**算法设计与分析实验报告**

**指导老师: 曲衍鹏**

**姓 名: 汪广鑫**

**学 号:1120200298**

**实验日期:2020.12.21**

**最短路径求解算法的实现及其时间复杂度分析**

**一、实验目的**

1、了解最短路径求解问题并进行算法实现。

2、熟悉并掌握Dijkstra算法与Floyd算法的思想。

3、分析Dijkstra算法与Floyd算法的优劣性。

**二、实验内容**

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(作时空分布图并做曲线拟合)，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。

**三、实验原理**

Dijkstra算法描述：

a) 初始化 G={V,E}

令 S={V0},T=V-S={其余顶点}，T中顶点对应的距离值

若存在<V0,Vi>，d(V0,Vi)为<V0,Vi>弧上的权值

若不存在<V0,Vi>，d(V0,Vi)为∞

b) 从T中选取一个与S中顶点有关联边且权值最小的顶点W，加入到S中

c) 对其余T中顶点的距离值进行修改：若加进W作中间顶点，从V0到Vi的距离值缩 短，则修改此距离值

d) 重复上述步骤2、3，直到S中包含所有顶点，即W=Vi为止

Floyd算法描述：

a)　初始化：D[u,v]=A[u,v]

b)　For k:=1 to n

For i:=1 to n

For j:=1 to n

If D[i,j]>D[i,k]+D[k,j] Then

D[i,j]:=D[i,k]+D[k,j];

c)　算法结束：D即为所有点对的最短路径矩阵

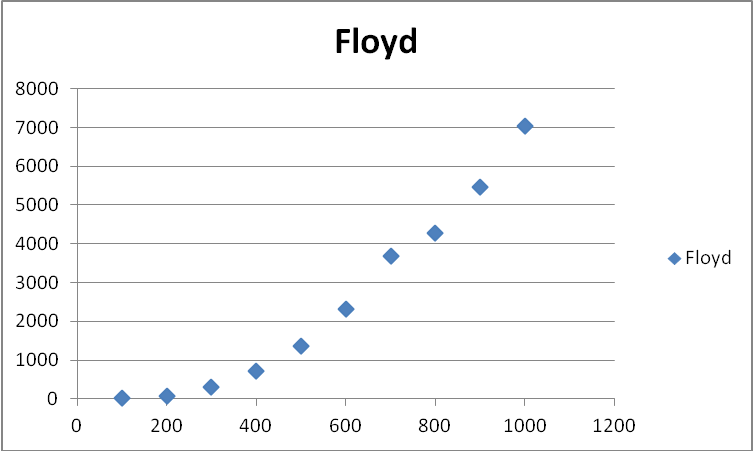
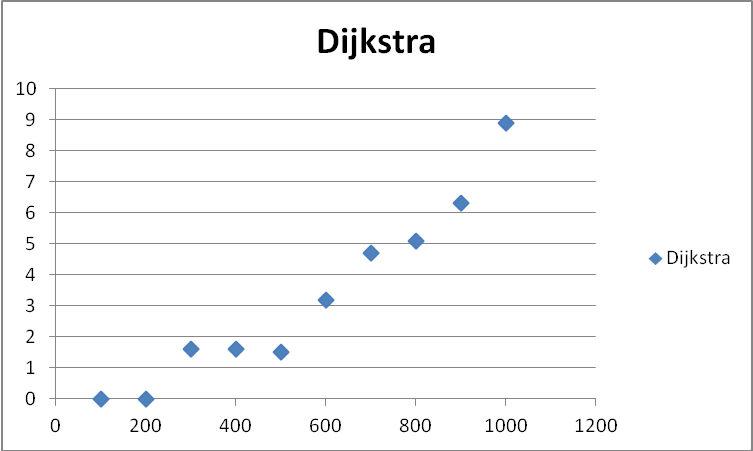
**四、实验数据**

|  |  |  |
| --- | --- | --- |
| 点数 | Dijkstra | Floyd |
| 100 | 0 | 16 |
| 200 | 0 | 79 |
| 300 | 1.6 | 312 |
| 400 | 1.6 | 718 |
| 500 | 1.5 | 1359 |
| 600 | 3.2 | 2312 |
| 700 | 4.7 | 3688 |
| 800 | 5.1 | 4287 |
| 900 | 6.3 | 5473 |
| 1000 | 8.9 | 7032 |

**五、实验代码**

附后。

**六、实验结果**



**七、实验总结**

通过本次实验，了解了有关最短路径求解的问题。Dijkstra算法求解单源点最短路径和Floyd算法求解多源点最短路径问题。Floyd算法适用于顶点数较少且给出的图比较稠密，因为在运行时间上要优于Dijkstra算法。

**主要代码**：

#include <iostream.h>

#include <fstream>

#include <stdio.h>

#include <time.h>

#include <windows.h>

using namespace std;

long getTime()

{

SYSTEMTIME sys;

GetLocalTime(&sys); //获取当前系统时间（可以精确到毫秒）

//printf("%d年%d月%d日 %d:%d:%d.%d",sys.wYear,sys.wMonth,sys.wDay,sys.wHour,sys.wMinute,sys.wSecond,sys,wMilliseconds);

return sys.wMinute\*60000+sys.wSecond\*1000+sys.wMilliseconds;

}

void main()

{

int i,j,k;

const int MAXV = 100000;

char \*\*text; //将文件夹中所有的文本文件存入此数组中

text=(char \*\*)malloc(sizeof(int \*)\*15);//文件夹中一共有15个文本文件

for (i=0;i<10;i++)

{

text[i]=(char \*)malloc(sizeof(int)\*12);

char temp[6];

itoa((i+1)\*100, temp, 10);

string name(temp);

name+=".txt";

for(j=0;j<name.length();j++)

text[i][j] = name[i];

text[name.length()] = '\0';

}

for (i=0;i<10;i++)

text[i]=(char \*)malloc(sizeof(int)\*12);//每个文本文件的名称的长度设置为12

text[0]="100.txt";

text[1]="200.txt";

text[2]="300.txt";

text[3]="400.txt";

text[4]="500.txt";

text[5]="600.txt";

text[6]="700.txt";

text[7]="800.txt";

text[8]="900.txt";

text[9]="1000.txt";

text[10]="1100.txt";

text[11]="1200.txt";

text[12]="1300.txt";

text[13]="1400.txt";

text[14]="1500.txt";

//建立exele表格文件存放查找的时间

ofstream o\_xlsfile;

o\_xlsfile.open("result.xls");

o\_xlsfile<<"点数\t"<<"Foly\t"<<"DJS\t\n";

for(int count=0;count<15;count++) //一共有15个文件所以COUNT<15

{

int dot=(count+1)\*100;//一共有307个点

o\_xlsfile<<dot<<"\t"; //将点数写入EXELE表格中

int \*\*matric; //存放图中各边权值的数组

matric=(int \*\*)malloc(sizeof(int \*)\*dot);

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

matric[i]=(int \*)malloc(sizeof(int)\*dot);

//从文本文件中读取数据

ifstream file;

file.open(text[count]);

if(file.is\_open())

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

for (int j=0;j<dot;++j)

{

file>>matric[i][j];

}

file.close();

int \*\*patch; //路径存放数组

patch=(int \*\*)malloc(sizeof(int \*)\*dot);

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

patch[i]=(int \*)malloc(sizeof(int)\*dot);

int \*\*a; //临时数组

a=(int \*\*)malloc(sizeof(int \*)\*dot);

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

a[i]=(int \*)malloc(sizeof(int)\*dot);

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

{

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

{

a[i][j] = matric[i][j];

patch[i][j] = j;

}

}

long starttime=getTime(); //Floyd开始时间

//FLOYD算法

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

{

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

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

{

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

{

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

patch[i][j] = patch[i][k];

}

}

}

long time=getTime()-starttime; //Floyd查找时间

cout<<(count+1)\*100<<"个点的floyd查找时间为"<<" "<<time<<"毫秒"<<endl;

o\_xlsfile<<time<<"\t";

//DJS算法

int DJSbegin=5; //查找5号结点到其它各点的路径

bool \*final;

final=(bool \*)malloc(sizeof(bool)\*(dot\*dot));

int \*D; //临时数组

D=(int \*)malloc(sizeof(int)\*dot);

//初始化

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

{

final[i]=false;

D[i]=matric[DJSbegin][i];

for (int w=0;w<dot;++w)

patch[i][w]=-1;

if(D[i]<MAXV)

{

patch[i][0]=DJSbegin;

patch[i][1]=i;

}

}

D[DJSbegin]=0;

final[DJSbegin]=true;

int min;

float sumtime=0;

for(int si=0;si<10;si++)

{

starttime=getTime(); //开始查找

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

{

min=MAXV;

for(int w=0;w<dot;++w)

if(!final[w])

if(D[w]<min)

{

i=w;

min=D[w];

}//if

final[i]=true;

for(w=0;w<dot;++w) //更新路径

if(!final[w]&&(min+matric[i][w]<D[w]))

{

D[w]=min+matric[i][w];

for (int s=0;s<dot;++s)

{

patch[w][s]=-1;

}

for (s=0;s<dot;++s)

{

if(patch[i][s]!=-1)

patch[w][s]=patch[i][s];

else

{

patch[w][s]=w;

break;

}

}

}//if

}//min-for

time=getTime()-starttime;

sumtime+=time;

}

cout<<(count+1)\*100<<"个点的DJS查找时间为 "<<sumtime/10<<"毫秒"<<endl;

cout<<endl;

o\_xlsfile<<sumtime/10<<"\n"; //将DJS查找时间写入EXELE表格中

}

o\_xlsfile.close();//关闭表格文件

}