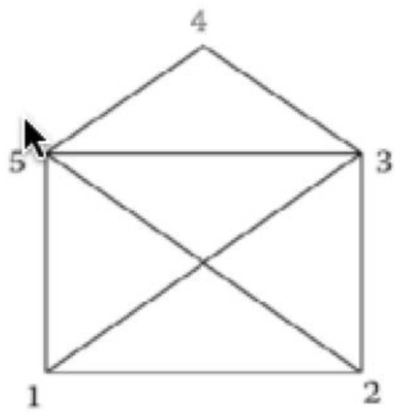
欧拉路径实验报告

2351114 朱俊泽

2.7 题目七 欧拉路径

2.7.1 问题描述

请你写一个程序，从下图所示房子的左下角（数字 1）开始，按照节点递增顺序，输出所有可以一笔画完的顶点顺序（欧拉路径），要求所有的边恰好都只画一次。例如，123153452 就是其中的一条路径。



2.7.2 基本要求

由于题目没有定义输入输出，那么我定义：
输入数据方式为： 输入 n 代表 n 个点 输入 m 代表 m 条边 之后 m 次输入 每次输入点 ai bi 代表两点之间有一条边 输出一条欧拉路径即可。

然后输出一条欧拉路径即可。如果没有就不输出。

2.7.3 数据结构设计

邻接矩阵：使用二维数组 `graph[MAX_N][MAX_N]` 表示图的结构。`graph[i][j]` 表示从节点 i 到节点 j 的边的数量。该设计便于快速检查节点间是否有边，以及移除边的操作。

```
int graph[MAX_N][MAX_N]; // 邻接矩阵存储图
```

欧拉路径存储：使用 `std::vector<int> eulerPath` 存储生成的欧拉路径。由于深度优先搜索的特性，路径是按逆序生成的，最终输出时逆序打印。

```
vector<int> eulerPath;  // 保存欧拉路径
```

辅助变量：n 和 m：节点数和边数。oddCount：记录奇度节点的数量，用于判断是否存在欧拉路径。startNode：欧拉路径的起点，默认是奇度节点，若无奇度节点，则任意节点都可以作为起点。

```
oddCount++;
startNode = i; // 欧拉路径的起点是奇度点
```

DFS 栈：递归调用实现深度优先搜索，隐式维护一个栈，便于按顺序遍历所有未访问的边。

2.7.4 功能说明（函数、类）

图的构建：程序通过 mmm 条边的信息构建无向图的邻接矩阵，支持多条边。

```
// 初始化邻接矩阵
for (int i = 0; i < m; ++i) {
    int a, b;
    cin >> a >> b;
    graph[a][b]++;
    graph[b][a]++; // 无向边
}
```

欧拉路径判定：使用度数统计法，检查每个节点的度数：奇度节点数为 0 或 2 时，存在欧拉路径。奇度节点数超过 2 时，输出 No Euler Path exists.。这个是由图的性质决定的。

```
// 检查图是否存在欧拉路径
int oddCount = 0, startNode = 1;
for (int i = 1; i <= n; ++i) {
    int degree = 0;
    for (int j = 1; j <= n; ++j) {
        degree += graph[i][j];
    }
    if (degree % 2 != 0) {
        oddCount++;
        startNode = i; // 欧拉路径的起点是奇度点
    }
}

if (oddCount != 0 && oddCount != 2) {
    cout << "No Euler Path exists." << endl;
    return 0;
}
```

寻找欧拉路径：使用深度优先搜索从起点出发遍历所有边。每次访问边后，将边从图中删除（将邻接矩阵对应值减 1），确保边仅访问一次。路径按逆序存储在 `eulerPath` 中，最终输出时逆序打印。

```
void dfs(int u) {
    for (int v = 1; v <= n; ++v) {
        while (graph[u][v] > 0) {
            graph[u][v]--;
            graph[v][u]--; // 无向图，两边都标记为访问过
            dfs(v);
        }
    }
    eulerPath.push_back(u);
}
```

2.7.5 调试分析（遇到的问题和解决方法）

注意是无向图，一开始输入边的 graph 操作是：

```
Cin>>a>>b;
Graph[a][b]++;
```

2.7.6 总结和体会

算法复杂度分析

时间复杂度：

1. 构建邻接矩阵：O(m)O(m)O(m)，其中 mmm 是边数。
2. 深度优先搜索：最多访问所有边一次，时间复杂度为 O(m)O(m)O(m)。
3. 总时间复杂度为 O(n2+m)O(n^2 + m)O(n2+m)，其中 n2n^2n2 来自邻接矩阵的存储。

空间复杂度：

1. 邻接矩阵占用 O(n2)O(n^2)O(n2) 空间。
2. 欧拉路径的栈占用 O(m)O(m)O(m) 空间。
3. 总空间复杂度为 O(n2)O(n^2)O(n2)。

2.7.7 代码

```
#include <iostream>

#include <vector>

#include <stack>


using namespace std;


const int MAX_N = 100; // 最大节点数
```

```
int n, m;                    // 节点数和边数
int graph[MAX_N][MAX_N]; // 邻接矩阵存储图
vector<int> eulerPath;    // 保存欧拉路径

// 深度优先搜索寻找欧拉路径
void dfs(int u) {
    for (int v = 1; v <= n; ++v) {
        while (graph[u][v] > 0) {
            graph[u][v]--;
            graph[v][u]--; // 无向图，两边都标记为访问过
            dfs(v);
        }
    }
    eulerPath.push_back(u);
}

int main() {
    cin >> n >> m;

    // 初始化邻接矩阵
    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        graph[a][b]++;
        graph[b][a]++; // 无向边
    }

    // 检查图是否存在欧拉路径
    int oddCount = 0, startNode = 1;
    for (int i = 1; i <= n; ++i) {
        int degree = 0;
        for (int j = 1; j <= n; ++j) {
            degree += graph[i][j];
        }
        if (degree % 2 != 0) {
            oddCount++;
            startNode = i; // 欧拉路径的起点是奇度点
        }
    }

    if (oddCount != 0 && oddCount != 2) {
        cout << "No Euler Path exists." << endl;
        return 0;
    }
```

```
    // 找到欧拉路径
    dfs(startNode);

    // 输出欧拉路径
    for (int i = eulerPath.size() - 1; i >= 0; --i) {
        cout << eulerPath[i];
        if (i > 0) cout << " ";
    }
    cout << endl;

    return 0;
}
```