图结构

目录

[图结构 1](#_Toc450586549)

[无向图的双联通性问题 2](#_Toc450586550)

[有向图的强连通性问题 5](#_Toc450586551)

[最小生成树 7](#_Toc450586552)

[最短路径的算法优化 10](#_Toc450586553)

## 无向图的双联通性问题

无向图通过tarjan算法可以求出割点以及其边双联通分量和点双连通分量。

以下代码可以求出其割点，边双联通分量，点双联通分量

#include <bits/stdc++.h>

using namespace std;

const int MAXN = 1000;

typedef pair<int, int> PII;

struct Edge

{

int u,v;

Edge(){}

Edge(int u, int v):u(u),v(v){}

};

vector<Edge> G[MAXN];

void addedge(int u, int v)

{

G[u].push\_back(Edge(u, v));

G[v].push\_back(Edge(v, u));

}

int n, m, tot, vbs, ves;

int dfn[MAXN], low[MAXN], vis[MAXN];

bool cut[MAXN];

deque<PII> stak;

set<int> belongs[MAXN];

void mark(int u, int id)

{

if(belongs[u].find(id) == belongs[u].end())

{

belongs[u].insert(id);

printf("%d ", u);

}

}

set< PII > bridge;

void addbrige(int u, int v)

{

if(u > v) swap(u, v);

bridge.insert(PII(u, v));

}

bool isbridge(int u, int v)

{

if(u > v) swap(u, v);

if(bridge.find(PII(u, v)) == bridge.end()) return false;

return true;

}

void tarjan(int now, int pre)

{

dfn[now] = low[now] = ++tot;

int child = 0;

for (int i = 0, len = G[now].size(); i < len; i++)

if(G[now][i].v != pre && dfn[G[now][i].v] < dfn[now])

{

int to = G[now][i].v;

stak.push\_back(PII(now, to));

if(dfn[to] == 0)

{

child++;

tarjan(to, now);

low[now] = min(low[now], low[to]);

if((pre == -1 && child > 1) || (pre != -1 && dfn[now] <= low[to]))

cut[now] = true;

if(low[to] > dfn[now])

addbrige(now, to);

if(low[to] >= dfn[now])

{

PII tmp;

printf("biconnectivity found %d:\n", ++vbs);

while(true)

{

tmp = stak.back();

stak.pop\_back();

mark(tmp.first, vbs);

mark(tmp.second, vbs);

if(tmp == PII(now, to))

break;

}

printf("\n");

}

}

else if(to != pre)

low[now] = min(low[now], dfn[to]);

}

}

void dfs\_no\_bridge(int now, int pre)

{

vis[now] = true;

printf("%d ", now);

for (int i = 0, len = G[now].size(); i < len; i++)

if(!isbridge(now, G[now][i].v) && !vis[G[now][i].v])

{

dfs\_no\_bridge(G[now][i].v, now);

}

}

int main()

{

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

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

G[i].clear(),belongs[i].clear();

bridge.clear();

memset(dfn, 0, sizeof(dfn));

memset(low, 0, sizeof(low));

memset(cut, 0, sizeof(cut));

memset(vis, 0, sizeof(vis));

stak.clear();

vbs = ves = tot = 0;

int u, v;

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

{

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

addedge(u, v);

}

printf("Point biconnectivity:\n");

for (int i = 1; i <= n; i++) if(dfn[i] == 0)

tarjan(i, -1);

printf("Briges:\n");

for (PII x : bridge)

{

printf("%d - %d\n", x.first, x.second);

}

printf("Edge biconnectivity:\n");

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

{

printf("biconnectivity found %d:\n", ++ves);

dfs\_no\_bridge(i, -1);

printf("\n");

}

return 0;

}

/\*

sample 1:

7 8

1 2

2 3

3 4

1 4

4 5

5 6

6 7

4 7

\*/

## 有向图的强连通性问题

有向图同样通过tarjan 算法求出其强连通分量。

/\*

vertex index from 1 to n;

created on 2016/4/14

\*/

#include <bits/stdc++.h>

using namespace std;

const int MAXN = 10000;

struct Edge

{

int u, v;

Edge(){}

Edge(int u, int v):u(u),v(v){}

};

vector<Edge> G[MAXN];

vector<int> sc[MAXN];

deque<int> stak;

bool instak[MAXN];

int n, m, dfn[MAXN], low[MAXN],tot,sccnt;

void addedge(int u, int v)

{

G[u].push\_back(Edge(u,v));

}

void tarjan(int u)

{

dfn[u] = low[u] = ++tot;

stak.push\_back(u);

instak[u] = true;

int v;

for (int i = 0, len = G[u].size(); i < len; i++)

{

int v = G[u][i].v;

if (dfn[v] == 0)

{

tarjan(v);

if(low[u] > low[v])

low[u] = low[v];

}

// else if(instak[v] && low[v] < low[u])

else if(instak[v] && dfn[v] < low[u])

low[u] = low[v];

}

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

{

sccnt++;

while(true)

{

v = stak.back();

stak.pop\_back();

sc[sccnt].push\_back(v);

instak[v] = false;

if(v == u) break;

}

}

}

int main()

{

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

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

G[i].clear(),sc[i].clear();

memset(instak, 0, sizeof(instak));

stak.clear();

sccnt = 0;

int u, v;

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

{

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

addedge(u,v);

}

tot = 0;

for (int i = 1; i <= n; i++) if(dfn[i] == 0)

tarjan(i);

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

for (int j = 0, len = sc[i].size(); j < len; j++)

printf("%d%c", sc[i][j], j == len-1? '\n':' ');

return 0;

}

/\*

sample 1:

6 8

1 2

1 3

2 4

3 4

3 5

4 1

4 6

5 6

\*/

## 最小生成树

Kruskak：将边排序，从小到大枚举边，只要该边所连的两个端点不在一个集合里面，该边一定在最小生成树上。用并查集维护集合。

#include <bits/stdc++.h>

using namespace std;

struct Edge

{

int u, v, c;

Edge(){}

Edge(int u, int v, int c):u(u),v(v),c(c){}

bool operator < (const Edge &e) const {

return c < e.c;

}

};

vector<Edge> ve;

int n, m;

int R[10020];

// int root(int x)

// {

// while(R[x] != x)

// x = R[x] = R[R[x]];

// return R[x];

// }

int root(int x)

{

if(R[x] == -1) return x;

if(R[x] != -1) R[x] = root(R[x]);

return R[x];

}

int main()

{

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

int u, v, c;

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

{

scanf("%d%d%d", &u,&n,&c);

ve.push\_back(Edge(u,n,c));

}

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

// R[i] = i;

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

sort(ve.begin(), ve.end());

int ans = 0;

int Ru, Rv;

for (int i = 0, len = ve.size(); i < len; i++)

{

Edge &now = ve[i];

Ru = root(now.u);

Rv = root(now.v);

if(Ru != Rv)

{

ans += now.c;

R[Ru] = Rv;

}

}

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

return 0;

}

Prime：每次扩展一个点所能到达的其他点，选择边权最小的边对应的点加入集合。

在选择点时可以用优先队列来优化复杂度。

#include <bits/stdc++.h>

using namespace std;

struct Edge

{

int u, v, c;

Edge(){}

Edge(int u, int v, int c):u(u),v(v),c(c){}

};

vector<Edge> G[10020];

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

{

G[u].push\_back(Edge(u,v,c));

G[v].push\_back(Edge(v,u,c));

}

int n, m;

int vis[10020];

int dist[10020];

int prim()

{

int ans = 0;

memset(vis, 0, sizeof(vis));

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

vis[1] = 1;

int minid, minc;

int now = 1;

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

{

for (int i = 0, len = G[now].size(); i < len; i++)

{

int to = G[now][i].v, c = G[now][i].c;

if(vis[to] == 1) continue;

if(dist[to] > c)

dist[to] = c;

}

minid = -1;

minc = 0x3f3f3f3f;

for (int i = 1; i <= n; i++) if((!vis[i]) && dist[i] < minc)

{

minid = i;

minc = dist[i];

}

ans += minc;

vis[minid] = 1;

now = minid;

}

return ans;

}

int main()

{

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

int u,v,c;

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

{

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

addedge(u,v,c);

}

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

return 0;

}

## 最短路径的算法优化

Floyed: 动态规划的思想求取任意两点间的最短路。

复杂度：n3 ;

不能判断出负环。

代码如下：

#include <iostream>

#include <cstdio>

#include <algorithm>

#include <cstring>

using namespace std;

const int maxn=110;

const int INF=100000000;

int dist[maxn][maxn], G[maxn][maxn];

int n, m, num, minc;

void floyd()

{

minc=INF;

// 求最小环

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

{

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

for(int j=i+1; j<k; j++)

{

int ans=dist[i][j]+G[i][k]+G[k][j];

if(ans<minc) //找到最优解

{

minc=ans;

}

}

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

for(int j=1; j<=n; j++)if(i!=j)

{

if(dist[i][j]>dist[i][k]+dist[k][j])

{

dist[i][j]=dist[i][k]+dist[k][j];

}

}

}

}

int main()

{

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

int x, y, c;

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

{

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

dist[i][j] = G[i][j] = INF;

dist[i][i] = 0;

}

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

{

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

G[x][y] = G[y][x] = dist[x][y] = dist[y][x] = c;

}

floyd();

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

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

printf("%d%c", dist[i][j]==INF? -1: dist[i][j], j == n? '\n':' ');

return 0;

}

/\*

4 4

1 2 10

2 3 10

3 4 -10

1 4 10

\*/

Dijstra：贪心求单元最短路

复杂度 n+mlogn

不能判断出负环

代码如下：

/\*

\*

\* Dijstra shortest path and minimal cost

\*

\*/

#include <bits/stdc++.h>

#define maxn 1020

#define INF 0x7f7f7f7f

using namespace std;

typedef pair<int, int> PII;

struct Edge

//l 为边的长度， c为费用

{

int u, v, l, c;

Edge(){}

Edge(int u,int v, int l, int c):u(u),v(v),l(l),c(c){}

};

struct Node

//Node 用于 priority\_queue 的记录

//v: node id

//l: length from start

//c: mincost

{

int v, l, c;

Node(){}

Node(int v, int l, int c):v(v),l(l),c(c){}

bool operator < (const Node &a) const

//priority\_queue 的优先级和 < 相反

{

if(l == a.l) return c > a.c;

return l > a.l;

}

};

vector<Edge>G[maxn];

priority\_queue<Node>pq;

int dist[maxn],cost[maxn],vis[maxn],tot;

void add\_edge(int u, int v, int l, int c)

{

G[u].push\_back(Edge(u, v, l, c));

}

// PII dijstra(int s, int d)

//start s, dest d

void dijstra(int s)

{

memset(dist, INF, sizeof(dist));

memset(cost, INF, sizeof(cost));

memset(vis, 0, sizeof(vis));

while(!pq.empty()) pq.pop();

pq.push(Node(s, 0, 0));

while(!pq.empty())

{

const Node nd = pq.top();

pq.pop();

if(vis[nd.v]) continue;

vis[nd.v] = true;

dist[nd.v] = nd.l;

cost[nd.v] = nd.c;

// if(nd.v == d) return make\_pair(dist[d], cost[d]);

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

{

Edge& e = G[nd.v][i];

if(!vis[e.v])

{

pq.push(Node(e.v, nd.l + e.l, nd.c+e.c));

}

}

}

//dist[d]: shortest distance

//cost[d]: mincost

// return make\_pair(dist[d], cost[d]);

}

int main()

{

int n, m;

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

int u, v, c;

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

{

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

add\_edge(u,v,c,0);

add\_edge(v,u,c,0);

}

dijstra(1);

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

{

printf("%d%c", dist[i], i==n?'\n':' ');

}

return 0;

}

/\*

4 4

1 2 10

2 3 10

3 4 -10

1 4 10

\*/

Bellman-ford：因为最短路的长度不会超过点的个数减一，所以对所有的点的最短路进行n-1次松弛操作，肯定可以得到最短路，如果能够进行n次松弛，那么肯定存在负环。

复杂度：n2

能够处理负环图

#include <bits/stdc++.h>

using namespace std;

const int maxn = 1020;

struct Edge

{

int u, v, c;

Edge(){}

Edge(int u, int v, int c):u(u),v(v),c(c){}

};

// vector<Edge> G[maxn];

vector<Edge> edges;

int dist[maxn];

int n, m;

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

{

edges.push\_back(Edge(u,v,c));

// edges.push\_back(Edge(v,u,c));

}

bool bellmanford(int s)

{

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

dist[s] = 0;

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

{

for (int j = 0, len = edges.size(); j < len; j++)

{

int u = edges[j].u, v = edges[j].v, c = edges[j].c;

if(dist[v] > dist[u] + c)

dist[v] = dist[u] + c;

}

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

// printf("%d ", dist[j]);

// printf("\n");

}

for (int i = 0, len = edges.size(); i < len; i++)

{

Edge& edge = edges[i];

if(dist[edge.v] > dist[edge.u] + edge.c)

return false;

}

return true;

}

int main()

{

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

int u,v,c;

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

{

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

addedge(u, v, c);

}

if(bellmanford(1))

{

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

printf("%d%c", dist[i], i == n? '\n' : ' ');

}

else

{

printf("No shortest path!!!\n");

}

return 0;

}