

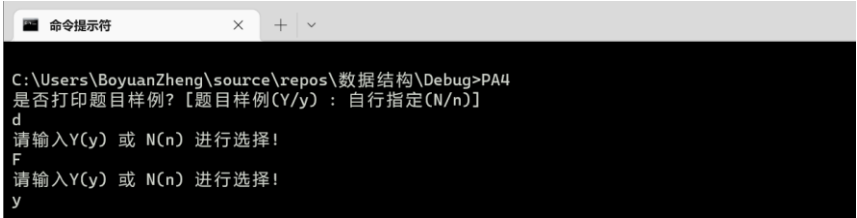
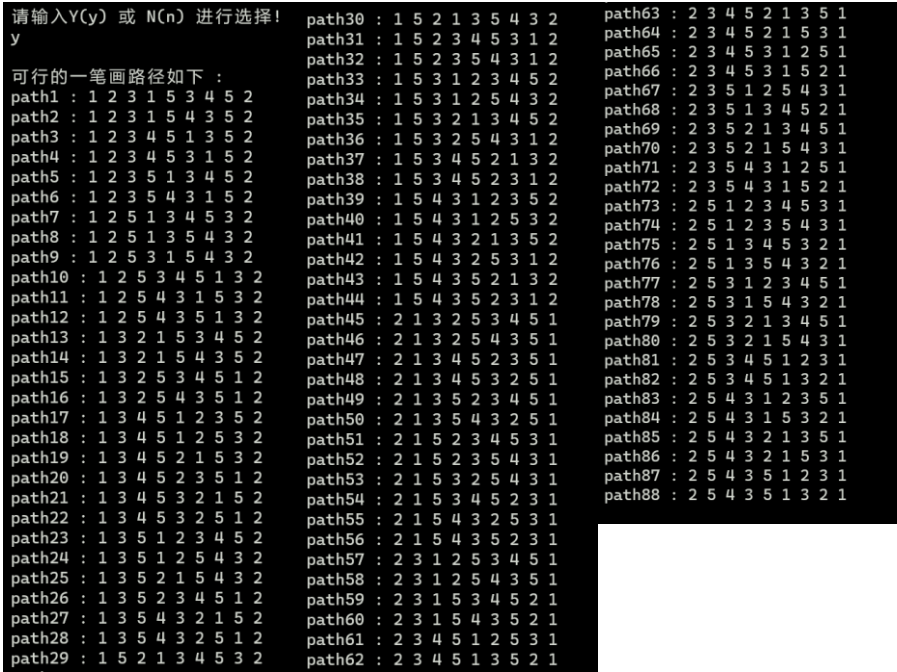
《数据结构》上机报告

2022 年 11 月 19 日

姓名：郑博远 学号：2154312 班级：计科 1 班 得分：_____

实验题目	欧拉路径（图的一笔画问题）	
实验目的	1. 掌握图的存储结构和基本操作； 2. 灵活运用图的遍历方法。	
问题描述	<p>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。</p> <div data-bbox="702 869 1077 1220"></div> <p>图 1 题面“房子”示意图</p>	
基本要求	<p>1. 图的存储：可以使用邻接矩阵 <code>map[][]</code> 存储此图，其中 <code>map</code> 的对角线 (<code>map[1][1]</code>, <code>map[2][2]</code>, <code>map[3][3]</code>, <code>map[4][4]</code>, <code>map[5][5]</code>) 和 <code>map[1][4]</code>, <code>map[4][1]</code>, <code>map[2][4]</code>, <code>map[4][2]</code> 为 0，其余元素为 1；</p> <p>2. 深度优先搜索：从顶点 1 开始，沿着一条未走过的边找到一个新的顶点，继续走一条未走过的边；当没有未走过的边时，则退回到上一个顶点，继续试探未走过的边，直到所有的边都被访问。</p> <p>3. 深度优先搜索 (DFS) 是个递归的过程。</p>	
选做要求		
	已完成选做内容（序号）	

数据结构设计	<p>使用邻接矩阵的方式来存储本题中的图。若两点之间（假设为 i、j）存在边则 $\text{map}[i][j] = 1$，否则 $\text{map}[i][j] = 0$。题目中的图结构存储的矩阵如下：</p> $\text{graph.map} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$ <p>具体数据结构如下：</p> <pre>struct Graph { //记录结点、边数 int vex_num, arc_num; //邻接矩阵 bool map[MAX_VEX_NUM + 1][MAX_VEX_NUM + 1]; };</pre>
功能(函数)说明	<pre>/** * @brief 深度优先搜索遍历图（用于判断图是否联通） * @param graph 判断的图 * @param cur 当前的结点 * @param vis 存储结点是否被访问过 */ void dfsGraph(const Graph& graph, int cur, bool* vis) /** * @brief 判断当前图是否存在可行的欧拉路径 * @param graph 判断的图 */ bool isEulerPathExist(const Graph& graph) /** * @brief 按照 "顶点 边数 边具体信息" 的方式输入图 * @param graph 输入的图 */ void inputGraph(Graph& graph) /** * @brief 深搜打印欧拉路径 * @param graph 遍历的图 * @param pst 当前所在端点</pre>

	<pre>* @param dep 扩展的深度 * @param s 输出的序列 */ void printEulerPath(Graph& graph, int pst, int dep, string s)</pre>
界面设计和 使用说明	<p>进入程序后，首先选择打印题目样例或自行指定图查找欧拉路径。若选择“Y”或“y”则程序以题目样例为数据的图进行欧拉路径的寻找，打印所有的一笔画路径；若选择“N”或“n”则程序进入手动输入模式。若输入的数字非以上两种选项，则程序给出输入错误提示，重复读入直至正确，如图 2 所示。</p> <div></div> <p>图 2 程序的模式选择与输入错误处理</p> <p>选择“Y”或“y”进入题目样例模式，程序根据内置的题目样例输出所有可行的欧拉路径共 88 条。</p> <div></div> <p>图 3 根据题目样例输出的 88 条欧拉路径</p>

选择“N 或“n”进入自行指定模式，分别输入顶点数、边数以及边的具体信息见图。若存在欧拉路径，则程序输出所有可能的欧拉路径；否则，程序给出错误提示。

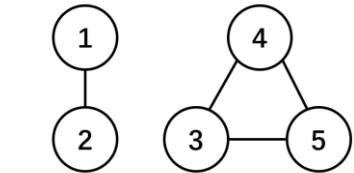
```
命令提示符
C:\Users\BoyuanZheng\source\repos\数据结构\Debug>PA4
是否打印题目样例?【题目样例(Y/y)：自行指定(N/n)】
n
请输入顶点数：1000
顶点个数应该不超过20!
请输入顶点数：F
输入格式非法!
请输入顶点数：3
请输入边数：3
请输入3组以边连接的顶点序号：
1 2
2 3
1 3

可行的一笔画路径如下：
path1：1 2 3 1
path2：1 3 2 1
path3：2 1 3 2
path4：2 3 1 2
path5：3 1 2 3
path6：3 2 1 3
```

图 4 程序的自行指定模式下的输入错误处理

若输入的图不存在欧拉路径，则程序给出提示。下方图 5 展示了两种不存在欧拉路径的情况：图非联通图，或图中边数为奇数的点数并非 0 或 2（奇数边数的点只能对应起点或终点）。

```
命令提示符
C:\Users\BoyuanZheng\source\repos\数据结构\Debug>PA4
是否打印题目样例?【题目样例(Y/y)：自行指定(N/n)】
N
请输入顶点数：5
请输入边数：4
请输入4组以边连接的顶点序号：
1 2
3 4
4 5
3 5
当前的图不存在可行的一笔画路径！
```



```
命令提示符
C:\Users\BoyuanZheng\source\repos\数据结构\Debug>PA4
是否打印题目样例?【题目样例(Y/y)：自行指定(N/n)】
N
请输入顶点数：5
请输入边数：5
请输入5组以边连接的顶点序号：
1 2
2 3
2 4
3 4
4 5
当前的图不存在可行的一笔画路径！
```

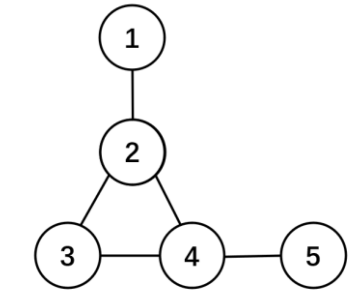
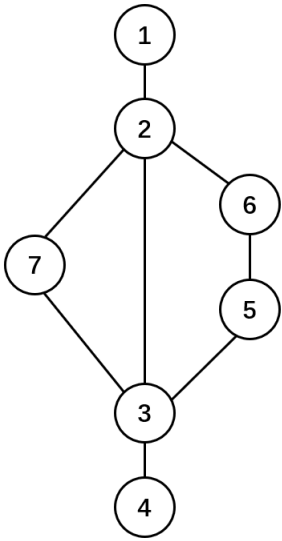


图 5 不存在欧拉路径的两种情况及程序处理

	<div><div><pre>命令提示符 C:\Users\BoyuanZheng\source\repos\数据结构\Debug>pa4 是否打印题目样例? [题目样例(Y/y) : 自行指定(N/n)] n 请输入顶点数 : 7 请输入边数 : 8 请输入8组以边连接的顶点序号 : 1 2 2 6 2 7 7 3 6 5 3 5 3 4 2 3 可行的一笔画路径如下 : path1 : 1 2 3 5 6 2 7 3 4 path2 : 1 2 3 7 2 6 5 3 4 path3 : 1 2 6 5 3 2 7 3 4 path4 : 1 2 6 5 3 7 2 3 4 path5 : 1 2 7 3 2 6 5 3 4 path6 : 1 2 7 3 5 6 2 3 4 path7 : 4 3 2 6 5 3 7 2 1 path8 : 4 3 2 7 3 5 6 2 1 path9 : 4 3 5 6 2 3 7 2 1 path10 : 4 3 5 6 2 7 3 2 1 path11 : 4 3 7 2 3 5 6 2 1 path12 : 4 3 7 2 6 5 3 2 1</pre></div><div></div></div> <div><p>图 6 程序输出所有存在的欧拉路径</p><p>图 6 展示了在存在欧拉路径的图相关信息输入后，程序在屏幕输出所有可行的所有一笔画路径。输入图中 7 个顶点、8 条边的对应信息后，程序给出“1 2 3 5 6 2 7 3 4”等可行的一笔画欧拉路径，以顶点的数字顺序依次列出。</p></div>
调试分析	<div><p>1. 存储图时，将结点的序号直接对应数组的下标，从而无需另外开变量进行存储。但题目中结点的下标从 1 开始，在程序中写入 map 数组时我忽略了这一点，导致了点与点之间连接边关系的错位。将下标为 0 的位置空出即可。后来由于要统计每个结点相邻的边数以判断图是否存在欧拉路径，我将每个结点所连接的边数存储在数组下标为 0 的位置处；</p><p>2. 结点的序号有可能大于 10，而先前的存储方式是将数字加上字符 0 转为对应的 ASCII 码加到字符串的末尾，这会导致大于 9 的数字输出出错。因此需要考虑 10 以上数字，优化转换方式。</p></div>
心得体会	<p>本次求一笔画的欧拉路径问题是对深度优先搜索（DFS）的一次简单应用，难度不大。即从顶点 1 开始，沿着一条未走过的边找到一个新的顶点，继续走一条未走过的边；当没有未走过的边时，则退回到上一个顶点，继续试探未走过的边，直到所有的边都被访</p>

问。当走完所有的边时，即可输出。这样便能输出所有的欧拉路径。

需要注意的是，不是所有图都存在欧拉路径。若图并非连通图，则显然不能通过一笔画完成，不存在欧拉路径。连通图的判定方式有深度优先搜索、广度优先搜索、并查集等方法，本次选择使用深搜实现。连通图中，若奇顶点（所连边数为奇数的顶点）个数为 0 或 2，则图存在欧拉路径。这是因为奇顶点由于进出次数不相等，只能作为图遍历时的起点或终点，因此非 0 或 2 个奇顶点的图不存在欧拉路径。我在输入时将每个顶点所连接的边记录下来，最后便可以 $O(n)$ 遍历所有结点来判断奇顶点的个数是否符合条件。

本次程序中将深度优先搜索 DFS 进行了两次运用，分别是判断图是否为连通图，以及遍历输出所有可行的欧拉路径。但由于使用方式不同，二者的时间复杂度有较大差异。

首先，分析 DFS 算法在判断连通图时的时间复杂度。由于每个点遍历后都将 vis 标记为 1，因此每个点（若是连通图能被遍历到）都有且仅有一次被遍历。而由于使用邻接矩阵存储，每次寻找下一个点都要 $O(n)$ 遍历所有点判断与当前点之间是否存在边，所以总的遍历时间复杂度为 $O(n^2)$ 。

其次，分析 DFS 算法在遍历输出可行欧拉路径时的时间复杂度。若设图中有 v 个结点与 e 条边，则遍历寻找欧拉路径时，需要遍历图中的 e 条边。每次遍历一条边时，都要在邻接矩阵寻找新的起点和终点之间是否存在边，即每次都需要遍历 v 个结点寻找。因此，使用邻接矩阵方式 DFS 遍历的时间复杂度为 $O(v^e)$ 。

附.完整代码

```
#include <iostream>
#include <conio.h>
#include <string>
#include <cstring>
using namespace std;

#define MAX_VEX_NUM 20

struct Graph {
    //记录结点、边数
    int vex_num = 5;
    int arc_num = 8;
    //邻接矩阵
    int map[MAX_VEX_NUM + 1][MAX_VEX_NUM + 1] =
    {
        //下标从1开始 下标0记录该点连接的边数
        {},
        {3, 0, 1, 1, 0, 1},
        {3, 1, 0, 1, 0, 1},
        {4, 1, 1, 0, 1, 1},
        {2, 0, 0, 1, 0, 1},
        {4, 1, 1, 1, 1, 0}
    };
};

/**
 * @brief 深度优先搜索遍历图（用于判断图是否联通）
 * @param graph 判断的图
 * @param cur 当前的结点
 * @param vis 存储结点是否被访问过
 */
void dfsGraph(const Graph& graph, int cur, bool* vis)
{
    vis[cur] = true;
    for (int i = 1; i <= graph.vex_num; i++)
        if (graph.map[cur][i] && !vis[i])
            dfsGraph(graph, i, vis);
}
```

```

/**
 * @brief 判断当前图是否存在可行的欧拉路径
 * @param graph 判断的图
 */
bool isEulerPathExist(const Graph& graph)
{
    //先判断图是否连通
    bool vis[MAX_VEX_NUM] = {0};
    dfsGraph(graph, 1, vis);
    for (int i = 1; i <= graph.vex_num; i++)
        if (!vis[i])
            return false;

    //再判断边数为奇数的点个数
    int odd_num = 0;
    for (int i = 1; i <= graph.vex_num; i++) {
        if (graph.map[i][0] % 2 == 1)
            odd_num++;
    }

    if (odd_num == 0 || odd_num == 2)
        return true;
    else
        return false;
}

/**
 * @brief 按照 "顶点 边数 边具体信息" 的方式输入图
 * @param graph 输入的图
 */
void inputGraph(Graph& graph)
{
    while (true) {
        cout << "请输入顶点数 : ";
        cin >> graph.vex_num;
        if (cin.good() && graph.vex_num <= MAX_VEX_NUM)
            break;
        else if (cin.good())
            cout << "顶点个数应该不超过" << MAX_VEX_NUM << "!" << endl;
        else
            cout << "输入格式非法!" << endl;
        cin.clear();
        cin.ignore(1024, '\n');
    }
}

```



```

    }

    while (true) {
        cout << "请输入边数 : ";
        cin >> graph.arc_num;
        if (cin.good())
            break;
        else
            cout << "输入格式非法! " << endl;
        cin.clear();
        cin.ignore(1024, '\n');
    }

    cout << "请输入" << graph.arc_num << "组以边连接的顶点序号 : " << endl;
    memset(graph.map, 0, sizeof(graph.map));
    for (int i = 0; i < graph.arc_num; i++) {
        int src, dst;
        while (true) {
            cin >> src >> dst;
            if (cin.good())
                break;
            else
                cout << "输入格式非法, 请再次输入该边的两顶点!" << endl;
            cin.clear();
            cin.ignore(1024, '\n');
        }
        //0表示两点之间无边, 1表示有边
        graph.map[src][dst] = 1;
        graph.map[dst][src] = 1;
        //下标0处记录每个结点连接的边数
        graph.map[src][0]++;
        graph.map[dst][0]++;
    }
}

/**
 * @brief 深搜打印欧拉路径
 * @param graph 遍历的图
 * @param pst 当前所在端点
 * @param dep 扩展的深度
 * @param s 输出的序列
 */

```

```

void printEulerPath(Graph& graph, int pst, int dep, string s)
{
    static int count = 0;
    if (dep >= graph.arc_num) {
        cout << "path" << ++count << " : " << s << endl;
        return;
    }
    else {
        for (int i = 1; i <= graph.vex_num; i++) {
            if (graph.map[pst][i]) {
                //删除走过的边
                graph.map[pst][i] = 0;
                graph.map[i][pst] = 0;

                printEulerPath(graph, i, dep + 1, s + to_string(i) + '
');

                graph.map[pst][i] = 1;
                graph.map[i][pst] = 1;
            }
        }
    }
}

int main(){
    Graph graph;

    cout << "是否打印题目样例? [题目样例(Y/y) : 自行指定(N/n)]" << endl;
    bool input;
    while (true) {
        char ch = _getche();
        if (ch == 'Y' || ch == 'y') {
            input = false;
            break;
        }
        else if (ch == 'N' || ch == 'n') {
            input = true;
            break;
        }
        cout << endl << "请输入Y(y) 或 N(n) 进行选择!" << endl;
    }
    cout << endl;
}

```

```
if (input)
    inputGraph(graph);

if (isEulerPathExist(graph)) {
    cout << endl << "可行的一笔画路径如下：" << endl;
    for (int i = 1; i <= graph.arc_num; i++)
        printEulerPath(graph, i, 0, to_string(i) + ' ');
}
else
    cout << "当前的图不存在可行的一笔画路径!" << endl;

return 0;
}
```