《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 23软件工程 | 姓名 | 梅子羽 | 学号 | 2327406107 |
| 实验布置日期 | | 2024.11.26 | | 提交  日期 | 2024.12.6 | | 成绩 |  |

课程实践实验6：图的实现及应用

## 一、问题描述及要求

题目1 无向图的实现

[问题描述]

（1）创建一个无向图的邻接矩阵和邻接表结构；

（2）在邻接矩阵结构下对该图进行深度优先搜索；

（3）在邻接表结构下对该图进行广度优先搜索。

题目2 社交网络模型

[问题描述]

设计并实现一个社交网络模型图。

要求：

（1）每个人的信息是一个顶点，两个人相互认识则构成边。

（2）根据输入的任意两个人的信息，给出他们之间的联系路径，最少经过多少人构成联系。

（3）可根据自己的创意添加更多的功能。

## 二、概要设计

#### 1. 对实验内容的理解

在本实验中，我们需要设计并实现一个图的数据结构，并在此基础上进行深度优先搜索（DFS）和广度优先搜索（BFS）。特别地，实验要求实现一个社交网络模型，其中每个人的信息表示为数字顶点，两个顶点之间的边表示两个人之间的认识关系。通过图的遍历算法，我们能够查询两个人之间的联系路径，最少经过多少人构成联系。

#### 2. 系统功能列表

**功能模块一：无向图实现**

* **功能1**：创建无向图的邻接矩阵和邻接表结构。
* **功能2**：在邻接矩阵上实现深度优先搜索（DFS）。
* **功能3**：在邻接表上实现广度优先搜索（BFS）。
* **功能4**：实现startToEnd()函数，查询并显示两个人（顶点）之间的联系路径，最少经过多少人。

**功能模块二：社交网络模型**

* **功能1**：每个人的信息作为数字顶点，两个顶点之间的边表示两个人之间的认识关系。
* **功能2**：查询任意两个人之间的联系路径，返回最短路径（通过startToEnd()实现）。

#### 3. 程序运行界面设计

程序启动后，界面提示用户选择功能：

1. 从文件加载图数据并进行遍历：
   * 选择加载graph1.txt文件，执行深度优先搜索（DFS）和广度优先搜索（BFS）。
2. 查询社交网络中任意两个人之间的联系路径：
   * 输入两个顶点（表示两个人），系统输出最短路径，表示他们之间的联系。

#### 4. 总体设计思路

**数据结构选择**：

1. **图的表示**：
   * **邻接矩阵**：二维数组edge[MaxSize][MaxSize]，表示图的边。
   * **邻接表**：vertex是一个存储顶点的链表结构，用于表示邻接表。
2. **社交网络模型**：
   * 顶点表示用户，边表示两个用户之间的认识关系。
   * 使用\*\*深度优先搜索（DFS）**和**广度优先搜索（BFS）\*\*来实现路径查找和最短路径计算。

**算法选择**：

1. **深度优先搜索（DFS）**：用于在邻接矩阵结构下遍历图。
2. **广度优先搜索（BFS）**：用于在邻接表结构下遍历图，特别适合用来查找最短路径。
3. **startToEnd()**：用于查询并输出两个顶点（人）之间的最短路径，已经在题目中通过DFS和BFS实现。

#### 5. 程序结构设计

**Graph类**：

* **成员变量**：
  + vertexNum：图的顶点数。
  + edgeNum：图的边数。
  + edge[MaxSize][MaxSize]：邻接矩阵。
  + vertex：邻接表。
* **成员方法**：
  + adjMatrixDfs：邻接矩阵的DFS遍历。
  + adjLineBfs：邻接表的BFS遍历。
  + startToEnd：查询两个顶点之间的最短路径，已经实现社交网络功能。

**社交网络模型**：

* 直接使用Graph类的startToEnd方法查询社交网络中的路径。

#### 6. 类之间的关系

* Graph类实现了图的基本操作（DFS、BFS）和社交网络模型（通过startToEnd()实现的最短路径查询）。社交网络功能无需额外的类设计，可以直接复用Graph类。

## 三、详细设计

#### 1. 主函数设计

主函数负责整个程序的执行流程，包括文件加载、用户输入处理、图的遍历算法调用以及社交网络模型的路径查询。具体流程如下：

1. **加载图文件**：
   * 主函数首先尝试打开graph1.txt文件，并将文件内容传递给ass1()函数来处理任务1：无向图的深度优先搜索（DFS）和广度优先搜索（BFS）遍历。
   * 随后，主函数尝试打开graph2.txt文件，并调用ass2()函数来执行任务2：社交网络中两个人之间的联系路径查询。
2. **任务1：无向图的DFS与BFS遍历**：
   * ass1()函数会首先检查文件是否成功打开。如果成功，函数会创建一个Graph对象，解析图的数据结构。
   * 用户输入起始节点，程序根据输入进行DFS和BFS遍历，输出每种遍历的节点访问顺序。
   * 对于DFS，程序会从起始节点开始，深度遍历所有未访问的邻接节点，直到图中所有可访问节点都被遍历为止。
   * 对于BFS，程序从起始节点开始，广度遍历所有邻接节点，确保每一层的节点都被访问完后再访问下一层。
3. **任务2：社交网络路径查询**：
   * ass2()函数同样会检查graph2.txt文件是否成功加载，并让用户输入两个节点（表示两个人）。
   * 程序会调用图的startToEnd()方法，计算从第一个节点到第二个节点的最短路径，并输出路径上的节点顺序。
   * 在社交网络模型中，路径的最短距离即为两个人之间需要最少经过多少人才能建立联系。

#### 2. 关键算法设计

在程序中，核心的算法包括深度优先搜索（DFS）、广度优先搜索（BFS）和社交网络路径查询。每种算法的设计理念如下：

1. **深度优先搜索（DFS）**：
   * **功能**：在无向图的邻接矩阵中实现深度优先搜索，从给定的起始节点出发，递归地访问所有相邻且未访问的节点，直到所有可能的路径都被遍历过。
   * **过程**：DFS会在访问每个节点时将其加入到路径中。当一个路径遍历完成后，程序会回溯到上一节点，继续尝试其他路径。该算法适合用于图的连通性检查或路径查找。
2. **广度优先搜索（BFS）**：
   * **功能**：在无向图的邻接表中实现广度优先搜索，从起始节点出发，先访问与起始节点直接相连的节点，再按顺序访问这些节点的相邻节点，直到所有节点都被访问。
   * **过程**：BFS通过使用队列来保证按照层次逐层访问节点。每次访问一个节点时，将其所有未访问的邻接节点加入队列，直到队列为空为止。
3. **社交网络路径查询**：
   * **功能**：根据输入的任意两个人的信息（即图中的两个节点），计算并输出两者之间的最短路径，路径上的节点表示从一个人到另一个人之间的最短联系人数。
   * **过程**：社交网络的路径查询通过遍历算法（DFS或BFS）实现，在计算路径时会记录访问过的节点和路径。最短路径通常通过BFS来实现，因为BFS能够保证找到从起始节点到目标节点的最短路径。

#### 3. 难点分析

1. **图的存储方式**：
   * 图的存储方式包括邻接矩阵和邻接表，邻接矩阵适用于密集图（边较多的图），而邻接表适用于稀疏图（边较少的图）。在实现过程中，需要根据图的实际情况选择合适的数据结构。
   * 在程序设计中，如何高效地处理图的数据存储和遍历是一个挑战，特别是在处理大规模图时，内存的消耗和访问效率需要重点考虑。
2. **路径查找的实现**：
   * 在startToEnd()函数中，社交网络的路径查询涉及到深度优先搜索（DFS）或广度优先搜索（BFS）来查找最短路径。虽然DFS可以查找所有路径，但要确保找到最短路径，BFS是更合适的选择。如何保证在遍历过程中正确记录路径和最短路径是一个重要问题。
3. **用户输入和错误处理**：
   * 用户输入是程序与外界交互的主要方式，需要确保输入的合法性和有效性。例如，起始节点和目标节点的输入需要检查是否在图的范围内，且不应访问已访问的节点。对于非法输入，需要给出清晰的错误提示，并让用户重新输入。

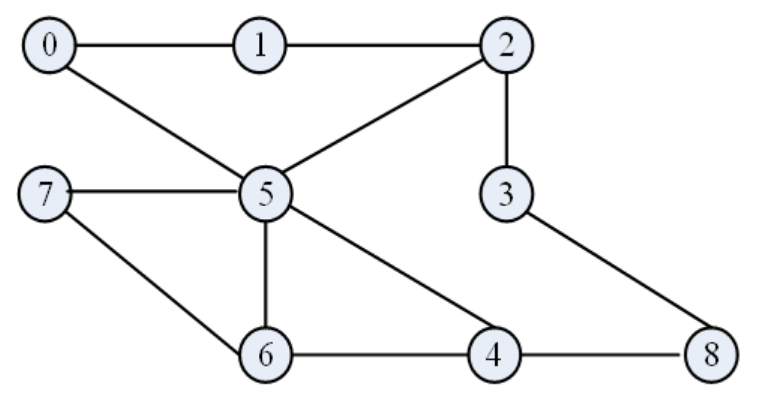
#### 4. 模块和类之间的关系

* **Graph类**：该类包含了图的基本数据结构（邻接矩阵和邻接表）和操作（如DFS、BFS、路径查询等）。它是程序的核心类，负责图的表示和图算法的实现。
* **主函数（main）**：主函数是程序的入口，负责加载文件、获取用户输入并调用Graph类的各个功能模块，确保程序流程的顺畅。

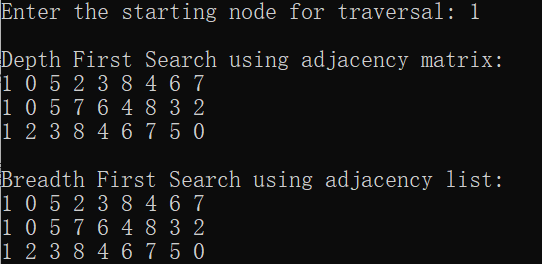
总体来说，本系统通过模块化设计和合理的类之间关系，能够有效实现图的遍历和路径查询功能，满足实验任务的需求。

## 四、实验分析与探讨

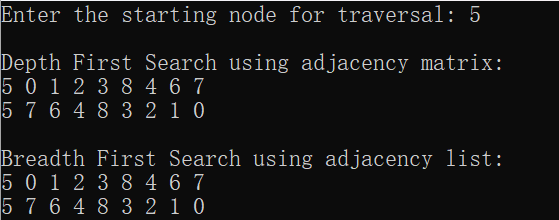
无向图如下，graph1和graph2文本文件中均记录该图，测试程序功能性



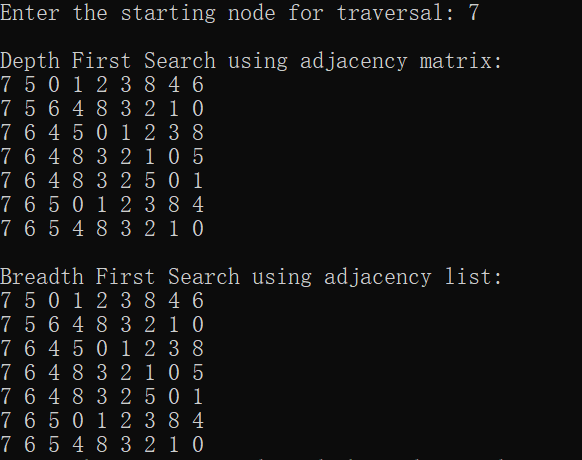
测试样例一：输入1，即测试该图从1节点开始在邻接矩阵结构下对该图进行深度优先搜索；在邻接表结构下对该图进行广度优先搜索，两者均打印出搜索所得路径，用于测试题目1的（1）（2）（3）功能，输出正确。



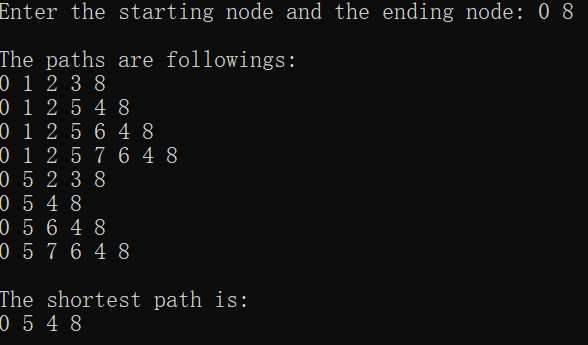
测试样例二：输入5，即测试该图从5节点开始在邻接矩阵结构下对该图进行深度优先搜索；在邻接表结构下对该图进行广度优先搜索，两者均打印出搜索所得路径，用于测试题目1的（1）（2）（3）功能，输出正确。



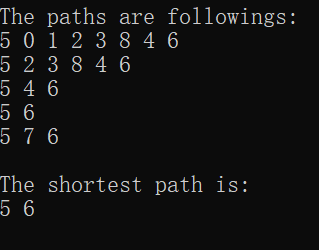
测试样例三：输入7，即测试该图从7节点开始在邻接矩阵结构下对该图进行深度优先搜索；在邻接表结构下对该图进行广度优先搜索，两者均打印出搜索所得路径，用于测试题目1的（1）（2）（3）功能，输出正确。



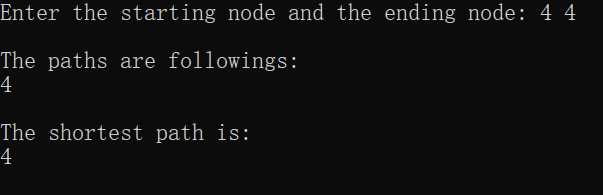
测试样例4：在输入7的基础上，再输入0和8，即测试该图的社交网络模型模块，测试该模块的（1）（2）功能，打印出从0-8所有路径和最短路径，输出正确。



测试样例5：在输入7的基础上，再输入5和6，即测试该图的社交网络模型模块，测试该模块的（1）（2）功能，打印出从5-6所有路径和最短路径，输出正确。



测试样例6,：在输入7的基础上，再输入4和4，即测试该图的社交网络模型模块，测试该模块的（1）（2）功能，打印出从4-4所有路径和最短路径，输出正确。



实验结果分析,：本次实验实现了无向图的深度优先，广度优先搜索和最短路径搜索三个内容，对于问题二，笔者认为没有用prim算法的必要，因为仅仅针对该题目来说，无法定义无向加权图的边权，所以无法使用该算法。当然涉及到最短路径的题目是需要prim算法或dijkstra算法的，这里笔者的处理是借用问题一的算法对边界条件进行了一些修改。

对于本次实验主要算法复杂度的分析，若图有V个节点和E条边，

 时间复杂度：

邻接矩阵构建：O(V^2) 邻接表构建：O(E)

DFS（邻接矩阵）：O(V^2) DFS（邻接表）：O(V + E)

BFS：O(V + E) 最短路径查询：O(V!)（最坏情况）

 空间复杂度：

邻接矩阵：O(V^2) 邻接表：O(V + E)

DFS：O(V) BFS：O(V)

最短路径查询：O(V)

## 五、小结

#### 1. 设计实现过程的小结

在本次设计中，我主要完成了两个任务：

1. **无向图的实现与遍历**：
   * 通过邻接矩阵和邻接表两种方式实现了无向图的存储，并设计了深度优先搜索（DFS）和广度优先搜索（BFS）算法，能够从给定的节点开始，遍历图中的所有节点。
   * 深度优先搜索使用递归方式遍历节点，广度优先搜索则通过队列实现层次遍历。
2. **社交网络模型与路径查询**：
   * 设计并实现了一个社交网络模型，其中每个人的信息表示为图中的一个顶点，人与人之间的联系表示为图中的一条边。
   * 实现了从任意两个人的信息（即两个节点）计算最短路径的功能，帮助确定两人之间最少需要经过多少人才能建立联系。

#### 2. 未完成部分与选做部分

在项目中，所有的基本要求和选做部分均已完成：

* **基本功能**：无向图的DFS与BFS遍历、社交网络的路径查询功能都已实现，并且能够根据用户输入的节点有效计算路径。
* **选做部分**：程序设计了图的两种存储方式（邻接矩阵和邻接表），在实际实现中，根据具体图的稠密程度选择合适的数据结构。该选做部分已经完成。

目前的程序实现了预期功能，但仍然可以在以下方面进一步完善：

* **输入文件的格式和错误处理**：
  + 对于输入文件graph1.txt和graph2.txt，文件格式较为简单，仅包含节点和边的信息，尚未考虑复杂的文件格式或图的多重边、环等特殊情况。未来可以考虑扩展文件格式，加入对多重边、带权边等更复杂图结构的支持。
  + 输入文件路径的异常处理也可以进一步完善，避免因文件路径不正确导致程序异常。
* **性能优化**：
  + 当前的图遍历算法（DFS和BFS）适用于小规模的图，对于大规模图的性能优化还有很大的提升空间。例如，当前实现中路径查询的最短路径查找通过DFS递归进行，效率较低，未来可以考虑更高效的算法，如Dijkstra或A\*算法。

#### 3. 程序的局限性

尽管该程序能满足实验的基本要求，但仍存在以下局限性：

* **图的规模限制**：目前程序采用了固定的最大节点数MaxSize，这限制了图的规模。若需要处理更大规模的图，程序需要改进内存管理，动态调整存储结构或使用更高效的图存储方式（如邻接链表、边列表等）。
* **路径查找的效率**：路径查找部分，目前使用的DFS方法可能导致性能瓶颈，特别是对于大图或稠密图，性能较差。可以考虑采用更高效的算法来提高查询效率。
* **图的扩展性**：当前程序仅支持无向图，若需要支持带权图、有向图等其他类型图结构，需要进一步扩展功能。

#### 4. 必要的补充说明

* **输入文件格式**：
  + graph1.txt：该文件应包含两个整数，表示节点数和边数，后续每一行包含两个整数，表示一条边的两个节点（无向图）。
  + graph2.txt：该文件与graph1.txt类似，用于任务2的社交网络模型。
* **程序的使用说明**：
  + 程序启动后，会首先加载graph1.txt和graph2.txt，然后要求用户输入起始节点，进行DFS和BFS遍历。接着，用户可以输入两个节点，查询它们之间的最短路径。
  + 在使用过程中，确保输入的节点编号在图的有效范围内，否则会提示错误并要求重新输入。

#### 5. 展望与进一步学习

未来，我计划进一步学习和完善以下方面：

* **图的更多算法**：除了DFS和BFS之外，我还可以学习和实现更复杂的图算法，如最短路径算法（Dijkstra算法、Bellman-Ford算法等）以及图的最小生成树算法（Kruskal、Prim算法等）。
* **数据结构的优化**：如针对稀疏图使用邻接表存储，针对密集图使用邻接矩阵，并学习如何在大规模数据中优化图的存储与遍历。
* **图的可视化**：未来可以尝试图形化显示图的结构，通过可视化工具帮助用户更直观地理解图的结构和遍历过程。

总之，整个项目让我对图的基本数据结构和图算法有了更深刻的理解，也让我意识到图算法在实际应用中的强大功能。在今后的学习中，我将继续优化和扩展图的应用，提升编程和算法分析能力。

## 附录：源代码

**1、实验环境：Dev-C++ 5.11 C++11标准**

2、

（1）//Graph.h

#ifndef GRAPH\_H

#define GRAPH\_H

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

#include <queue>

using namespace std;

const int MaxSize = 100;

struct EdgeNode

{

int adjvex;

EdgeNode \*next;

EdgeNode(int x): adjvex(x), next(nullptr) {}

};

class Graph

{

private:

vector<EdgeNode\*> vertex;

int edge[MaxSize][MaxSize];

int vertexNum, edgeNum;

void adjMatrixDfsHelper(int node, vector<int> &path, bool visited[]);

void adjLineBfsHelper(int node);

void startToEndHelper(int startNode, int endNode, vector<int> &path, vector<int> &shortestPath, bool visited[]);

public:

Graph(ifstream &ifs);

void adjMatrixDfs(int node);

void adjLineBfs(int node);

void startToEnd(int startNode, int endNode);

};

#endif

（2）//Graph.cpp

#include "Graph.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

#include <queue>

using namespace std;

Graph::Graph(ifstream &ifs)

{

string line;

getline(ifs, line);

istringstream ss(line);

ss >> vertexNum >> edgeNum;

while(getline(ifs, line))

{

ss.clear();

ss.str(line);

int pa, pb;

ss >> pa >> pb;

edge[pa][pb] = 1;

edge[pb][pa] = 1;

}

for(int i = 0; i < vertexNum; i ++)

{

vertex.push\_back(new EdgeNode(i));

EdgeNode \*p = vertex[i];

for(int j = 0; j < vertexNum; j ++)

{

if(edge[i][j] == 1)

{

p->next = new EdgeNode(j);

p = p->next;

}

}

}

}

void Graph::adjMatrixDfsHelper(int node, vector<int> &path, bool visited[])

{

if(visited[node])

return;

visited[node] = 1;

path.push\_back(node);

if(path.size() == vertexNum)

{

for(int i = 0; i < vertexNum; i ++)

cout << path[i] << " ";

cout << endl;

}

for(int j = 0; j < vertexNum; j ++)

{

if(edge[node][j] == 1)

adjMatrixDfsHelper(j, path, visited);

}

visited[node] = 0;

path.pop\_back();

}

void Graph::adjMatrixDfs(int node)

{

bool visited[MaxSize] = {0};

vector<int> path;

adjMatrixDfsHelper(node, path, visited);

}

void Graph::adjLineBfsHelper(int node)

{

queue<vector<bool> > v;

queue<vector<int> > q;

vector<int> path;

path.push\_back(node);

vector<bool> visited(vertexNum, 0);

visited[node] = 1;

v.push(visited);

q.push(path);

while(!q.empty())

{

path = q.front();

visited = v.front();

q.pop();

v.pop();

if(path.size() == vertexNum)

{

for(int i = 0; i < vertexNum; i ++)

cout << path[i] << " ";

cout << endl;

continue;

}

int lastNode = path.back();

EdgeNode \*p = vertex[lastNode]->next;

while(p != nullptr)

{

if(visited[p->adjvex])

{

p = p->next;

continue;

}

vector<bool> newVisited = visited;

newVisited[p->adjvex] = 1;

vector<int> newPath = path;

newPath.push\_back(p->adjvex);

v.push(newVisited);

q.push(newPath);

p = p->next;

}

}

}

void Graph::adjLineBfs(int node)

{

adjLineBfsHelper(node);

}

void Graph::startToEndHelper(int startNode, int endNode, vector<int> &path, vector<int> &shortestPath, bool visited[])

{

if(visited[startNode])

return;

visited[startNode] = 1;

path.push\_back(startNode);

if(startNode == endNode)

{

for(int i = 0; i < path.size(); i ++)

cout << path[i] << " ";

cout << endl;

if(path.size() < shortestPath.size())

shortestPath = path;

visited[startNode] = 0;

path.pop\_back();

return;

}

for(int j = 0; j < vertexNum; j ++)

{

if(edge[startNode][j] == 1)

startToEndHelper(j, endNode, path, shortestPath, visited);

}

visited[startNode] = 0;

path.pop\_back();

}

void Graph::startToEnd(int startNode, int endNode)

{

bool visited[MaxSize] = {0};

vector<int> path;

vector<int> shortestPath(vertexNum + 1, 0);

cout << "\nThe paths are followings:" << endl;

startToEndHelper(startNode, endNode, path, shortestPath, visited);

if(shortestPath.size() <= vertexNum)

{

cout << "\nThe shortest path is:" << endl;

for(int i = 0; i < shortestPath.size(); i ++)

cout << shortestPath[i] << " ";

cout << endl;

}

}

（3）//main.cpp

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

#include "Graph.h"

#include <iostream>

#include <string>

#include <cmath>

#include <iomanip>

#include <ctime>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <cstdlib>

#include <map>

#include <fstream>

#include <sstream>

#include <ctime>

#include <queue>

using namespace std;

void ass1(ifstream &ifs)

{

if(!ifs)

{

cerr << "Error: Failed to open the file 'graph1.txt'." << endl;

return;

}

// ´´½¨Í¼

Graph graph(ifs);

int startNode;

cout << "Enter the starting node for traversal: ";

cin >> startNode;

// ¼ì²éÊäÈë½ÚµãÊÇ·ñÓÐÐ§

if(startNode < 0 || startNode >= MaxSize)

{

cerr << "Error: Invalid starting node! Please enter a valid node index (0-" << MaxSize - 1 << ")." << endl;

return;

}

cout << "\nDepth First Search using adjacency matrix:" << endl;

graph.adjMatrixDfs(startNode);

cout << "\nBreadth First Search using adjacency list:" << endl;

graph.adjLineBfs(startNode);

}

void ass2(ifstream &ifs)

{

if(!ifs)

{

cerr << "Error: Failed to open the file 'graph2.txt'." << endl;

return;

}

// ´´½¨Í¼

Graph graph(ifs);

int startNode, endNode;

cout << "Enter the starting node and the ending node: ";

cin >> startNode >> endNode;

if(startNode < 0 || startNode >= MaxSize)

{

cerr << "Error: Invalid starting or ending node! Please enter a valid node index (0-" << MaxSize - 1 << ")." << endl;

return;

}

graph.startToEnd(startNode, endNode);

}

int main()

{

ifstream ifs1("graph1.txt");

ass1(ifs1);

ifstream ifs2("graph2.txt");

ass2(ifs2);

return 0;

}