**算法分析与设计实验报告**

**最短路径求解算法的实现及其时间复杂度分析**

**作者: 赵彤彤（1120160311）**

指导老师: 蒋 波

实验日期: 2016.12

## 一、实验目的

(1) 编程实现求解最短路径的Dijkstra算法与Floyd算法，并采用事后分析方法，分析算法的时间复杂度。

(2) 采用文件方式输入数据，并采用有向图的邻接矩阵存储输入数据，图的顶点个数为1000个。Dijkstra算法的源点设定为第一个的顶点。

提示：可以采用伪随机函数生成数据，并以顺序文件的形式加以保存，以便反复测试用。约定：每个输入数据是不大于100的非负整数。假设aij是邻接矩阵中的数据元素，当i=j时，aij=0。当i≠j时，若aij大于100，则aij视为无穷大，并设置aij的值为106；若aij小于5，则设置aij的值为0。

(3) 对每组数据，首先对500个顶点的数据，运行Dijkstra算法和Floyd算法，并分别统计算法运行时间。然后，每增加20个顶点再计算一次并统计时间，直到计算到1000个顶点。这样，对每个算法而言，每次运算可得到26个值。

(4) 要求至少构造10组数据，并统计每组数据的运行时间，计算平均值。

(5) 将每个算法的运行时间平均值复制到EXCEL表中，构造运行时间散点图，给出运行时间散点图的模拟曲线与模拟方程。

(6) 根据实验结果，撰写实验报告。

## 二、实验内容

## 1、Dijstra算法思想

Dijstra：这个算法是通过为每个顶点*v*保留目前为止所找到的从s到v的最短路径来工作的。

把顶点集合V分成两组：

（1）S：已求出的顶点的集合（初始时只含有源点V0）。

（2）V-S=T：尚未确定的顶点集合。

将T中顶点按递增的次序加入到S中，保证：

（1）从源点V0到S中其他各顶点的长度都不大于从V0到T中任何顶点的最短路径长度。

（2）每个顶点对应一个距离值。

S中顶点：从V0到此顶点的长度。

T中顶点：从V0到此顶点的只包括S中顶点作中间顶点的最短路径长度。

依据：可以证明V0到T中顶点Vk的，或是从V0到Vk的直接路径的权值；或是从V0经S中顶点到Vk的路径权值之和。

### 2、Floyd算法思想

（1）从任意一条单边路径开始。所有两点之间的距离是边的权，如果两点之间没有边相连，则权为无穷大。

（2）对于每一对顶点 u 和 v，看看是否存在一个顶点 w 使得从 u 到 w 再到 v 比已知的路径更短。如果是更新它。

把图用邻接矩阵G表示出来，如果从Vi到Vj有路可达，则G[i,j]=d，d表示该路的长度；否则G[i,j]=无穷大。定义一个矩阵D用来记录所插入点的信息，D[i,j]表示从Vi到Vj需要经过的点，初始化D[i,j]=j。把各个顶点插入图中，比较插点后的距离与原来的距离，G[i,j] = min( G[i,j], G[i,k]+G[k,j] )，如果G[i,j]的值变小，则D[i,j]=k。在G中包含有两点之间最短道路的信息，而在D中则包含了最短通路径的信息。

## 三、实验数据

表4.2 D、F时间

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 测试次数 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| D时间(ms) | 0.295 | 0.29 | 0.32 | 0.23 | 0.25 | 0.24 | 0.2 | 0.28 | 0.3 | 0.3 |
| F时间(ms) | 2.3 | 2.25 | 2.2 | 2.29 | 2.3 | 2.29 | 2.25 | 2.3 | 2.3 | 2.392 |

## 四、程序代码（主要代码）

typedef struct

{

//char vexs[MAXV];//di

int edges[MAXV][MAXV];

int n; // 顶点个数

//int e; //边数

}MGraph; //表示邻接矩阵的数据类型

MGraph \* createmgraph(int n);

void writeToFile(int n,string fileName);

void readFromFile(MGraph \*g, string fileName);

void dijkstra(MGraph \*G,int v,int dist[],int path[]);

void floyd(MGraph \*G,int FDist[][MAXV], int FPath[][MAXV]);

int \_tmain(int argc, \_TCHAR\* argv[])

{

cout<<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<" \* 最短路径 \*"<<endl;

cout<<" \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"请输入顶点数:"<<endl;

while( ++status){

clock\_t startTime, endTime;//记录运行时间

const int source=0;//源点

int n=0;//顶点数

int DDist[MAXV],DPath[MAXV];//迪杰斯特拉

int FDist[MAXV][MAXV];

int FPath[MAXV][MAXV];

if(status>1)

cout<<"请输入顶点数:"<<endl;

cin>>n;

MGraph \* g= createmgraph(n );

writeToFile(g->n,"Graph.txt");

readFromFile(g,"Graph.txt");

//Dijkstra

cout << "\nDijkstra 最短路径:" << endl;

startTime = clock();

for(int i = 0;i<1000;i++)

{

dijkstra(g,source,DDist,DPath);

}

endTime = clock();

//cout << "startTime="<<startTime << ",endTime= " << endTime << endl;

cout << "Dijkstra "<<"顶点："<<g->n<<"个, 用时: " << (double)(endTime - startTime)/1000<<"秒！"<< endl;

//cout << "path is:" << endl;

//void disDPath(MGraph \*G,int dist[],int path[],int s[],int v);

//printfDPath(DPath, DDist, end);

startTime = clock();

for(int i = 0;i<1000;i++)

{

floyd(g,FDist,FPath);

}

endTime = clock();

//cout << "startTime="<<startTime << ",endTime= " << endTime << endl;

cout << "\nFloyd 最短路径:" << endl;

cout << "Floyd "<<"顶点："<<g->n<<"个, 用时: " << (double)(endTime - startTime)/1000<<"秒！\n"<< endl;

}

system("pause");

return 0;

}

MGraph \* createmgraph(int n) //建立有向图的邻接矩阵

{

MGraph \*g =(MGraph\*)malloc(sizeof(MGraph));

;

int i,j;

g->n=n;//顶点个数

for(i=0;i<g->n;i++) //建立图的邻接矩阵

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

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

return g;

}

void dijkstra(MGraph \*G,int v,int dist[],int path[])

{

int set[MAXV];

int min, i, j, u;

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

{

dist[i] = G->edges[v][i];

set[i] = 0;

if (G->edges[v][i] < INF)

path[i] = v;

else

path[i] = -1;

}

set[v] = 1;

path[v] = -1;

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

{

min = INF;

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

if (set[j] == 0 && dist[j] < min)

{

u = j;

min = dist[j];

}

set[u] = 1;

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

{

if (set[j] == 0 && dist[u] + G->edges[u][j] < dist[j])

{

dist[j] = dist[u] + G->edges[u][j];

path[j] = u;

}

}

}

//cout << endl << "Dijkstra complete" << endl;

}

void floyd(MGraph \*G,int FDist[][MAXV], int FPath[][MAXV])

{

int i, j, k;

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

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

{

FDist[i][j] = G->edges[i][j];

FPath[i][j] = -1;

}

for (k = 0; k < G->n; ++k)

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

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

if (FDist[i][j]>FDist[i][k] + FDist[k][j])

{

FDist[i][j] = FDist[i][k] + FDist[k][j];

FPath[i][j] = k;

}

//cout << "Floyd complete" << endl;

}

## 五、实验结果及其分析

### 1、程序结果输出图:

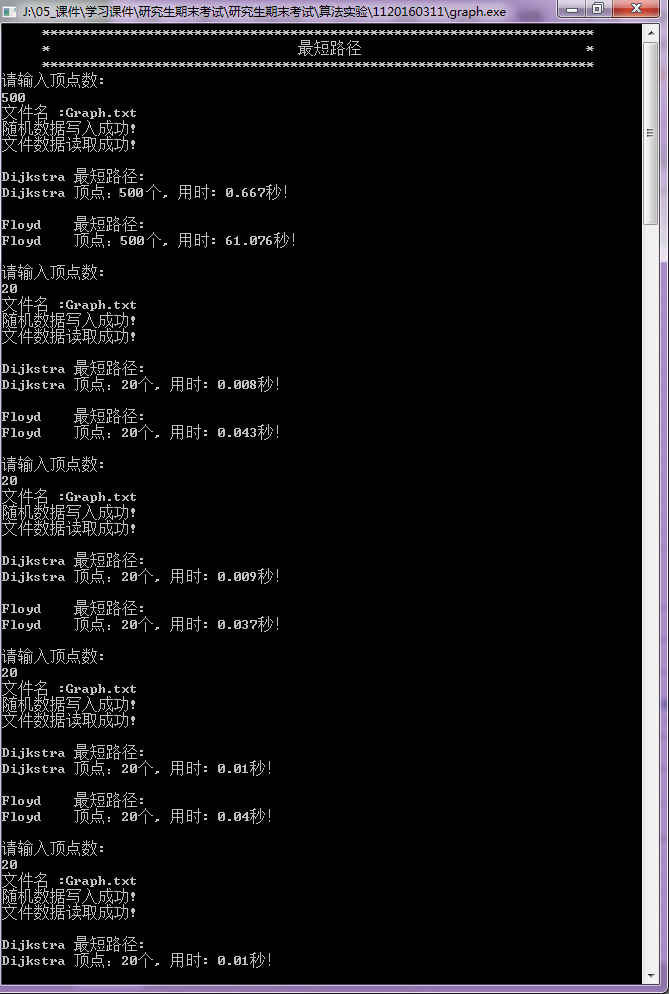


图 5.1 程序结果输出图

### 2、Dijkstra算法运行时间散点图：

图 5.2 Dijkstra算法运行时间散点图

### 3.Floyd算法运行时间散点图：

图 5.3 Floyd算法运行时间散点图

### 4实验结果分析

首先从理论分析，Dijkstra的时间复杂度为O（n2），Floyd的时间复杂度为O（n3）；再根据实验的结果我们也可以证明理论的正确性。

通过此次的实验，让我们更加深刻的理解图的最短路径问题，也更加了解Dijkstra和Floyd算法；同时也加强了我们的编程能力。