**第七章 图 实验报告**

班级：2018211310 姓名：刘立敏 学号：2018211398 分工：程序编写及报告撰写者

班级：2018211310 姓名：余子劲 学号：2018211403 分工：实验讨论与算法优化改进

班级：2018211310 姓名：庞可涵 学号：2018211392 分工：实验讨论与算法优化改进

问题：

已知某图是边带权(权值为正数)的有向无环图,采用邻接表存储,求出图中距离最远的两个结点。

**1.需求分析**

题目要求对一个给定的正权值有向无环图，求出图中距离最远的两个结点。输入的是这张图的所有信息，包括顶点数，边数以及顶点和边的信息，输出的是所有顶点中距离最远的两个，以及它们之间的距离。

由于题目中只说明了是“距离最远”，此处可以有两种理解。一是按“**利用最短路径算法求出所有顶点间的最短距离，然后找出这些距离中的最大值**”理解；二是按“**利用最长路径算法求出所有顶点间的最长距离，然后找出这些距离中的最大值**”理解。因此，我们将这个实验分成两部分，分别按照两种理解来设计算法。

我们事先设定了允许最多输入的顶点数（MAXVEX）和边数（MAXEDGE）分别为200和10000，如果大于这些值，程序将终止并输出“顶点数（边数）过大，程序终止”作为提示，此时可以修改MAXVEX和MAXEDGE以满足要求。

**2.概要设计**

思路概述：①最短路径最长：将图中的每个顶点作为源点，进行一次dijkstra算法，求出顶点之间的全部距离，这些距离中最大值就是目标。

②最长路径最长：最长路径又称关键路径，可以利用拓扑排序的思想，每访问到一个顶点，就将当前顶点入队列，并调整它所有的后继节点，使得路径的长度尽可能的长。最终可以把所有顶点间的最远距离求出来，并找出其中的最大值。

题目明确要求了采用邻接表的存储结构。需要说明的是，我们的程序中虽然使用了距离矩阵（类似邻接矩阵），但这只是为了将所有顶点之间的距离保存起来，方便后续程序添加一些新的功能。事实上，算法本体上完全是只用了邻接表的存储结构。

我们存储图的结构定义如下（存放在struct.h中）

/\*一些声明\*/

typedef int Vertextype; //顶点数据类型，默认为int

typedef int Edgetype; //边权值数据类型，默认为int

const int MAXVEX = 200; //最大顶点数

const int MAXEDGE = 10000; //最大边数

const int INF = 0x3f3f3f3f; //无穷大的值，表示无路径

Edgetype M[MAXVEX][MAXVEX]; //保存最短路径长度的矩阵（全局变量）

typedef struct edgenode //边表

{

Edgetype weight;

int adjvex;

struct edgenode\* nextedge;

}Edgenode, \* Edge;

typedef struct //顶点结构

{

Vertextype data;

Edge firstedge;

}Vertex, Adjlist;

typedef struct //图结构

{

int numvex, numedge;

Adjlist adjlist[MAXVEX];

}graph, \* Graph;

用到的函数有

Graph CreateGraph(); //建立图

void traceback(Graph G, int n, int\* P); //打印路径

void Save2Matrix(Graph G, Edgetype \* D, int sourse); //保存路径长度至矩阵M中

void Print\_Matrix(Graph G); //打印距离矩阵

void Dijkstra(Graph G, int n); //求出源点n到达各顶点的最短路径

void Find\_Max\_Shortest\_Path(Graph G); //从矩阵M中找到最短路径长度最大的两个顶点

void KeyPath(Graph G, int sourse); //求出源点n到达各顶点的关键路径 (最长路径）

void Find\_Max\_Longest\_Path(Graph G); //从矩阵M中找到最长路径长度最大的两个顶点

**3.详细设计**

**包含头文件struct.h、Print.h、Longest.h、Shortest.h以及源文件main.cpp**

①输入图的数据，创建图（存放在Print.h中）

输入格式样例如下：

3 3 //第一行是顶点数v和边数e，这里表示三个顶点和三条边

c b a //第二行是v个顶点的信息，这里表示顶点1,2,3的数据依次为c,b,a

1 2 5 //接下来e行中，每行的三个数依次是边的起点，终点和权值

1 3 4

2 3 7

Graph CreateGraph() //建立图

{

Graph G = (Graph)malloc(sizeof(graph));

if (G == NULL)

{

cout << "内存不足" << endl;

exit(1);

}

cout << "请输入图的顶点数和边数" << endl;

cin >> G->numvex >> G->numedge;

if(G->numvex>=MAXVEX)

{

cout<<"顶点数过大，程序终止"<<endl;

exit(1);

}

if(G->numedge>=MAXEDGE)

{

cout<<"边数过大，程序终止"<<endl;

exit(1);

}

cout << "请依次输入每个顶点的数据，用空格间隔" << endl;

for (int i = 1; i <= G->numvex; i++)

{

cin >> G->adjlist[i].data;

G->adjlist[i].firstedge = NULL;

}

int u, v;

Edgetype w;

cout << "请依次输入所有的边的起点，终点，权值" << endl;

for (int i = 1; i <= G->numedge; i++)

{

Edge e = (Edge)malloc(sizeof(edgenode));

if (e == NULL)

{

cout << "内存不足" << endl;

exit(1);

}

cin >> u >> v >> w;

e->weight = w;

e->nextedge = G->adjlist[u].firstedge;

e->adjvex = v;

G->adjlist[u].firstedge = e;

}

return G;

}

②关键算法

**第一部分：最短路径（Dijkstra）（存放在Shortest.h中）**

void Dijkstra(Graph G, int n) //求出源点n到达各顶点的最短路径

{

bool S[MAXVEX]; //标记是否已经确定了最短路径，true表示已确定

Edgetype D[MAXVEX]; //存储当前最短路径长度

int P[MAXVEX]; //存储当前访问到的顶点在当前最短路径上的前驱

for (int i = 1; i <= G->numvex; i++)

{

S[i] = false;

D[i] = INF;

P[i] = -1;

}

D[n] = 0;

int tmp = n; //tmp用于保存源点，以便后面恢复

while (!S[n])

{

S[n] = true;

Edge e = G->adjlist[n].firstedge;

while (e) //更新到达当前所有可达顶点的最短路径

{

if (D[n] + e->weight < D[e->adjvex])

{

D[e->adjvex] = D[n] + e->weight;

P[e->adjvex] = n;

}

e = e->nextedge;

}

Edgetype nowmin = INF;

int now = 1;

for (int i = 1; i <= G->numvex; i++) //找出D[i]中的最小值，存到nowmin中， 并将i存到now中

if (nowmin > D[i] && S[i] != 1)

{

nowmin = D[i];

now = i;

}

n = now; //n改为D[i]最小的那个顶点，继续下次循环

}

n = tmp;

Save2Matrix(G, D, n); //保存从源点n出发到达各顶点的最短路径长度

}

**输出距离（最短路径）最远的两个顶点（存放在Shortest.h中）**

void Find\_Max\_Shortest\_Path(Graph G) //从矩阵M中找到最短路径长度最大的两个顶点

{

Edgetype maxl = -INF; //用maxl保存所有距离中的最大值

int x = 1, y = 1;

for (int i = 1; i <= G->numvex; i++) //遍历距离矩阵，更新maxl

for (int j = 1; j <= G->numvex; j++)

if (M[i][j] != INF && maxl < M[i][j])

//if (!(fabs(M[i][j] - INF) <= 1e-6) && maxl < M[i][j]) //如果边权值是浮点类型

{

maxl = M[i][j];

x = i; y = j;

}

cout << "图中距离最远的两个顶点为" << x << '(' << G->adjlist[x].data << ')' << "和";

cout << y << '(' << G->adjlist[y].data << ')' << ",路径长度为" << maxl << endl<<endl;

}

**第二部分：最长路径（Keypath）（存放在Longest.h中）**

void KeyPath(Graph G, int sourse) //求出源点sourse到达各顶点的关键路径 (最长路径）

{

Edgetype D[MAXVEX]; //存储当前最长路径的长度

queue<int>Q; //队列Q声明

for (int i = 1; i <= G->numvex; i++)

D[i] = INF;

int now = sourse;

D[now] = 0;

Q.push(now);

while (!Q.empty()) //将顶点依次入队列出队列直到队列为空

{

Q.pop();

Edge e = G->adjlist[now].firstedge;

while (e) //更新所有后继顶点的最长路径长度，使之尽可能大

{

Q.push(e->adjvex);

if (D[e->adjvex] == INF) //原本没有路径，现在有了

D[e->adjvex] = D[now] + e->weight;

if (D[e->adjvex] < D[now] + e->weight)

D[e->adjvex] = D[now] + e->weight;

e = e->nextedge;

}

now = Q.front(); //下个循环更新当前队首顶点的所有后继

}

Save2Matrix(G,D,sourse); //存到矩阵M中

}

**输出距离（最长路径）最远的两个顶点（存放在Longest.h中）**

void Find\_Max\_Longest\_Path(Graph G) //从矩阵M中找到最长路径长度最大的两个顶点

{

Edgetype maxl = 0;

int x = 1, y = 1;

for (int i = 1; i <= G->numvex; i++)

for (int j = 1; j <= G->numvex; j++)

if (M[i][j] != INF && M[i][j] > maxl)

{

maxl = M[i][j];

x = i; y = j;

}

cout << "图中距离最远的两个顶点为" << x << '(' << G->adjlist[x].data << ')' << "和";

cout << y << '(' << G->adjlist[y].data << ')' << ",路径长度为" << maxl << endl<<endl;

}

**③其他的一些用来输出的函数（非题目要求的输出）（存放在Print.h中）**

void traceback(Graph G, int n, int\* P) //打印最短路径

{

if (P[n] != -1)

traceback(G, P[n], P);

cout << n << '(' << G->adjlist[n].data << ')' << ' ';

}

void Save2Matrix(Graph G, Edgetype \* D, int sourse) //保存路径长度至矩阵M中

{

for (int i = 1; i <= G->numvex; i++)

M[sourse][i] = D[i];

}

void Print\_Matrix(Graph G) //打印距离矩阵

{

for (int i = 1; i <= G->numvex; i++)

{

for (int j = 1; j <= G->numvex; j++)

if (M[i][j] != INF)

cout << M[i][j] << ' ';

else cout << "∞ ";

cout << endl;

}

}

**4.调试分析报告**

①实验遇到的问题以及解决、改进方案：

由于一开始我只做了最短路径最长的部分，所以先入为主地想通过修改dijkstra算法或Froyd算法来实现最长路径最长的部分

但是由于dijkstra算法每次确定了一个顶点的路径后，之后便不再修改该顶点的最长路径长度，可是事实上后面可能会有一条更长的路径到达该顶点，所以要保证算法正确就必须得修改该顶点的最长路径长度。因此最长路径问题是不能用dijkstra算法求解的。

然后考虑了froyd算法，但是这道题不能用邻接矩阵来存储图，如果需要用链式存储的话，将需要用到十字链表，所以也放弃了这个想法。

最后，换了个思路，意识到最长路径就是关键路径，因此它实际上就相当于课件上的求事件发生的最晚时间问题，即利用了拓扑排序的思想。至此，问题就得以解决。

②算法的时空分析

**时间复杂度**：

Dijkstra算法：T（n）=O（n²）

Keypath函数：T（n）=O（n²）

由于是对每个顶点执行一次算法，所以无论是求最短路径的dijkstra算法还是求最长路径的Keypath函数，总的时间复杂度均为O(n³)

另外，Print\_Matrix函数的时间复杂度T（n）=O（n²）

Save2Matrix函数的时间复杂度T（n）=O（n）

traceback函数的时间复杂度T（n）=O（n）

综上，程序最主要的部分在于调用n次Dijkstra函数或者Keypath函数

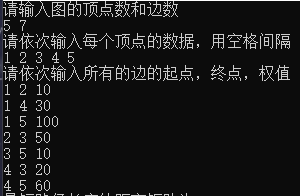
因此，最终时间复杂度为为O(n³)

**空间复杂度：**

存储图用的邻接表的空间复杂度为 S(n)= O（n²）

**5.用户使用说明**

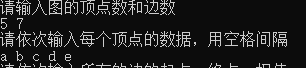
输入情况如图所示

****

输出采用下面的方式

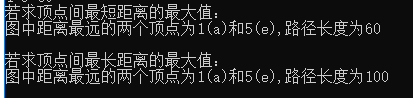
对于每个顶点，我们先输出顶点的编号值，再在括号里面体现顶点的实际数据

例如：假如顶点类型为字符型

****

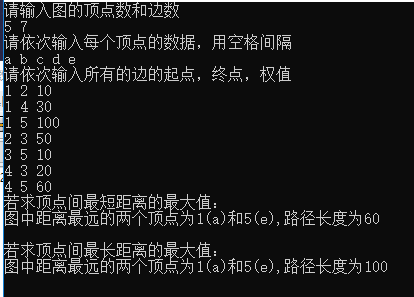
输入如上 代表顶点1 2 3 4 5的数据依次为a b c d e

那么输出的结果如下所示

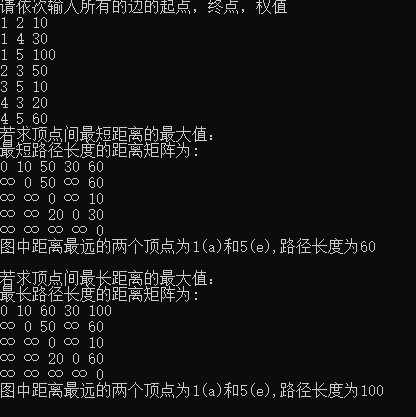
****

**6.测试结果**

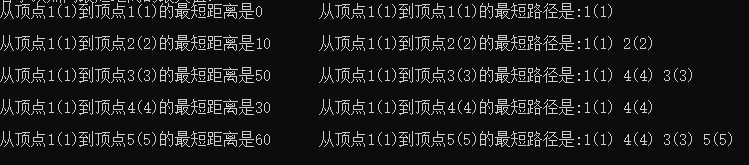
测试数据1：

****

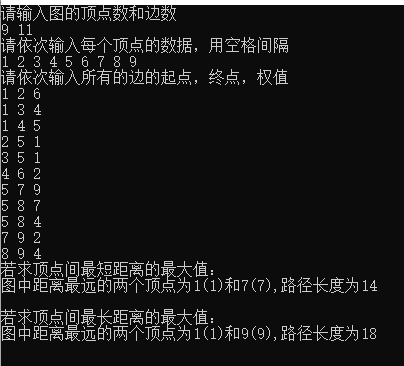
另，如果要打印距离矩阵，结果如下：

****

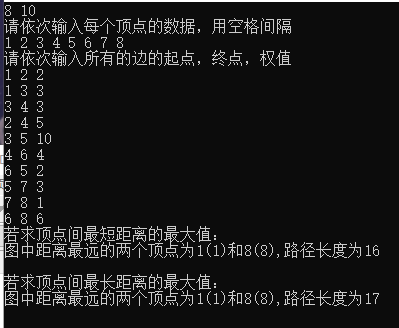
如果要打印最短路径，结果示意如下

****

测试数据2：



测试数据3：



测试数据4：

