#include <iostream> // Dijkstra 迪杰斯特拉

#include <stack> // 输出最短路径

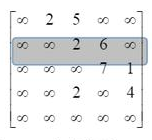
using namespace std;

//———————————————1 数据结构—————————————————————

const int N = 100; // 城市的个数，用于初始化固定数组

点不连通

const int INF = 1e7; // 初始化无穷大为10000000，表示从一点到另一点不连通

int map[N][N]; // 带权邻接矩阵，表示点的连通和距离

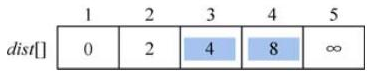
int n,m; // n表示城市的个数，m为城市间路线的条数，用于循环输入

int dist[N]; // 表示从源点u到城市j的最短路径距离为dist[j]，中间值和输出

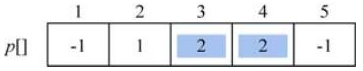
int p[N]; // 表示从源点u到城市j的最短路径的前驱节点为p[j]，中间值和输出

bool flag[N]; // 如果flag[i]等于true，说明顶点i已经加入到集合S;否则顶点i属于集合V-S

// 集合V表示全部顶点，集合S按贪心策略已搜索的最短路径顶点

void Dijkstra(int u)

{

//———————————————2 初始化——————————————————————

for(int i=1; i<=n; i++) // ①

{ // 初始化源点u到其他各个顶点的最短路径长度

dist[i] =map[u][i];

flag[i]=false;

if(dist[i]==INF)

p[i]=-1; // 源点u到该顶点的路径长度为无穷大，说明顶点i与源点u不相邻

else

p[i]=u; // 说明顶点i与源点u相邻，设置顶点i的前驱p[i]=u

}

dist[u] = 0;

flag[u]=true; // 初始时，集合S中只有一个元素：源点u

for(int k=1; k<=n; k++) // ②

{

//———————————————3 在集合V-S（!flag[m]）中找距离源点u最近的顶点t——

int temp = INF, t = u;

for(int m=1; m<=n; m++) // ③ 在集合V-S中寻找距离源点u最近的顶点t

if(!flag[m]&&dist[m]<temp)

{

t=m;

temp=dist[m];

}

if(t==u) return ; // 找不到t（t未更新），跳出循环②

flag[t]= true; // 否则，将t加入集合

//———————————————4 找到t点后，更新从源点u经t后到集合V-S中与t邻接的顶点的距离

for(int j=1;j<=n;j++) // ④ 更新集合V-S中与t邻接的顶点到源点u的距离

if(!flag[j]&& map[t][j]<INF) // 前者表示j在V-S中，后者表示T与J相邻

if(dist[j]>(dist[t]+map[t][j])) // 经T到J的路径最短

{

dist[j]=dist[t]+map[t][j]; // 更新源点到j的最短距离

p[j]=t; // 更新j的前驱

}

}

}

void findpath(int u) // 源点到其它各顶点的最短路径

{

int x; // 前驱

stack<int>s; // 利用库<stack>创建一个栈s

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

{

x=p[i];

while(x!=-1)

{

s.push(x); // 将前驱依次压入栈中

x=p[x];

}

cout<<"源点"<<u<<"到顶点"<<i<<"的最短距离为：";

if(dist[i] == INF)

cout << "sorry,无路可达"<<endl;

else

{

cout <<dist[i];

cout<<"，\t路径为：";

while(!s.empty())

{

cout<<s.top()<<"→"; // 输出栈顶元素

s.pop(); // 依次出栈

}

cout<<i<<endl;

}

}

}

void printmatrix(int n) // 输出图的带权邻接矩阵

{

cout <<"输出矩阵map："<<endl;

cout<<"\t";

for(int k=1;k<=n;++k) // 输出列标题

cout<<"\t"<<k;

cout<<endl;

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

{

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

{

if(j==1) // 输出行标题

cout << "\t" << i;

if(map[i][j]==INF)

cout << "\t" << "∞";

else

cout << "\t" << map[i][j];

if(j==n)

cout<<endl;

}

}

}

void initmatrix(int m,int n)

{

int u,v,w;

for(int i=1;i<=n;i++) // 初始化图的邻接矩阵

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

map[i][j]=INF; // 初始化邻接矩阵为无穷大

while(m--)

{

cin >> u >> v >> w;

map[u][v] = map[u][v]<w?map[u][v]:w; // 邻接矩阵储存，保留最小的距离

}

}

int main()

{

cout << "请输入城市的个数："<<endl;

cin >> n;

cout << "请输入城市之间的路线的个数："<<endl;

cin >>m;

cout << "请输入城市之间的路线以及距离："<<endl;

initmatrix(m,n);

int st;

cout<<"请输入小明所在的位置："<<endl;

cin>>st;

printmatrix(n);

Dijkstra(st);

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

cout <<"小明从"<<st<<" to "<<k;

if(dist[k] == INF)

cout << "的最短距离为:sorry,无路可达"<<endl;

else

cout << "的最短距离为:"<<dist[k]<<endl;

}

findpath(st); // 主函数中st为源点

system("pause");

return 0;

}

输出矩阵map：

1 2 3 4 5

1 ∞ 2 5 ∞ ∞

2 ∞ ∞ 2 6 ∞

3 ∞ ∞ ∞ 7 1

4 ∞ ∞ 2 ∞ 4

5 ∞ ∞ ∞ ∞ ∞

源点1到顶点1的最短距离为：0，路径为：1

源点1到顶点2的最短距离为：2，路径为：1→2

源点1到顶点3的最短距离为：4，路径为：1→2→3

源点1到顶点4的最短距离为：8，路径为：1→2→4

源点1到顶点5的最短距离为：5，路径为：1→2→3→5