

判断连通图

并查集方法

```
1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  struct Edge{
6      int u,v,w;
7  }edge[200005];
8
9  int fa[5005],n,m;
10
11 int find(int x){
12     while(x!=fa[x]) x=fa[x]=fa[fa[x]];
13     return x;
14 }
15
16 void merge_set(){
17     int eu,ev;
18     FOR(i,0,m-1){
19         eu=find(edge[i].u),ev=find(edge[i].v);
20         fa[ev]=eu;
21     }
22 }
23
24 bool is_connected(){
25     int cnt=0;
26     FOR(i,1,n){
27         if(fa[i]==i) cnt++;
28     }
29     if(cnt==1) return true;
30     else return false;
31 }
32
33 int main(){
34     cin>>n>>m;
35     FOR(i,1,n)
36         fa[i]=i;
37     FOR(i,0,m-1){
38         int u,v,w;
39         cin>>u>>v>>w;
40         edge[i]={u,v,w};
41     }
42     merge_set();
43     if(is_connected()) cout<<"connected";
```

```
44     else cout<<"disconnected";
45     return 0;
46 }
```

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](#)

```
1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  const int INF=(1<<30);
4  using namespace std;
5
6  const int maxn=1e4+7;
7  int n,m,s;//点、边、出发点
8  int f[maxn][maxn];
9
10 void floyd(){
11     FOR(k,1,n){
12         FOR(i,1,n){
13             if(i==k or f[i][k]==INF) continue;
14             FOR(j,1,n){
15                 f[i][j]=min(f[i][j],f[i][k]+f[k][j]);
16             }
17         }
18     }
19     f[s][s]=0;
20 }
21
22 int main(){
23     cin>>n>>m>>s;
24     FOR(i,1,n)
25         FOR(j,1,n)
26             f[i][j]=INF;
27     int u,v,w;
28     FOR(i,1,m){
29         cin>>u>>v>>w;
30         f[u][v]=min(f[u][v],w);//去重边
31     }
32     floyd();
33     FOR(i,1,n){
34         if(f[s][i]!=INF) cout<<f[s][i]<<" ";
35         else cout<<INT_MAX<<" ";
36     }
```

```
37     return 0;
38 }
```

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

```
1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  const int maxn=1e4+7;
6  const int maxm=5e5+7;
7  const int INF=(1<<30);
8
9  struct Edge{
10     int u,v,w;
11 };
12
13 Edge edge[maxm];
14 int dist[maxn]; // 结点到源点最小距离
15 int n,m,s; // 结点数, 边数, 源点
16
17 // 初始化图
18 void init(){
19     FOR(i,1,n)
20         dist[i]=INF;
21     FOR(i,1,m){
22         int u,v,w;
23         cin>>u>>v>>w;
24         edge[i]={u,v,w};
25         if(u==s) dist[v]=w;
26     }
27     dist[s]=0;
28 }
29
30 void relax(int u,int v,int w){
31     if(dist[v]>dist[u]+w) dist[v]=dist[u]+w;
32 }
33
34 bool Bellman_Ford(){
35     FOR(i,1,n-1)
36         FOR(j,1,m)
37             relax(edge[j].u,edge[j].v,edge[j].w);
38     bool flag=true;
39     FOR(i,1,m) // 判断是否有负环路
40         if(dist[edge[i].v]>dist[edge[i].u]+edge[i].w){
```

```

41         flag=false;
42         break;
43     }
44     return flag;
45 }
46 int main(){
47     cin>>n>>m>>s;
48     init();
49     if(Bellman_Ford()){
50         FOR(i,1,n){
51             if(dist[i]!=INF) cout<<dist[i]<<" ";
52             else cout<<INT_MAX<<" ";
53         }
54     }
55     return 0;
56 }

```

SPFA 算法 - 链式前向星 (100分)

[参考](#)

```

1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  const int maxn=1e4+7;
6  const int maxm=5e5+7;
7  const int INF=(1<<30);
8
9  int n,m,s,adt=0;
10 int dis[maxn],vst[maxn],head[maxm];
11
12 struct Edge{
13     int to,nxt,w;
14 }e[maxm];
15
16 void add(int u,int v,int w){
17     e[++adt]={v,head[u],w};
18     head[u]=adt;
19 }
20
21 void SPFA(){
22     queue<int> q;
23     FOR(i,1,n){
24         dis[i]=INF;
25         vst[i]=0;//记录点i是否在队列中
26     }
27     dis[s]=0;
28     q.push(s);

```

```

29     vst[s]=1;//第一个顶点入队，进行标记
30     while(!q.empty()){
31         int u=q.front();//取出队首
32         q.pop();
33         vst[u]=0;//出队标记
34         for(int i=head[u];i;i=e[i].nxt){
35             int v=e[i].to,w=e[i].w;
36             if(dis[v]>dis[u]+w){
37                 dis[v]=dis[u]+w;
38                 if(!vst[v]){
39                     vst[v]=1;//标记入队
40                     q.push(v);
41                 }
42             }
43         }
44     }
45     dis[s]=0;
46 }
47
48 int main(){
49     cin>>n>>m>>s;
50     FOR(i,1,m){
51         int u,v,w;
52         cin>>u>>v>>w;
53         add(u,v,w);
54     }
55     SPFA();
56     FOR(i,1,n){
57         if(dis[i]!=INF) cout<<dis[i]<<" ";
58         else cout<<INT_MAX<<" ";
59     }
60     return 0;
61 }

```

Dijkstra 算法 - 链式前向星 (100分)

[参考](#)

```

1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  const int maxn=1e4+7;
6  const int maxm=5e5+7;
7  const int INF=(1<<30);
8
9  int n,m,s;
10 int dis[maxn],vst[maxn];
11 int head[maxm],adt;

```

```

12
13 struct Edge{
14     int to,nxt,w;
15 }e[maxm];
16
17 void add(int u,int v,int w){
18     e[++adt]={v,head[u],w};
19     head[u]=adt;
20 }
21
22 void dijkstra(){
23     int u=s;
24     while(!vst[u]){
25         int min0=INF;
26         vst[u]=1;
27         for(int i=head[u];i;i=e[i].nxt){
28             int v=e[i].to,w=e[i].w;
29             if(!vst[v]) dis[v]=min(dis[v],dis[u]+w);
30         }
31         FOR(i,1,n)
32             if(!vst[i] and dis[i]<min0) min0=dis[i],u=i;
33     }
34 }
35
36 int main(){
37     cin>>n>>m>>s;
38     FOR(i,1,n)
39         dis[i]=INF;
40     dis[s]=0;
41     FOR(i,1,m){
42         int u,v,w;
43         cin>>u>>v>>w;
44         add(u,v,w);
45     }
46     dijkstra();
47     FOR(i,1,n){
48         if(dis[i]!=INF) cout<<dis[i]<<" ";
49         else cout<<INT_MAX<<" ";
50     }
51     return 0;
52 }

```

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）](#)

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

```

2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  const int maxn=1e5+7;//弱化版是1e4
6  const int maxm=2e5+7;//弱化版是4e5
7  const int INF=(1<<30);
8
9  int n,m,s;
10 int dis[maxn],vst[maxn];
11 int head[maxm],adt;
12
13 struct Edge{
14     int to,nxt,w;
15 }e[maxm];
16
17 void add(int u,int v,int w){
18     e[++adt]={v,head[u],w};
19     head[u]=adt;
20 }
21
22 struct node{
23     int dis,u;
24     bool operator <(const node &x)const{
25         return x.dis<dis;
26     }
27 };
28
29 void dijkstra(){
30     priority_queue<node> q;
31     q.push({0,s});
32     dis[s]=0;
33     while(!q.empty()){
34         int u=q.top().u;
35         q.pop();
36         if(vst[u]) continue;
37         vst[u]=1;
38         for(int i=head[u];i;i=e[i].nxt){
39             int v=e[i].to,w=e[i].w;
40             if(dis[v]>dis[u]+w){
41                 dis[v]=dis[u]+w;
42                 if(!vst[v]) q.push({dis[v],v});
43             }
44         }
45     }
46 }
47
48 int main(){
49     cin>>n>>m>>s;
50     FOR(i,1,n)

```

```

51     dis[i]=INF;
52     dis[s]=0;
53     FOR(i,1,m){
54         int u,v,w;
55         cin>>u>>v>>w;
56         add(u,v,w);
57     }
58     dijkstra();
59     FOR(i,1,n){
60         if(dis[i]!=INF) cout<<dis[i]<<" ";
61         else cout<<INT_MAX<<" ";
62     }
63     return 0;
64 }

```

树的存储和遍历

```

1  #include<iostream>
2  #include<queue>
3  #include<stack>
4  using namespace std;
5  const int N=1e5+10;
6
7  struct Edge{//边的结构体
8      int to,nxt;
9  }e[N*2];
10
11  int adt,head[N];
12
13  void add(int u,int v){//加边建树
14      e[++adt]={v,head[u]};
15      head[u]=adt;
16  }
17
18  int fa[N],cntp[N];
19
20  void dfs(int p1){//递归实现dfs
21      cntp[p1]=0;
22      for(int i=head[p1];i!=0;i=e[i].nxt){
23          int p2=e[i].to;
24          if(p2==fa[p1]) continue;
25          fa[p2]=p1;//点p2的父节点是点p1
26          dfs(p2);
27          cntp[p1]++;//统计子节点个数
28      }
29  }
30
31  void dfs2(){//栈实现dfs

```



```

32     stack<int> s;
33     s.push(1);
34     while(!s.empty()){
35         int p1=s.top();//访问栈顶
36         s.pop();//出栈
37         cntp[p1]=0;
38         for(int i=head[p1];i!=0;i=e[i].nxt){
39             int p2=e[i].to;
40             if(p2==fa[p1]) continue;
41             fa[p2]=p1;//点p2的父节点是点p1
42             s.push(p2);//入栈
43             cntp[p1]++;//统计子节点个数
44         }
45     }
46 }
47
48 void bfs(){//队列实现bfs
49     queue<int> q;
50     q.push(1);
51     while(!q.empty()){
52         int p1=q.front();//访问队首
53         q.pop();//出队
54         cntp[p1]=0;
55         for(int i=head[p1];i!=0;i=e[i].nxt){
56             int p2=e[i].to;
57             if(p2==fa[p1]) continue;
58             fa[p2]=p1;//点p2的父节点是点p1
59             q.push(p2);//入队
60             cntp[p1]++;//统计子节点个数
61         }
62     }
63 }
64
65 int main() {
66     int n;
67     cin>>n;
68     for(int i=1;i<=n-1;i++){
69         int u,v;
70         scanf("%d%d",&u,&v);
71         add(u,v);
72         add(v,u);
73     }
74     //dfs(1);
75     //bfs();
76     dfs2();
77     for(int i=1;i<=n;i++){
78         cout<<i<<" : "<<cntp[i]<<endl;
79     }
80     return 0;

```

```

81 }
82 /*
83 stdin
84 9
85 1 2
86 1 9
87 2 3
88 2 4
89 4 5
90 4 6
91 4 7
92 5 8
93 */
94
95 /*
96 stdout
97 1: 2
98 2: 2
99 3: 0
100 4: 3
101 5: 1
102 6: 0
103 7: 0
104 8: 0
105 9: 0
106 */

```

最小生成树

[P3366 【模板】最小生成树](#)

[参考](#)

Prim 算法

```

1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  const int INF=(1<<30);
6  #define maxn 5005
7  #define maxm 200005
8  struct edge{
9      int v,w,next;
10 }e[maxm<<1]; //无向图开两倍数组
11 int n,m;
12 int dis[maxn],now=1,ans;
13 //dis[i]表示已经加入最小生成树的点到点i的最短距离

```

```

14  bool vis[maxn];
15
16  int head[maxn],cnt;
17  void add(int u,int v,int w){
18      e[++cnt].v=v;
19      e[cnt].w=w;
20      e[cnt].next=head[u];
21      head[u]=cnt;
22  }
23
24  int prim(){
25      FOR(i,2,n)
26          dis[i]=INF;
27      for(int i=head[1];i;i=e[i].next)
28          dis[e[i].v]=min(dis[e[i].v],e[i].w);
29      FOR(j,1,n-1){
30          int minn=INF;
31          vis[now]=1;
32          FOR(i,1,n){
33              if(!vis[i] and minn>dis[i]){
34                  minn=dis[i];
35                  now=i;
36              }
37          }
38          if(minn==INF) return 0;
39          ans+=minn;
40          for(int i=head[now];i;i=e[i].next){
41              int v=e[i].v;
42              if(!vis[v] and dis[v]>e[i].w) dis[v]=e[i].w;
43          }
44      }
45      return ans;
46  }
47
48  int main(){
49      cin>>n>>m;
50      FOR(i,1,m){
51          int u,v,w;
52          cin>>u>>v>>w;
53          add(u,v,w),add(v,u,w);
54      }
55      int res=prim();
56      if(res) cout<<res;
57      else cout<<"orz";
58      return 0;
59  }

```

Kruskal 算法

```

1  #include<bits/stdc++.h>
2  #define FOR(i,a,b) for(int i=(a);i<=(b);++i)
3  using namespace std;
4
5  struct Edge{
6      int u,v,w;
7  }edge[200005];
8
9  int fa[5005],n,m;
10
11 bool cmp(Edge a,Edge b){
12     return a.w<b.w;
13 }
14
15 int find(int x){
16     while(x!=fa[x])
17         x=fa[x]=fa[fa[x]];
18     return x;
19 }
20
21 int kruskal(){
22     int eu,ev,cnt=0,ans=0;
23     sort(edge,edge+m,cmp);
24     FOR(i,0,m-1){
25         eu=find(edge[i].u),ev=find(edge[i].v);
26         if(eu==ev) continue;
27         ans+=edge[i].w;
28         fa[ev]=eu;
29         if(++cnt==n-1) break;
30     }
31     return ans;
32 }
33
34 bool is_connected(){
35     int cnt=0;
36     FOR(i,1,n){
37         if(fa[i]==i) cnt++;
38     }
39     if(cnt==1) return true;
40     else return false;
41 }
42
43 int main(){
44     cin>>n>>m;
45     FOR(i,1,n)
46         fa[i]=i;
47     FOR(i,0,m-1){
48         int u,v,w;
49         cin>>u>>v>>w;

```

```
50     edge[i]={u,v,w};
51 }
52 int res=kruskal();
53 if(is_connected()) cout<<res;
54 else cout<<"orz";
55 return 0;
56 }
```