filein.in
1  51
2  8 12
3  1 2 50
4  1 3 40
5  1 6 42
6  2 5 41
7  2 6 22
8  2 4 1
9  5 7 32
10 5 6 9
11 4 6 11
12 7 8 35
13 7 8 23
14 6 8 7
```

拓扑+贪心：

```
3
0.000000 s
PS D:\code repository\code>
```

dfs 分支定界：

```
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
3
0.001000 s
PS D:\code repository\code>
```

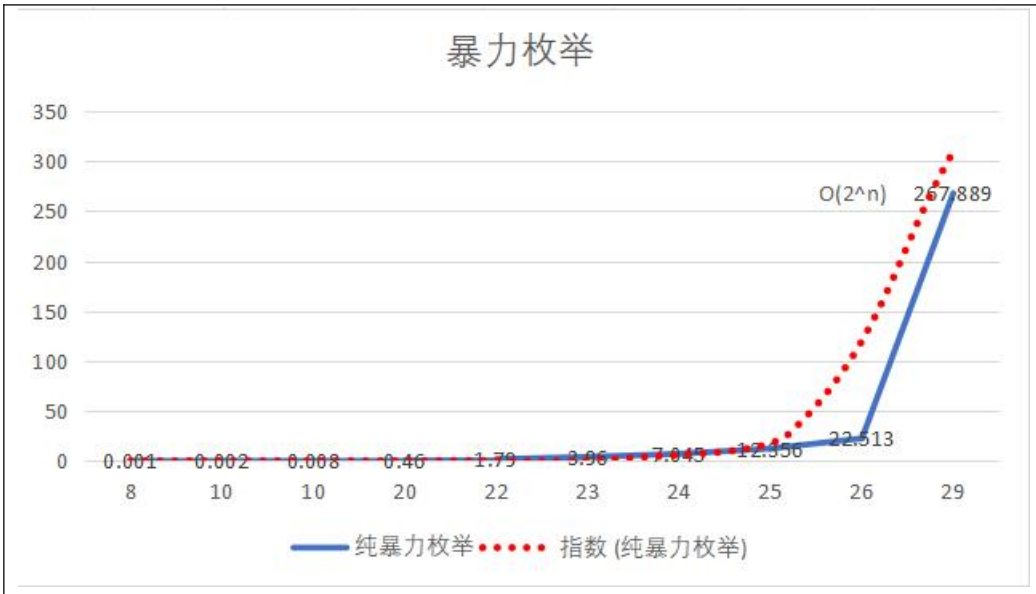
回溯：

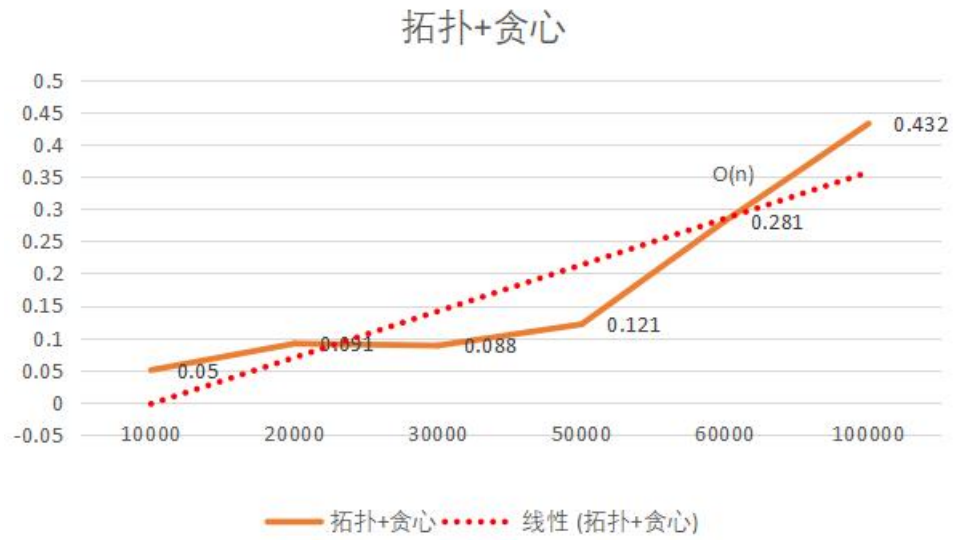
```
3
0.000000 s
```

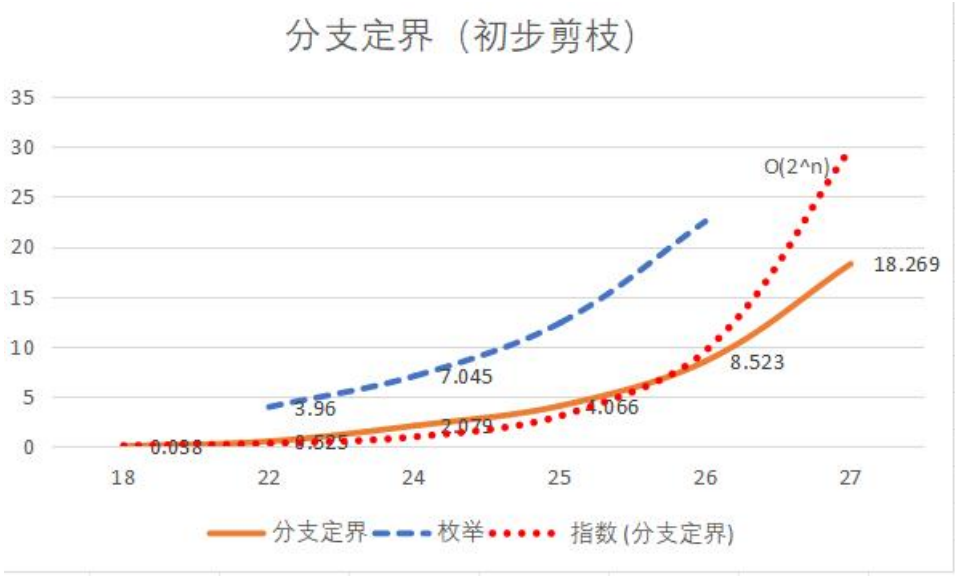
4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

贪心的方法只能保证大部分的结果是正确的，但是也有的图无法得到正确结果

以下是性能比较，可以看出贪心的复杂度接近 O(n)，而分支定界和回溯低于 O(2^n)







5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

（1）拓扑+贪心

```
#include<iostream>
```

```
#include<vector>
#include<queue>
#include<ctime>
#include<fstream>
using namespace std;
int pmax,pmin,d;//pmax到pmin的距离不超过d，cost是距离的一个函数，假设cost=距离，pmin=0，d=pmax
int p[20005];
int num=0;
int din[20005]={0};//入度
struct node{
    int v,p;
};
vector<pair<int,int>>g[20005];

void bianli(int u){
    for(auto vv:g[u]){
        if(p[u]-vv.second>=pmin){
            p[vv.first]=max(p[vv.first],p[u]-vv.second);//从一个点出发判断指向的另一点的p是否>=pmin，更新p值
        }
        else{
            p[u]=pmax;//如果p<pmin，说明u要放一个放大器，重新遍历u的邻边，更新它们的p值
            num++;
            bianli(u);
            return;
        }
    }
}

int main(){
    clock_t start,end;
    start=clock();
    freopen("filein.in","r",stdin);
    cin>>pmax;//pmin=0,d=pmax-pmin
    int n,m;
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v,c;
        cin>>u>>v>>c;
        g[u].push_back({v,c});
        din[v]++;
    }
    for(int i=0;i<=n;i++) p[i]=-1;
    queue<node>q;
```

```
    for(int i=1;i<=n;i++){
        if(din[i]==0){
            q.push((node){i,pmax});//把入度为零的点先放入队列
            p[i]=pmax;
        }
    }
    while(!q.empty()){
        node tmp=q.front();
        q.pop();
        int u=tmp.v;
        bianli(u);          //更新p
        for(auto vv:g[u]){
            din[vv.first]--;//更新入度
            if(din[vv.first]==0) q.push((node){vv.first,p[vv.first]});//入度为
零的点放入队列
        }
    }
    cout<<num<<'\n';
    end=clock();
    printf( "%f s\n", (double)(end - start) / CLOCKS_PER_SEC );
}
```

（2）dfs分支定界

```
#include<iostream>
#include<algorithm>
#include<queue>
#include<cstring>
#include<ctime>
using namespace std;
const int maxn=10005;
struct edge{
    int u,v,next,cost;
};

int n,ans,pmax;
int pmin=0;
int din[maxn],vis[maxn],head[maxn],status[maxn];
int cnt,p[maxn],topo[maxn];
edge edg[maxn];
bool flag;

int tt=0;
void addedg(int u,int v,int w){
    edg[tt].u=u,edg[tt].v=v,edg[tt].cost=w;
    din[v]++;
    edg[tt].next=head[u];
```

```
    head[u]=tt;
    tt++;
}

void topusort(){
    priority_queue<int>q;
    cnt=0;
    for(int i=1;i<=n;i++){
        if(din[i]==0) q.push(-i);
    }
    while(!q.empty()){
        int u=-1*q.top();
        q.pop();
        topo[++cnt]=u;
        for(int i=head[u];i!=-1;i=edg[i].next){
            int v=edg[i].v;
            if(--din[v]==0) q.push(-v);
        }
    }
}

void dfs(int x){
    if(!flag) return ;
    vis[x]=1;
    for(int i=head[x];i!=-1;i=edg[i].next){
        int v=edg[i].v,w=edg[i].cost;
        if(p[x]>=w){
            if(vis[v]) continue;
            vis[v]=1;
            dfs(v);
        }
        else{
            flag=0;
            return ;
        }
    }
}

void judge(){  //判断这种情况的每个点的 p 是否>pmin,如果都大于, dfs 修改 p 值
    status[1]=1;
    for(int i=1;i<=n;i++){
        vis[i]=0;
        if(status[i]) p[i]=pmax;
        else p[i]=-1;
    }
    for(int i=1;i<=cnt;i++){
```

```
        int u=topo[i];
        for(int j=head[u];j!=-1;j=edg[j].next){
            int v=edg[j].v,w=edg[j].cost;
            p[v]=max(p[v],p[u]-w);
        }
    }
    flag=1;
    for(int i=1;i<=n;i++){
        if(p[i]<pmin){
            flag=0;
            return ;
        }
    }
    dfs(1);
}



void meiju(int x,int tot){  //枚举所有情况，进行递归
    if(tot>ans) return;
    if(x>n){
        judge();
        if(flag) ans=min(ans,tot);
        return;
    }
    status[x]=1;
    meiju(x+1,tot+1);
    status[x]=0;
    meiju(x+1,tot);
}

int main(){
    clock_t start,end;
    freopen("filein.in","r",stdin);
    start=clock();
    ans=maxn;
    memset(p,0,sizeof(p));
    memset(din,0,sizeof(din));
    memset(vis,0,sizeof(vis));
    memset(status,0,sizeof(status));
    for(int i=1;i<=maxn;i++) head[i]=-1;
    int m;
    cin>>pmax>>n>>m;
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
```

```
        addedg(u,v,w);
    }
    topusort();
    meiju(2,0);
    cout<<ans;
    end=clock();
    printf( "\n%f s\n", (double)(end - start) / CLOCKS_PER_SEC );
}
```

（3）回溯

```
#include<iostream>
#include<algorithm>
#include<queue>
#include<cstring>
#include<ctime>
using namespace std;
const int maxn=10005;
struct edge{
    int u,v,next,cost;
};
struct node{
    int p,num,tot,tag,father;
    bool operator < (const node& y) const{
        return tot>y.tot;
    }
};

int n,ans,pmax,pmin;
int din[maxn],vis[maxn],head[maxn],status[maxn],mp[maxn][maxn]={0};
int cnt,p[maxn],topo[maxn];
edge edg[maxn];

int tt=0;
void addedg(int u,int v,int w){
    edg[tt].u=u,edg[tt].v=v,edg[tt].cost=w;
    din[v]++;
    edg[tt].next=head[u];
    head[u]=tt;
    tt++;
    if(!mp[v][u]) mp[v][u]=w;
    mp[v][u]=min(mp[v][u],w);
}

void topusort(){
    priority_queue<int>q;
    cnt=0;
```

```
    for(int i=1;i<=n;i++){
        if(din[i]==0) q.push(-i);
    }
    while(!q.empty()){
        int u=-1*q.top();
        q.pop();
        topo[++cnt]=u;
        for(int i=head[u];i!=-1;i=edg[i].next){
            int v=edg[i].v;
            if(--din[v]==0) q.push(-v);
        }
    }
}

void fun(){
    priority_queue<node>q;
    node tree[maxn];
    int tag=0;
    node tmp;
    tmp.p=pmax;tmp.tot=0;tmp.num=1;tmp.tag=++tag;tmp.father=0;
    q.push(tmp);
    tree[tmp.tag]=tmp;
    while(!q.empty()){
        tmp=q.top();
        q.pop();
        if(tmp.num==n){
            ans=tmp.tot;
            break;
        }
        int v=topo[tmp.num+1];
        node grn=tree[tmp.tag];
        tmp.num++;tmp.p=-1;
        while(grn.tag){
            if(mp[v][topo[grn.num]]){
                tmp.p=max(tmp.p,tree[grn.tag].p-mp[v][topo[grn.num]]);
            }
            grn=tree[grn.father];
        }
        tmp.father=tmp.tag;
        tmp.tag=++tag;
        bool cc=0;
        for(int i=head[v];i!=-1;i=edg[i].next){
            int w=edg[i].cost;
            if(tmp.p-w<0){
                cc=1;
                break;
```

```
            }
        }
        if(tmp.p>=0&&cc==0){//不用放大器
            q.push(tmp);
            tree[tmp.tag]=tmp;
            tmp.tag=++tag;
        }
        tmp.tot++;tmp.p=pmax;tree[tmp.tag]=tmp;
        q.push(tmp);
    }
}

int main(){
    clock_t start,end;
    freopen("filein.in","r",stdin);
    start=clock();
    tt=0;
    ans=maxn;
    memset(p,0,sizeof(p));
    memset(din,0,sizeof(din));
    memset(vis,0,sizeof(vis));
    memset(status,0,sizeof(status));
    for(int i=1;i<=maxn;i++) head[i]=-1;
    int m;
    cin>>pmax>>n>>m;
    start=clock();
    for(int i=1;i<=m;i++){
        int u,v,w;
        cin>>u>>v>>w;
        addedg(u,v,w);
    }
    topusort();
    fun();
    cout<<ans;
    end=clock();
    printf( "\n%f s\n", (double)(end - start) / CLOCKS_PER_SEC );
}
```