求有

A 点到

B点的最短路径。

输入：四个数 n,m,a,b，和图。 其中 n 为图中结点数目，m为边数目，a, b 分别为A 和B 对应的结点编号

输出格式：一个数，表示a 到b的最短距离。

样例：

7 11 5 4

2 4 2

1 4 3

7 2 2

3 4 3

5 7 5

7 3 3

6 1 1

6 3 4

2 4 3

5 6 3

7 2 1

输出：7

**代码：**

#include<iostream>

using namespace std;

#define error 0

#define ok 1

#define duplicate 2

#define INFTY 32767

typedef int elemtype ;

typedef int status ;

typedef struct enode

{

int adjvex;

elemtype w;

struct enode \*nextarc;

}enode;

typedef struct

{

int n;

int e;

enode \*\*a;//指向一维指针数组

}lgraph;

status init(lgraph \*lg,int nsize) //初始化

{

int i;

lg->n=nsize;

lg->e=0;

lg->a=(enode\*\*)malloc(nsize\*sizeof(enode\*));

if(!lg->a)

return error;

else

{

for(i=0;i<lg->n;i++)

lg->a[i]=NULL;

return ok;

}

}

status exist(lgraph \*lg,int u, int v) //边的搜索

{

enode \*p;

if(u<0||v<0||u>lg->n-1||v>lg->n-1||u==v) //判断输入边是否正确

return error;

p=lg->a[u];

while(p&&p->adjvex!=v)

p=p->nextarc;

if(!p)

return error;

else

return ok;

}

status insert(lgraph \*lg,int u,int v,elemtype w) //边的插入

{

enode \*p;

if(u<0||v<0||u>lg->n-1||v>lg->n-1||u==v)

return error;

if(exist(lg,u,v))

return duplicate;

p=(enode\*)malloc(sizeof(enode));

p->adjvex=v;

p->w=w;

p->nextarc=lg->a[u];//新结点插入到单链表最前面

lg->a[u]=p;

lg->e++;

return ok;

}

int choose(int \*d,int \*s,int n) //选出最小的d[i]

{ // d[i]保存从源点到顶点最短路径长度

int i,minpos;

elemtype min;

min=INFTY;

minpos=-1;

for(i=0;i<n;i++) //找到d[i]最小的，即最短路径

{

if(d[i]<min&&!s[i])

{

min=d[i];

minpos=i;

}

}

return minpos; //返回下标

}

status dijkstra(int v,int v2,int \*d,int \*path,lgraph \*g) //迪杰斯特拉算法最短路径

{

int i,k;

int distance; //最短距离

enode \*p;

p=g->a[v];//工作指针

int \*s;

s=(int\*)malloc(sizeof(int)\*g->n); //申请s数组标记 结点是否被访问

if(v<0||v>g->n-1) //不合理数据判断

{

return error;

}

for(i=0;i<g->n;i++) //初始化1，所有结点初始化

{

d[i]=INFTY; //保存从源点到顶点i的当前最短路径长度。（源点到顶点有边则为边的权值）

path[i]=-1; //保存从源点到顶点i最短路径上i的直接前驱顶点

s[i]=0; //为1则源点到顶点i的最短路径确定，0则无

}

while(p) //初始化2，将与源点连接的结点数据初始化

{

d[p->adjvex]=p->w;

if(p->adjvex!=v&&d[p->adjvex]<INFTY) //path[p]=v

{

path[p->adjvex]=v;

}

p=p->nextarc;

}

s[v]=1;

d[v]=0; //v为源点

for(i=1;i<g->n;i++)

{

k=choose(d,s,g->n); //找到d[]最小的小标

if(k==-1) //若没找到

{

continue; //直接进行下一次循环

}

s[k]=1; //k加入s中

if(s[v2]==1) //判断终点，若到达终点输出最短路径

{

cout<<d[v2];

return 0;

}

p=g->a[k];

if(p==NULL)

{

continue;

}

while(p)

{

if(!s[p->adjvex]&&d[k]+p->w<d[p->adjvex]) //更新d和path

{

d[p->adjvex]=d[k]+p->w;

distance=d[p->adjvex];

path[p->adjvex]=k;

}

p=p->nextarc;

}

}

return ok;

}

int main()

{

lgraph g;

int enode,edge; //结点 边

int u,v;

int v1,v2; //起点终点

int d[100];

int path[100];

elemtype w; //权值

cin>>enode>>edge>>v1>>v2;

init(&g,enode);

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

{

cin>>u>>v>>w;

insert(&g,u,v,w);

}

dijkstra(v1,v2,d,path,&g);

return 0;

}

**代码结果：**



