代码：

#include"stdio.h"

#include"stdlib.h"

#include"string.h"

#define M 30 /\*预定义图的最大顶点数\*/

#define FINITY 5000 //用 5000代替无穷

//最小生成树算法的结构体

typedef struct {

char vexs[M]; //顶点的数据域

int edges[M][M]; //定义的邻接矩阵

int n, e; //图中的顶点数和边数

}Mgraph;

typedef struct edgedata

{

int beg, en; //beg,en都是顶点序号

int length; //边长

}edge;

void create(Mgraph \*g, char \*filename, int c) //c = 0表示创建无向图

{

int weight = 0; //边的权值

int front, rear; //变得前驱和后驱

FILE \*file;

file = fopen(filename, "r"); //从文件读取图的边信息

if (file)

{

fscanf(file, "%d%d\n", &g->n, &g->e); //读取图的顶点数与边数

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

{

fscanf(file, "%c", &g->vexs[i]); //读取图中顶点的信息域

}

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

{

for (int j = 0; j < g->n; j++)

{

if (i == j)

g->edges[i][j] = 0; //对角线上的均为0

else

g->edges[i][j] = FINITY; //没有赋值的域均加成无

}

}

for (int k = 0; k < g->e; k++)

{

fscanf(file, "%d%d%d", &front, &rear, &weight);

g->edges[front][rear] = weight; //只些相应顺序的，即有向图

if (c == 0)

g->edges[rear][front] = weight; //对称图形，即为无向图

}

fclose(file);

}

else

{

g->n = 0;

printf("文件打开失败！\n");

}

}

// 图的最短路径

typedef enum{FALSE,TRUE} boolean; /\*FALSE 为0，TRUE为1\*/

typedef int dist[M]; //距离向量类型

typedef int path[M]; //路径类型

/\*

Dijkstra算法求单源最短路径

g 有向图

root 根节点

p 路径 p[i] 用于保存最短路径上第i个顶点的前驱顶点的序号

d 距离向量 用于保存从源点出发的顶点到其他顶点的距离向量

\*/

void Dijkstra(Mgraph g,int root,path p, dist d)

{

int v = 0; //用于临时保存

int min = 0;

boolean final[M]; //表示当前元素是否已经求出最短路径，定义为布尔类型

for (int i = 0; i < g.n; i++) //初始化各种集合与距离向量

{

final[i] = FALSE; //全部赋值为FALSE

d[i] = g.edges[root][i];

if (d[i] < FINITY && d[i] != 0) //非自我路径和达不到的路径

p[i] = root; //前驱结点为根结点

else

p[i] = -1; //当前结点无前驱结点

}

final[root] = TRUE; //初始时s只有一个根节点root一个结点

d[root] = 0;

printf("%d到其他顶点的最短距离为：\n", root);

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

{

min = FINITY;

for (int k = 0; k < g.n; ++k) //循环找出最小边入结点

{

if (!final[k] && d[k] < min) //如果还没有求出最短路径并且存在更短的路径

{

v = k; //将初始结点替换为当前结点

min = d[k]; //min保存最短路径

}

}

printf("\n%d-->%d 路径长度： %d\n",root,g.vexs[v], min);

if (min == FINITY) //如果得出的路径无法达到

return;

final[v] = TRUE; //该结点已经求出最短路径

//修改S到V-S中各结点的距离

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

if (!final[i] && (min + g.edges[v][i] < d[i]))

{

d[i] = min + g.edges[v][i]; //最短路径等于之前的最短路径加上最近的最短路径

p[i] = v; //将 路径向量赋值

}

}

}

void print\_dpd(Mgraph g, path p, dist d)

{

int st[M], pre, top = -1;//定义栈st并初始化空栈

printf("\n\n最短路径为：\n\n");

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

{

printf("路径长度：%3d ", d[i]); //将距离向量输出

st[++top] = i; //将i入栈

pre = p[i]; //储存当前

while (pre != -1)

{

st[++top] = pre; //让pre入栈

pre = p[pre];

}

printf("路径走向：");

while (top > 0)

printf("%2d", st[top--]);

printf("\n");

}

}

int main()

{

Mgraph g;

char filename[20] = "D:\\Desktop\\Test.txt";

path p;

dist d;

int root;

create(&g, filename, 1);

printf("请输入源点：\n");

scanf("%d", &root);

Dijkstra(g, root, p, d);

print\_dpd(g, p, d);

system("pause");

return 0;

}

文件格式:



输出结果：

