哈尔滨工业大学(深圳)

《数据结构与算法》实验报告

实验 3

图型结构及其应用

学　院: 机电工程与自动化学院

姓　名: 吴俊达

学　号: 210320621

专　业: 自动化

日　期: 2023-5-1

一、问题分析

题 1：使用邻接矩阵或邻接表构建给定**无向图**，判断该图是否连通，并计算每个节点的度。

题 1 分析：选择采用**邻接矩阵**来构建无向图（采用二维数组存储邻接矩阵）；利用**图的广度优先遍历和(或)深度优先遍历**来判断图是否连通，通过对邻接矩阵进行搜索、判断每两个节点的间距来计算每个节点的度。

题 2：计算无向图的聚类系数（Clustering coefficient）。单个节点的聚类系数 (Local Cluster Coefficient) 是**所有与它相连的节点之间所连的边数**①，除以**这些节点之间可以连出的最大边数**②。图的聚类系数是其所有节点聚类系数的均值（即，所有节点聚类系数之和除以节点数）。对于度 0 或者 1 的节点，聚类系数为 0③。

题 2 分析：上述问题中①转化为求无向图中某节点的度（见题 1 分析）；②转化为利用邻接矩阵判断两个节点间是否相连、并计算出多个节点中两两相连的结点对数。③注意特殊判断。

题 3：若图连通，使用 Dijkstra 算法计算节点 1 到节点 3 的单源最短路径。

题 3 分析：首先要判断图是否连通（题 1）；其次判断图是否有节点 1 和节点 3（与节点总个数作比较）；第三是利用 Dijkstra 算法计算节点 1 到**所有节点**的最短路径，最后输出节点 1 到节点 3 的最短路径。

题 4（附加题 1）：若图连通，使用 Floyd 算法计算该图的直径和半径。涉及概念有：①节点距离：两个节点间的最短路径长度。②节点离心率 (Eccentricity)：节点到其他节点距离的最大值。③图直径 (Diameter)：图中所有节点的最大离心率。④图半径 (Radius)：图中所有节点的最小离心率。

题 4 分析：首先要判断图是否连通（题 1）；其次用一个二维数组存放每两个节点间的距离，利用 Floyd 算法计算节点间的最短路径；最后根据上述二维数组中的数据计算每个节点到其他节点距离的最大值（节点离心率），进而计算图的直径和半径。

二、详细设计

2.1 设计思想

图的信息至少有以下几个要素：图的结点数、边数、邻接矩阵。所以采用结构体来存储这个图（结构体具体定义在 2.2 存储结构及操作 一节中）。

程序首先读取自身目录下的 stu.in 文件。

为使计算机容易识别，stu.in 文件的输入格式设置如下：

```
7        // 节点数
8        // 边数
// 以下 8 行描述结点之间的连接关系，节省篇幅仅列出部分
0 6 1  //节点 0 到节点 6 有一条长度为 1 的边
……
1 3 8
```

如果读取失败则输出提示信息并退出程序。如果读取成功，则：

1. 程序读入前两个数字 n 和 e，调用 createGraph 函数初始化有 n 个节点和 e 条边的图。
2. 读入若干行，填充邻接矩阵（两个节点间的距离信息）。
3. 调用判断图是否连通的 isConnected 函数并输出图是否连通的信息（连通返回 1，不连通返回 0）；
4. 创建用于存放节点的度的数组，并调用计算各节点度的 nodeDegree 函数，最后输出各节点度的信息；
5. 调用计算图的聚类系数的 clusteringCoefficient 函数并输出图的聚类系数；
6. 若图是连通的（利用 isConnected 函数判断），则调用 dijkstra 函数利用 dijkstra 算法计算两节点（例如，节点 1 到节点 3）的最短距离、输出并写入表示路径的数组 path，调用 printPath 函数输出路径，最后再调用 Radius 和 Diameter 函数利用 Floyd 算法计算图的半径和直径并输出。【事实上，若图不是连通的，也可以利用 Dijkstra 算法和 Floyd 算法，所以“若图是连通的”可以不用判断，只需在对应函数中增加路径不存在等情况的判断即可，操作上的更改和具体逻辑分别在 2.2 节和 2.3 节作了说明】

以上 6 步进行完毕后，程序继续读入两个数，如果读取成功，则再重复进行以上 6 步，如果读取失败，则程序结束。

（各函数的定义、操作和算法思想在 2.2 存储结构及操作 一节中）

2.2 存储结构及操作

1. 存储结构：

利用结构体来存储图的相关信息。结构体定义如下：

```
typedef char vextype[20];
typedef struct {
    int N, E;//N 是顶点数，E 是边数
    int **matrix;//储存邻接矩阵（二维数组）
    vextype *vertex;//存储节点的名字
} Graph;
```

增加存储节点名字的接口，是为了与实际问题相衔接。实际问题中往往是把一个实体对象抽象为一个节点，其必然有对应的名字。在本程序中，直接将各节点的 ID 设置为该节点的名字。

在 bfs_graph 函数中使用数组的存储结构维护了一个队列，来实现图的广度优先遍历。

2. 涉及的操作

函数名	形参表	返回值	功能简述
createGraph	int n, int e	Graph	创建一个节点数为 n，边数为 e 的图
printPath	int d, int *diameter_path, Graph g	void	输出路径
bfs_graph	Graph g, int node_id, bool* visited	int	图的广度优先遍历。连通返回 1，否则返回 0
dfs_graph	Graph g,int node_id,bool* visited	void	图的深度优先遍历。利用引用传参给 isConnected 函数
isConnected	Graph g	int	判断图是否连通，连通返回 1，不连通返回 0
nodeDegree	Graph g, int *node_degree	void	计算每个点的度
clusteringCoefficient	Graph g	double	计算图的聚类系数
find_closest_node	int total_node_num,bool visited[], int *path_length_to_each	int	距离起始点（源点）最近的点
dijkstra	Graph g, int start, int end, int *path	int	使用 dijkstra 算法计算单源最短路径
Eccentricity	Graph g,int* distance	void	使用 Floyd 算法计算图上各点的离心率
Diameter	Graph g	int	计算图的直径
Radius	Graph g	int	计算图的半径

以下是各函数实现的具体操作：（各函数中涉及分配内存处均检查是否返回空指针[分配内存失败]，对应错误码为 1）

1. createGraph 函数：图的创建和初始化。将节点数和边数存储于结构体中，并利用 malloc 函数初始化邻接矩阵和顶点名字。
2. printPath 函数：打印路径。传入表示路径的数组和对应的路径总长，每打印路径中的一步的同时计算现有路径总长，若现有路径总长达到预期的路径总长，则表示路径输出完毕，退出该函数。
3. isConnected 函数：判断图是否连通。创建 bool 型 visited 数组。若调用 bfs_graph 函数（见以下第 4 点），则将其返回值直接返回即可；若调用 dfs_graph 函数（见以下第 5 点），则检查 visited 数组是否全为 true，若是则说明图为连通，返回 1；反之则说明图非连通，返回 0。
4. bfs_graph 函数：图的广度优先遍历。形参 node_id 表示遍历的起始点，bool 型 visited 数组用于标记某节点是否找到过（初始全为 false）。创建一个队列来存放节点 ID（用数组的存储结构实现，采用少用一个元素形式），并用变量 nodenum 来标记已找到节点数目。使 0 号节点先入队，当队列非空时，每次都使队头对应节点出队并访问队头对应节点，若某节点与当前节点的距离不为无穷大(有边相连)且未被访问过，将此节点入队（相当于把搜索向前推进一步，之后遍历与它相连的节点）并将它的找到标记置为 true，已找到节点数目增加 1。在以上每一步寻找中，如果已找到节点总数等于图中节点总数，则说明已经找到了所有节点，直接返回 1；否则，若到函数末尾，已找到节点总数还是小于图中节点总数，则图不是连通的（某一连通分量没有包含所有的节点），返回 0。
5. dfs_graph 函数：图的深度优先遍历。bool 型 visited 数组用于标记某节点是否访问过。从某一由形参 node_id 指定的起始节点开始，对于所有和起始节点有边相连且未访问过的节点，访问它们并以它们为起始作深度优先搜索(递归过程)，若该节点的所有节点均被遍历，则退出函数。visited 数组通过引用传递方式传回到调用它的 isConnected 函数中（见第 3 点）。
6. nodeDegree 函数：计算节点的度。首先，将传入的 node_degree 数组（准确地说是给 node_degree 指针分配的内存区域）中每个元素均置为 0。因为是无向图，所以只用遍历邻接矩阵的右上三角即可(不包含主对角线)，对于每个邻接矩阵元素，只要不是 max_dis（距离无穷大），则节点 i 和 j 的度同时增 1。
7. clusteringCoefficient 函数：计算图的聚类系数。先初始化聚类系数变量为 0。对于每个节点分别计算聚类系数：首先定义一数组，标记与某节点相连的节点 ID，定义变量存储与某节点相连的节点数(节点的度)和与某节点相连的节点之间的边数。接着，通过遍历邻接矩阵的对应一行，找到所有与某节点相连的节点，存于上述数组中，同时计算出该节点的度。如果节点的度为 0 或 1，聚类系数为 0，不参与下面计算；否则，通过在邻接矩阵中查找上述数组中存放的各节点之间的距离（若不为无穷大则两节点间有边相连），计算这些节点之间的边数。最后利用聚类系数公式计算出单个节点的聚类系数，最后将

所有节点的聚类系数取平均即可得图的聚类系数。

8. dijkstra 函数：用 Dijkstra 算法计算单源最短路径。首先判断传入的节点是否合法。路径长度（返回值）初始化为 max_dis（无穷大），用 closest_node 变量来标记离源点最近的节点 ID，初始化访问标记数组 visited 为 false(0)。创建 path_length_to_each 数组存储从源点到每个节点的最短路径长度，并初始化数组为邻接矩阵中对应值。创建 last_node 数组标记到达某一个节点上一步所到达的节点，并将源点可到达的节点(除自身外)的"上一节点标记"记为源点 ID，其余记为-1(表示没有从源点到该点的路径)。初始状态时，已访问的只有源点，然后从未访问顶点中选择到源点路径长度最短的顶点（利用 find_closest_node 函数[见第 9 点]处理，用变量 closest_node 存储），并更新源点到未访问过顶点的距离，更新方法是：原来的当前最短路径长度值与从源点过顶点 closest_node 到达该顶点的路径长度中的较小者；同时更新该顶点的“上一节点标记”。此过程不断重复，直到全部顶点都被访问过一次（find_closest_node 函数返回-1，见第 9 点）为止。这样就计算出了从源点到所有节点的最短路径。接下来是输出到某个特定顶点 end 的路径。如果 end 的 "上一节点标记"为-1，则说明没有到该节点的路径；否则就从当前节点向前寻找，将路径逆序依次存入 path[g.N]、path[g.N-1]、……等中，直到向前找到的节点是-1(这就说明上次已经找到源点，源点已经加入到路径中)。因为路径是从后往前存储的，源点 ID 所在内存单元与 path 有偏移，利用 memmove 函数移动对齐。最后返回路径长度数组中 end 下标对应的元素，**如果没有路径或节点非法，返回的都是 max_dis【针对非连通图路径可能不存在的特殊处理】**，否则返回路径长度。
9. find_closest_node 函数：找到距源点最近的点。total_node_num 表示节点总数，visited[]是访问标记，path_length_to_each 是源点到每个节点的距离数组（整型指针表示）。遍历一遍整个 path_length_to_each 数组，假如某节点未被访问过且源点到此节点的距离更小，则返回值更新为这个节点 ID。如果始终没能找到这样的点，则返回-1。
10. Eccentricity 函数：利用 Floyd 算法计算节点离心率。创建一个二维数组来存储任意两点间的最短距离，初始值为邻接矩阵中的值。每次将节点 i 加入已访问集合，并修改 A[j][k]的值，修改方法是：A[j][k] = Min{ A[j][k] , (A[j][i]+A[i][k]) }，直到图中所有顶点都加进已访问集合为止。最后求解每个节点的离心率。
11. Diameter 函数：调用 Eccentricity 函数计算图中各点的离心率，通过按引用传递获得存储图中各点离心率的数组，找出该数组中最大值即可。
12. Radius 函数：同上 Diameter 函数，找出存储图中各点离心率的数组中的最小值即可。

2.3 程序整体流程图

如下两张图所示。图 1 为根据问题分析得出的程序流程图（有判断图是否连通的步骤）；图 2 去掉了判断图是否连通的步骤（在 Dijkstra 函数中对于路径不存在的情况做了对应的设计）

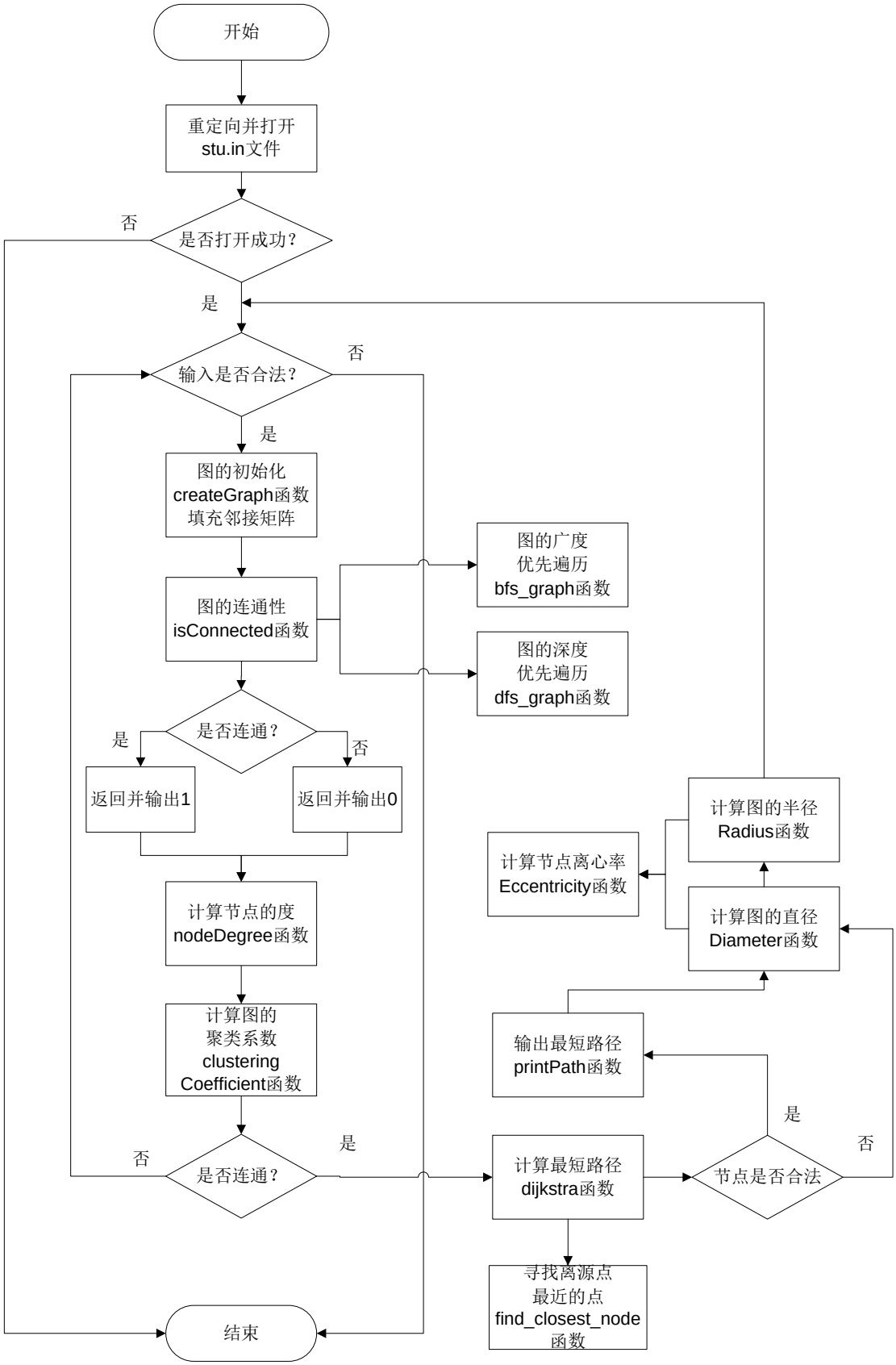

开始
重定向并打开
stu.in文件
否
是否打开成功？
是
否
输入是否合法？
是
图的初始化
createGraph函数
填充邻接矩阵
图的连通性
isConnected函数
图的广度
优先遍历
bfs_graph函数
图的深度
优先遍历
dfs_graph函数
是否连通？
是
否
返回并输出1
返回并输出0
计算节点的度
nodeDegree函数
计算图的
聚类系数
clustering
Coefficient函数
否
是否连通？
是
计算最短路径
dijkstra函数
寻找离源点
最近的点
find_closest_node
函数
节点是否合法
是
否
输出最短路径
printPath函数
计算图的直径
Diameter函数
计算节点离心率
Eccentricity函数
计算图的半径
Radius函数
结束

图 1

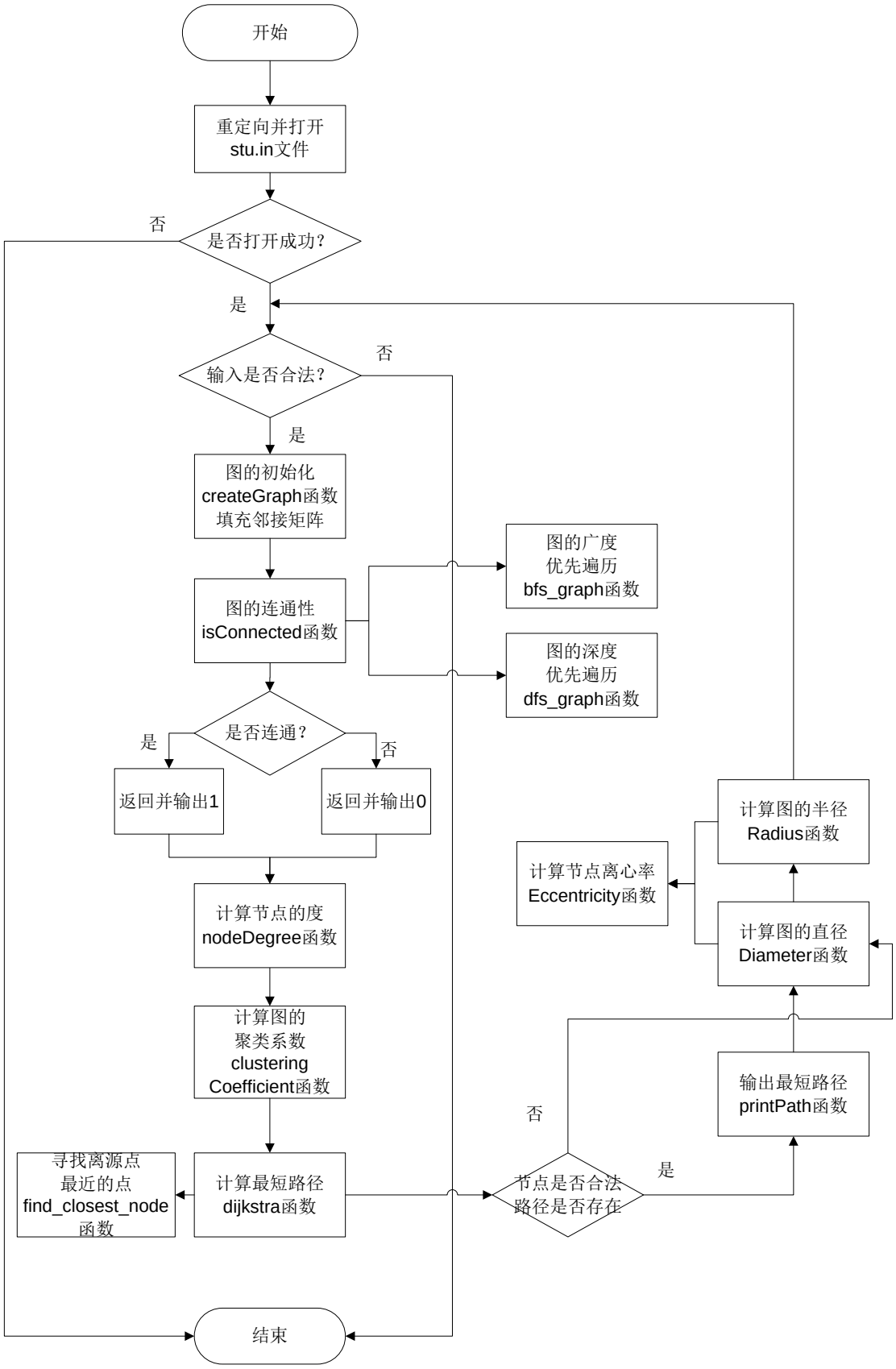

开始
重定向并打开
stu.in文件
否
是否打开成功?
是
输入是否合法?
否
是
图的初始化
createGraph函数
填充邻接矩阵
图的连通性
isConnected函数
图的广度
优先遍历
bfs_graph函数
图的深度
优先遍历
dfs_graph函数
是否连通?
是
否
返回并输出1
返回并输出0
计算节点的度
nodeDegree函数
计算图的
聚类系数
clustering
Coefficient函数
寻找离源点
最近的点
find_closest_node
函数
计算最短路径
dijkstra函数
节点是否合法
路径是否存在
否
是
输出最短路径
printPath函数
计算图的半径
Radius函数
计算节点离心率
Eccentricity函数
计算图的直径
Diameter函数
结束

图 2

三、用户手册

（一）数据的输入格式

待输入数据应保存在与源代码/源程序同一文件夹下的 stu.in 文件中。

对于每个图，其有关信息应按照以下的格式输入：

```
7          // 节点数
8          // 边数
// 以下 8 行描述结点之间的连接关系，节省篇幅仅列出部分
0 6 1   //节点 0 到节点 6 有一条长度为 1 的边
……
1 3 8
```

请确保输入格式正确。边数以下的若干行，行数须和边数相同。例如上面实例中，某图有 8 条边，则边数以下应有 8 行表示每条边长度和所连接结点的信息。

不同的图的信息，按照上述格式依次录入 stu.in 文件即可，无需空行。每张图的处理结果会分别输出。

（二）实现各种功能的操作方式

程序首先读取自身目录下的 stu.in 文件。如果读取失败程序会输出提示信息并退出程序。如果读取成功，则：*（斜体字表明不必须）*

1. **图的初始化：**程序读入前两个数字 n 和 e，传入 createGraph 函数初始化有 n 个节点和 e 条边的图。再读入若干行，填充邻接矩阵（两个节点间的距离信息）。
2. **图的连通性判断：**调用判断图是否连通的 isConnected 函数（传入图 g 即可）并输出图是否连通的信息（连通返回 1，不连通返回 0）；
3. **计算节点的度：**创建用于存放节点的度的数组，并调用计算各节点度的 nodeDegree 函数（传入图 g 和上述数组即可），最后输出各节点度的信息；
4. **计算图的聚类系数：**调用计算图的聚类系数的 clusteringCoefficient 函数（传入图 g 即可）并输出图的聚类系数；
5. **计算最短路径：***若图是连通的（利用 isConnected 函数判断）*，则调用 dijkstra 函数（传入图 g、源点 start、终点 end 和用于保存路径的 int 型指针 path 即可。**请务必保证源点和终点是存在的！**）利用 dijkstra 算法计算某两个节点间的最短距离、输出并写入表示路径的数组 path，调用 printPath 函数（传入图 g、路径数组 path 和路径总长 d）输出路径，最后再调用 Radius 和 Diameter 函数（均传入图 g 即可）利用 Floyd 算法计算图的半径和直径并输出。

以上 5 步进行完毕后，程序继续读入两个数，如果读取成功，则再重复进行以上 5 步，如果读取失败，则程序结束。

程序输出样例为：

case 1: // 1 为序号，即代表它是 stu.in 文件里的第 1 个图

connected: 1 // 是否连通，连通为 1，不连通为 0

degree distribution:

node0:4,node1:1,node2:1,node3:1,node4:1, // 每个节点的度

clustering coefficient:0.000000 // 聚类系数

// 若判断图是否连通，则只有图连通时才出现以下 4 行结果

// 否则，路径存在时如下，路径不存在时输出 Path from x to y is not found!

the shortest path between 1 and 3: 4 // 从 1 到 3 节点的最短路径长为 4

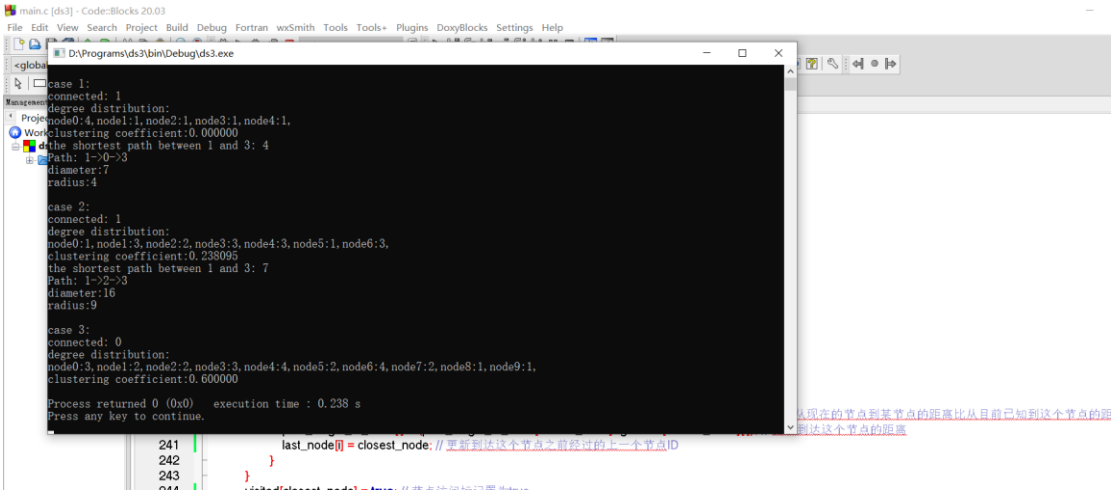
Path: 1->0->3 //从 1 到 3 节点的最短路

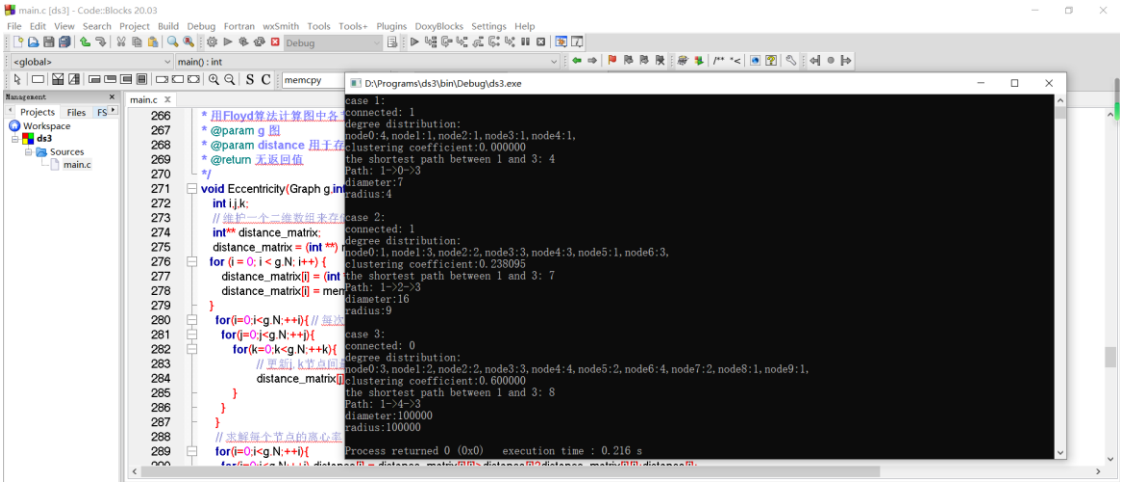
diameter:7 // 图的直径

radius:4 // 图的半径

四、结果

（上图：有判断图是否连通的版本，下图：不判断图是否连通的版本）





五、总结

该实验涉及的数据结构有队列、图、数组（线性表），算法有图的广度优先遍历算法（广度优先搜索）、深度优先遍历算法（深度优先搜索）、Dijkstra 算法和 Floyd 算法。

遇到的问题较多，以下列举几个较难发现、收获较大、或影响较大的问题：

（一）算法部分

1. 无向图的广度优先遍历中，一个节点的找到标记置为 true 和找到节点数增 1 这一步，应该在这个节点入队的时候就完成，而不是在这个节点出队被访问到的时候才完成（之所以不称作访问节点数而称为找到节点数，也是出于这方面的考虑）。若不如此，则这个节点会在被访问到之前重复地被其他节点找到。另外，如果节点数达到了图的总结点数，说明已经找到了所有节点，可以直接返回 1，有利于加快程序运行速度，对于稠密图尤其合适。
2. 无向图的深度优先遍历算法的自然语言描述为：

   - 所有顶点访问标志visited[]设置为FALSE
   - 从某顶点v_0开始，设v=v_0
   1. 如果visited[v]=FALSE，则访问该顶点，且设visited[v]=TRUE
   2. 如果找到当前顶点的一个新的相邻顶点w，设v=w，重复1
   3. 否则（说明当前顶点的所有相邻顶点都已被访问过，或者当前顶点没有相邻顶点），如果当前顶点是v_0，退出；否则返回上一级顶点，重复2

   第 3 点在程序中的转化是个难点。最后我使用了 if(i==g.N)return; 来实现。这个语句放在函数的最后一行。对于当前顶点是起始顶点 v_0 的情况，这个语句能实现退出函数；否则，这个语句能实现退出对于这个顶点的搜索，上一级顶点的循环语句能继续进行（即“返回上一级顶点”）。

（二）程序设计部分（细节问题）

1. 在利用 memset 函数给 int 型数组快速赋初值时，发现利用 memset 函数给数组赋初值 1 会出现混乱的结果。检查后发现，memset 函数是按**字节（以八位为单位）**对内存块进行初始化，所以不能用它将 int 数组初始化为 0 和-1 之外的其他值（除非该值高字节和低字节相同）。具体地说，在用 memset 给 int 型数组赋初值-1 时，-1 的二进制码为（11111111）$_2$，则 memset 把每个 int 型数据所占的 4 个字节均赋值为 11111111，则每个 int 型数据实际对应的值就是（11111111 11111111 11111111 11111111）$_2$，十进制仍为-1（赋值为 0 的情况也类似）；而用 memset 给 int 型数组赋初值 1 时，1 的二进制码为（00000001）$_2$，则 memset 把每个 int 型数据所占的 4 个字节均赋值为 00000001，则每个 int 型数据实际对应的值就是（00000001 00000001 00000001 00000001）$_2$，对应的十进制值为 16843009，所以产生错误。
2. int 型指针+1 即向前移动一个 int 型的存储单元，不需要乘 sizeof(int)。
3. memcpy 函数在拷贝源和目标间有重叠时不能保证拷贝内容正确，使用 memmove 函数可以解决这个问题。

4. 在函数中给指针分配的空间在退出函数时要释放掉。

我的收获是：

1. 要善用调试工具，如 Code::Blocks 中的断点调试、Watches 视窗等。
2. 要细心、耐心、有恒心，要对边界条件作充分的检查，考虑一些异常情况，提高程序的健壮性。
3. 要认真理解引入各类数据结构的意义；要认真领悟经典算法的精髓，例如“空间换时间（存储信息，化遍历为查表）”等。