**数据结构实验报告**

**实验题目：图中最短距离查找**

|  |  |  |  |
| --- | --- | --- | --- |
| 班级：2020211305 | 姓名：缪奇志 | 学号：2020211345 | 分工：样例测试 |
| 班级：2020211305 | 姓名：倪玮昊 | 学号：2020211346 | 分工：思路分析 |
| 班级：2020211305 | 姓名：杨智杰 | 学号：2020211356 | 分工：程序编写 |

1. **需求分析**

题目任务：

某图是边带权(权值为正数)的有向无环图,要求采用邻接表的数据结构完成该图的创建和存储,并求出图中每一对的最短距离中最远的两个结点。

程序的输入和输出：

输入：首先输入顶点的总数目；在确定顶点的总数目之后，依次输入每一条边，按照（起点，终点，权重）来输入。最终成功构建图并完成储存。

输出：

得到在每队顶点的最短距离之后，输出这些最短距离中最远的路径的起点，终点，权重。

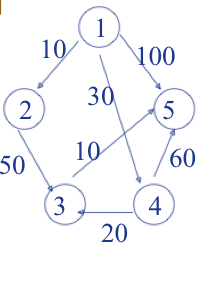
程序功能实现分为3个模块：

1. 模块 1：输入顶点数目和每条边的完成图的创建和存储。 利用创建好的两种不同的数据结构和循环结构实现图的创建和存储同时还要对用户的非法输入进行判断。
2. 模块2： 根据构建好的数据结构，求出每一对之间的最短距离，在得到这些最短距离中最远的一对。利用两个两个函数的调用和双循环嵌套的算法遍历求出每队顶点之间的最短距离并比较得出最远距离，保存这一个最远距离的起点和终点和距离。
3. 模块3：输出之前求出的最短距离中最远距离的始点和终点。利用之前的存储结构可以直接输出对应的结果。

正确数据输出结果范例：

黑板上的字

描述已自动生成



文本

描述已自动生成

错误数据输出结果范例：

****

**文本

描述已自动生成**

1. **概要设计**

**算法思路概述：**

本程序主要由三个主要的功能模块组成。

1. 模块一：利用邻接表数据结构完成图的创建 思路：利用创建好的两种不同的数据结构和循环结构实现图的创建和存储同时还要对用户的非法输入进行判断。
2. 模块二： 根据构建好的数据结构，求出每一对之间的最短距离，在得到这些最短距离中最远的一对。 思路：利用两个两个函数的调用和双循环嵌套的算法遍历求出每一对顶点之间的最短距离并比较得出最远距离，保存这一个最远距离的起点和终点和距离。
3. 模块3：输出之前求出的最短距离中最远距离的顶点和终点。利用之前的存储结构可以直接输出对应的结果。

**数据结构类型的定义：**

定义结构体Vertex用来表示一个顶点。

struct Vertex

{

int data;

Edge\* firstedge;

};

定义结构体Edge用来作为边表节点

struct Vertex

{

int data;

Edge\* firstedge;

};

定义全局变量ShortestPath[VertexSize + 1][VertexSize + 1]用来存储最短路径长度，0为未寻找/自环路径为0，-1为找不到（无穷大）。

**主程序流程：**

**int main(void)**

**{**

**Vertex\* V = NULL;**

**int VertexNum;**

**if (!StructGraph(V, VertexNum))**

**return 0;**

**long long MaximumShoterestLength = GetMaximumShortestPath(V, VertexNum);**

**Output(V, VertexNum, MaximumShoterestLength);**

**return 0;**

**}**

**各层次模块层次调用关系：**

main先调用了**StructGraph**构建了无环有向图结构并对其进行存储，接着调用**GetMaximumShortestPath**函数遍历找到最短路径的最远距离，其中**GetMaximumShortestPath**函数调用了**DFSShortestPath**函数用于找到特定起点和终点的距离。最后调用**Output**函数实现对查找结果的输出，程序结束。

1. **详细设计**

各种函数的设计：

1.完成有向无环图的构建与存储：

static bool StructGraph(Vertex\*& V, int& VertexNum)

{

printf("请输入有向无环图顶点个数：");

scanf("%d", &VertexNum);

while VertexNum < 0 do

{

printf("输入非法！请重新输入：");

scanf("%d", &VertexNum);

}

V (Vertex\*)malloc(sizeof(Vertex) \* (VertexNum + 1));

if !v==1 then

{

printf("顶点个数过多，分配空间不足！\n");

return false;

}

for int i 1 to VertexNum do

V[i].firstedge NULL;

i++;

printf("请依次输入每条边：\n1.点集中各个点请命名为1到%d\n2.边的格式为“始点 终点 权重”\n3.以“0 0 0”作为结束输入\n输入开始！\n", VertexNum);

int st, ed;

long long weight;

scanf("%d %d %lld", &st, &ed, &weight);

while st || ed || weight==1 do

if (1 <= st && st <= VertexNum && 1 <= ed && ed <= VertexNum && weight > 0) then

Edge\* edge = (Edge\*)malloc(sizeof(Edge));

if (!edge) then

printf("边数过多，分配空间不足！\n");

return false;

edge->adjvex ed;

edge->weight weight;

edge->nextedge V[st].firstedge;

V[st].firstedge edge;

else then

printf("输入的边非法！请重新输入（已输入的正确边仍保存）：\n");

scanf("%d %d %lld", &st, &ed, &weight);

end while

return true;

2.在每一对最短距离中找到最远距离：

static long long GetMaximumShortestPath(Vertex\* V, int VertexNum)

{

memset(ShortestPath, 0, sizeof(ShortestPath));

long long len -1, cmp;

for int i 1 to VertexNum do

for int j 1 to VertexNum do

if i == j then

continue;

if ShortestPath[i][j]==ture then

cmp ShortestPath[i][j];

else then

cmp DFSShortestPath(V, VertexNum, i, j);

if cmp != INF && len < cmp then

len cmp;

jj+1;

end for

ii+1

end for

return len;

3.找到特点起点和终点的最短距离：

static long long DFSShortestPath(Vertex\* V, int VertexNum, int st, int ed)

if st == ed then

return 0;

if ShortestPath[st][ed]==true then

return ShortestPath[st][ed];

long long ans = INF;

for Edge\* edge V[st].firstedge to edge==false do

int pos edge->adjvex;

long long partlen = DFSShortestPath(V, VertexNum, pos, ed);

if partlen == INF then

continue;

else then

long long templen edge->weight + partlen;

if templen < ans then

ans templen;

edge = edge->nextedge;

end for

ShortestPath[st][ed] ans;

return ans;

4.完成最短距离中的最远距离的两节点输出：

static void Output(Vertex\* V, int VertexNum, long long len)

{

printf("图中最短距离最远的路径长度为%lld!\n始点\t终点\n", len);

for int i 1 to VertexNum do

for int j 1 to VertexNum do

if i == j then

continue;

if ShortestPath[i][j] == len then

printf("%d\t%d\n", i, j);

j++;

end for

i++;

end for

return;

**函数调用关系图：**

图示

描述已自动生成

1. **调试分析报告**

**调试过程中遇到的问题：**

问题：

调用创建图的函数后图无法被访问到。  
解决方法：

对指向顶点数组的指针V进行传引用，这样子就不再出现形参指向的地址实参无法访问的情况。  
问题：

求最短路径长度的时候有些深度搜索的过程会重复，时间复杂度会大大增加。  
解决方法：

键一个存储从i点到j点的最短路径长度的数组，当遇到同样始点和终点的更小规模的问题时可以先访问最短路径长度数组，如果已经求过可以直接使用，如果没有再调用求最短路径长度的递归函数进行求解，然后再把求解得到的最短路径长度存储到数组中。  
问题：

最短路径中距离最长的路径可能不只一条  
解决方法：

首先求出最长的最短路径的长度len，然后访问最短路径数组，对数组中所有长度为len的元素，输出其横纵坐标（对应始点和终点）

**算法时空分析：**

设输入的顶点数为n；输入边数为m；

StructGraph:O(m\*n)

GetMaximumShortestPath:O（n^2)

DFSShortestPath:O(m\*n)

Output:(n^2)

1. **用户使用说明**

步骤1：输入顶点的总数目；在确定顶点的总数目是否合理。

步骤2：如果输入顶点数合理，则依次输入每一条边，按照（起点，终点，权重）来输入。

步骤3：若输入完成则输入0 0 0来表示输入完成

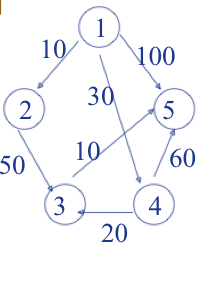
1. **测试结果**

样例1：

黑板上的字

描述已自动生成

样例2：



文本

描述已自动生成

样例3：

