**无向图的两个节点有没有连通**

**无向图的连通性（并查集）**

**无向图有没有环（并查集+所有点相连+（点的个数=边的个数+1））**

**无向图加一条边有没有环（并查集+两个点的上级是不是同一个点）**

<https://blog.csdn.net/qq_40772692/article/details/79667455>

#include <iostream>

#include <cstring>

using namespace std;

int pre[1001];

int find(int a)//判断两个点有没有连通只要判断find(a)==find(b)就OK

{

int r=a;

while(pre[r]!=r)

r=pre[r];

int i=a,j;

while(i!=r)

{

j=pre[i];

pre[i]=r;

i=j;

}

return r;

}

void join(int x,int y)

{

int fx=find(x),fy=find(y);

if(fx!=fy)

pre[fx]=fy;

}

int main()

{

int n,m;

while(cin>>n&&n)

{

cin>>m;

int sum=0;//sum表示整个无向图有多少个分支

memset(pre,0,sizeof(pre));

for(int i=1;i<=n;i++)

pre[i]=i;

for(int j=0;j<m;j++)

{

int a,b;

cin>>a>>b;

join(a,b);

}

for(int i=1;i<=n;i++)

{

if(pre[i]==i)

sum++;

}

cout<<sum<<endl;

}

return 0;

}

**有向图有没有环（DFS+判断点有没有被走过）**

**有向图最大强连通分量（Tarjan）**

#include <cstdio>

#include <stack>

#include <cstring>

#include <iostream>

using namespace std;

int n,m,idx=0,k=1,Bcnt=0; //Bcnt为强连通分量个数

int head[100];

int ins[100]={0};

int dfn[100]={0},low[100]={0};

int Belong[100];//存储强连通分量信息

stack <int> s;

struct edge

{

int v,next;

}e[100];

int min(int a,int b)

{

return a<b?a:b;

}

void adde(int u,int v)

{

e[k].v=v;

e[k].next=head[u];

head[u]=k++;

}

void readdata()

{

int a,b;

memset(head,-1,sizeof(head));

scanf("%d%d",&n,&m);

for(int i=1;i<=m;i++)

{

scanf("%d%d",&a,&b);

adde(a,b);

}

}

void tarjan(int u)

{

int v;

dfn[u]=low[u]=++idx;//每次dfs，u的次序号增加1

s.push(u);//将u入栈

ins[u]=1;//标记u在栈内

for(int i=head[u];i!=-1;i=e[i].next)//访问从u出发的边

{

v=e[i].v;

if(!dfn[v])//如果v没被处理过

{

tarjan(v);//dfs(v)

low[u]=min(low[u],low[v]);//u点能到达的最小次序号是它自己能到达点的最小次序号和连接点v能到达点的最小次序号中较小的

}

else if(ins[v])low[u]=min(low[u],dfn[v]);//如果v在栈内，u点能到达的最小次序号是它自己能到达点的最小次序号和v的次序号中较小的

}

if(dfn[u]==low[u])

{

Bcnt++;

do

{

v=s.top();

s.pop();

ins[v]=0;

Belong[v]=Bcnt;

}while(u != v);

}

}

void work()

{

for(int i=1;i<=n;i++)if(!dfn[i])tarjan(i);

printf("\n");

for(int i = 1;i <= 6;i++)printf("%d %d\n",dfn[i],low[i]);

printf("共有%d强连通分量，它们是:\n",Bcnt);

for(int i=1;i<=Bcnt;i++)

{

printf("第%d个:",i);

for(int j=1;j<=n;j++)

{

if(Belong[j]==i)printf("%d ",j);

}

printf("\n");

}

}

int main()

{

readdata();

work();

return 0;

}

/\*

6 8

1 2

1 3

2 4

3 4

3 5

4 1

4 6

5 6

\*/

**最小生成树（Prim, Kruskal）**

**https://blog.csdn.net/qq\_39630587/article/details/77427044**

**Prim**

#include<stdio.h>

#define MAX 100

#define MAXCOST 0x7fffffff

int graph[MAX][MAX];

void prim(int graph[][MAX], int n)

{

int lowcost[MAX];

int mst[MAX];

int i, j, min, minid, sum = 0;

for (i = 2; i <= n; i++)

{

lowcost[i] = graph[1][i];//lowcost存放顶点1可达点的路径长度

mst[i] = 1;//初始化以1位起始点

}

mst[1] = 0;

for (i = 2; i <= n; i++)

{

min = MAXCOST;

minid = 0;

for (j = 2; j <= n; j++)

{

if (lowcost[j] < min && lowcost[j] != 0)

{

min = lowcost[j];//找出权值最短的路径长度

minid = j; //找出最小的ID

}

}

printf("V%d-V%d=%d\n",mst[minid],minid,min);

sum += min;//求和

lowcost[minid] = 0;//该处最短路径置为0

for (j = 2; j <= n; j++)

{

if (graph[minid][j] < lowcost[j])//对这一点直达的顶点进行路径更新

{

lowcost[j] = graph[minid][j];

mst[j] = minid;

}

}

}

printf("最小权值之和=%d\n",sum);

}

int main()

{

int i, j, k, m, n;

int x, y, cost;

//freopen("1.txt","r",stdin);//文件输入

scanf("%d%d",&m,&n);//m=顶点的个数，n=边的个数

for (i = 1; i <= m; i++)//初始化图

{

for (j = 1; j <= m; j++)

{

graph[i][j] = MAXCOST;

}

}

for (k = 1; k <= n; k++)

{

scanf("%d%d%d",&i,&j,&cost);

graph[i][j] = cost;

graph[j][i] = cost;

}

prim(graph, m);

return 0;

}

**Kruskal**

#include <stdio.h>

#include <iostream>

#include <algorithm>

using namespace std;

#define MAXE 100

#define MAXV 100

typedef struct{

int vex1; //边的起始顶点

int vex2; //边的终止顶点

int weight; //边的权值

}Edge;

void kruskal(Edge E[],int n,int e)

{

int i,j,m1,m2,sn1,sn2,k,sum=0;

int vset[n+1];

for(i=1;i<=n;i++) //初始化辅助数组

vset[i]=i;

k=1;//表示当前构造最小生成树的第k条边，初值为1

j=0;//E中边的下标，初值为0

while(k<e)//生成的边数小于e时继续循环

{

m1=E[j].vex1;

m2=E[j].vex2;//取一条边的两个邻接点

sn1=vset[m1];

sn2=vset[m2];

//分别得到两个顶点所属的集合编号

if(sn1!=sn2)//两顶点分属于不同的集合，该边是最小生成树的一条边

{//防止出现闭合回路

printf("V%d-V%d=%d\n",m1,m2,E[j].weight);

sum+=E[j].weight;

k++; //生成边数增加

if(k>=n)

break;

for(i=1;i<=n;i++) //两个集合统一编号

if (vset[i]==sn2) //集合编号为sn2的改为sn1

vset[i]=sn1;

}

j++; //扫描下一条边

}

printf("最小权值之和=%d\n",sum);

}

//int fun(Edge arr[],int low,int high)

// {

// int key;

// Edge lowx;

// lowx=arr[low];

// key=arr[low].weight;

// while(low<high)

// {

// while(low<high && arr[high].weight>=key)

// high--;

// if(low<high)

// arr[low++]=arr[high];

//

// while(low<high && arr[low].weight<=key)

// low++;

// if(low<high)

// arr[high--]=arr[low];

// }

// arr[low]=lowx;

// return low;

// }

//void quick\_sort(Edge arr[],int start,int end)

//{

// int pos;

// if(start<end)

// {

// pos=fun(arr,start,end);

// quick\_sort(arr,start,pos-1);

// quick\_sort(arr,pos+1,end);

// }

//}

bool up(Edge a,Edge b)

{

return a.weight<b.weight;

}

int main()

{

Edge E[MAXE];

int nume,numn;

//freopen("1.txt","r",stdin);//文件输入

printf("输入顶数和边数:\n");

scanf("%d%d",&numn,&nume);

for(int i=0;i<nume;i++)

scanf("%d%d%d",&E[i].vex1,&E[i].vex2,&E[i].weight);

sort(E,E+nume,up);

//quick\_sort(E,0,nume-1);

kruskal(E,numn,nume);

return 0;

}

**网络最大流（Ford-Fulkerson）：**

**Ford-Fulkerson:**

#include<stdio.h>

#include<cstring>

#include<vector>

#include<algorithm>

using namespace std;

const int maxn = 1000+10;

const int INF = 1000000000;

struct edge{

int to, cap, rev;

};

vector<edge> G[maxn];

bool used[maxn];

int V,E;

/\*向图中增加一条从s到t为cap的边\*/

void add\_edge(int from, int to, int cap)

{

G[from].push\_back((edge){to, cap, G[to].size()});

G[to].push\_back((edge){from, 0, G[from].size() - 1});

}

/\*通过DFS找增广路\*/

int dfs(int v, int t, int f)

{

if(v == t) return f;

used[v] = true;

for(int i=0; i<G[v].size(); i++)

{

edge &e = G[v][i];

if(!used[e.to] && e.cap > 0)

{

int d = dfs(e.to, t, min(f, e.cap));

if(d > 0)

{

e.cap -= d;

G[e.to][e.rev].cap += d;

return d;

}

}

}

return 0;

}

/\*求解从s到t的最大流\*/

int max\_flow(int s, int t)

{

int flow = 0;

for(; ;)

{

memset(used, 0, sizeof(used));

int f = dfs(s, t, INF);

if(f == 0) return flow;

flow += f;

}

}

int main()

{

scanf("%d%d",&V,&E);//节点数，边数

int s,t,e;

for(int i=0; i<E; i++)

{

scanf("%d%d%d",&s,&t,&e);//节点数从1开始

add\_edge(s-1, t-1, e);

}

printf("%d\n", max\_flow(0, V-1));//第一个参数表示源节点，第二个表示汇聚节点

return 0;

}

**Dinic:**

#include <iostream>

#include <stdio.h>

#include <cstring>

#include <queue>

using namespace std;

const int inf = 0x7fffffff;

const int mn = 1e4 + 10, mm = 1e5 + 10;

int edge;

int fr[mn];

int cur[2 \* mn]; /// 当前弧优化

int lv[4 \* mn];

struct node

{

int to, val, nx, fan;

} e[2 \* mm];

void addedge(int u, int v, int w)

{

edge++;

e[edge].to = v, e[edge].val = w, e[edge].nx = fr[u], e[edge].fan = edge + 1;

fr[u] = edge;

edge++;

e[edge].to = u, e[edge].val = 0, e[edge].nx = fr[v], e[edge].fan = edge - 1;

fr[v] = edge;

}

void bfs(int s)

{

memset(lv, 0, sizeof(lv));

lv[s] = 1;

queue<int> q;

q.push(s);

while (!q.empty())

{

int t = q.front();

q.pop();

for (int i = fr[t]; i != -1; i = e[i].nx)

{

if (e[i].val > 0 && !lv[e[i].to])

{

lv[e[i].to] = lv[t] + 1;

q.push(e[i].to);

}

}

}

}

int dfs(int s, int t, int f)

{

if (s == t)

return f;

for (int i = cur[s]; i != -1; i = e[i].nx)

{

if (e[i].val > 0 && lv[s] < lv[e[i].to])

{

int d = dfs(e[i].to, t, min(f, e[i].val));

if (d > 0)

{

e[i].val -= d;

e[e[i].fan].val += d;

return d;

}

}

}

return 0;

}

int main()

{

memset(fr, -1, sizeof fr);

int n, m, s, t;

scanf("%d %d %d %d", &n, &m, &s, &t);//n为节点个数，m为边个数，s为源节点，t为汇聚节点

while (m--)

{

int u, v, w;

scanf("%d %d %d", &u, &v, &w);

addedge(u, v, w);

}

int flow = 0;//最大流

while (1)

{

bfs(s);

if (!lv[t])//没找到增广路径

break;

for (int i = 1; i <= n; i++)

cur[i] = fr[i];

int f = 0;

while ((f = dfs(s, t, inf)) > 0)

flow += f;

}

printf("%d\n", flow);

return 0;

}

**最短路径：Dijkstra**



每次寻找已知节点到达未知节点距离最短的一个节点，那么这个位置节点的最短距离我们就知道了，这个未知节点变为已知节点，再去更新未知节点的距离

#include<iostream>

#include<cstring>

#include<stdio.h>

using namespace std;

#define INF 0x7f7f7f7f

const int N = 105; //点的个数上限

int maze[N][N];

int dis[N];

bool vis[N];

//点的个数和边的条数

int n,m;

void init()

{

memset(maze,INF,sizeof(maze));

memset(dis,INF,sizeof(dis));

memset(vis,false,sizeof(vis));

}

void dijkstra(int st)

{

dis[st]=0;

for(int i=1; i<=n; i++)

{

//找到和起点距离最短的点

int minx=INF;

int minmark;

for(int j=1; j<=n; j++)

{

if(vis[j]==false&&dis[j]<=minx)

{

minx=dis[j];

minmark=j;

}

}

//并标记

vis[minmark]=true;

//更新所有和它连接的点的距离

for(int j=1; j<=n; j++)

{

if(vis[j]==false&&dis[j]>dis[minmark]+maze[minmark][j])

dis[j]=dis[minmark]+maze[minmark][j];

}

}

}

int main()

{

int s;

scanf("%d %d %d",&n,&m,&s);//节点个数，边数，源节点

//每组数据都要初始化

init();

for(int i=1; i<=m; i++)

{

int x,y,len;

scanf("%d %d %d",&x,&y,&len);

if(x!=y&&maze[x][y]>len)

{

maze[y][x]=len;

maze[x][y]=len;

}

}

//以1为起点跑一次dij

dijkstra(s);

//输出到n的距离

for(int i=1;i<=n;i++)

printf("%d ",dis[i]);//源节点到各点的距离

return 0;

}

**最短路径：SPFA**

#include <iostream>

#include <algorithm>

#include <queue>

#include <cstring>

using namespace std;

const int maxN = 200010 ;

struct Edge

{

int to , next , w ;

} e[ maxN ];

int n,m,cnt,p[ maxN ],Dis[ maxN ];

int In[maxN ];

bool visited[ maxN ];

void Add\_Edge ( const int x , const int y , const int z )

{

e[ ++cnt ] . to = y ;

e[ cnt ] . next = p[ x ];

e[ cnt ] . w = z ;

p[ x ] = cnt ;

return ;

}

bool Spfa(const int S)

{

int i,t,temp;

queue<int> Q;

memset ( visited , 0 , sizeof ( visited ) ) ;

memset ( Dis , 0x3f , sizeof ( Dis ) ) ;

memset ( In , 0 , sizeof ( In ) ) ;

Q.push ( S ) ;

visited [ S ] = true ;

Dis [ S ] = 0 ;

while( !Q.empty ( ) )

{

t = Q.front ( ) ;Q.pop ( ) ;visited [ t ] = false ;

for( i=p[t] ; i ; i = e[ i ].next )

{

temp = e[ i ].to ;

if( Dis[ temp ] > Dis[ t ] + e[ i ].w )

{

Dis[ temp ] =Dis[ t ] + e[ i ].w ;

if( !visited[ temp ] )

{

Q.push(temp);

visited[temp]=true;

if(++In[temp]>n)return false;

}

}

}

}

return true;

}

int main ( )

{

int S , T ;

scanf ( "%d%d%d%d" , &n , &m , &S , &T ) ;//节点数，边数，源节点，目的节点

for(int i=1 ; i<=m ; i++ )

{

int x , y , w ;//开始节点，结束节点，权值

scanf ( "%d%d%d" , &x , &y , &w ) ;

Add\_Edge ( x , y , w ) ;

}

if ( !Spfa ( S ) ) printf ( "FAIL!\n" ) ;

else printf ( "%d\n" , Dis[ T ] ) ;

return 0;

}