|  |  |
| --- | --- |
| haut | 河南工业大学信息科学与工程学院 |

**《算法设计与分析》实验报告**

|  |  |
| --- | --- |
| 课程名称： | 算法设计与分析 |
| 学生姓名： | 刘文博 |
| 学生学号： | 201716040224 |
| 学生专业： | 软件工程1702 |

**实验二：最短路径**

苗建雨

2019 年 5 月 9 日

实验日期： 2019 年 5 月 16 日 班级： 软件工程1702

学号（后四位）：\_\_\_0224\_\_\_\_\_\_ 姓名：\_\_\_\_刘文博\_\_\_\_\_ 成绩：

**一、 实验目的：**

熟练掌握利用动态规划方法解决问题的基本思想，学会如何将问题化为多阶段图的方法，并能对具体问题写出正确的递推公式。

**二、实验要求：**

熟练掌握利用动态规划方法解决问题的基本思想，学会如何将问题化为多阶段图的方法，并能对具体问题写出正确的递推公式。并用代码实现。

**三、实验内容：**

**1.** **Dijkstra算法**

通过对最短路径求解的学习，掌握动态规划算法的基本思想。Dijkstra算法是一种经

典的使用动态规划思想解决问题的算法，其主要思想是：初始集合S包含路径的起点，随

后每次从集合之外的点中选择一个到起点最近的点

D[i] = min{D[j]|vj ∈ V − S}

加入该集合，最后按照如下条件更新所有经过该点的路径的长度

D[k] = D[i] + arcs[i][k], if D[i] + arcs[i][k] < D[k];

**代码实现：**

#include "stdio.h"

#include "stdlib.h"

#include "io.h"

#include "math.h"

#include "time.h"

#define OK 1

#define ERROR 0

#define TRUE 1

#define FALSE 0

#define MAXEDgE 20

#define MAXVEX 20

#define INFINITY 65535

typedef int Status;

typedef struct

{

int vexs[MAXVEX];

int arc[MAXVEX][MAXVEX];

int numVertexes, numEdges;

}Mgraph;

typedef int Patharc[MAXVEX];

typedef int ShortPathTable[MAXVEX];

void CreateMgraph(Mgraph \*g)

{

int i, j;

/\* printf("请输入边数和顶点数:"); \*/

g->numEdges=16;

g->numVertexes=9;

for (i = 0; i < g->numVertexes; i++)/\* 初始化图 \*/

{

g->vexs[i]=i;

}

for (i = 0; i < g->numVertexes; i++)/\* 初始化图 \*/

{

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

{

if (i==j)

g->arc[i][j]=0;

else

g->arc[i][j] = g->arc[j][i] = INFINITY;

}

}

g->arc[0][1]=1;

g->arc[0][2]=5;

g->arc[1][2]=3;

g->arc[1][3]=7;

g->arc[1][4]=5;

g->arc[2][4]=1;

g->arc[2][5]=7;

g->arc[3][4]=2;

g->arc[3][6]=3;

g->arc[4][5]=3;

g->arc[4][6]=6;

g->arc[4][7]=9;

g->arc[5][7]=5;

g->arc[6][7]=2;

g->arc[6][8]=7;

g->arc[7][8]=4;

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

{

for(j = i; j < g->numVertexes; j++)

{

g->arc[j][i] =g->arc[i][j];

}

}

}

void ShortestPath\_Dijkstra(Mgraph g, int v0, Patharc \*P, ShortPathTable \*D)

{

int v,w,k,min;

int final[MAXVEX];

/\* 初始化数据 \*/

for(v=0; v<g.numVertexes; v++)

{

final[v] = 0;

(\*D)[v] = g.arc[v0][v];

(\*P)[v] = 0;

}

(\*D)[v0] = 0;

final[v0] = 1;

/\* 开始主循环，每次求得v0到某个v顶点的最短路径 \*/

for(v=1; v<g.numVertexes; v++)

{

min=INFINITY;

for(w=0; w<g.numVertexes; w++) /\* 寻找离v0最近的顶点 \*/

{

if(!final[w] && (\*D)[w]<min)

{

k=w;

min = (\*D)[w]; /\* w顶点离v0顶点更近 \*/

}

}

final[k] = 1; /\* 将目前找到的最近的顶点置为1 \*/

/\* 修正当前最短路径及距离 \*/

for(w=0; w<g.numVertexes; w++)

{

/\* 如果经过v顶点的路径比现在这条路径的长度短的话 \*/

if(!final[w] && (min+g.arc[k][w]<(\*D)[w]))

{

/\* 说明找到了更短的路径，修改D[w]和P[w] \*/

(\*D)[w] = min + g.arc[k][w]; /\* 修改当前路径长度 \*/

(\*P)[w]=k;

}

}

}

}

int main(void)

{

int i,j,v0;

Mgraph g;

Patharc P;

ShortPathTable D;

v0=0;

CreateMgraph(&g);

ShortestPath\_Dijkstra(g, v0, &P, &D);

printf("最短路径倒序如下:\n");

for(i=1;i<g.numVertexes;++i)

{

printf("v%d - v%d : ",v0,i);

j=i;

while(P[j]!=0)

{

printf("%d ",P[j]);

j=P[j];

}

printf("\n");

}

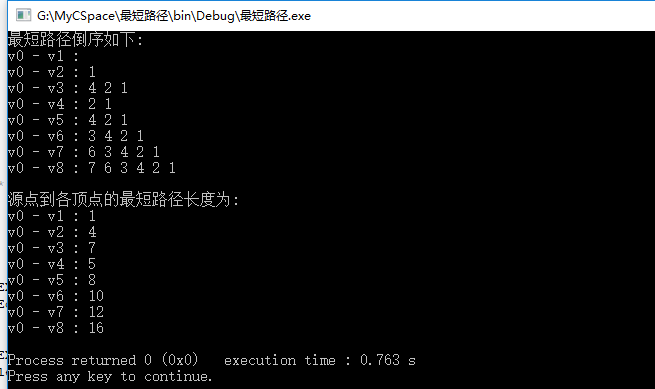
printf("\n源点到各顶点的最短路径长度为:\n");

for(i=1;i<g.numVertexes;++i)

printf("v%d - v%d : %d \n",g.vexs[0],g.vexs[i],D[i]);

return 0;

}**实验截图：**



**四、实验结果&总结：**

Dijkstra不能含有负权值，可以想象一下，如果有一条负边在图中。Dijkstra维护的优先队列，每次都是取一条最小的边出来的，而这条负边，可以使这条路无限小。

(min+g.arc[k][w]<(\*D)[w]) 这句判断的代码很重要，有时候经过两个点比直接到达所花销更小，这一点需要注意。