# Floyd算法

班级：2015211306 姓名：李航 学号：2015211284 分工：代码+文档

班级：2015211306 姓名：魏晓 学号：2015211301 分工：代码+文档

## 需求分析

一个以邻接表形式存储的有向图，求出其中路径的权值最大的两个结点。首先应考虑到我们需要用到Floyd算法，我们需要先求出任意两个结点之间的最短路径，然后取出这些最短路径的最大值，这个最大值所联系的两个结点就是我们所要求的结点。首先输入整个图（邻接表方式），转换成矩阵存储，利用Floyd算法求出所有最短路径，取出所有最短路径的最大值。测试的图如下：

6

4

11

3 2

正确的结果应该是c->b，c到b的距离最远，为7，其余的输出结果都是错误的

## 概要设计

typedef struct ArcNode{ //弧信息

int adjvex; //该弧所指向的顶点位置

struct ArcNode \*nextare;//指向下一条弧的指针

int weight; //该弧的权值

}ArcNode;

typedef struct VNode{

char vertex; //顶点信息，用一个字符表示一个顶点

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

}VNode;

这是用邻接表存储该图的结构，顶点结构和弧的结构；

typedef struct{

int arcs[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]; //邻接矩阵

int vexnum; //当前顶点数目

}Arccell;

这是邻接矩阵的存储结构

主要的实现函数为：

main

{

VNode AdjList[MAX\_VERTEX\_NUM]; //定义一个邻接表

Arccell Matrix; //定义一个邻接矩阵

init(AdjList,Matrix); //初始化邻接表

ListToMatrix(AdjList,Matrix); //用邻接矩阵存储该图

ShortestPath\_FLOYD(Matrix); //Floyd算法求出最短路径

FarthestVertex(AdjList,Matrix); //求出最远距离并输出

}

## 详细设计

下面给出重要的四个函数的详细算法：

void init(VNode AdjList[],Arccell &Matrix) //初始化邻接表

{

int i=0;

char ch;

cout<<"请输入顶点个数：";

cin>>Matrix.vexnum;

cout<<"请输入所有的顶点（用字母表示,'#'代表输入结束）：";

cin>>ch;

for(;ch!='#';++i)

{

AdjList[i].vertex=ch;

AdjList[i].firstarc=NULL;

cin>>ch;

}

if(i!=Matrix.vexnum)

cout<<"输入错误"<<endl;

else

{

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

cout<<i<<"."<<AdjList[i].vertex<<" ";

cout<<endl;

ArcNode \*last;

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

{

cout<<"顶点"<<AdjList[i].vertex<<"的邻接点位置和路径权值（-1代表输入结束）：";

int num;

cin>>num;

while(num!=-1)

{

ArcNode \*p=(ArcNode\*)malloc(sizeof(ArcNode));

if(AdjList[i].firstarc==NULL)

AdjList[i].firstarc=p;

else

last->nextare=p;

last=p;

p->nextare=NULL;

p->adjvex=num;

cin>>num;

p->weight=num;

cin>>num;

}

}

cout<<"初始化结束"<<endl;

}

}

void ListToMatrix(VNode AdjList[],Arccell &Matrix) //该函数将邻接表存储的图转化为邻接矩阵存储

{

int i,j;

for(i=0;i<Matrix.vexnum;++i) //初始化矩阵

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

if(i==j)

Matrix.arcs[i][j]=0;

else

Matrix.arcs[i][j]=MAX\_NUM;

for(i=0;i<Matrix.vexnum;++i) //邻接表转矩阵

{

ArcNode \*p=AdjList[i].firstarc;

for(;p;p=p->nextare)

Matrix.arcs[i][p->adjvex]=p->weight;

}

cout<<"邻接表转矩阵后的矩阵为：\n";

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

{

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

cout<<Matrix.arcs[i][j]<<" ";

cout<<endl;

}

}

void ShortestPath\_FLOYD(Arccell &Matrix) //FLOYD算法求出最短路径

{

int u,v,w;

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

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

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

if(Matrix.arcs[v][u]+Matrix.arcs[u][w]<Matrix.arcs[v][w])

Matrix.arcs[v][w]=Matrix.arcs[v][u]+Matrix.arcs[u][w];

}

void FarthestVertex(VNode AdjList[],Arccell Matrix) //求出最远顶点，并且输出程序结果

{

int i,j,m,n;

int MAX=0;

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

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

{

if(Matrix.arcs[i][j]>MAX)

{

MAX=Matrix.arcs[i][j];

m=i;

n=j;

}

}

cout<<"距离最远的两个结点为: "<<AdjList[m].vertex<<" "<<AdjList[n].vertex<<endl;

cout<<"距离为"<<MAX<<endl;

}

for(i=0;i<Matrix.vexnum;++i) //看最远距离是否唯一

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

{

if(Matrix.arcs[i][j]==MAX&&(i!=m||j!=n))

{

cout<<"距离最远的两个结点为: "<<AdjList[m].vertex<<" "<<AdjList[n].vertex<<endl;

cout<<"距离为"<<MAX<<endl;

}

}

这里定义了两个重要的符号常量：

#define MAX\_VERTEX\_NUM 20 //所允许的最大顶点数

#define MAX\_NUM 1000 //理论上的无穷大，这里的所有权值都远小于这个值

## 调试分析报告

首先我觉得还是在读取字符和数字时比较麻烦，只能黑框输入各个点的逻辑关系。其实算法的设计比较简单。除了Floyd算法的时间复杂度为O（n^3），其余函数时间复杂度都为O（n^2），所以总的时间复杂度为O（n^3）等于Floyd算法的时间复杂度。

## 用户使用说明

在输入某一顶点的邻接点位置和该路径权值的时候，首先输入邻接点位置，该位置的序号程序会给出，然后输入权值，中间空格分开；如果不止一个邻接点，那么接着输入另一个邻接点位置和权值，与上一个邻接点和权值用空格分开，最后输入-1，再换行，表示输入结束。如果没有邻接点（出度为0），那么直接输入-1然后换行表示输入结束。

在输入顶点时，最后输入”#”代表输入结束。其余的程序会给出充分的提示，用户友好度较好。

## 测试结果

输入的图在需求分析中已经给出，如下：

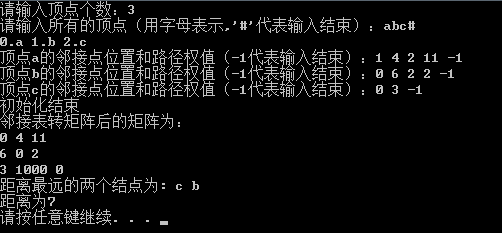
6

4

11

3 2

整个程序的运行程序如下：



程序的输出结果与预期的结果相符，程序运行正确。