简单路径

算法：

void Floyd(MGraph N,PathMatrix path,ShortPathLength dist)

{

int u,v,w,i;

for(v=0;v<N.vexnum;v++) /\*初始化数组path和dist\*/

for(w=0;w<N.vexnum;w++)

{

dist[v][w]=N.arc[v][w].adj; /\*初始时，顶点v到顶点w的最短路径为v到w的弧的权值\*/

for(u=0;u<N.vexnum;u++)

path[v][w][u]=0; /\*路径矩阵初始化为零\*/

if(dist[v][w]<INFINITY) /\*如果v到w有路径，则由v到w的路径经过v和w两点\*/

{

path[v][w][u]=1;

path[v][w][w]=1;

}

}

for(u=0;u<N.vexnum;u++)

for(v=0;v<N.vexnum;v++)

for(w=0;w<N.vexnum;w++)

if(dist[v][u]<INFINITY&&dist[u][w]<INFINITY&&dist[v][u]+dist[u][w]<dist[v][w])

/\*从v经u到w的一条路径为当前最短的路径\*/

{

dist[v][w]=dist[v][u]+dist[u][w]; /\*更新v到w的最短路径\*/

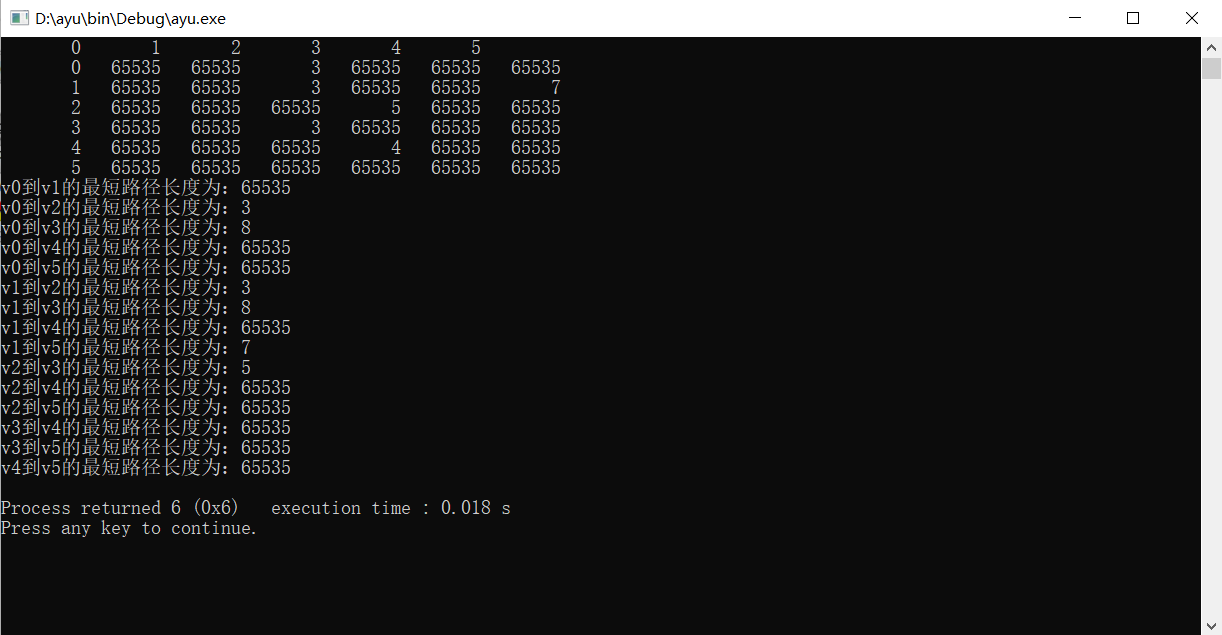
for(i=0;i<N.vexnum;i++)

path[v][w][i]=path[v][u][i]||path[u][w][i];

}

}

运行截图：



求长度最短的路径

算法：

void ShortestPath\_DIJ()

{

int i = 0, min = 0;

for (v = 0; v < n; v++) //循环 初始化

{

final[v] = 0; D[v] = arcs[v0][v];

for (w = 0; w < n; w++) p[v][w] = 0;//设空路径

if (D[v] < MAX) {p[v][v0] = 1; p[v][v] = 1;}

}

D[v0] = 0; final[v0]=0; //初始化 v0顶点属于集合S

//开始主循环 每次求得v0到某个顶点v的最短路径 并加v到集合S中

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

{

min = MAX;

for (w = 0; w < n; w++)

{

//我认为的核心过程--选点

if (!final[w]) //如果w顶点在V-S中

{

//这个过程最终选出的点 应该是选出当前V-S中与S有关联边

//且权值最小的顶点 书上描述为 当前离V0最近的点

if (D[w] < min) {v = w; min = D[w];}

}

}

final[v] = 1; //选出该点后加入到合集S中

for (w = 0; w < n; w++)//更新当前最短路径和距离

{

/\*在此循环中 v为当前刚选入集合S中的点

则以点V为中间点 考察 d0v+dvw 是否小于 D[w] 如果小于 则更新

比如加进点 3 则若要考察 D[5] 是否要更新 就 判断 d(v0-v3) + d(v3-v5) 的和是否小于D[5]

\*/

if (!final[w] && (min+arcs[v][w]<D[w]))

{

D[w] = min + arcs[v][w];

// p[w] = p[v];

p[w][w] = 1; //p[w] = p[v] +　[w]

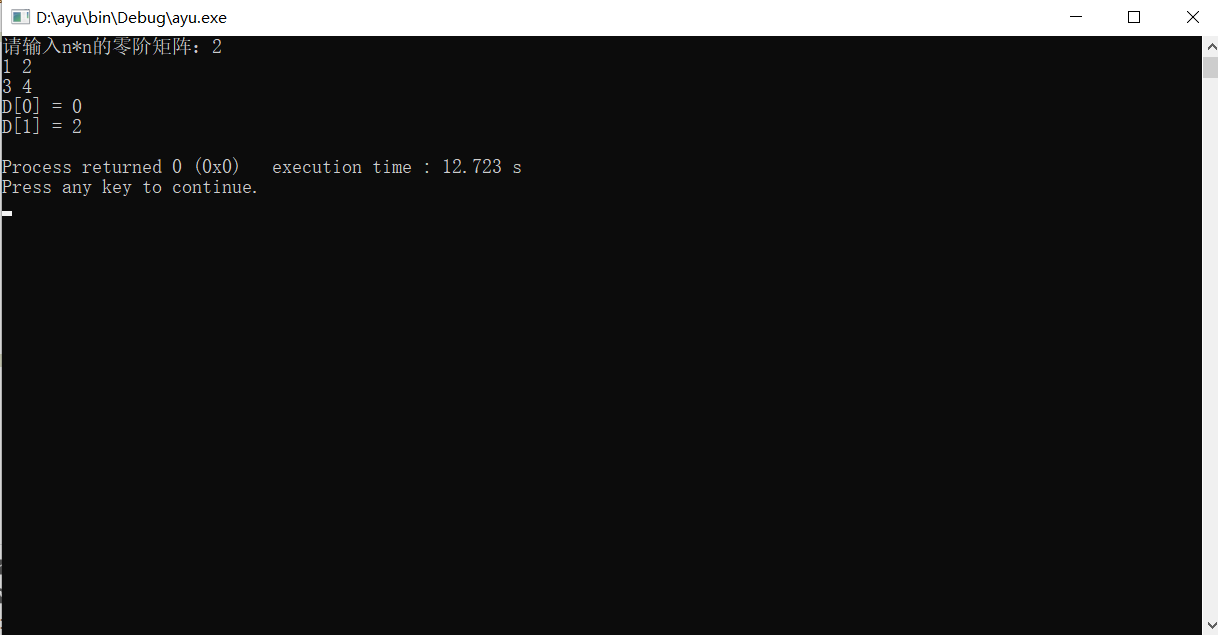
}

}

}

}

运行截图：



源代码（简单路径）：

#include<stdio.h>

#include<string.h>

#include<malloc.h>

#include<stdlib.h>

#define INFINITY 65535 /\*定义一个无限大的值\*/

#define MaxSize 50 /\*最大顶点个数\*/

typedef char VertexType[4];

typedef char InfoPtr;

typedef int VRType;

typedef enum{DG,DN,UG,UN}GraphKind; /\*图的类型：有向图、有向网、无向图和无向网\*/

typedef int PathMatrix[MaxSize][MaxSize][MaxSize]; /\*定义一个保存最短路径的三维数组\*/

typedef int ShortPathLength[MaxSize][MaxSize]; /\*定义一个保存从顶点v到顶点w的最短距离的数组\*/

typedef struct

{

VRType adj; /\*对于无权图，用1表示相邻，0表示不相邻；对于带权图，存储权值\*/

InfoPtr \*info; /\*与弧或边的相关信息\*/

}ArcNode,AdjMatrix[MaxSize][MaxSize];

typedef struct /\*图的类型定义\*/

{

VertexType vex[MaxSize]; /\*用于存储顶点\*/

AdjMatrix arc; /\*邻接矩阵，存储边或弧的信息\*/

int vexnum,arcnum; /\*顶点数和边（弧）的数目\*/

GraphKind kind; /\*图的类型\*/

}MGraph;

typedef struct /\*添加一个存储网的行、列和权值的类型定义\*/

{

int row;

int col;

int weight;

}GNode;

void CreateGraph(MGraph \*N,GNode \*value,int vnum,int arcnum,VertexType \*ch)

/\*采用邻接矩阵表示法创建有向网N\*/

{

int i,j,k;

N->vexnum=vnum;

N->arcnum=arcnum;

for(i=0;i<vnum;i++) /\*将各个顶点赋值给vex域\*/

strcpy(N->vex[i],ch[i]);

for(i=0;i<N->vexnum;i++) /\*初始化邻接矩阵\*/

for(j=0;j<N->vexnum;j++)

{

N->arc[i][j].adj=INFINITY;

N->arc[i][j].info=NULL; /\*弧的信息初始化为空\*/

}

for(k=0;k<arcnum;k++)

{

i=value[k].row;

j=value[k].col;

N->arc[i][j].adj=value[k].weight;

}

N->kind=DN; /\*图的类型为有向网\*/

}

void Floyd(MGraph N,PathMatrix path,ShortPathLength dist)

{

int u,v,w,i;

for(v=0;v<N.vexnum;v++) /\*初始化数组path和dist\*/

for(w=0;w<N.vexnum;w++)

{

dist[v][w]=N.arc[v][w].adj; /\*初始时，顶点v到顶点w的最短路径为v到w的弧的权值\*/

for(u=0;u<N.vexnum;u++)

path[v][w][u]=0; /\*路径矩阵初始化为零\*/

if(dist[v][w]<INFINITY) /\*如果v到w有路径，则由v到w的路径经过v和w两点\*/

{

path[v][w][u]=1;

path[v][w][w]=1;

}

}

for(u=0;u<N.vexnum;u++)

for(v=0;v<N.vexnum;v++)

for(w=0;w<N.vexnum;w++)

if(dist[v][u]<INFINITY&&dist[u][w]<INFINITY&&dist[v][u]+dist[u][w]<dist[v][w])

/\*从v经u到w的一条路径为当前最短的路径\*/

{

dist[v][w]=dist[v][u]+dist[u][w]; /\*更新v到w的最短路径\*/

for(i=0;i<N.vexnum;i++) /\*从v到w的路径经过从v到u和从u到w的所有路径\*/

path[v][w][i]=path[v][u][i]||path[u][w][i];

}

}

void DisplayGraph(MGraph N)

/\*输出邻接矩阵存储表示的图N\*/

{

int i,j;

for(i=0;i<N.vexnum;i++)

printf("%8d",i);

printf("\n");

for(i=0;i<N.vexnum;i++)

{

printf("%8d",i);

for(j=0;j<N.vexnum;j++)

printf("%8d",N.arc[i][j].adj);

printf("\n");

}

}

void main()

{

int i,j,vnum=6,arcnum=8;

MGraph N;

GNode value[]={{1,2,3},{0,2,3},{1,5,6},{1,5,7},

{2,3,5},{3,2,3},{4,3,4},{5,23,0}};

VertexType ch[]={"v0","v1","v2","v3","v4","v5"};

PathMatrix path; /\*用二维数组存放最短路径所经过的顶点\*/

ShortPathLength dist; /\*用一维数组存放最短路径长度\*/

CreateGraph(&N,value,vnum,arcnum,ch); /\*创建有向网N\* \*/

DisplayGraph(N); /\*输出有向网N\*/

Floyd(N,path,dist);

for(i=0;i<N.vexnum;i++)

{

for(j=i+1;j<N.vexnum;j++)

printf("%s到%s的最短路径长度为：%d\n",N.vex[i],N.vex[j],dist[i][j]);

}

}

源代码（最短路径）：

#include <stdio.h>

#include <stdlib.h>

#define MAX 1000000

int arcs[10][10];//邻接矩阵

int D[10];//保存最短路径长度

int p[10][10];//路径

int final[10];//若final[i] = 1则说明 顶点vi已在集合S中

int n = 0;//顶点个数

int v0 = 0;//源点

int v,w;

void ShortestPath\_DIJ()

{

int i = 0, min = 0;

for (v = 0; v < n; v++) //循环 初始化

{

final[v] = 0; D[v] = arcs[v0][v];

for (w = 0; w < n; w++) p[v][w] = 0;//设空路径

if (D[v] < MAX) {p[v][v0] = 1; p[v][v] = 1;}

}

D[v0] = 0; final[v0]=0; //初始化 v0顶点属于集合S

//开始主循环 每次求得v0到某个顶点v的最短路径 并加v到集合S中

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

{

min = MAX;

for (w = 0; w < n; w++)

{

//我认为的核心过程--选点

if (!final[w]) //如果w顶点在V-S中

{

//这个过程最终选出的点 应该是选出当前V-S中与S有关联边

//且权值最小的顶点 书上描述为 当前离V0最近的点

if (D[w] < min) {v = w; min = D[w];}

}

}

final[v] = 1; //选出该点后加入到合集S中

for (w = 0; w < n; w++)//更新当前最短路径和距离

{

/\*在此循环中 v为当前刚选入集合S中的点

则以点V为中间点 考察 d0v+dvw 是否小于 D[w] 如果小于 则更新

比如加进点 3 则若要考察 D[5] 是否要更新 就 判断 d(v0-v3) + d(v3-v5) 的和是否小于D[5]

\*/

if (!final[w] && (min+arcs[v][w]<D[w]))

{

D[w] = min + arcs[v][w];

// p[w] = p[v];

p[w][w] = 1; //p[w] = p[v] +　[w]

}

}

}

}

int main()

{

int i, j;

printf("请输入n\*n的零阶矩阵：");

scanf("%d", &n); //顶点个数

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

{

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

{

scanf("%d",&arcs[i][j]); //用来存储邻接矩阵

}

}

ShortestPath\_DIJ();

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

printf("D[%d] = %d\n",i,D[i]);

return 0;

}