南京邮电大学

实 验 报 告

（2024 / 2025 学年 第 一 学期）

课程名称	离散数学
实验名称	图的随机生成及欧拉（回）路的确定
实验时间	2024 年 11 月 13 日
指导单位	计算机学院计算机科学与技术系
指导教师	柯昌博

学生姓名	于明宏	班级学号	B23041011
学院(系)	计算机学院	专　　业	信息安全

实 验 报 告

实验名称	图的随机生成及欧拉（回）路的确定			指导教师	柯昌博
实验类型	验证	实验学时	4	实验时间	2024.11.13

一、 实验目的和要求

为了进一步理解图的表示方法和欧拉（回）路的判定定理，具体的实验要求如下：

程序的能够根据输入的 n 和 m，随机生成具有 n 个结点 m 个边的简单无向图（能够判断 n 和 m 的合理性），然后判断图的连通性，如果这个图是个连通图，再计算图中度数是奇数的结点个数，判断是欧拉图还是半欧拉图，如果是欧拉图或者半欧拉图，然后根据输入打印一个欧拉(回)路,或者所有的欧拉(回)路。

【其他要求】

1)变量、函数命名符合规范。

2)注释详细：每个变量都要求有注释说明用途；函数有注释说明功能，对参数、返回值也要以注释的形式说明用途；关键的语句段要求有注释解释。

3)程序的层次清晰，可读性强。

二、实验环境(实验设备)

硬件：微型计算机

软件：Windows 操作系统、Microsoft Visual C++ 2022

三、实验原理及内容

1、这个程序实现了一个用于分析无向图的连通性的工具。在输入要生成的无向图的结点数和边数后，随机生成一个符合该条件的图，将其以邻接矩阵的形式输出。随后，判断连通性，输出其可达矩阵。最后，判断是否为欧拉图或者半欧拉图，如果是，则输出一个可行的欧拉(回)路。

2、C++源代码：

```
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>
using namespace std;

class EXP4 {
private:
    vector<vector<int>> graph;                // Adjacency matrix for the original random graph
    vector<vector<int>> connectedMatrix;    // Adjacency matrix for connections
    vector<vector<int>> reachableMatrix;    // Reachability matrix
    vector<int> degrees;                    // Array to store degrees of nodes
    int n;                                  // Total number of nodes
    int edge;                               // Total number of edges
    int sumDegrees;                         // Sum of all degrees
    int has;                                        // Flag to indicate if Euler path/circuit is found
    vector<vector<int>> vis;                // Temporary matrix for visited edges
    vector<int> node;                       // Array to store nodes in Euler path
    int count;                                  // Number of nodes in the current Euler path

public:
    EXP4(int size)
        :       graph(size,       vector<int>(size)),       connectedMatrix(size, vector<int>(size)),
        reachableMatrix(size,  vector<int>(size)),  degrees(size,  0),  vis(size, vector<int>(size, 0)),
        node(size, 0), n(0), edge(0), sumDegrees(0), has(0), count(0) {}
```

```
        // Generates a random adjacency matrix for an undirected graph with n nodes and m edges
        void randGraph(int n, int m) {
            srand(time(0));
            EXP4::n = n;
            graph = vector<vector<int>>(n, vector<int>(n, 0));    // Initialize adjacency matrix with 0s
            int edgesAdded = 0;
            while (edgesAdded < m) {
                int i = rand() % n;
                int j = rand() % n;
                if (i != j && graph[i][j] == 0) {  // Avoid self-loops and duplicate edges
                    graph[i][j] = 1;
                    graph[j][i] = 1;  // Ensure symmetry for undirected graph
                    edgesAdded++;
                }
            }
            connectedMatrix = graph;
        }

        // Matrix addition
        vector<vector<int>> addition(const vector<vector<int>>& m1, const vector<vector<int>>& m2) {
            vector<vector<int>> tmp(n, vector<int>(n, 0));
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    tmp[i][j] = m1[i][j] + m2[i][j];
                }
            }
            return tmp;
        }

        // Matrix multiplication
```

```
        vector<vector<int>> multiplication(const vector<vector<int>>& m1, const vector<vector<int>>& m2) {
            vector<vector<int>> tmp(n, vector<int>(n, 0));
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    for (int k = 0; k < n; k++) {
                        tmp[i][j] += m1[i][k] * m2[k][j];
                    }
                }
            }
            return tmp;
        }

        // Calculate the reachability matrix
        void getReachableMatrix() {
            reachableMatrix = graph;
            for (int i = 0; i < n - 1; i++) {
                graph = multiplication(graph, connectedMatrix);
                reachableMatrix = addition(graph, reachableMatrix);
            }
        }

        // Check if the graph is reachable (i.e., all nodes are accessible from any other node)
        bool isReachable() {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (reachableMatrix[i][j] == 0) return false;
                }
            }
            return true;
        }

        // Convert the reachability matrix to a binary matrix
        void unitization() {
```

```
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (reachableMatrix[i][j] != 0) reachableMatrix[i][j] = 1;
            }
        }
    }

    // Check if the graph is an Eulerian graph or semi-Eulerian
    int isEulerMap() {
        if (n == 0) return 0;
        if (!isReachable()) return 0;
        int num = 0;
        for (int i = 0; i < n; i++) {
            if (degrees[i] % 2 == 1) num++;
        }
        if (num == 2) return 1;  // Semi-Eulerian
        if (num == 0) return 2;  // Eulerian
        return 0;
    }

    // Calculate the degrees of all nodes
    void getDegrees() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (connectedMatrix[i][j] != 0) degrees[i]++;
            }
        }
        for (int i = 0; i < n; i++) sumDegrees += degrees[i];
        edge = sumDegrees / 2;
    }

    // Print the reachability matrix
    void printReachableMatrix() {
        cout << "Reachable matrix:" << endl;
        for (int i = 0; i < n; i++) {
```

```
                for (int j = 0; j < n; j++) {
                    cout << reachableMatrix[i][j] << " ";
                }
                cout << endl;
            }
            cout << endl;
        }

        // Print the adjacency matrix
        void printConnectedMatrix() {
            cout << "Connected matrix:" << endl;
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    cout << connectedMatrix[i][j] << " ";
                }
                cout << endl;
            }
            cout << endl;
        }

        // Core algorithm to find the Eulerian path or circuit
        void getEulerMap(int currentIndex) {
            if (has == 1) return;
            if (count == edge + 1) {
                for (int i = 0; i < count; i++) {
                    if (i == 0) cout << node[i];
                    else cout << " -> " << node[i];
                }
                cout << endl;
                has = 1;
            }
            else {
                for (int i = 0; i < n; i++) {
                    if (connectedMatrix[currentIndex][i] == 1 && vis[currentIndex][i]
== 0) {
```

```
                vis[i][currentIndex] = vis[currentIndex][i] = 1;
                node[count++] = i;
                getEulerMap(i);
                count--;
                vis[i][currentIndex] = vis[currentIndex][i] = 0;
            }
        }
    }
}

// Print the Eulerian path or circuit
void printEulerMap() {
    int result = isEulerMap();
    if (result == 1) {
        cout << "Euler road:" << endl;
        for (int i = 0; i < n; i++) {
            int t = 0;
            for (int j = 0; j < n; j++) {
                if (connectedMatrix[i][j] == 1) t++;
            }
            if (t % 2 == 1) {
                node[count++] = i;
                getEulerMap(i);
                count--;
                break;
            }
        }
    }
    else if (result == 2) {
        cout << "Euler circuit:" << endl;
        node[count++] = 0;
        getEulerMap(0);
        count--;
    }
    else {
```

```
                cout << "Neither Euler road nor Euler circuit is found." << endl;
            }
        }
    };

    int main() {
        int n, m;
        EXP4 e(100);
        cout << "Number of node: ";
        cin >> n;
        cout << "Number of edge: ";
        cin >> m;
        cout << endl;
        if (n <= 0 || m <= 0 || m * 2 > n * n - n) {  // Validate the relationship between nodes and edges
            cout << "Wrong relationship between n and m!" << endl;
            return -1;
        }
        e.randGraph(n, m);          // Generate random adjacency matrix
        e.printConnectedMatrix();    // Print adjacency matrix
        e.getReachableMatrix();      // Compute reachability matrix
        e.unitization();             // Convert reachability matrix to binary form
        e.printReachableMatrix();    // Print reachability matrix
        cout << "This is " << (e.isReachable() ? "a " : "an un") << "reachable graph." << endl;
        e.getDegrees();              // Calculate degrees of nodes
        e.printEulerMap();           // Print Eulerian path or circuit
        return 0;
    }
```

3、运行结果：

```
Number of node: 6
Number of edge: 10

Connected matrix:
0 1 1 0 1 1
```

```
1 0 1 0 0 1
1 1 0 1 0 1
0 0 1 0 1 1
1 0 0 1 0 0
1 1 1 1 0 0

Reachable matrix:
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1
1 1 1 1 1 1

This is a reachable graph.
Euler road:
1 -> 0 -> 2 -> 1 -> 5 -> 0 -> 4 -> 3 -> 2 -> 5 -> 3
```

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

说明：这部分内容主要包括：在编程、调试或测试过程中遇到的问题及解决方法、本次实验的心得体会、进一步改进的设想等。

（一）实验中遇到的主要问题及解决方法

1.问题：在随机生成图的连通性判断中，由于算法对连通性矩阵的更新未完全考虑，导致判断出错。

解决方法：重新检查连通性判断算法，确保在计算可达矩阵时累加所有可能的路径，最终通过调试和测试验证了连通性计算的正确性。

2.问题：在判断欧拉图和半欧拉图时，由于对节点度数的统计和偶数判定未完全覆盖所有情况，导致部分情况下的判定错误。

解决方法：增加节点度数的检查环节，确保在统计节点度数时能够完整覆盖所有节点。同时在偶数判定中进行了改进，确保结果准确无误。

3.问题：在输出欧拉路径时，由于路径回溯过程中的条件设置不严谨，导致路径中存在重复的边。

解决方法：通过设置访问标志矩阵，确保每条边在生成路径的过程中仅使用一次，进而避免了重复问题。

（二）实验心得

通过本次实验，我对图论中的连通性和欧拉图判定方法有了更深入的理解。实验过程中，通过编程实现从图的随机生成到连通性判断再到欧拉路径输出的全过程，使我加深了对理论知识的理解。特别是通过实现欧拉路径的回溯算法，提升了我在数据结构和算法应用方面的能力。同时，这次实验让我认识到在实现复杂算法时注重细节的重要性，通过多次测试确保结果的准确性是十分必要的。

（三）意见与建议（没有可省略）

可以提供更多的时间上机操作，以确保更多程序设计思路得以实现，提升掌握程度和编程能力。

五、支撑毕业要求指标点

支撑毕业要求的指标点为：

☐ 1-4 掌握计算机科学与技术领域的专业知识，能将专业知识用于分析和解决计算机领域复杂工程问题。

☑ 2-1 能够应用数学、自然科学和工程科学的基本知识，识别和分析计算机领域复杂工程问题的特征。

六、指导教师评语 (含学生能力达成度的评价)					
成　绩		批阅人		日　期	

如果不太想写太多字，“指导教师评语”也可以设计为如下的各选择项用打勾形式（仅仅作为一个简单示例，请各课程负责人根据课程和实验情况以及支撑的指标点来自行设定选择项，同一门课程的不同实验评分细则项允许存在不同）：

	评分项	优秀	良好	中等	合格	不合格
评	遵守实验室规章制度					
	学习态度					
	算法思想准备情况					
	程序设计能力					
分	解决问题能力					
	课题功能实现情况					
	算法设计合理性					
	算法效能评价					
	回答问题准确度					
细	报告书写认真程度					
	内容详实程度					
	文字表达熟练程度					
	其它评价意见					
则	本次实验能力达成评价（总成绩）					