大连理工大学图论模版

目录

1. 最小生成树（Kruscal 算法）：

思想：对边排序，维护并查集，每次选端点不在同一并查集的最小的边，再并合并这两个端点所在的并查集。

```
struct Side{
    int u,v,w;
}side[maxn];
int fa[maxn];
int get_fa(int x){
    if(x==fa[x])return x;
    else return fa[x]=get_fa(fa[x]);
}
bool cmp(const Side &a,const Side &b){
    return a.w<b.w;
}
int kruscal(int n,int m){
    int ans=0;
    sort(side,side+m,cmp);
    for(int i=0;i<n;i++)fa[i]=i;
    for(int i=0;i<m;i++){
        int a=get_fa(side[i].u);
        int b=get_fa(side[i].v);
        if(a!=b){
            ans+=side[i].w;
            fa[a]=b;
        }
    }
    int k=get_fa(0);
    for(int i=1;i<n;i++)if(get_fa(i)!=k)return -1;
    return ans;
}
```

2. 最小生成树（Prim 算法）：

思想：贪心地从小到大扩展最小生成树的大小，维护当前最小生成树到各点的最近距离，每次选取一个离当前生成树最近的点加进生成树中。

```
int grid[maxn][maxn];
int dis[maxn];
bool flag[maxn];
int prim(int n){
    int ans=0;
    flag[0]=true;
    for(int i=1;i<n;i++){dis[i]=grid[0][i];flag[i]
=false;}
    for(int i=1;i<n;i++){
        int d=INF,p=-1;
        for(int j=1;j<n;j++){
            if(!flag[j]&&dis[j]<d){
                d=dis[j];
                p=j;
            }
        }
        if(p==-1)return -1;
        ans+=dis[p];
        flag[p]=true;
        for(int j=1;j<n;j++)dis[j]=min(grid[p][j],
dis[j]);
    }
    return ans;
}
```

3. 单源最短路（Dijkstra 算法）：

思想：贪心，维护 dis 数组为起点每个点的最短距离，每次可确定 dis 中最小的点的距离即为到此点的最短距离，然后用这个点来更新 dis。

注意：必须保证边权非负

```
struct Side{
    int to,next,w;
}side[maxm];
int node[maxn],dis[maxn],top;
void add_side(int u,int v,int w){
    side[top]=(Side){v,node[u],w};node[u]=top++;
    side[top]=(Side){u,node[v],w};node[v]=top++;
}
class Cmp{
public:
    bool operator () (const pair<int,int> &a,const
 pair<int,int> &b){
        return a.first>b.first;
    }
};
priority_queue<pair<int,int>,vector<pair<int,int>
 >,Cmp>q;
int start,end,n,m;
int dijkstra(){
    for(int i=0;i<n;i++)dis[i]=INF;
    q.push(make_pair(0,start));
    while(!q.empty()){
        pair<int,int> tmp=q.top();q.pop();
        int u=tmp.second;
        if(dis[u]!=INF)continue;
        dis[u]=tmp.first;
        for(int i=node[u];i!=-1;i=side[i].next){
            q.push(make_pair(dis[u]+side[i].w,sid
e[i].to));
```

```
        }
    }
    return dis[end]==INF?-1:dis[end];
}
```

4. 单源最短路算法（SPFA）：

思想：Bellman-Ford 的优化，可以记录点出队次数来判断有无负环

优化：

(1) SLF：Small Label First

思想：设队首元素为 i，队列中要加入节点 j，在 $d_j \le d_i$ 时加到队首而不是队尾，否则和普通的 SPFA 一样加到队尾

(2) LLL：Large Label Last

思想：设队列 Q 中的队首元素为 i，距离标号的平均值为 $\overline{d} = \frac{\sum_{j \in Q} d_j}{|Q|}$，每次出队时，若 $d_i > \overline{d}$，把 i 移到队列末尾，如此反复，直到找到一个 i 使 $d_i \le \overline{d}$，将其出队。

```
struct Side{
    int to,next,w;
}side[maxm];
int node[maxn],dis[maxn],top;
void add_side(int u,int v,int w){
    side[top]=(Side){v,node[u],w};node[u]=top++;
    side[top]=(Side){u,node[v],w};node[v]=top++;
}
int start,end,n,m;
bool inqueue[maxn];
queue<int>q;
int spfa(){
    for(int i=0;i<n;i++){
        inqueue[i]=false;
        dis[i]=INF;
    }
    dis[start]=0;
    q.push(start);
    while(!q.empty()){
        int u=q.front();q.pop();
        inqueue[u]=false;
        for(int i=node[u];i!=-1;i=side[i].next){
            int v=side[i].to;
            if(dis[u]+side[i].w<dis[v]){
                dis[v]=dis[u]+side[i].w;
                if(!inqueue[v]){
                    inqueue[v]=true;
                    q.push(v);
                }
            }
        }
    }
    return dis[end]==INF?-1:dis[end];
}
```

5. 多源最短路算法（Floyd 算法）：

设 dis[i][i]=INF，自己到自己的最短路径；

可设 pre[i][j]数组来记录中转节点，从而记录路径

```
void floyd(){
    for(int k=0;k<n;k++){
        for(int i=0;i<n;i++){
            for(int j=0;j<n;j++){
                dis[i][j]=min(dis[i][k]+dis[k]
[j],dis[i][j]);
            }
        }
    }
}
```

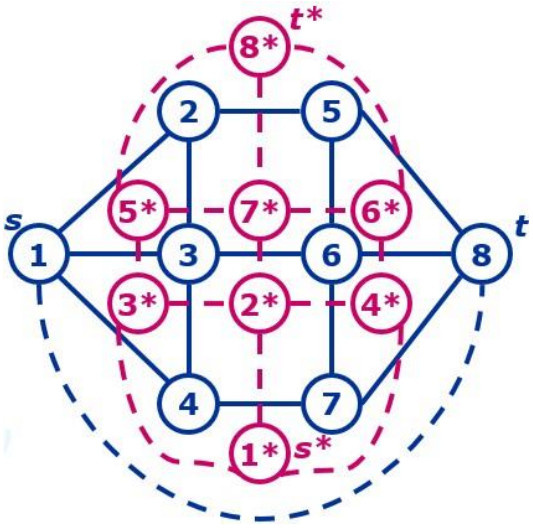
6. 最大流（ISAP 算法）：

```
const int maxn = 210;
const int maxm = 500;
const int INF = 0x7FFFFFF;
#define mem(name,value) memset((name),(value),size
of(name))
struct Side{
    int to,next,c;
}side[maxm];
int top,node[maxn];
void add_side(int u,int v,int c,int rc){
    side[top]=(Side){v,node[u],c};node[u]=top++;
    side[top]=(Side){u,node[v],rc};node[v]=top++;
}
int start,end,cnt,dis[maxn],gap[maxn];
int get_flow(int u,int flow){
    //printf("%d %d\n",u,flow);
    if(u==end)return flow;
    int ans=0;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to,c=side[i].c;
        if(dis[u]>dis[v]&&c){
            int f=get_flow(v,min(flow-ans,c));
            ans+=f;
```

```
            side[i].c-=f;
            side[i^1].c+=f;
            if(ans==flow)return ans;
        }
    }
    if(!(--gap[dis[u]]))dis[start]=cnt+2;
    gap[++dis[u]]++;
    return ans;
}
int main(){
    int n,m;
    while(~scanf("%d%d",&m,&n)){
        top=0;
        mem(node,-1);
        mem(gap,0);
        mem(dis,0);
        while(m--){
            int u,v,w;
            scanf("%d%d%d",&u,&v,&w);
            add_side(u,v,w,0);
        }
        int ans=0;
        start=1;
        end=n;
        cnt=n;
        gap[0]=cnt;
        while(dis[start]<cnt)ans+=get_flow(start,I
NF);
        printf("%d\n",ans);
    }
}
```

7. 平面图最大流算法（对偶图）

转化示意图图如下：



需要：S，T 在图的边界上

建立如上图的对偶图，然后求 S*到 T*的最短路即原图的一个最小割，即最大流。

8. 最小费用最大流（SPFA 实现）：

```
struct Side{
    int to,next,c,w;
}side[maxm];
int node[maxn],top,dis[maxn],pre[maxn];
void add_side(int u,int v,int c,int w){
    side[top]=(Side){v,node[u],c,w};
    node[u]=top++;
    side[top]=(Side){u,node[v],0,-w};
    node[v]=top++;
}
bool inque[maxn];
int start,end,cnt;
queue<int>q;
bool spfa(){
    for(int i=0;i<cnt;i++){
        dis[i]=INF;
        inque[i]=false;
        pre[i]=-1;
    }
    dis[start]=0;
    q.push(start);
    while(!q.empty()){
        int u=q.front();q.pop();
        inque[u]=false;
        for(int i=node[u];i!=-1;i=side[i].next){
            int v=side[i].to;
            if(side[i].c&&dis[u]+side[i].w<dis
[v]){
                dis[v]=dis[u]+side[i].w;
                pre[v]=i;
                if(!inque[v]){
                    inque[v]=true;
                    q.push(v);
                }
            }
        }
    }
    return dis[end]!=INF;
}
int flow;
int maxFlowMinCost(){
    int ans=0;
```

```
    flow=0;
    while(spfa()){
        int u=end;
        int aug=INF;
        while(u!=start){
            aug=min(aug,side[pre[u]].c);
            u=side[pre[u]^1].to;
        }
        u=end;
        flow+=aug;
        while(u!=start){
            side[pre[u]].c-=aug;
            side[pre[u]^1].c+=aug;
            ans+=aug*side[pre[u]].w;
            u=side[pre[u]^1].to;
        }
    }
    return ans;
}
```

9. 最小费用最大流（ZKW 费用流）：

类似 sap 利用距离来分层

```
struct Side{
    int to,next,c,w;
}side[maxm];
int node[maxn],top;
void add_side(int u,int v,int c,int w){
    side[top]=(Side){v,node[u],c,w};node[u]=top++;
    side[top]=(Side){u,node[v],0,-w};node[v]=top+
+;
}
int dis[maxn],cur[maxn],vis[maxn];
int start,end,cnt;
int ans,sum_flow,need_flow;
int augment(int u,int flow){
    if(u==end){
        ans+=flow*dis[start];
        sum_flow+=flow;
        return flow;
    }
    vis[u]=true;
    for(int &i=cur[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(vis[v] ||side[i].c==0|| dis[v]+side[i].
w!=dis[u])continue;
        int aug=augment(v,min(flow,side[i].c));
        if(aug){
            side[i].c-=aug;
            side[i^1].c+=aug;
            return aug;
        }
    }
    return 0;
}
bool adjust(){
    int delta=INF;
    for(int u=0;u<cnt;u++)if(vis[u]){
        for(int i=node[u];i!=-1;i=side[i].next){
            if(side[i].c==0)continue;
            int v=side[i].to;
            if(!vis[v])delta=min(side[i].w+dis[v]
-dis[u],delta);
        }
    }
    if(delta==INF)return false;
    for(int i=0;i<cnt;i++)if(vis[i]){dis[i]+=delt
a;cur[i]=node[i];}
    return true;
}
void minCostmaxFlow(){
    memset(dis,0,sizeof(dis));
    for(int i=0;i<cnt;i++)cur[i]=node[i];
    do{
        do
        memset(vis,0,sizeof(vis));
        while(augment(start,INF));
    }while(adjust());
}
```

10. 强联通分量（Tarjan 算法）：

用 low 来存储点属于哪个强联通分量

```
int dfn[maxn],low[maxn];
int sum;
int t;
stack<int>q;
void dfs(int u){
    dfn[u]=low[u]=t++;
    q.push(u);
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(!dfn[v])dfs(v);
        if(dfn[v]!=-1)low[u]=min(low[u],low[v]);
    }
    int v;
```

```
    if(low[u]==dfn[u]){
        do{
            v=q.top();
            q.pop();
            dfn[v]=-1;
            low[v]=sum;
        }while(v!=u);
        sum++;
    }
}
```

11. 双连通分量：

```
int dfn[maxn],low[maxn],cnt,sum;
vector<int>ddc[maxn];
stack<int>q;
void dfs(int u){
    dfn[u]=low[u]=cnt++;
    q.push(u);
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(!dfn[v]){
            dfs(v);
            low[u]=min(low[u],low[v]);
            if(dfn[u]<=low[v]){
                sum++;
                int t;
                do{
                    t=q.top();
                    q.pop();
                    ddc[sum].push_back(t);
                }while(t!=u);
                q.push(u);
            }
        }else low[u]=min(low[u],dfn[v]);
    }
}
```

12. 割边（桥）：

low[v]>dfn[u]

```
int vis[maxn],low[maxn],dfn[maxn],cnt;
map<int,int>num[maxn];//记录重边的个数
void dfs(int u,int fa){
    vis[u]=1;
    dfn[u]=low[u]=cnt++;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(v!=fa&&vis[v]==1)low[u]=min(low[u],dfn
[v]);
        if(vis[v]==0){
            dfs(v,u);
            low[u]=min(low[u],low[v]);
            if(low[v]>dfn[u]&&num[u][v]==1){
                //则这条边是割边
            }
        }
    }
    vis[u]=2;
}
```

13. 割点（关节点）：

low[v]>=dfn[u]

```
void dfs(int u){
    vis[u]=1;
    dfn[u]=low[u]=cnt++;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(!vis[v]){
            dfs(v);
            low[u]=min(low[u],low[v]);
            if(dfn[u]<=low[v])vis[u]++;
        }else low[u]=min(low[u],dfn(v));
    }
    if(u==root&&vis[u]>2||u!=root&&v[u]>1){
        //则该点是割点
    }
}
```

14. 无源汇上下界可行流：

设 du[i]表示 i 节点入流之和与出流之和的差

加附加源点 S 汇点 T，如 du[i]>0，S->i 连容量为 du[i]的边；反之连，i->T 容量为-du[i]的边。求 S-T 最大流，判断是否满流

```
const int maxn = 210;
const int maxm = 50000;
const int INF = 0x3ffffff;
struct Side{
    int to,next,c;
}side[maxm*2];
int node[maxn],dis[maxn],gap[maxn],start,end,cnt,t
op;
void add_side(int u,int v,int c){
    side[top]=(Side){v,node[u],c};node[u]=top++;
    side[top]=(Side){u,node[v],0};node[v]=top++;
}
int get_flow(int u,int flow){
    if(u==end)return flow;
```

```
    int ans=0;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to,c=side[i].c;
        if(dis[u]>dis[v]&&c){
            int f=get_flow(v,min(flow-ans,c));
            ans+=f;
            side[i].c-=f;
            side[i^1].c+=f;
            if(ans==flow)return flow;
        }
    }
    if(!(--gap[dis[u]]))dis[start]=cnt+2;
    gap[++dis[u]]++;
    return ans;
}
int up[maxm],low[maxm],du[maxn],id[maxm];
int main(){
    int n,m;
    memset(node,-1,sizeof(node));
    scanf("%d%d",&n,&m);
    cnt=n+2;start=0;end=n+1;
    for(int i=0;i<m;i++){
        int u,v;
        scanf("%d%d%d%d",&u,&v,&low[i],&up[i]);
        du[v]+=low[i];
        du[u]-=low[i];
        add_side(u,v,up[i]-low[i]);
        id[i]=top-1;
    }
    int target=0;
    for(int i=1;i<=n;i++){
        if(du[i]>0){add_side(start,i,du[i]);target
+=du[i];}
        else if(du[i]<0)add_side(i,end,-du[i]);
    }
    int ans=0;
    gap[0]=cnt;
    while(dis[start]<cnt)ans+=get_flow(start,INF);
    if(ans==target){
        printf("YES\n");
        for(int i=0;i<m;i++)printf("%d\n",low[i]+s
ide[id[i]].c);
    }else printf("NO\n");
}
```

15. 有源汇上下界最大流：

设源点汇点为 S 和 T，添加边 T->S，容量[0,INF]，添加超级源汇 SS，TT 然后求可行流，判断是否有解。之后，删掉 SS，TT。求 S-T 最大流即是答案

```
//ZOJ - 3229
const int maxn = 1500;
const int maxm = 400000;
const int INF = 0x3fffffff;
struct Side{
    int to,next,c;
}side[maxm*2];
int node[maxn],top,cnt,start,end;
void add_side(int u,int v,int c){
    side[top]=(Side){v,node[u],c};node[u]=top++;
    side[top]=(Side){u,node[v],0};node[v]=top++;
}
int dis[maxn],gap[maxn];
int get_flow(int u,int flow){
    if(u==end)return flow;
    int ans=0;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to;
        if(dis[u]>dis[v]&&side[i].c){
            int f=get_flow(v,min(side[i].c,flow-an
s));
            side[i].c-=f;
            side[i^1].c+=f;
            ans+=f;
            if(ans==flow)return flow;
        }
    }
    if(!(--gap[dis[u]]))dis[start]=cnt+2;
    gap[++dis[u]]++;
    return ans;
}
int low[400][1000],up[400][1000],id[400][1000],c[4
00];
int d[400],g[1000];
int du[maxn];
int s,e;
int main(){
    int n,m;
    while(~scanf("%d%d",&n,&m)){
        top=0;
        memset(node,-1,sizeof(node));
        memset(du,0,sizeof(du));
        memset(dis,0,sizeof(dis));
        memset(gap,0,sizeof(gap));
```

```
        cnt=n+m+4;
        s=n+m;
        e=n+m+1;
        start=n+m+2;
        end=n+m+3;
        for(int i=0;i<m;i++){
            scanf("%d",&g[i]);
            add_side(i,e,INF);
            du[i]-=g[i];
            du[e]+=g[i];
        }
        for(int i=0;i<n;i++){
            int u=m+i;
            scanf("%d%d",&c[i],&d[i]);
            add_side(s,u,d[i]);
            for(int j=0;j<c[i];j++){
                int v;
                scanf("%d%d%d",&v,&low[i][j],&up
[i][j]);
                add_side(u,v,up[i][j]-low[i][j]);
                du[v]+=low[i][j];
                du[u]-=low[i][j];
                id[i][j]=top-1;
            }
        }
        int target=0,ans=0;
        add_side(e,s,INF);
        for(int i=0;i<=e;i++){
            if(du[i]>0)target+=du[i];
            if(du[i]>0)add_side(start,i,du[i]);
            else if(du[i]<0)add_side(i,end,-du
[i]);
        }
        gap[0]=cnt;
        while(dis[start]<cnt)ans+=get_flow(start,I
NF);
        if(ans==target){
            memset(gap,0,sizeof(gap));
            memset(dis,0,sizeof(dis));
            dis[start]=dis[end]=cnt;
            cnt-=2;
            node[start]=node[end]=-1;
            start=s;
            end=e;
            gap[0]=cnt;
            ans=0;
            while(dis[start]<cnt)ans+=get_flow(st
art,INF);
            printf("%d\n",ans);
            for(int i=0;i<n;i++){
                for(int j=0;j<c[i];j++){
                    printf("%d\n",side[id[i][j]].
c+low[i][j]);
                }
            }
        }else printf("-1\n");
        printf("\n");
    }
}
```

16. 有源汇上下界最小流：

设 du[i]表示 i 节点入流之和与出流之和的差

然后添加附加源汇 SS,TT，如果 du[i]>0，添加 SS->i 容量 du[i]，如果 du[i]<0，添加 i->TT 容量-du[i]。

求 SS->TT 最大流，记录流量；

添加边 T->S，容量为无穷大

求 SS->TT 最大流，记录流量；

累计这两次的流量，判断是否满流，即有无可行解

最后：T->S 的反向弧上的流量即为 S->T 的最小流

```
//sgu 176 flow construction
const int maxn = 110;
const int maxm = 11000;
const int INF = 0x3ffffff;
struct Side{
    int to,next,c;
}side[maxm*2];
int node[maxn],dis[maxn],gap[maxn],start,end,cnt,t
op;
void add_side(int u,int v,int c){
    side[top]=(Side){v,node[u],c};node[u]=top++;
    side[top]=(Side){u,node[v],0};node[v]=top++;
}
int get_flow(int u,int flow){
    if(u==end)return flow;
    int ans=0;
    for(int i=node[u];i!=-1;i=side[i].next){
        int v=side[i].to,c=side[i].c;
        if(dis[u]>dis[v]&&c){
            int f=get_flow(v,min(flow-ans,c));
            ans+=f;
            side[i].c-=f;
            side[i^1].c+=f;
            if(ans==flow)return flow;
```

```
        }
    }
    if(!(--gap[dis[u]]))dis[start]=cnt+2;
    gap[++dis[u]]++;
    return ans;
}
int up[maxm],low[maxm],du[maxn],id[maxm];
int main(){
    int n,m;
    memset(node,-1,sizeof(node));
    scanf("%d%d",&n,&m);
    for(int i=0;i<m;i++){
        int u,v,c,t;
        scanf("%d%d%d%d",&u,&v,&c,&t);
        if(t)up[i]=low[i]=c;
        else{up[i]=c;low[i]=0;}
        add_side(u,v,up[i]-low[i]);
        id[i]=top-1;
        du[u]-=low[i];
        du[v]+=low[i];
    }
    int s=0,t=n+1;
    cnt=n+2;
    int target=0;
    for(int i=1;i<=n;i++){
        if(du[i]>0){add_side(s,i,du[i]);target+=du
[i];}
        else if(du[i]<0)add_side(i,t,-du[i]);
    }
    start=s,end=t;
    int ans=0;
    gap[0]=cnt;
    while(dis[start]<cnt)ans+=get_flow(start,INF);
    add_side(n,1,INF);
    int mark=top-1;
    memset(dis,0,sizeof(dis));
    memset(gap,0,sizeof(gap));
    gap[0]=cnt;
    while(dis[start]<cnt)ans+=get_flow(start,INF);
    if(ans==target){
        printf("%d\n",side[mark].c);
        for(int i=0;i<m;i++){
            printf("%d%c",side[id[i]].c+low[i],i=
=m-1?'\n':' ');
        }
    }else{
        printf("Impossible\n");
    }
}
```

17. 全局最小割 stoerwagner

```
const int INF = 1000000;
const int N  = 110;
int side[N][N];
int grid[N][N];
int w[N];
bool vis[N];
bool deleted[N];
bool choose[N];
int K;

queue<int>qa,qb;
vector<int>nodes[N];

int prim(int k,int n,int &s,int &t){
    s=0;
    while(deleted[s])s++;
    for(int i=0;i<n;i++){
        w[i]=grid[s][i];
        vis[i]=false;
    }
    vis[s]=true;
    int p;
    int Max;
    for(int i=1;i<k;i++){
        Max=-INF;
        for(int j=0;j<n;j++)if(!vis[j]&&!deleted
[j]){
            if(w[j]>Max){
                Max=w[j];p=j;
            }
        }
        if(i==k-2)s=p;
        if(i==k-1)t=p;
        vis[p]=true;
        for(int i=0;i<n;i++)if(!vis[i]&&!deleted
[i]){
            w[i]+=grid[i][p];
        }
    }
    return w[t];
}
int stoerwagner(int n){
```

```
    if(n<=1)return 0;
    int min_cut=INF,s,t;
    for(int i=0;i<n;i++){
        deleted[i]=false;
        nodes[i].clear();
        nodes[i].push_back(i);
    }
    for(int i=1;i<n;i++){
        int cut=prim(n-i+1,n,s,t);
        if(cut<min_cut){
            min_cut=cut;
            for(int j=0;j<n;j++)choose[j]=0;
            for(int j=0;j<nodes[t].size();j++)choo
se[nodes[t][j]]=1;
        }
        for(int i=0;i<nodes[t].size();i++){
            nodes[s].push_back(nodes[t][i]);
        }
        deleted[t]=true;
        for(int i=0;i<n;i++){
            if(i==s)continue;
            if(!deleted[i]){
                grid[s][i]+=grid[t][i];
                grid[i][s]+=grid[i][t];
            }
        }
    }
    for(int i=0;i<n;i++){
        if(choose[i])qa.push(i);
        else qb.push(i);
    }
    return min_cut;
}
int solve(vector<int> a){
    int n=a.size();
    if(n<=1)return 1;
    int t;
    for(int i=0;i<n;i++)for(int j=0;j<n;j++){
        grid[i][j]=side[a[i]][a[j]];
    }
    if(stoerwagner(n)>=K){
        while(!qa.empty())qa.pop();
        while(!qb.empty())qb.pop();
        return 1;
    }
    vector<int>x,y;
    while(!qa.empty()){
        x.push_back(a[qa.front()]);qa.pop();
    }
    while(!qb.empty()){y.push_back(a[qb.front()]);
qb.pop();};
    return solve(x)+solve(y);
}
int main(){
    int n,m;
    while(~scanf("%d%d%d",&n,&m,&K)){
        memset(side,0,sizeof(side));
        while(m--){
            int u,v;
            scanf("%d%d",&u,&v);
            u--;v--;
            side[u][v]=side[v][u]=1;
        }
        vector<int>graph;
        for(int i=0;i<n;i++){
            graph.push_back(i);
        }
        printf("%d\n",solve(graph));
    }
}
```

18. 最小树形图——朱刘算法

```
//poj 3164
const double INF = 1e15;
struct Side{
    int u,v;
    double w;
}side[10001];
double p[101][2];
double get_len(int i){
    int u=side[i].u,v=side[i].v;
    double ans=(p[u][0]-p[v][0])*(p[u][0]-p[v][0])
+(p[u][1]-p[v][1])*(p[u][1]-p[v][1]);
    return sqrt(ans);
}
double ans;
int pre[101],id[101],vis[101],idx,n,m;
double in[101];
bool mst(){
    ans=0;
    int root=1;
    while(true){
        for(int i=1;i<=n;i++)in[i]=INF;
```

```
        for(int i=0;i<m;i++){
            int v=side[i].v;
            int u=side[i].u;
            if(u!=v&&side[i].w<in[v]){
                pre[v]=u;
                in[v]=side[i].w;
            }
        }
        for(int i=1;i<=n;i++)if(i!=root&&in[i]==INF)return false;
        memset(id,0,sizeof(id));
        memset(vis,0,sizeof(vis));
        idx=0;
        in[root]=0;
        for(int i=1;i<=n;i++){
            ans+=in[i];
            int v=i;
            while(vis[v]!=i&&v!=root&&!id[v]){
                vis[v]=i;
                v=pre[v];
            }
            if(v!=root&&!id[v]){
                ++idx;
                for(int u=pre[v];u!=v;u=pre[u]){
                    id[u]=idx;
                }
                id[v]=idx;
            }
        }
        //printf("%d\n",idx);
        if(idx==0)break;
        for(int i=1;i<=n;i++)if(!id[i])id[i]=++idx;
        for(int i=0;i<m;i++){
            int v=side[i].v;
            side[i].v=id[v];
            side[i].u=id[side[i].u];
            if(side[i].v!=side[i].u){
                side[i].w-=in[v];
            }
        }
        n=idx;
        root=id[root];
    }
    return true;
}
```

```
int main(){
    while(~scanf("%d%d",&n,&m)){
        for(int i=1;i<=n;i++)scanf("%lf%lf",&p[i][0],&p[i][1]);
        for(int i=0;i<m;i++){
            scanf("%d%d",&side[i].u,&side[i].v);
            side[i].w=get_len(i);
        }
        if(mst())printf("%.2lf\n",ans);
        else printf("poor snoopy\n");
    }
}
```