电子科技大学公共管理学院

数据结构与算法应用实验报告

|  |  |
| --- | --- |
| 题目： | 社交网络图中结点的“重要性”计算 |
| 学号： | 2020160101014 |
| 姓名： | 黄雨杭 |
| 学号： | 2020160101018 |
| 姓名： | 李婷 |
| 学号： | 2020160101022 |
| 姓名： | 冯宝文 |
| 学号： | 2020160101029 |
| 姓名： | 饶芮瑞 |
| 学号： | 2020160101008 |
| 姓名： | 向韵竹 |
| 完成日期： | 2021.12.23 |

目录

[一、 问题描述 3](#_Toc20227)

[二、 需求分析 5](#_Toc10137)

[三、 概要设计 6](#_Toc31175)

[四、 详细设计 7](#_Toc23516)

[五、 调试分析 1](#_Toc9840)3

[六、 测试结果 1](#_Toc17093)4

[七、 总结 1](#_Toc3842)9

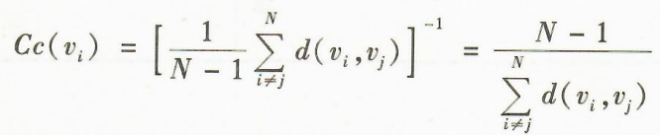
[附录：程序代码 2](#_Toc9185)2

1. **问题描述**

1.1 实验内容

在社交网络中，个人或单位（结点）之间通过某些关系（边）联系起来。他们受到这些关系的影响，这种影响可以理解为网络中相互连接的结点之间蔓延的一种相互作用，可以增强也可以减弱。而结点根据其所处的位置不同，其在网络中体现的重要性也不尽相同。

“紧密度中心性”是用来衡量一个结点到达其他结点的“快慢”的指标，即一个有较高中心性的结点比有较低中心性的结点能够更快地（平均意义下）到达网络中的其他结点，因而在该网络的传播过程中有更重要的价值。在有N个结点的网络中，结点vi的“紧密度中心性”Cc（vi）数学上定义为到其余所有结点vj（j≠i）的最短距离d（vi，vj）的平均值的倒数



对于非连通图，所有结点的紧密度中心性都是0。

本实验给定一个无权的无向图以及其中的一组结点，要求计算这组结点中每个结点的紧密度中心性。

1.2 实验要求

1.先按以下输入输出要求进行设计 ：

（1）输入说明：输入第1行给出两个正整数N和M，其中N（N≤10）是图中结点个数，顺便假设结点从1～N编号；M( M≤N\*（N-1）/2 )是边的条数。随后的M行中，每行给出一条边的信息，即该边连接的两个结点编号，中间用空格分隔。最后一行给出需要计算紧密度中心性的这组结点的个数K（K≤N）以及K个结点编号，用空格分隔。

（2）输出说明：按照“Cc（i）＝x．x x”的格式输出K个给定结点的紧密度中心性，每个输出占行，结果精确到小数点后2位。

1.3 测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 输入 | 输出 | 说明 |
| 1 | 9 14  1 2  1 3  1 4  2 3  3 4  4 5  4 6  5 6  5 7  5 8  6 7  6 8  7 8  7 9  3 3 4 9 | Cc(3) = 0.47  Cc(4) = 0.62  Cc(9) = 0.35 | 一般情况测试 |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
| 2 | 5 8  1 2  1 3  1 4  2 3  3 4  4 5  2 5  3 5  2 4 3 | Cc(4) = 0.80  Cc(3) = 1.00 | 紧密度中心性取到最大值1：结点3到其余结点距离都是1，其紧密度中心性达到最大 |
| 3 | 6 8  1 2  1 3  1 4  2 3  3 4  4 5  2 5  3 5  2 4 3 | Cc(4) = 0.00  Cc(3) = 0.00 | 非连通图，紧密度中心性都是0 |
| 4 | 10 45  45条不重复的边  10个结点 | 略 | 边界测试：最大N和M |

2.再考虑实际的社交网络，比如朋友圈或人与人之间的合作网络等等，根据兴趣来选择。对顶点进行紧密度中心性的计算，对计算结果进行分析，并输出分析结果。

3.程序也通过菜单选择一些相应功能，比如增加新朋友、查询朋友的朋友、谁是最重要的那个人、谁更需要帮助等等，将重要性计算分析放在菜单功能中，请自行设计。

4. 社交网络图中结点的 “重要性”还可以通过其他因素来体现。 例如,结点的“介数”表示一个网络中经过该结点的最短路径的数量。一个结点的介数越大,其在结点之间的通信中所起的作用也越大。为了更准确地衡量这种作用,一般采用结点的“介数中心性”来描述，可查询学习并尝试实现。

1. **需求分析**

2.1 输入

1. 输入主要分为两部分，指定数据构建图和指定需要计算紧密中心性的结点。首先指定图的结点个数和图的边数，然后输入有边连接的两个结点序号并以空格间隔，将边插入到图中，以此建立一个无权无向图。其次，在成功构建图之后，输入需要输入紧密中心性的的节点艺术以及对应的节点序号，以此为指令计算紧密中心性。
2. 本题目为了使边界测试更加便利且符合无权无向图的特征，指定结点个数N不超过10，边数M不超过N(N-1)/2。因此，当用户输入不符合要求时，需要建立人机交互来提示用户重新按要求输入。

2.2 输出

1. 输出主要分为三部分，输出图、输出指定结点的紧密中心性、输出图的类型（连通、非连通）以及标志结束。首先，在图被成功构建之后，会输出图的邻接表或者邻接矩阵，帮助用户直观地看到图的结构关系。其次，在获取到指定结点之后，会以Cc（i）=x. xx的格式输出结点的紧密度中心性。其中，i是结点的编号，各个结点的紧密度中心性的输出之间会进行换行。然后，根据图的构建指令输出图的类型。最后，标志程序结束。
2. 由于“紧密度中心性”是用来衡量一个结点到达其他结点的“快慢”的指标，对于连通图，结点的“紧密度中心性”为到其余所有结点的最短距离平均值的倒数；而对于非连通图，所有结点的紧密度中心性都是0，所以需要判断图的类型再进行紧密中心性的计算。

2.3 功能

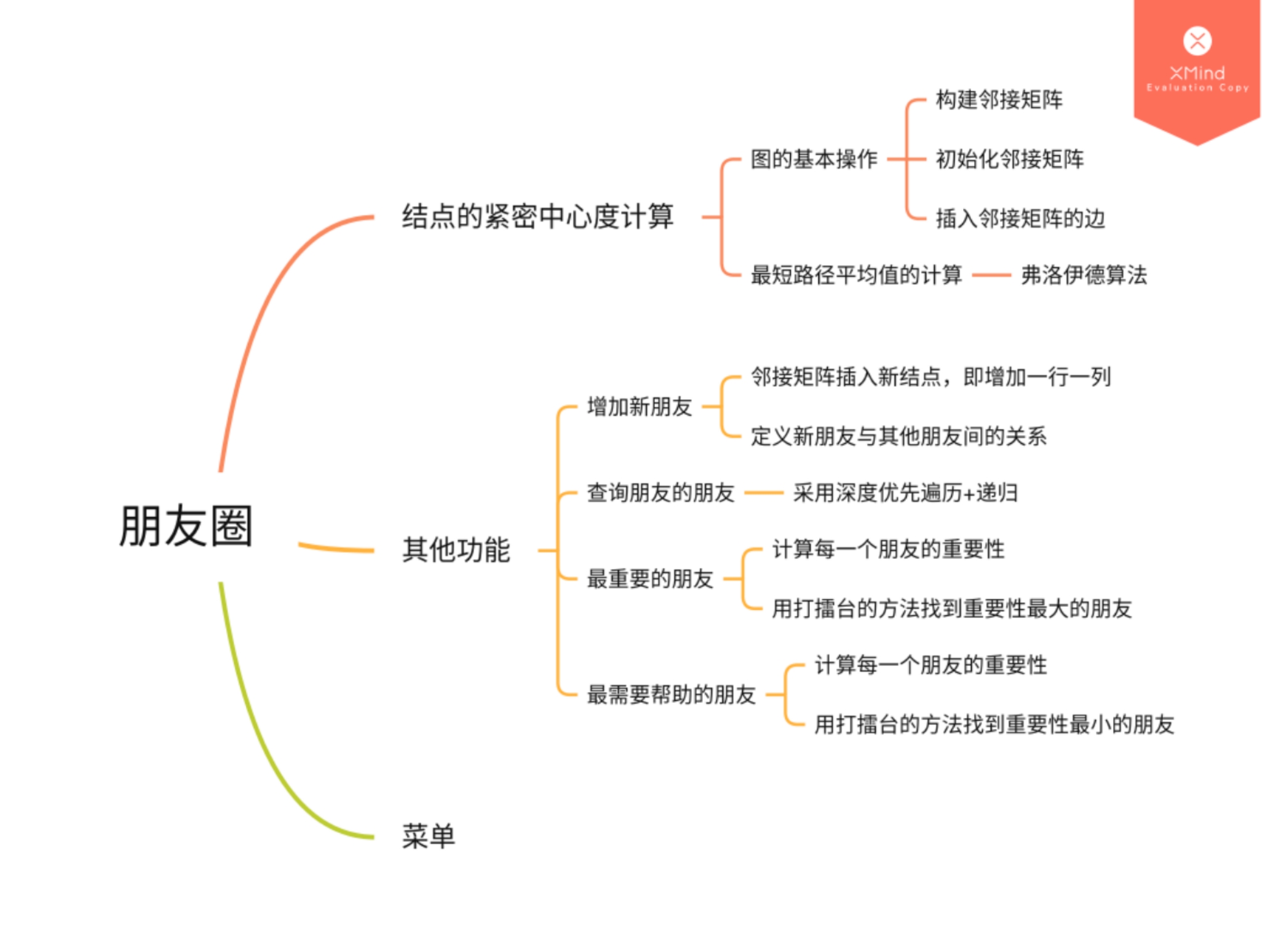
在社交网络中，个人或单位（结点）之间通过某些关系（边）联系起来。他们受到这些关系的影响，这种影响可以理解为网络中相互连接的结点之间蔓延的一种相互作用，可以增强也可以减弱。而结点根据其所处的位置不同，其在网络中体现的重要性也不尽相同。“紧密度中心性”是用来衡量一个结点到达其他结点的“快慢”的指标，即一个有较高中心性的结点比有较低中心性的结点能够更快地（平均意义下）到达网络中的其他结点，因而在该网络的传播过程中有更重要的价值。本程序运用了数据结构中图的数据类型，实现了对于社交网络的模拟；并且可以对需要考察的结点的紧密度中心性进行计算和输出。

在此基础上，本程序也模拟了“朋友圈”这个具体化的场景，增加了“加入朋友”、“查询朋友的朋友”、“谁是最重要的朋友”、“谁最需要帮助”的功能。

1. **概要设计**

3.1 图的邻接矩阵存储结构+弗洛伊德算法

3.1.1程序结构图



3.1.2抽象数据类型的定义

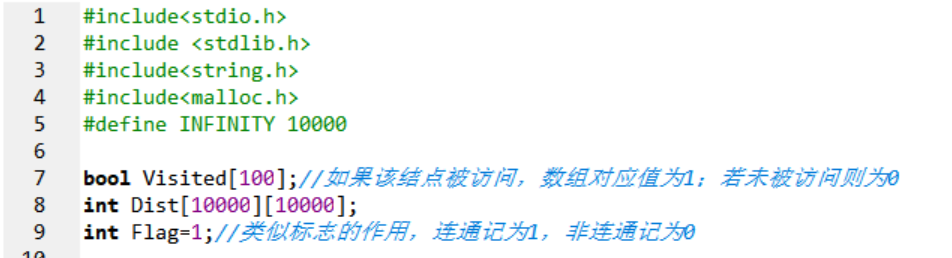
1. 使用邻接矩阵存储结构实现图的定义及其相关操作，以存储结点以及结点之间的联系。
2. 因为图是无权无向图，综合考虑之后采用弗洛伊德算法实现求最短路径的需求。

3.1.3程序模块之间的调用关系

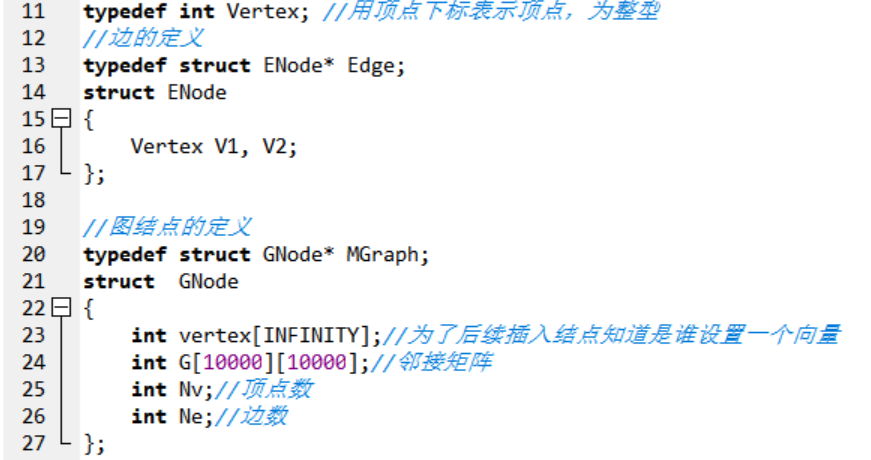
1. 在实现图的定义时，使用CreateGragh函数建立一个有N个顶点但没有边的初始图，使用Insert函数给图插入对应数量的边，实现图的构建。其中，使用了定义图结点的结构体GNode，定义边结点的结构体ENode。
2. 在求节点的紧密度中心性时，设置一个int型变量flag发挥标志的作用，连通记为1，非连通记为0。利用弗洛伊德算法求最短路径
3. 在其他功能中，插入新朋友、查询朋友的朋友以及计算最重要的朋友和最需要帮助的朋友的程序都是建立在计算紧密度中心性的前提下的，并且彼此的改变也相互影响。
4. **详细设计**

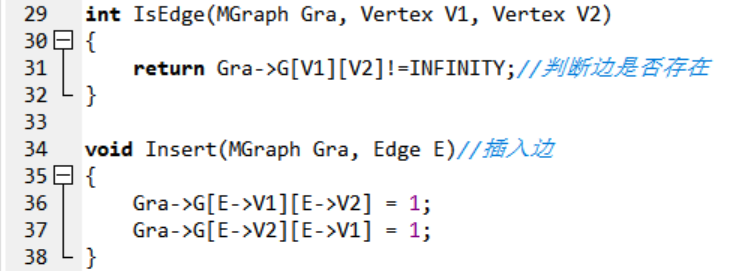
4.1 图的邻接矩阵存储结构+队列的顺序存储结构

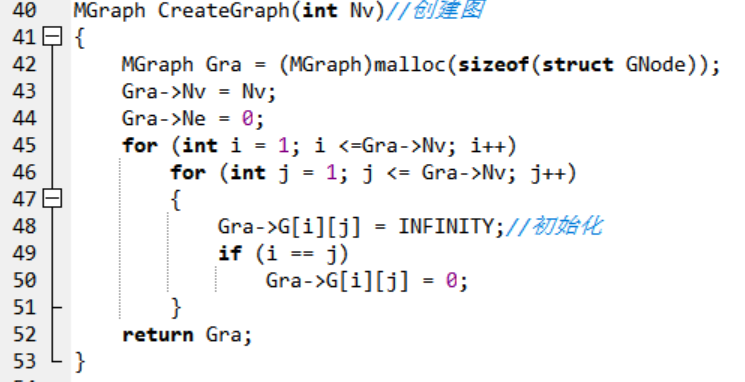
1.头文件与全局变量



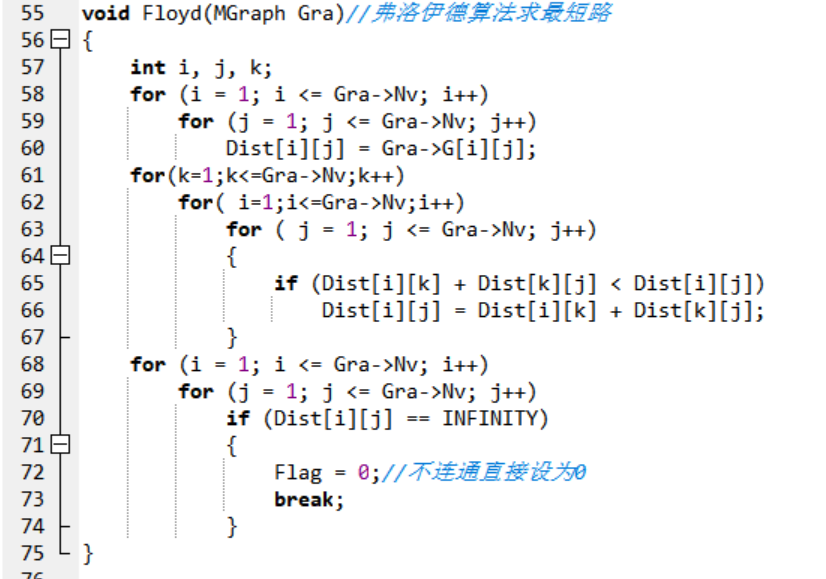
2.图的邻接矩阵定义与操作



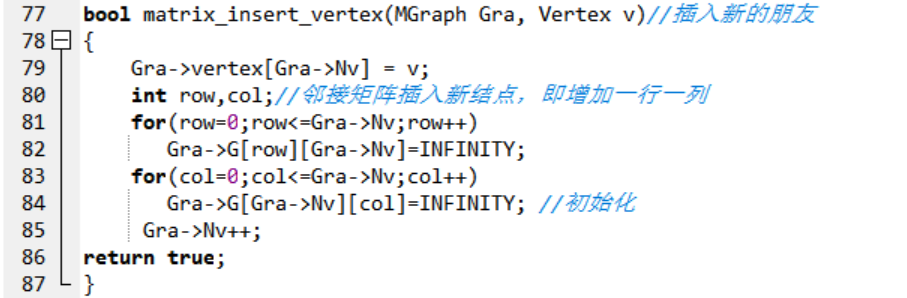




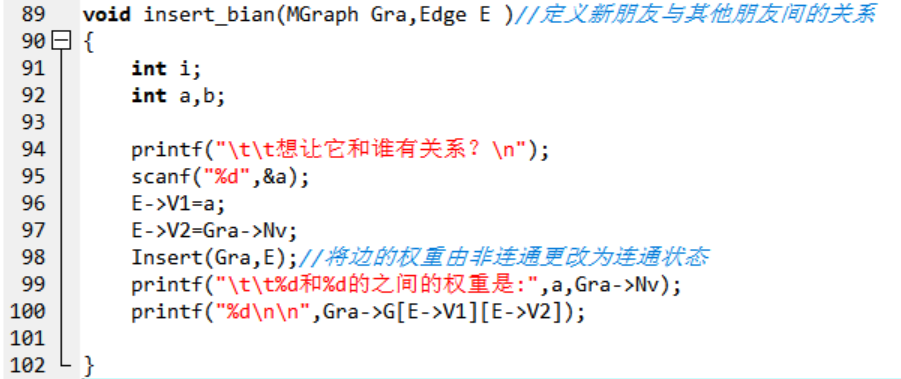
3.弗洛伊德算法求结点紧密度中心性



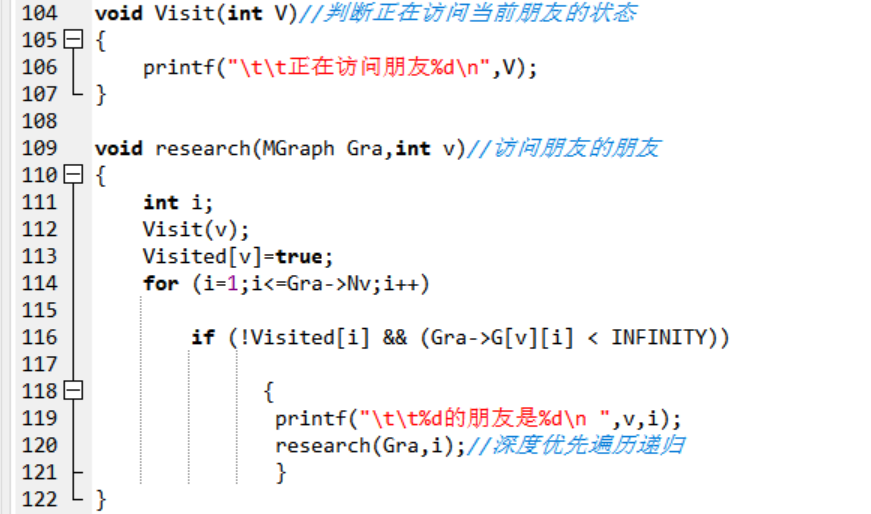
4.插入新朋友



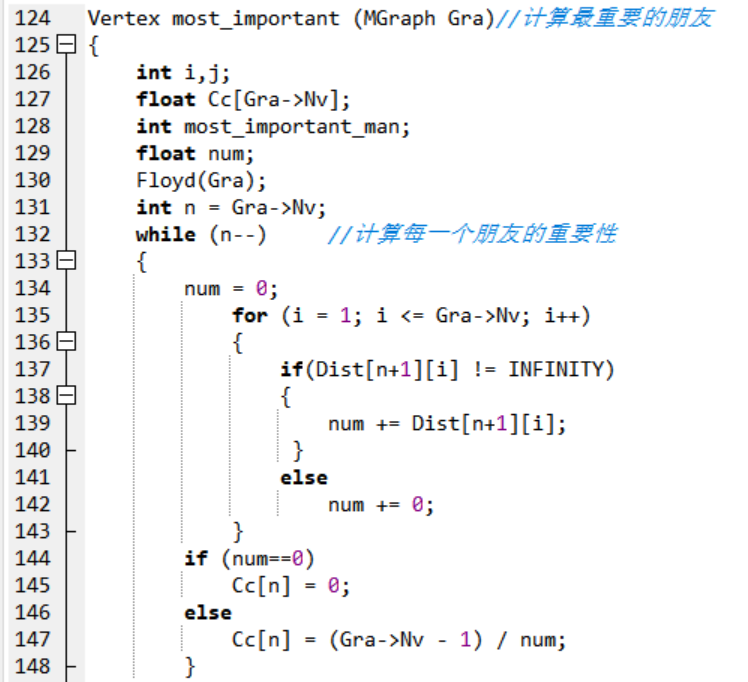
5.定义新朋友与其他朋友的关系

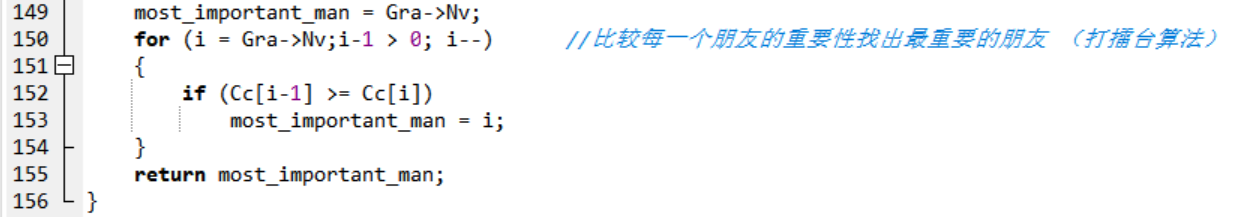


6.访问朋友的朋友

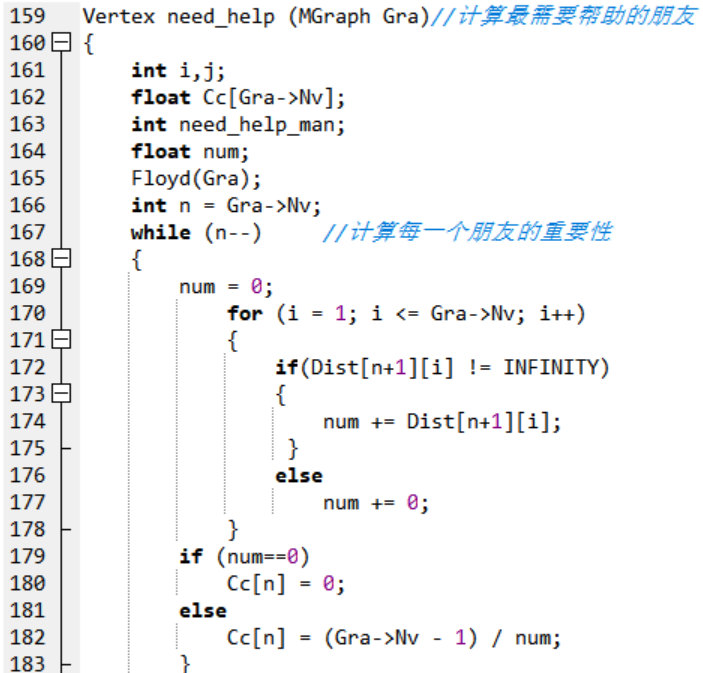


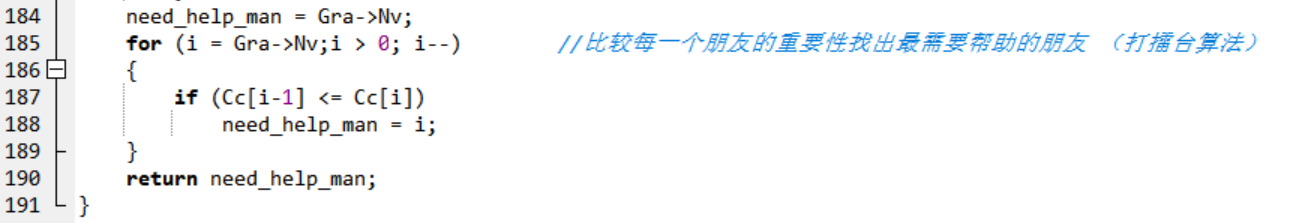
7.计算最重要的朋友



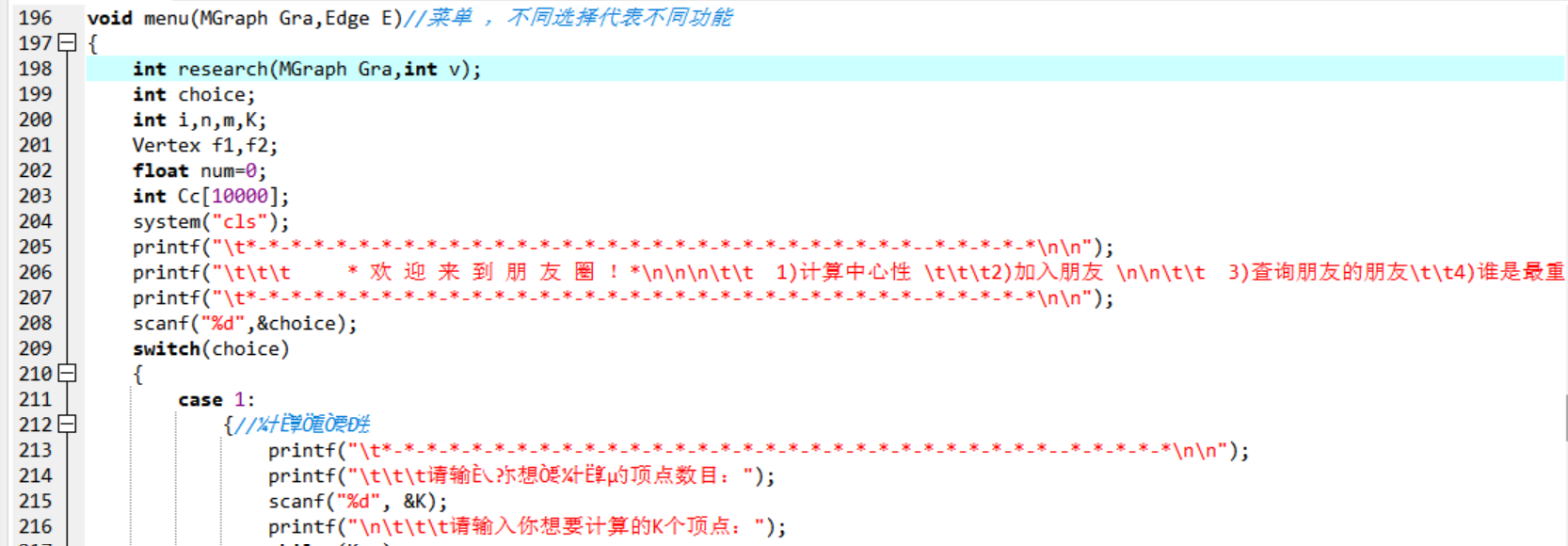


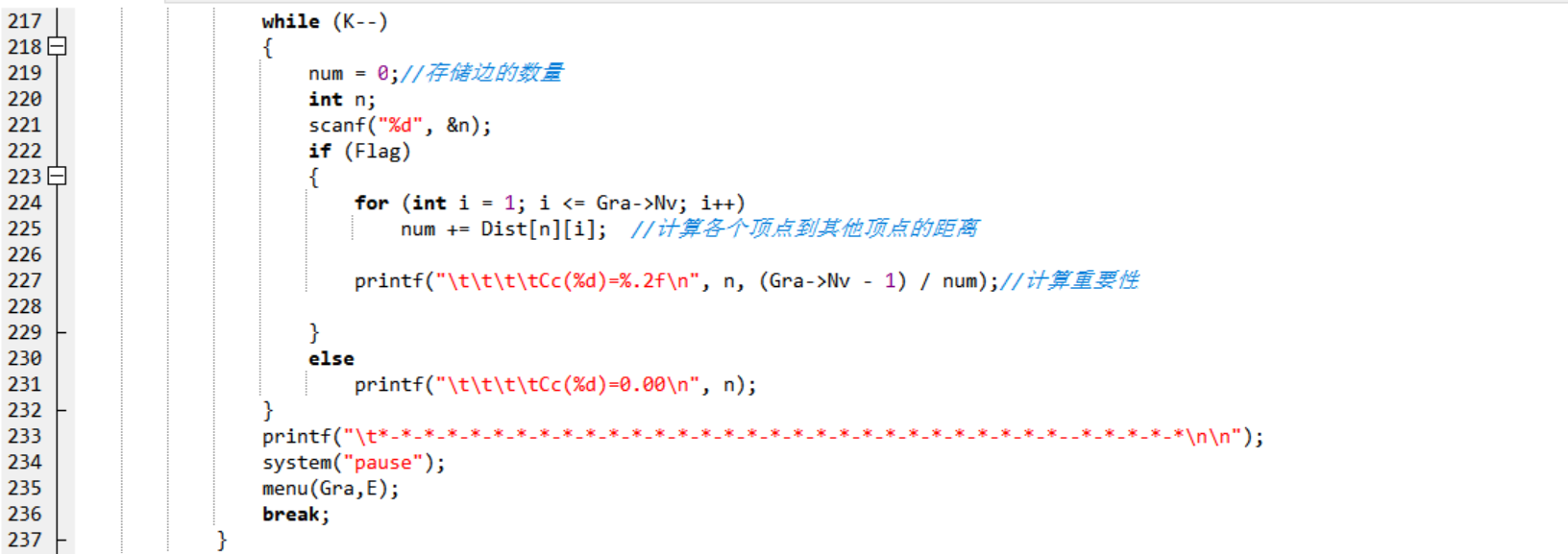
8.计算最需要帮助的朋友

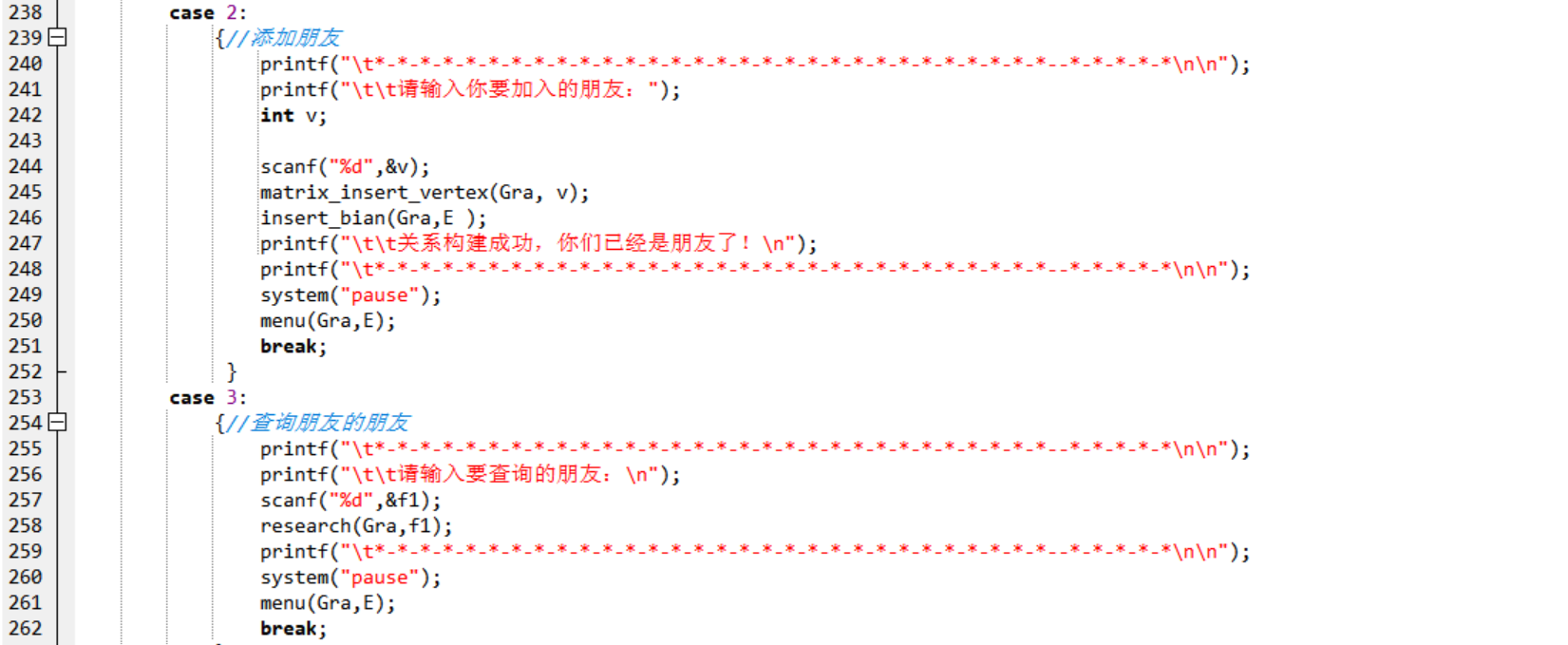


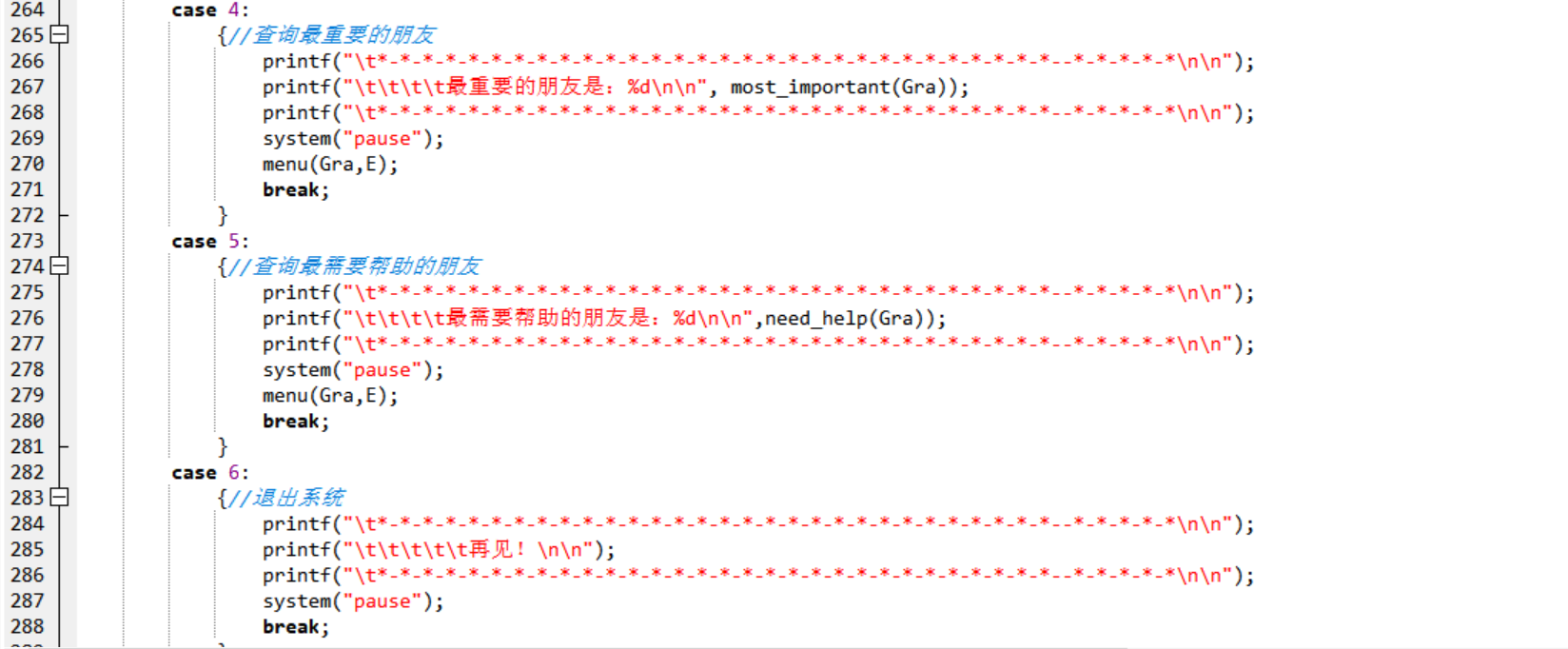


9.菜单

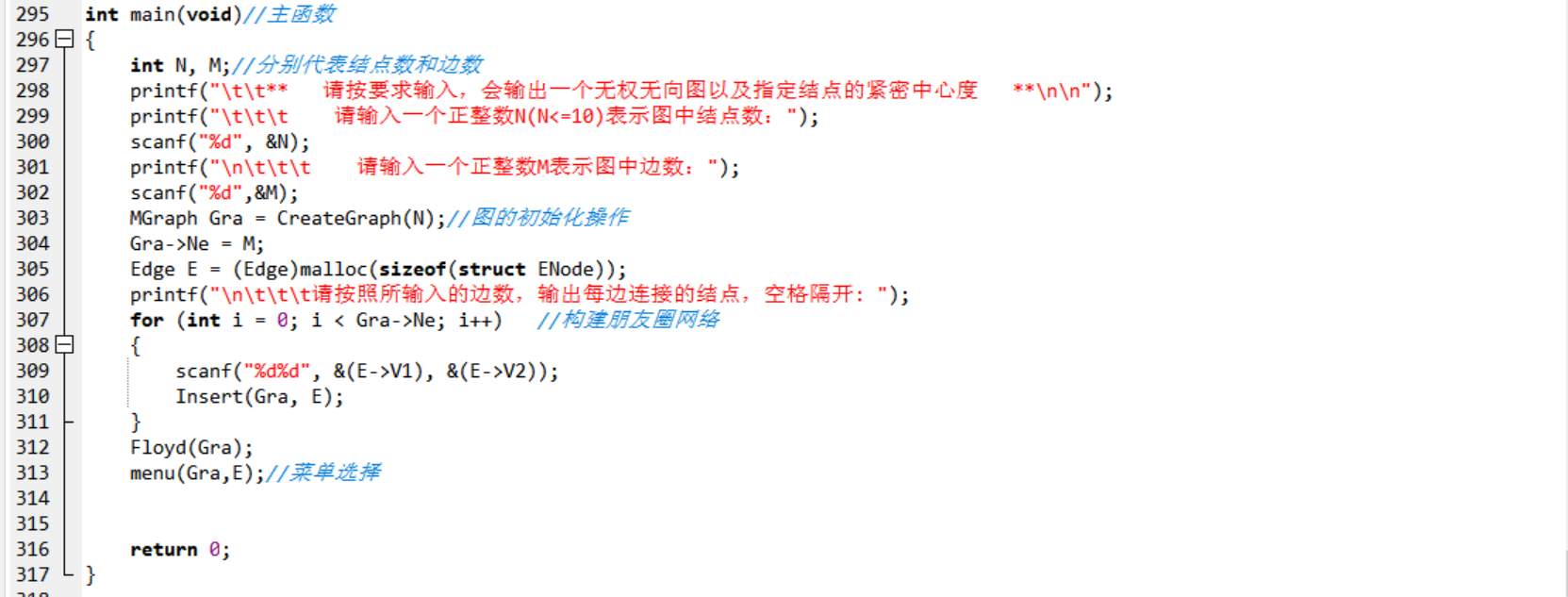








10.主函数



1. **调试分析**

5.1问题与解决方案

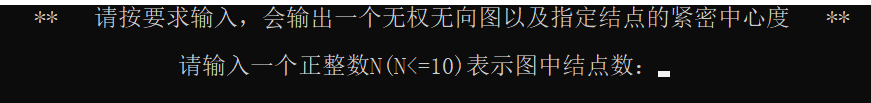
1. 对于具体计算紧密度中心性算法的选择上，我们认为仅用边是否存在的遍历算法可读性不够 故选用多对多的弗洛伊德算法，并设立了一个int型变量flag，连通时为1，不连通时为0。
2. 开始没有考虑到判断所构建的图是否是连通图，所以对于第三组数据非连通图结点的紧密中心性仍然有值、而不是0。所以设立一个全局变量对于进入队列的结点计数，若所有结点都入列过，则为连通图，否则为非连通图；若为非连通图，则所有结点的紧密中心性均为0。
3. 在利用深度优先遍历计算单源最短路径时，对于结点值V以及结点数组下标i的使用有些混乱（关系是V = i + 1），所以需要理清楚循环变量的初始值时0或1，范围是<或<=，注意下标问题。
4. 在遍历结点时，为了识别结点是否被访问以及其对应最短路径是多少，设置了bool型标志访问数组和int型路径数组，但在循环检查时发现有些结点本未被访问但却被标志已访问。所以从循环条件出发，发现问题的原因是在计算同一个图的多个结点时，每次对某一结点进行遍历求解最短路径时、未对标志访问数组进行初始化重置，导致已经计算结点之后标志访问数组的值会影响后续操作。

5.2 经验与体会

1. 设计程序时充分分析需要实现的功能，避免因为细节的疏忽而造成后续的大调整（虽然这次实验并没有出现大调整，但需要注意的是对于需求的完整分析）。
2. 设计程序时要理清思路，要有一个完整的思路和整体逻辑框架，尽量简明扼要和优化结构，让人易懂，对于存储结构、算法应用以及各函数之间的关系考虑清楚再去实现。
3. 注意根据题目进行精准设计，注重细节，创造人机交互良好的界面。
4. 可以充分利用数据结构不同的实现方式分别对程序功能进行实现，加深对于知识点的理解和掌握。
5. 注意算法的应用要根据题目要求和实际情况灵活调整，以更好地实现不同功能。
6. **测试结果**

STEP1:

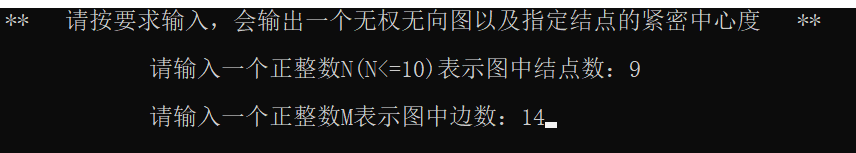
当提示输入图中结点数时，应当按提示的要求输入一个小于等于10的正整数，并以Enter键结束。若输入符合要求，程序会自动跳转至下一步。



如9

STEP2:

当提示输入边数时，应当按要求输入一个小于最大边数的正整数，最大边数由结点数所决定。（M <= N\*(N-1)/2）同样，以Enter键结束。若输入符合要求，程序会自动跳转至下一步。



STEP3:

此时，需要按边数输入相应的边，每条边由其连接的两个结点决定，两个结点之间用空格隔开，每条边之间使用Enter进行确认。

2

如：

1 2

1 3

1 4

2 3

3 4

4 5

4 6

5 6

5 7

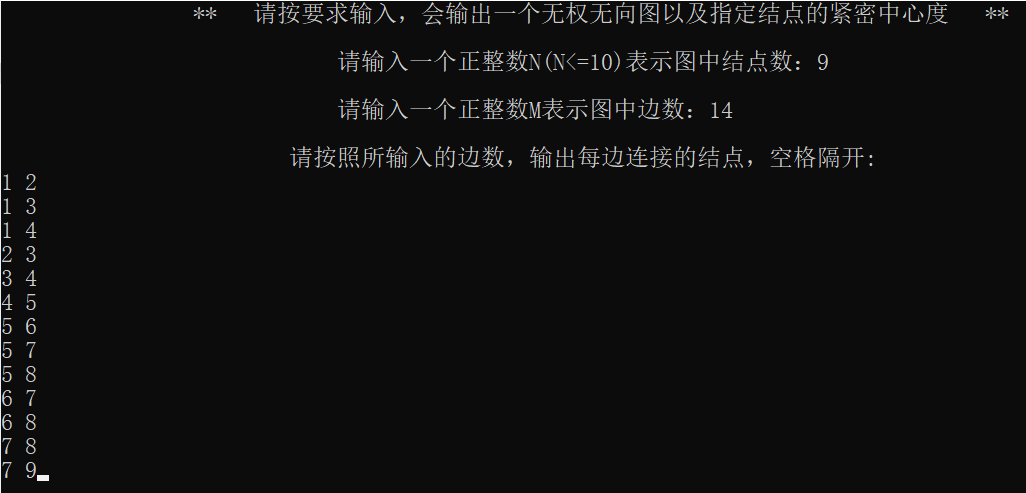
5 8

6 7

6 8

7 8

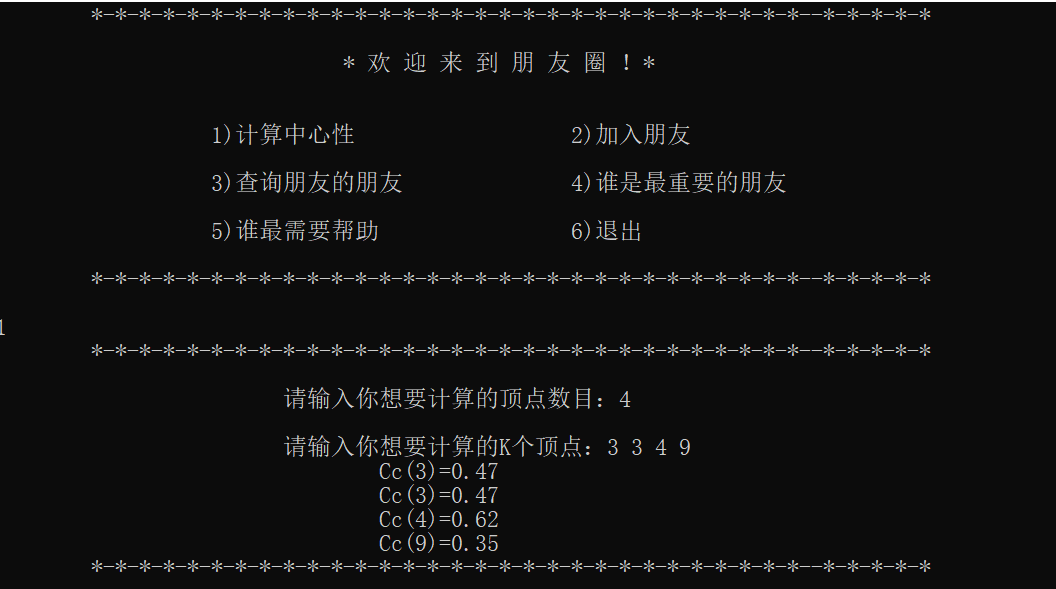
7 9



STEP4:

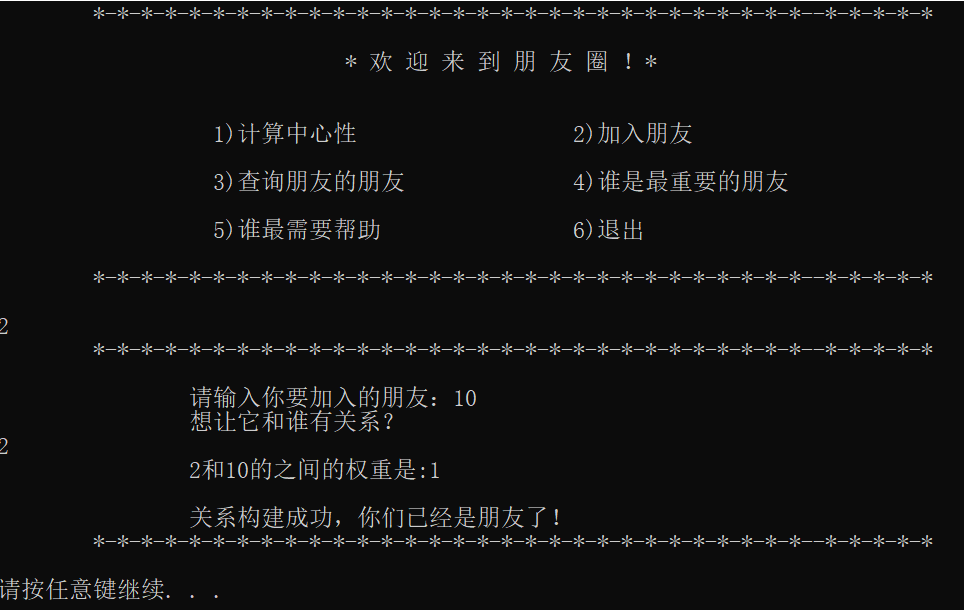
来到朋友圈，首先输入“1”计算紧密度中心性，需要输入需要计算紧密度中心性的结点的个数并输入对应的编号，每个数之间都需要用空格隔开，最后以Enter键结束。

如：3 3 4 9



STEP5:

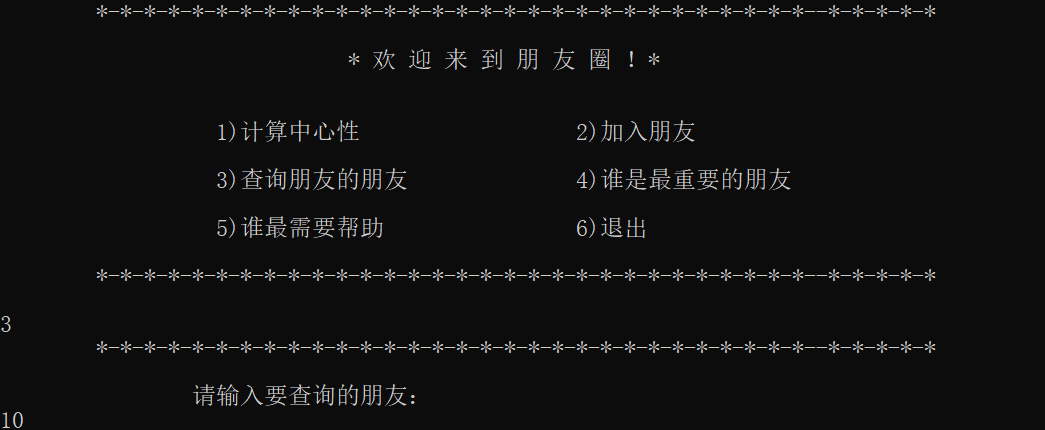
按任意键继续，来到第二个功能——加入朋友，输入“2”，再输入想要加入的朋友，输入选择新朋友的朋友。



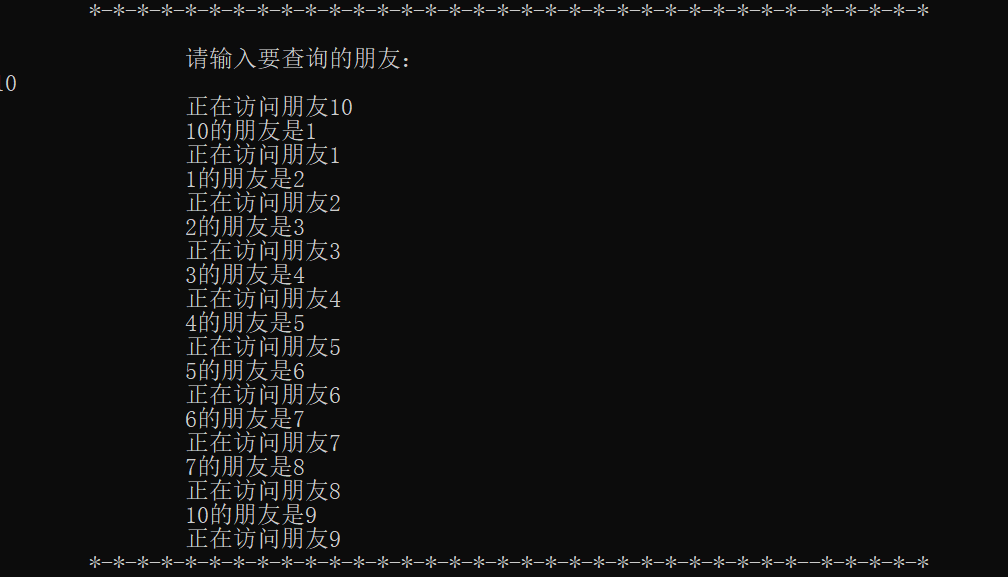
STEP6:

按任意键继续，来到第三个功能——查询朋友的朋友，输入“3”，再输入要查询的朋友

如10

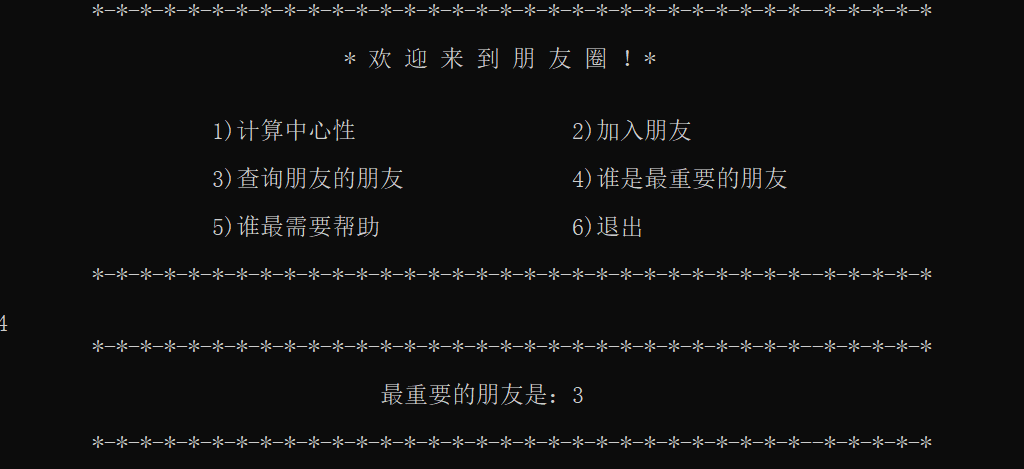


得到以下关系图：



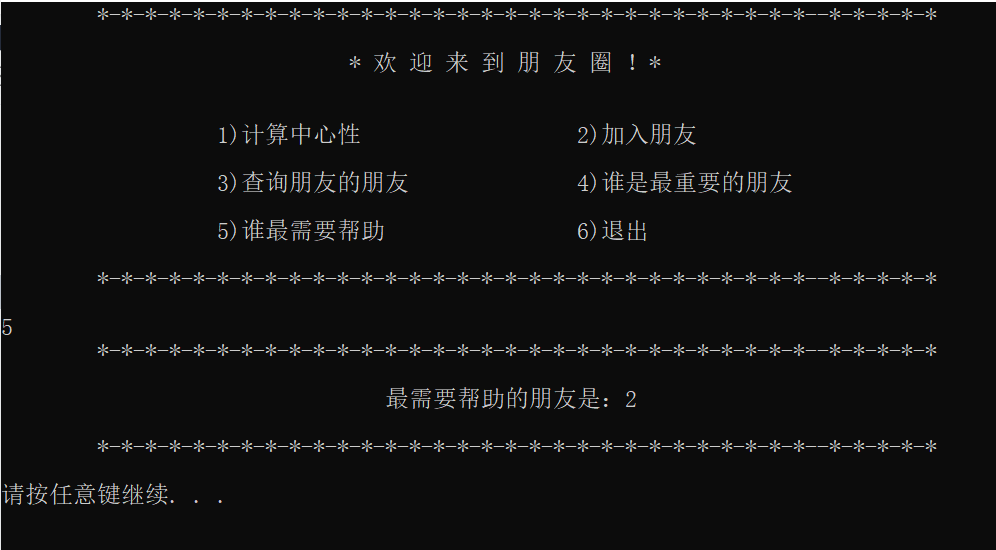
STEP7:

按任意键继续，来到第四个功能——谁是最重要的朋友，输入“4”，得到如下结果：



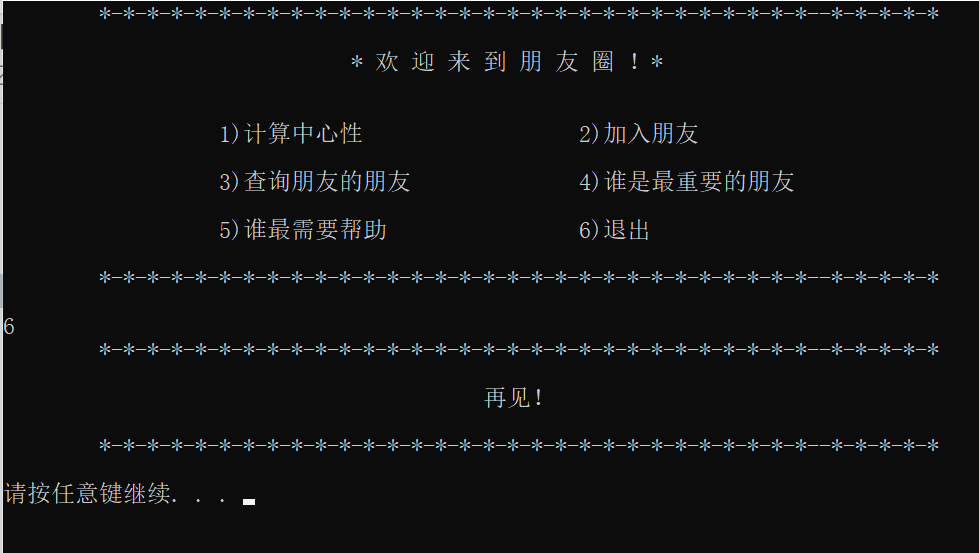
STEP8:

按任意键继续，来到第五个功能——谁最需要帮助，输入“5”，得到如下结果：



STEP9:

按任意键继续，选择退出，输入“6”，得到如下结果：、



1. **总结**

7.1 实验收获

1.本次实验主要解决了结点的紧密度中心性的算法问题。与前几次作业，以及上学期的C语言的很多实验相比，本次实验的现实意义更强，是利用数据结构所学知识真正去解决来源于现实生活当中的问题。我们将社交重要性的问题放在一个具体的情景中“朋友圈”，将程序问题现实化，具体化让我们切实地体会到了，这门课程的意义不仅仅在于各类数据类型的定义与实现，更在于实际应用和解决现实问题。

2.通过规范化的实验操作与实验报告的撰写，也规范了我们完成小项目的流程。从一步步地进行问题分析和任务定义，到数据类型和系统设计的初步构思，再到实际的编码实现和静态检查，最后再进行调试和改进。这样的过程虽然与原本直接写代码相比，似乎多出了很多步骤，但其实也减少了在实际编码实现中产生的阻力，帮助我们可以更加高效地完成复杂度远比原本高很多的程序。

3.同时，对于同样的数据结构，我们也尝试采用了不同的实现方式进行编码实现。而我们可以看到，虽然各部分函数的内部实现是不同的，但最终的解决问题的主函数都是相同的。在实际应用当中，我们也可以尝试借助已经定义好的API或者第三方库来解决现实问题。我们学习数据结构，不仅要学习算法的具体实现，对内在逻辑进行理解，更要学会熟练掌握利用定义的类的基础操作实现实际需求，即利用算法解决现实问题的能力。

7.2 问题解决

1. 最开始我们对具体计算紧密度中心性算法的选择上还有些具体不定，最后我们认为仅用边是否存在的遍历算法可读性不够，故选用多对多的弗洛伊德算法。
2. 开始没有考虑到判断所构建的图是否是连通图，所以对于第三组数据非连通图结点的紧密中心性仍然有值、而不是0。所以设立一个全局变量对于进入队列的结点计数，若所有结点都入列过，则为连通图，否则为非连通图；若为非连通图，则所有结点的紧密中心性均为0。
3. 在利用深度优先遍历计算单源最短路径时，对于结点值V以及结点数组下标i的使用有些混乱（关系是V = i + 1），所以需要理清楚循环变量的初始值时0或1，范围是<或<=，注意下标问题。
4. 在遍历结点时，为了识别结点是否被访问以及其对应最短路径是多少，设置了bool型标志访问数组和int型路径数组，但在循环检查时发现有些结点本未被访问但却被标志已访问。所以从循环条件出发，发现问题的原因是在计算同一个图的多个结点时，每次对某一结点进行遍历求解最短路径时、未对标志访问数组进行初始化重置，导致已经计算结点之后标志访问数组的值会影响后续操作。

7.3 程序调试

1. 首先，要可以读懂编译器进行编译后的报错信息，大致了解常见错误以及出错后对应的大致情况，让排错有迹可循。虽然目前在进行过一段时间的代码编写后，一定程度上减少了输入中文字符或是拼写的错误。但相对应的，随着学习的深入，其他一些编程语言的书写习惯也会导致我们有时会因一时疏忽出现一些低级错误。这是我们应当避免以及首先需要排查的错误。
2. 其次，随着我们需要编写的程序结构越来越复杂，涉及的模块如结构体、函数越来越多，很多时候需要在编写程序时尽可能全部完成后再进行编译和排错。面对这类新出现的情况，一方面我们要区分由于程序编写不完整而导致的错误，另一方面在编写程序时尽可能编写完成程序后再进行排错。这样不仅可以让我们防止一些由于程序不完整而导致的奇怪错误影响我们的进度，又可以方便我们排查到一些由于模块之间相互调用而可能会出现的错误。
3. 同时，我们要善于使用编译器自带的调试功能。在DEV C++中，我们可以通过修改编译器选项，令其产生调试信息，以实现调试功能。善于使用断点和变量查看，有助于我们利用调试功能解决较长程序当中出现的错误，方便我们查看究竟是哪一个部分在运行过程中出错，从而缩小排查的范围，直到找到错误。
4. 但随着数据结构课程的不断推进，我们广泛地使用结构体来帮助我们实现数据结构。而调试功能并不能很好地查看结构体变量内的变量情况。这时需要我们善于用printf语句，在一些操作的关键节点对一些结构体或是变量的内容进行查看，来帮助我们不通过调试的方式也方便地查看其内容，从而实现缩小排

查范围的作用。

7.4 思考认识

在学习数据结构这门课程的时候，或多或少总会让我在脑中把它和面向对象程序设计纠缠在一起。在我看来，数据结构与面向对象都是想要将生活中具体的、现实的事务，通过代码的方式进行抽象，试图将更多的我们遇到的非结构化的问题转化为结构化的问题，从而更好地让计算机对其进行处理，并实现一定的功能。不同的就是，数据结构的抽象程度似乎更高一点，更接近逻辑与过程，而面向对象更多的还是想要模拟现实生活中的具体事物。

而在我们逐渐深入了解之后，这样的想法也越来越深入。我们常用的工具就是结构体、数组、指针、链表，而我们也一直在使用typedef来把原本一些基础的类型赋予他们更接近现实的意义。这样的过程让我产生了一种感悟：我们是通过一些基础的数据类型，对现实生活中更加复杂的事务和流程进行模拟，其本质是将非结构化的问题转化为结构化的问题，其目的是将计算机强大的计算功能更好地运用于现实生活和生产实践中去。

因此，我们学习数据结构，不应当仅仅停留在数据类型或是算法本身，也不应该仅仅停留在使用代码将其实现上。因为实现其数据类型的定义或是算法仅仅是最基础的部分，我们应该更加深入地思考这样的数据类型可以帮助我们拟合和解决哪些现实生活中实际存在的问题，我们可以通过算法和代码帮助我们克服生活中的哪些困难或是带来哪些便利。更进一步的，当我们在生活实际中遇到一些复杂问题时，我们应当学会思考，如何将这类复杂的问题进行分析和细化，从而通过一些基本的数据类型来帮助我们解决这样的问题。

总的来说，我们作为信息管理与信息系统专业的同学，面对这门课程，我们应当树立一定的学科思维：一方面，应当深入理解数据结构这门课程的内容，懂得这些常见的数据类型可以帮助我们解决哪些现实的问题；另一方面，我们更要领会数据结构这门课程的思想，学会使用计算机以及代码作为我们的工具，通过将现实生活中的问题抽象为一定的数据类型及其操作，帮助我们分析和解决现实生活中的问题。

**附录：程序代码**

**#include<stdio.h>**

**#include <stdlib.h>**

**#include<string.h>**

**#include<malloc.h>**

**#define INFINITY 10000**

**bool Visited[100];//如果该结点被访问，数组对应值为1；若未被访问则为0**

**int Dist[10000][10000];**

**int Flag=1;//类似标志的作用，连通记为1，非连通记为0**

**typedef int Vertex; //用顶点下标表示顶点，为整型**

**//边的定义**

**typedef struct ENode\* Edge;**

**struct ENode**

**{**

**Vertex V1, V2;**

**};**

**//图结点的定义**

**typedef struct GNode\* MGraph;**

**struct GNode**

**{**

**int vertex[INFINITY];//为了后续插入结点知道是谁设置一个向量**

**int G[10000][10000];//邻接矩阵**

**int Nv;//顶点数**

**int Ne;//边数**

**};**

**int IsEdge(MGraph Gra, Vertex V1, Vertex V2)**

**{**

**return Gra->G[V1][V2]!=INFINITY;//判断边是否存在**

**}**

**void Insert(MGraph Gra, Edge E)//插入边**

**{**

**Gra->G[E->V1][E->V2] = 1;**

**Gra->G[E->V2][E->V1] = 1;**

**}**

**MGraph CreateGraph(int Nv)//创建图**

**{**

**MGraph Gra = (MGraph)malloc(sizeof(struct GNode));**

**Gra->Nv = Nv;**

**Gra->Ne = 0;**

**for (int i = 1; i <=Gra->Nv; i++)**

**for (int j = 1; j <= Gra->Nv; j++)**

**{**

**Gra->G[i][j] = INFINITY;//初始化**

**if (i == j)**

**Gra->G[i][j] = 0;**

**}**

**return Gra;**

**}**

**void Floyd(MGraph Gra)//弗洛伊德算法求最短路**

**{**

**int i, j, k;**

**for (i = 1; i <= Gra->Nv; i++)**

**for (j = 1; j <= Gra->Nv; j++)**

**Dist[i][j] = Gra->G[i][j];**

**for(k=1;k<=Gra->Nv;k++)**

**for( i=1;i<=Gra->Nv;i++)**

**for ( j = 1; j <= Gra->Nv; j++)**

**{**

**if (Dist[i][k] + Dist[k][j] < Dist[i][j])**

**Dist[i][j] = Dist[i][k] + Dist[k][j];**

**}**

**for (i = 1; i <= Gra->Nv; i++)**

**for (j = 1; j <= Gra->Nv; j++)**

**if (Dist[i][j] == INFINITY)**

**{**

**Flag = 0;//不连通直接设为0**

**break;**

**}**

**}**

**bool matrix\_insert\_vertex(MGraph Gra, Vertex v)//插入新的朋友**

**{**

**Gra->vertex[Gra->Nv] = v;**

**int row,col;//邻接矩阵插入新结点，即增加一行一列**

**for(row=0;row<=Gra->Nv;row++)**

**Gra->G[row][Gra->Nv]=INFINITY;**

**for(col=0;col<=Gra->Nv;col++)**

**Gra->G[Gra->Nv][col]=INFINITY; //初始化**

**Gra->Nv++;**

**return true;**

**}**

**void insert\_bian(MGraph Gra,Edge E )//定义新朋友与其他朋友间的关系**

**{**

**int i;**

**int a,b;**

**printf("\t\t想让它和谁有关系？\n");**

**scanf("%d",&a);**

**E->V1=a;**

**E->V2=Gra->Nv;**

**Insert(Gra,E);//将边的权重由非连通更改为连通状态**

**printf("\t\t%d和%d的之间的权重是:",a,Gra->Nv);**

**printf("%d\n\n",Gra->G[E->V1][E->V2]);**

**}**

**void Visit(int V)//判断正在访问当前朋友的状态**

**{**

**printf("\t\t正在访问朋友%d\n",V);**

**}**

**void research(MGraph Gra,int v)//访问朋友的朋友**

**{**

**int i;**

**Visit(v);**

**Visited[v]=true;**

**for (i=1;i<=Gra->Nv;i++)**

**if (!Visited[i] && (Gra->G[v][i] < INFINITY))**

**{**

**printf("\t\t%d的朋友是%d\n ",v,i);**

**research(Gra,i);//深度优先遍历递归**

**}**

**}**

**Vertex most\_important (MGraph Gra)//计算最重要的朋友**

**{**

**int i,j;**

**float Cc[Gra->Nv];**

**int most\_important\_man;**

**float num;**

**Floyd(Gra);**

**int n = Gra->Nv;**

**while (n--) //计算每一个朋友的重要性**

**{**

**num = 0;**

**for (i = 1; i <= Gra->Nv; i++)**

**{**

**if(Dist[n+1][i] != INFINITY)**

**{**

**num += Dist[n+1][i];**

**}**

**else**

**num += 0;**

**}**

**if (num==0)**

**Cc[n] = 0;**

**else**

**Cc[n] = (Gra->Nv - 1) / num;**

**}**

**most\_important\_man = Gra->Nv;**

**for (i = Gra->Nv;i-1 > 0; i--) //比较每一个朋友的重要性找出最重要的朋友 （打擂台算法）**

**{**

**if (Cc[i-1] >= Cc[i])**

**most\_important\_man = i;**

**}**

**return most\_important\_man;**

**}**

**Vertex need\_help (MGraph Gra)//计算最需要帮助的朋友**

**{**

**int i,j;**

**float Cc[Gra->Nv];**

**int need\_help\_man;**

**float num;**

**Floyd(Gra);**

**int n = Gra->Nv;**

**while (n--) //计算每一个朋友的重要性**

**{**

**num = 0;**

**for (i = 1; i <= Gra->Nv; i++)**

**{**

**if(Dist[n+1][i] != INFINITY)**

**{**

**num += Dist[n+1][i];**

**}**

**else**

**num += 0;**

**}**

**if (num==0)**

**Cc[n] = 0;**

**else**

**Cc[n] = (Gra->Nv - 1) / num;**

**}**

**need\_help\_man = Gra->Nv;**

**for (i = Gra->Nv;i > 0; i--) //比较每一个朋友的重要性找出最需要帮助的朋友 （打擂台算法）**

**{**

**if (Cc[i-1] <= Cc[i])**

**need\_help\_man = i;**

**}**

**return need\_help\_man;**

**}**

**void menu(MGraph Gra,Edge E)//菜单 ，不同选择代表不同功能**

**{**

**int research(MGraph Gra,int v);**

**int choice;**

**int i,n,m,K;**

**Vertex f1,f2;**

**float num=0;**

**int Cc[10000];**

**system("cls");**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t\t \* 欢 迎 来 到 朋 友 圈 ！\*\n\n\n\t\t 1)计算中心性 \t\t\t2)加入朋友 \n\n\t\t 3)查询朋友的朋友\t\t4)谁是最重要的朋友\n\n\t\t 5)谁最需要帮助\t\t6)退出\n\n");**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**scanf("%d",&choice);**

**switch(choice)**

**{**

**case 1:**

**{//计算重要性**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t\t请输入你想要计算的顶点数目：");**

**scanf("%d", &K);**

**printf("\n\t\t\t请输入你想要计算的K个顶点：");**

**while (K--)**

**{**

**num = 0;//存储边的数量**

**int n;**

**scanf("%d", &n);**

**if (Flag)**

**{**

**for (int i = 1; i <= Gra->Nv; i++)**

**num += Dist[n][i]; //计算各个顶点到其他顶点的距离**

**printf("\t\t\t\tCc(%d)=%.2f\n", n, (Gra->Nv - 1) / num);//计算重要性**

**}**

**else**

**printf("\t\t\t\tCc(%d)=0.00\n", n);**

**}**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**menu(Gra,E);**

**break;**

**}**

**case 2:**

**{//添加朋友**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t请输入你要加入的朋友：");**

**int v;**

**scanf("%d",&v);**

**matrix\_insert\_vertex(Gra, v);**

**insert\_bian(Gra,E );**

**printf("\t\t关系构建成功，你们已经是朋友了！\n");**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**menu(Gra,E);**

**break;**

**}**

**case 3:**

**{//查询朋友的朋友**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t请输入要查询的朋友：\n");**

**scanf("%d",&f1);**

**research(Gra,f1);**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**menu(Gra,E);**

**break;**

**}**

**case 4:**

**{//查询最重要的朋友**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t\t\t最重要的朋友是：%d\n\n", most\_important(Gra));**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**menu(Gra,E);**

**break;**

**}**

**case 5:**

**{//查询最需要帮助的朋友**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t\t\t最需要帮助的朋友是：%d\n\n",need\_help(Gra));**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**menu(Gra,E);**

**break;**

**}**

**case 6:**

**{//退出系统**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**printf("\t\t\t\t\t再见！\n\n");**

**printf("\t\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*--\*-\*-\*-\*-\*\n\n");**

**system("pause");**

**break;**

**}**

**}**

**}**

**int main(void)//主函数**

**{**

**int N, M;//分别代表结点数和边数**

**printf("\t\t\*\* 请按要求输入，会输出一个无权无向图以及指定结点的紧密中心度 \*\*\n\n");**

**printf("\t\t\t 请输入一个正整数N(N<=10)表示图中结点数：");**

**scanf("%d", &N);**

**printf("\n\t\t\t 请输入一个正整数M表示图中边数：");**

**scanf("%d",&M);**

**MGraph Gra = CreateGraph(N);//图的初始化操作**

**Gra->Ne = M;**

**Edge E = (Edge)malloc(sizeof(struct ENode));**

**printf("\n\t\t\t请按照所输入的边数，输出每边连接的结点，空格隔开: ");**

**for (int i = 0; i < Gra->Ne; i++) //构建朋友圈网络**

**{**

**scanf("%d%d", &(E->V1), &(E->V2));**

**Insert(Gra, E);**

**}**

**Floyd(Gra);**

**menu(Gra,E);//菜单选择**

**return 0;**

**}**