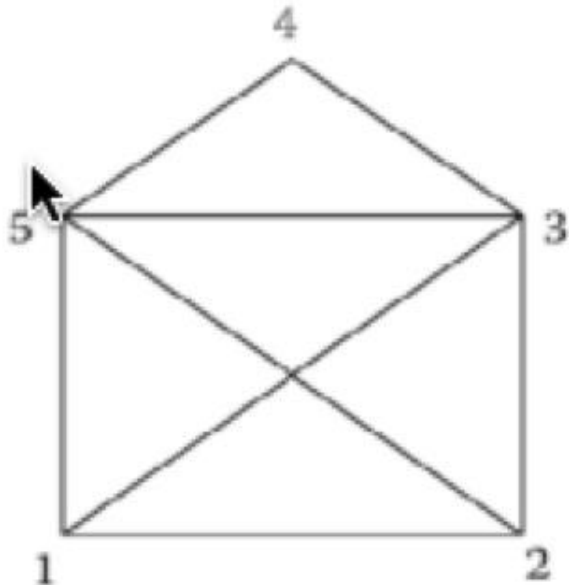


《数据结构》上机报告

2020 年 12 月 11 日

姓名： 王上游 学号： 1850767 班级： 19 计科 2 班 得分：

实验题目	一笔画	
实验目的	1、掌握图的结构和基本操作； 2、灵活运用图的遍历方法。	
问题描述	<p>圣诞节马上到了，我们用一笔画画出圣诞老人的房子吧。现在的问题是，一共有多少种画法呢？</p> <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顺序，要求一条边不能画两次。</p> 	
基本要求	(1) 程序要添加适当的注释，程序的书写要采用 缩进格式 。 (2) 程序要具在一定的 健壮性，即当输入数据非法时， 程序也能适当地做出反应，如 插入删除时指定的位置不对 等等。 (3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。 (4) 根据实验报告模板详细书写实验报告，在实验报告中给出主要算法的复杂度分析。	
选做要求	无	
	已完成选做内容（序号）	无

数据结构设计	<p>程序采用邻接矩阵保存图，一维数组保存各点入度，一维数组保存递归过程中的路径</p> <pre> int G[MAXN][MAXN];//存图 int CNT[MAXN];//存每个点度的奇偶性 int path[MAXM]; </pre>
功能(函数)说明	<pre> void print_path(int path[MAXM]);//打印一条路径 void dfs(int u, int(*g)[MAXN], int path[MAXM], int pp);//dfs 递归深搜所有可能情况 void input_and_init();//输入初始化 void mode1();//输出全部路径 void mode2();//输出指定起点的全部路径 </pre>
界面设计和使用说明	 <p>请输入无向图顶点数:5 请输入无向图边数:8 请输入8条边的起点终点序号，范围[1-5]: 1 2 1 3 1 5 2 3 2 5 3 4 3 5 4 5 1. 输出所有顶点开始全部路径 2. 输出指定顶点开始的全部路径 请输入[1/2]:2 请输入指定起点:1</p> <p>界面设计友好，首先输入图的顶点数、边数。然后依次输入各边信息。最后选择输出图中所有一笔画路径还是指定起点的一笔画路径。按字典序输出所有路径，最后输出路径总数。 程序可移植性强，可以求解无向图图的一笔画路径（欧拉路径）问题。对各种输入错误（范围错误、自环等）均作了错误处理，健壮性强。</p>

程序计算出该图中从 1 处发的一笔画路径共有 44 条。

```
请输入无向图顶点数:5
请输入无向图边数:8
请输入8条边的起点终点序号, 范围[1-5]:
1 2
1 3
1 5
2 3
2 5
3 4
3 5
4 5
1. 输出所有顶点开始全部路径
2. 输出指定顶点开始的全部路径
请输入[1/2]:2
请输入指定起点:1
1 : 1->2->3->1->5->3->4->5->2
2 : 1->2->3->1->5->4->3->5->2
3 : 1->2->3->4->5->1->3->5->2
4 : 1->2->3->4->5->3->1->5->2
5 : 1->2->3->5->1->3->4->5->2
6 : 1->2->3->5->4->3->1->5->2
7 : 1->2->5->1->3->4->5->3->2
8 : 1->2->5->1->3->5->4->3->2
9 : 1->2->5->3->1->5->4->3->2
10 : 1->2->5->3->4->5->1->3->2
11 : 1->2->5->4->3->1->5->3->2
12 : 1->2->5->4->3->5->1->3->2
13 : 1->3->2->1->5->3->4->5->2
14 : 1->3->2->1->5->4->3->5->2
15 : 1->3->2->5->3->4->5->1->2
16 : 1->3->2->5->4->3->5->1->2
17 : 1->3->4->5->1->2->3->5->2
18 : 1->3->4->5->1->2->5->3->2
19 : 1->3->4->5->2->1->5->3->2
20 : 1->3->4->5->2->3->5->1->2
21 : 1->3->4->5->3->2->1->5->2
22 : 1->3->4->5->3->2->5->1->2
23 : 1->3->5->1->2->3->4->5->2
24 : 1->3->5->1->2->5->4->3->2
25 : 1->3->5->2->1->5->4->3->2
26 : 1->3->5->2->3->4->5->1->2
27 : 1->3->5->4->3->2->1->5->2
28 : 1->3->5->4->3->2->5->1->2
29 : 1->5->2->1->3->4->5->3->2
30 : 1->5->2->1->3->5->4->3->2
31 : 1->5->2->3->4->5->3->1->2
32 : 1->5->2->3->5->4->3->1->2
33 : 1->5->3->1->2->3->4->5->2
34 : 1->5->3->1->2->5->4->3->2
35 : 1->5->3->2->1->3->4->5->2
36 : 1->5->3->2->5->4->3->1->2
37 : 1->5->3->4->5->2->1->3->2
38 : 1->5->3->4->5->2->3->1->2
39 : 1->5->4->3->1->2->3->5->2
40 : 1->5->4->3->1->2->5->3->2
41 : 1->5->4->3->2->1->3->5->2
42 : 1->5->4->3->2->5->3->1->2
43 : 1->5->4->3->5->2->1->3->2
44 : 1->5->4->3->5->2->3->1->2
共有上述共44条从1开始的路径
```

心得体会	<p>(对整个实验过程做出总结, 对重要的算法做出性能分析。)</p> <p>一笔画问题的实质为无向图的欧拉路径问题。无向图欧拉路径有一个重要的结论是: 一个图存在欧拉路径, 当且仅当其为连通图, 且所有点度数均为偶数, 或者有且仅有两个点度数为奇数。若所有点度数均为偶数, 则从每个点出发, 均可找到能返回自身的欧拉路径(欧拉回路)若有且仅有两个点度数为奇数, 则所有路径都是两个奇点一个作为起点, 另一个作为终点。一条路径的逆路径必然也是一条欧拉路径, 因此以两点做起点的欧拉路径数量相同。本题中, 1 和 2 是奇点, 因此只有 1, 2 出发有一笔画路径。且各有 44 条。</p> <p>判断是否存在欧拉路径或求解出一条欧拉路径的算法复杂度与无向图边数成线性关系 $O(n + m)$, n 为总点数, m 为总边数。但是 DFS 递归暴力搜索所有可能路径并输出的过程, 算法复杂度难以分析, 实质上是一个阶乘级别的 NPC 问题, 因为欧拉路径总数的理论上界趋近边数的全排列数 $m!$。调试过程中可以发现, 输出 5-10 个点的稠密图的所有欧拉路径就已经相当耗时。</p>
代码实现	<pre> #define _CRT_SECURE_NO_WARNINGS #include <iostream> #include <cstring> #include <cstdio> #include <iomanip> #include <conio.h> using namespace std; #ifdef _MSC_VER #pragma warning(disable:6031) #endif const int MAXN = 6; const int MAXM = MAXN * (MAXN - 1) / 2; //完全无向图边数 int count_paths = 0; //路径总数 int G[MAXN][MAXN]; //存图 int CNT[MAXN]; //存每个点度的奇偶性 int N, M; //点数, 边数 int s; char mode; void print_path(int path[MAXN]) { cout << setw(4) << ++count_paths << " : "; for (int i = 0; i <= M; i++) cout << path[i] << ((i == M) ? "\n" : "->"); } </pre>

```

}

void dfs(int u, int(*g)[MAXN], int path[MAXM], int pp)
{
    path[pp++] = u; //当前节点进入顺序表，指针pp后移
    if (pp == M + 1) //顺序表里M+1个点，说明恰好画完了
        print_path(path);
    for (int v = 1; v <= N; v++)
    {
        if (g[u][v])
        {
            g[u][v] -= 1;
            g[v][u] -= 1;
            dfs(v, g, path, pp);
            //dfs恢复边
            g[u][v] += 1;
            g[v][u] += 1;
        }
    }
    pp--;
}

```

```

void input_and_init()
{
    while (1)
    {
        cout << "请输入无向图顶点数:";
        if (!(cin >> N))
        {
            cin.clear();
            cin.ignore(0x7fffffff, '\n');
        }

        if (N > 0)
            break;
        else
            cout << "输入有误，请重新输入" << endl;
    }

    while (1)
    {
        cout << "请输入无向图边数:";
        if (!(cin >> M))
        {

```

```

        cin.clear();
        cin.ignore(0x7fffffff, '\n');
    }

    if (M >= 0 && M <= N * (N - 1) / 2)
        break;
    else
        cout << "输入有误，请重新输入" << endl;
}

cout << "请输入" << M << "条边的起点终点序号，范围[1-" << N <<
"]:" << endl;
memset(CNT, 0, sizeof CNT);
memset(G, 0, sizeof G);
int u, v;
for (int i = 1; i <= M; ++i)
{
    while (1)
    {
        if (!(cin >> u >> v))
        {
            cin.clear();
            cin.ignore(0x7fffffff, '\n');
            cout << "输入有误， ";
        }
        else if (u < 1 || u > N || v < 1 || v > N)
            cout << "输入不合法，序号错误[1," << N << "], ";
        else if (u == v)
            cout << "输入不合法，不允许自环， ";
        else
            break;
        cout << "请重新输入" << endl;
    }
    G[u][v] += 1;
    G[v][u] += 1;
    CNT[u] ^= 1; //利用了异或运算，0表示度为偶数，1表示度为奇
数。
    CNT[v] ^= 1;
}

while (1)
{
    cout << "1.输出所有顶点开始全部路径" << endl;
    cout << "2.输出指定顶点开始的全部路径" << endl;
}

```

```

        cout << "请输入[1/2]:";
        mode = _getche();
        if (mode == '1' || mode == '2')
            break;
        else
            cout << "输入有误,请重新输入" << endl;
    }
    cout << endl;
    if (mode == '2')
    {
        while (1)
        {
            cout << "请输入指定起点:";
            if (!(cin >> s))
            {
                cin.clear();
                cin.ignore(0x7fffffff, '\n');
            }

            if (s >= 1 && s <= N)
                break;
            else
                cout << "输入有误, 请重新输入" << endl;
        }
    }
}

void mode1()
{
    int u;
    for (u = 1; u <= N; ++u) { //注意判断图是否从1点开始
        if (CNT[u]) break;
    }

    if (u == N + 1) //都为偶节点, 从每一个开始
    {
        for (int i = 1; i <= N; i++)
        {
            int g[MAXN][MAXN];
            int cnt[MAXN];
            memcpy(g, G, sizeof(G));
            memcpy(cnt, CNT, sizeof(CNT));
            int path[MAXM];
            int pp = 0;

```

```

        dfs(i, g, path, pp);
    }
}
else//有奇节点，从所有奇节点开始
{
    for (int i = 1; i <= N; i++)
    {
        if (CNT[i])
        {
            int g[MAXN][MAXN];
            int cnt[MAXN];
            memcpy(g, G, sizeof(G));
            memcpy(cnt, CNT, sizeof(CNT));
            int path[MAXM];
            int pp = 0;
            dfs(i, g, path, pp);
        }
    }
}
cout << "共有上述共" << count_paths << "条路径" << endl;
}

void mode2()
{
    int u;
    for (u = 1; u <= N; ++u)//判断图是否从1点开始
        if (CNT[u]) break;
    if (u == N + 1 || CNT[s])//都为偶节点或者s是奇节点
    {
        int g[MAXN][MAXN];
        int cnt[MAXN];
        memcpy(g, G, sizeof(G));
        memcpy(cnt, CNT, sizeof(CNT));
        int path[MAXM];
        int pp = 0;
        dfs(s, g, path, pp);
        cout << "共有上述共" << count_paths << "条从" << s << "开始
的路径" << endl;
    }
    else
        cout << "不存在从" << s << "开始的路径" << endl;
}

int main()

```



```
{  
    input_and_init();  
    if (mode == '1')  
        mode1();  
    else if (mode == '2')  
        mode2();  
    return 0;  
}
```