如果你看完之后还不知道怎么建，不用担心，你不是一个人……

其实这个建树过程写出来并不难，看一下代码就知道了。大致思路是：一开始所有点都在同一个集合内。然后每次选一个集合，对其中两个点做最小割，按照源汇集分割这个集合，并添加一条边。最后所有集合都剩下一个点，树也就建完了。共需要做O(n)次最大流。

现在有了这棵树，应该怎么办呢？
很简单：选择树中权值最小的边，那么最优解一定是先在一侧走完，再经过这条边，再走另外一侧（只经过该权值最小的边一次，最优方案必然如此）。然后分成两部分递归下去即可。

所以这篇题解的核心其实是贴一下Gomory-Hu树的代码，相信我，代码真的非常简单……

代码

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<vector>
#include<queue>
using namespace std;
const int SIZEN=210,INF=0x7fffffff/2;
class EDGE{
public:
    int from,to,cap,flow;
};
vector<EDGE> edges;
vector<int> c[SIZEN];
int S,T;
bool visit[SIZEN]={0};
int depth[SIZEN]={0};
int cur[SIZEN]={0};
void clear_graph(void){
    edges.clear();
    for(int i=0;i<SIZEN;i++) c[i].clear();
}
void clear_flow(void){
    for(int i=0;i<edges.size();i++) edges[i].flow=0;
}
void addedge_2(int from,int to,int cap){//加双向边
    EDGE temp;
    temp.from=from,temp.to=to,temp.cap=cap,temp.flow=0;
    edges.push_back(temp);
    temp.from=to,temp.to=from,temp.cap=cap,temp.flow=0;
    edges.push_back(temp);
    int tot=edges.size()-2;
```

```
    c[from].push_back(tot);
    c[to].push_back(tot+1);
}
bool BFS(void){
    memset(visit,0,sizeof(visit));
    memset(depth,-1,sizeof(depth));
    queue<int> Q;
    Q.push(S);visit[S]=true;depth[S]=0;
    while(!Q.empty()){
        int x=Q.front();Q.pop();
        for(int i=0;i<c[x].size();i++){
            EDGE &now=edges[c[x][i]];
            if(!visit[now.to]&&now.cap>now.flow){
                visit[now.to]=true;
                depth[now.to]=depth[x]+1;
                Q.push(now.to);
            }
        }
    }
    return visit[T];
}
int DFS(int x,int a){
    if(x==T||!a) return a;
    int flow=0,cf=0;
    for(int i=cur[x];i<c[x].size();i++){
        cur[x]=i;
        EDGE &now=edges[c[x][i]];
        if(depth[x]+1==depth[now.to]){
            cf=DFS(now.to,min(a,now.cap-now.flow));
            if(cf){
                flow+=cf;
                a-=cf;
                now.flow+=cf,edges[c[x][i]^1].flow-=cf;
            }
            if(!a) break;
        }
    }
    if(!flow) depth[x]=-1;
    return flow;
}
int Dinic(void){
    int flow=0;
    while(BFS()){
        memset(cur,0,sizeof(cur));
        flow+=DFS(S,INF);
    }
    return flow;
}
int N,M;
int fa[SIZEN],falen[SIZEN];
int now;
void find_min(int x,int fa){
    for(int i=0;i<c[x].size();i++){
        EDGE &e=edges[c[x][i]];
```

```
        if(e.to!=fa&&e.cap!=-1){
            if(now==-1||e.cap<edges[now].cap) now=c[x][i];
            find_min(e.to,x);
        }
    }
}
void Solve(int x){
    now=-1;
    find_min(x,0);
    if(now==-1){
        printf("%d ",x);
        return;
    }
    edges[now].cap=edges[now^1].cap=-1;
    int p=now;
    Solve(edges[p].from);
    Solve(edges[p].to);
}
int ans=0;
void build_tree(void){//建树
    for(int i=1;i<=N;i++) fa[i]=1;
    for(int i=2;i<=N;i++){
        clear_flow();
        S=i,T=fa[i];
        falen[i]=Dinic();
        BFS();
        for(int j=i+1;j<=N;j++)
            if(visit[j]&&fa[j]==fa[i]) fa[j]=i;
    }
    clear_graph();
    for(int i=2;i<=N;i++)
        addedge_2(i,fa[i],falen[i]),ans+=falen[i];
}
void answer(void){
    printf("%d\n",ans);
    Solve(1);
    printf("\n");
}
void init(void){
    scanf("%d%d",&N,&M);
    int a,b,w;
    for(int i=1;i<=M;i++){
        scanf("%d%d%d",&a,&b,&w);
        addedge_2(a,b,w);
    }
}
int main(){
    //freopen("pumpingstations.in","r",stdin);
    //freopen("pumpingstations.out","w",stdout);
    init();
    build_tree();
    answer();
    return 0;
}
```

建树就是build_tree函数。