判断连通图

并查集方法

```cpp
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

struct Edge{
    int u,v,w;
}edge[200005];

int fa[5005],n,m;

int find(int x){
    while(x!=fa[x]) x=fa[x]=fa[fa[x]];
    return x;
}

void merge_set(){
    int eu,ev;
    FOR(i,0,m-1){
        eu=find(edge[i].u),ev=find(edge[i].v);
        fa[ev]=eu;
    }
}

bool is_connected(){
    int cnt=0;
    FOR(i,1,n){
        if(fa[i]==i) cnt++;
    }
    if(cnt==1) return true;
    else return false;
}

int main(){
    cin>>n>>m;
    FOR(i,1,n)
        fa[i]=i;
    FOR(i,0,m-1){
        int u,v,w;
        cin>>u>>v>>w;
        edge[i]={u,v,w};
    }
    merge_set();
    if(is_connected()) cout<<"connected";
```

```
    else cout<<"disconnected";
    return 0;
}
```

最短路

P3371 【模板】单源最短路径（弱化版）

Floyd 算法 - 邻接矩阵 (70分)

参考

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
const int INF=(1<<30);
using namespace std;

const int maxn=1e4+7;
int n,m,s;//点、边、出发点
int f[maxn][maxn];

void floyd(){
    FOR(k,1,n){
        FOR(i,1,n){
            if(i==k or f[i][k]==INF) continue;
            FOR(j,1,n){
                f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
            }
        }
    }
    f[s][s]=0;
}

int main(){
    cin>>n>>m>>s;
    FOR(i,1,n)
        FOR(j,1,n)
            f[i][j]=INF;
    int u,v,w;
    FOR(i,1,m){
        cin>>u>>v>>w;
        f[u][v]=min(f[u][v],w);//去重边
    }
    floyd();
    FOR(i,1,n){
        if(f[s][i]!=INF) cout<<f[s][i]<<" ";
        else cout<<INT_MAX<<" ";
    }
```

```
    return 0;
}
```

Bellman-Ford 算法 - 直接存边 (70分)

参考

参考代码对于源点的距离赋值顺序有误

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

const int maxn=1e4+7;
const int maxm=5e5+7;
const int INF=(1<<30);

struct Edge{
    int u,v,w;
};

Edge edge[maxm];
int dist[maxn];//结点到源点最小距离
int n,m,s;//结点数,边数,源点

// 初始化图
void init(){
    FOR(i,1,n)
        dist[i]=INF;
    FOR(i,1,m){
        int u,v,w;
        cin>>u>>v>>w;
        edge[i]={u,v,w};
        if(u==s) dist[v]=w;
    }
    dist[s]=0;
}

void relax(int u,int v,int w){
    if(dist[v]>dist[u]+w) dist[v]=dist[u]+w;
}

bool Bellman_Ford(){
    FOR(i,1,n-1)
        FOR(j,1,m)
            relax(edge[j].u,edge[j].v,edge[j].w);
    bool flag=true;
    FOR(i,1,m)//判断是否有负环路
        if(dist[edge[i].v]>dist[edge[i].u]+edge[i].w){
```

```
            flag=false;
            break;
        }
    return flag;
}
int main(){
    cin>>n>>m>>s;
    init();
    if(Bellman_Ford()){
        FOR(i,1,n){
            if(dist[i]!=INF) cout<<dist[i]<<" ";
            else cout<<INT_MAX<<" ";
        }
    }
    return 0;
}
```

SPFA 算法 - 链式前向星 (100分)

参考

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

const int maxn=1e4+7;
const int maxm=5e5+7;
const int INF=(1<<30);

int n,m,s,adt=0;
int dis[maxn],vst[maxn],head[maxm];

struct Edge{
    int to,nxt,w;
}e[maxm];

void add(int u,int v,int w){
    e[++adt]={v,head[u],w};
    head[u]=adt;
}

void SPFA(){
    queue<int> q;
    FOR(i,1,n){
        dis[i]=INF;
        vst[i]=0;//记录点i是否在队列中
    }
    dis[s]=0;
    q.push(s);
```

```
    vst[s]=1;//第一个顶点入队，进行标记
    while(!q.empty()){
        int u=q.front();//取出队首
        q.pop();
        vst[u]=0;//出队标记
        for(int i=head[u];i;i=e[i].nxt){
            int v=e[i].to,w=e[i].w;
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                if(!vst[v]){
                    vst[v]=1;//标记入队
                    q.push(v);
                }
            }
        }
    }
    dis[s]=0;
}

int main(){
    cin>>n>>m>>s;
    FOR(i,1,m){
        int u,v,w;
        cin>>u>>v>>w;
        add(u,v,w);
    }
    SPFA();
    FOR(i,1,n){
        if(dis[i]!=INF) cout<<dis[i]<<" ";
        else cout<<INT_MAX<<" ";
    }
    return 0;
}
```

Dijkstra 算法 - 链式前向星 (100分)

参考

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

const int maxn=1e4+7;
const int maxm=5e5+7;
const int INF=(1<<30);

int n,m,s;
int dis[maxn],vst[maxn];
int head[maxm],adt;
```

```cpp

struct Edge{
    int to,nxt,w;
}e[maxm];

void add(int u,int v,int w){
    e[++adt]={v,head[u],w};
    head[u]=adt;
}

void dijkstra(){
    int u=s;
    while(!vst[u]){
        int min0=INF;
        vst[u]=1;
        for(int i=head[u];i;i=e[i].nxt){
            int v=e[i].to,w=e[i].w;
            if(!vst[v]) dis[v]=min(dis[v],dis[u]+w);
        }
        FOR(i,1,n)
            if(!vst[i] and dis[i]<min0) min0=dis[i],u=i;
    }
}

int main(){
    cin>>n>>m>>s;
    FOR(i,1,n)
        dis[i]=INF;
    dis[s]=0;
    FOR(i,1,m){
        int u,v,w;
        cin>>u>>v>>w;
        add(u,v,w);
    }
    dijkstra();
    FOR(i,1,n){
        if(dis[i]!=INF) cout<<dis[i]<<" ";
        else cout<<INT_MAX<<" ";
    }
    return 0;
}
```

Dijkstra 算法 - 链式前向星 & 优先队列 (100分)

参考

P4779 【模板】单源最短路径（标准版）

```cpp
#include<bits/stdc++.h>
```

```cpp
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

const int maxn=1e5+7;//弱化版是1e4
const int maxm=2e5+7;//弱化版是4e5
const int INF=(1<<30);

int n,m,s;
int dis[maxn],vst[maxn];
int head[maxm],adt;

struct Edge{
    int to,nxt,w;
}e[maxm];

void add(int u,int v,int w){
    e[++adt]={v,head[u],w};
    head[u]=adt;
}

struct node{
    int dis,u;
    bool operator <(const node &x)const{
        return x.dis<dis;
    }
};

void dijkstra(){
    priority_queue<node> q;
    q.push({0,s});
    dis[s]=0;
    while(!q.empty()){
        int u=q.top().u;
        q.pop();
        if(vst[u]) continue;
        vst[u]=1;
        for(int i=head[u];i;i=e[i].nxt){
            int v=e[i].to,w=e[i].w;
            if(dis[v]>dis[u]+w){
                dis[v]=dis[u]+w;
                if(!vst[v]) q.push({dis[v],v});
            }
        }
    }
}

int main(){
    cin>>n>>m>>s;
    FOR(i,1,n)
```

```
        dis[i]=INF;
    dis[s]=0;
    FOR(i,1,m){
        int u,v,w;
        cin>>u>>v>>w;
        add(u,v,w);
    }
    dijkstra();
    FOR(i,1,n){
        if(dis[i]!=INF) cout<<dis[i]<<" ";
        else cout<<INT_MAX<<" ";
    }
    return 0;
}
```

树的存储和遍历

```
#include<iostream>
#include<queue>
#include<stack>
using namespace std;
const int N=1e5+10;

struct Edge{//边的结构体
    int to,nxt;
}e[N*2];

int adt,head[N];

void add(int u,int v){//加边建树
    e[++adt]={v,head[u]};
    head[u]=adt;
}

int fa[N],cntp[N];

void dfs(int p1){//递归实现dfs
    cntp[p1]=0;
    for(int i=head[p1];i!=0;i=e[i].nxt){
        int p2=e[i].to;
        if(p2==fa[p1]) continue;
        fa[p2]=p1;//点p2的父节点是点p1
        dfs(p2);
        cntp[p1]++;//统计子节点个数
    }
}

void dfs2(){//栈实现dfs
```

```
    stack<int> s;
    s.push(1);
    while(!s.empty()){
        int p1=s.top();//访问栈顶
        s.pop();//出栈
        cntp[p1]=0;
        for(int i=head[p1];i!=0;i=e[i].nxt){
            int p2=e[i].to;
            if(p2==fa[p1]) continue;
            fa[p2]=p1;//点p2的父节点是点p1
            s.push(p2);//入栈
            cntp[p1]++;//统计子节点个数
        }
    }
}

void bfs(){//队列实现bfs
    queue<int> q;
    q.push(1);
    while(!q.empty()){
        int p1=q.front();//访问队首
        q.pop();//出队
        cntp[p1]=0;
        for(int i=head[p1];i!=0;i=e[i].nxt){
            int p2=e[i].to;
            if(p2==fa[p1]) continue;
            fa[p2]=p1;//点p2的父节点是点p1
            q.push(p2);//入队
            cntp[p1]++;//统计子节点个数
        }
    }
}

int main() {
    int n;
    cin>>n;
    for(int i=1;i<=n-1;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        add(u,v);
        add(v,u);
    }
    //dfs(1);
    //bfs();
    dfs2();
    for(int i=1;i<=n;i++){
        cout<<i<<": "<<cntp[i]<<endl;
    }
    return 0;
```

```
}
/*
stdin
9
1 2
1 9
2 3
2 4
4 5
4 6
4 7
5 8
*/

/*
stdout
1: 2
2: 2
3: 0
4: 3
5: 1
6: 0
7: 0
8: 0
9: 0
*/
```

最小生成树

P3366 【模板】最小生成树

参考

Prim 算法

```
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

const int INF=(1<<30);
#define maxn 5005
#define maxm 200005
struct edge{
    int v,w,next;
}e[maxm<<1];//无向图开两倍数组
int n,m;
int dis[maxn],now=1,ans;
//dis[i]表示已经加入最小生成树的的点到点i的最短距离
```

```cpp
bool vis[maxn];

int head[maxn],cnt;
void add(int u,int v,int w){
    e[++cnt].v=v;
    e[cnt].w=w;
    e[cnt].next=head[u];
    head[u]=cnt;
}

int prim(){
    FOR(i,2,n)
        dis[i]=INF;
    for(int i=head[1];i;i=e[i].next)
        dis[e[i].v]=min(dis[e[i].v],e[i].w);
    FOR(j,1,n-1){
        int minn=INF;
        vis[now]=1;
        FOR(i,1,n){
            if(!vis[i] and minn>dis[i]){
                minn=dis[i];
                now=i;
            }
        }
        if(minn==INF) return 0;
        ans+=minn;
        for(int i=head[now];i;i=e[i].next){
            int v=e[i].v;
            if(!vis[v] and dis[v]>e[i].w) dis[v]=e[i].w;
        }
    }
    return ans;
}

int main(){
    cin>>n>>m;
    FOR(i,1,m){
        int u,v,w;
        cin>>u>>v>>w;
        add(u,v,w),add(v,u,w);
    }
    int res=prim();
    if(res) cout<<res;
    else cout<<"orz";
    return 0;
}
```

Kruskal 算法

```cpp
#include<bits/stdc++.h>
#define FOR(i,a,b) for(int i=(a);i<=(b);++i)
using namespace std;

struct Edge{
    int u,v,w;
}edge[200005];

int fa[5005],n,m;

bool cmp(Edge a,Edge b){
    return a.w<b.w;
}

int find(int x){
    while(x!=fa[x])
        x=fa[x]=fa[fa[x]];
    return x;
}

int kruskal(){
    int eu,ev,cnt=0,ans=0;
    sort(edge,edge+m,cmp);
    FOR(i,0,m-1){
        eu=find(edge[i].u),ev=find(edge[i].v);
        if(eu==ev) continue;
        ans+=edge[i].w;
        fa[ev]=eu;
        if(++cnt==n-1) break;
    }
    return ans;
}

bool is_connected(){
    int cnt=0;
    FOR(i,1,n){
        if(fa[i]==i) cnt++;
    }
    if(cnt==1) return true;
    else return false;
}

int main(){
    cin>>n>>m;
    FOR(i,1,n)
        fa[i]=i;
    FOR(i,0,m-1){
        int u,v,w;
        cin>>u>>v>>w;
```

```
        edge[i]={u,v,w};
    }
    int res=kruskal();
    if(is_connected()) cout<<res;
    else cout<<"orz";
    return 0;
}
```