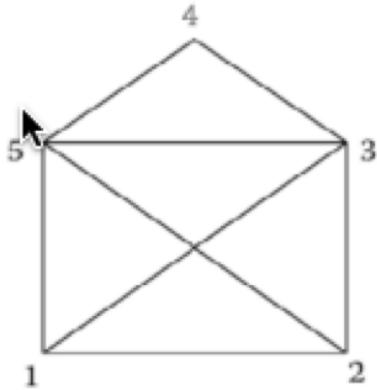


# 《数据结构》上机报告

2020 年 12 月 6 日

姓名： 林日中 学号： 1951112 班级： 10072602 得分： \_\_\_\_\_

实验题目	图的应用实验报告 - 一笔画	
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	<p>(1) 程序要添加适当的注释，程序的书写要采用缩进格式。</p> <p>(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。</p> <p>(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。</p> <p>(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。</p>	
	已完成基本内容（序号）：	(1)(2)(3)(4)
选做要求		
	已完成选做内容（序号）：	无

<p>数据结构设计</p>	<p>在本次上机实验中，我设计了两个数据结构：enum GraphType 和 class MGraph，分别表示图的类型和图。</p> <p>enum GraphType 中有五个成员 UNDEFINED, DG, DN, UDG, UDN，分别表示未定义、有向图、有向网络、无向图和无向网络。它表示的是图的类型，在构建图的时候可以用来判断是否需要进行输入权重 weight、双向构建 arc 等相关操作。</p> <p>class MGraph 中有6个成员变量，它们的名称和功能分别为：int **base 是指向图的各元素对应的矩阵的指针；GraphType type 是标记本对象类型的枚举类型；int vexNum, int arcNum 分别记录了本对象的点、边数量信息；int *vexs 记录了各元素对应的名称，在本程序中以 int 表示；int *path 记录了由起点开始的路径，数组存储的是所经过顶点的序号。</p>
<p>功能(函数)说明</p>	<pre> class MGraph { protected:     int **base = nullptr; // pointer to the matrix     GraphType type = UNDEFINED;     int vexNum = 0;     int arcNum = 0;     int *vexs = nullptr;     int *path = nullptr;     inline bool isNetwork() const { return type == DN    type == UDN; }     inline bool isUndirected() const { return type == UDG    type == UDN; }      int locate(const int v) const...      int printPath(const int *path, const int edgeCount) const...  public:     MGraph(const GraphType t, const int vn, const int an)...      ~MGraph() ...      int DFS_OnePath(         const int currentArc = 0,         const int edgeCount = 0,         int ansNum = 0) ... }; // class MGraph </pre> <p>class MGraph 的构造函数包装了从 stdin 读入各顶点和边的信息，用于构建图；isNetWork()函数和 isUndirected()函数分别返回一个 bool 值用于判断本对象是否为网络、是否无向，便于进行相关操作；locate()函数负责按值搜索定位顶点，并返回顶点的序号；printPath()函数用于打印完整的一笔画路径；DFS_OnePath()函数封装了寻找所有一笔画路径的过程。</p>
<p>开发环境</p>	<p>Windows 10, C++ language, Visual Studio Code with g++</p>
<p>调试分析</p>	<p>本程序采用重定向的方式输入所需数据。我将题目所给图片转换为数字，并放入 hw7_6_input.txt 中，其内容如下。</p> <pre> 5 8 1 2 </pre>

```
1 3
1 5
2 3
2 5
3 4
3 5
4 5
```

经过测试，本程序功能完好，能够输出所有可能的边遍历序列。

输出的所有文本如下：

```
Please input the numbers of vertices and arcs respectively:
Please input src and dst (and weight, if needed) of each arc:
Found accessible paths are:
Path No.1: 1->2->3->1->5->3->4->5->2
Path No.2: 1->2->3->1->5->4->3->5->2
Path No.3: 1->2->3->4->5->1->3->5->2
Path No.4: 1->2->3->4->5->3->1->5->2
Path No.5: 1->2->3->5->1->3->4->5->2
Path No.6: 1->2->3->5->4->3->1->5->2
Path No.7: 1->2->5->1->3->4->5->3->2
Path No.8: 1->2->5->1->3->5->4->3->2
Path No.9: 1->2->5->3->1->5->4->3->2
Path No.10: 1->2->5->3->4->5->1->3->2
Path No.11: 1->2->5->4->3->1->5->3->2
Path No.12: 1->2->5->4->3->5->1->3->2
Path No.13: 1->3->2->1->5->3->4->5->2
Path No.14: 1->3->2->1->5->4->3->5->2
Path No.15: 1->3->2->5->3->4->5->1->2
Path No.16: 1->3->2->5->4->3->5->1->2
Path No.17: 1->3->4->5->1->2->3->5->2
Path No.18: 1->3->4->5->1->2->5->3->2
Path No.19: 1->3->4->5->2->1->5->3->2
Path No.20: 1->3->4->5->2->3->5->1->2
Path No.21: 1->3->4->5->3->2->1->5->2
Path No.22: 1->3->4->5->3->2->5->1->2
Path No.23: 1->3->5->1->2->3->4->5->2
Path No.24: 1->3->5->1->2->5->4->3->2
Path No.25: 1->3->5->2->1->5->4->3->2
Path No.26: 1->3->5->2->3->4->5->1->2
Path No.27: 1->3->5->4->3->2->1->5->2
Path No.28: 1->3->5->4->3->2->5->1->2
Path No.29: 1->5->2->1->3->4->5->3->2
Path No.30: 1->5->2->1->3->5->4->3->2
Path No.31: 1->5->2->3->4->5->3->1->2
Path No.32: 1->5->2->3->5->4->3->1->2
Path No.33: 1->5->3->1->2->3->4->5->2
Path No.34: 1->5->3->1->2->5->4->3->2
Path No.35: 1->5->3->2->1->3->4->5->2
Path No.36: 1->5->3->2->5->4->3->1->2
Path No.37: 1->5->3->4->5->2->1->3->2
Path No.38: 1->5->3->4->5->2->3->1->2
Path No.39: 1->5->4->3->1->2->3->5->2
Path No.40: 1->5->4->3->1->2->5->3->2
Path No.41: 1->5->4->3->2->1->3->5->2
Path No.42: 1->5->4->3->2->5->3->1->2
Path No.43: 1->5->4->3->5->2->1->3->2
Path No.44: 1->5->4->3->5->2->3->1->2
Done! There are 44 paths found!
```

综上，本实验设计的数据结构很好地完成了题目对图的功能的要求，功能完好，使用方便，并且能够较好地处理非法数据的输入并反馈相关错误提示；程序界面友好，具有比较恰当的人性化提醒，具有一定的鲁棒性。

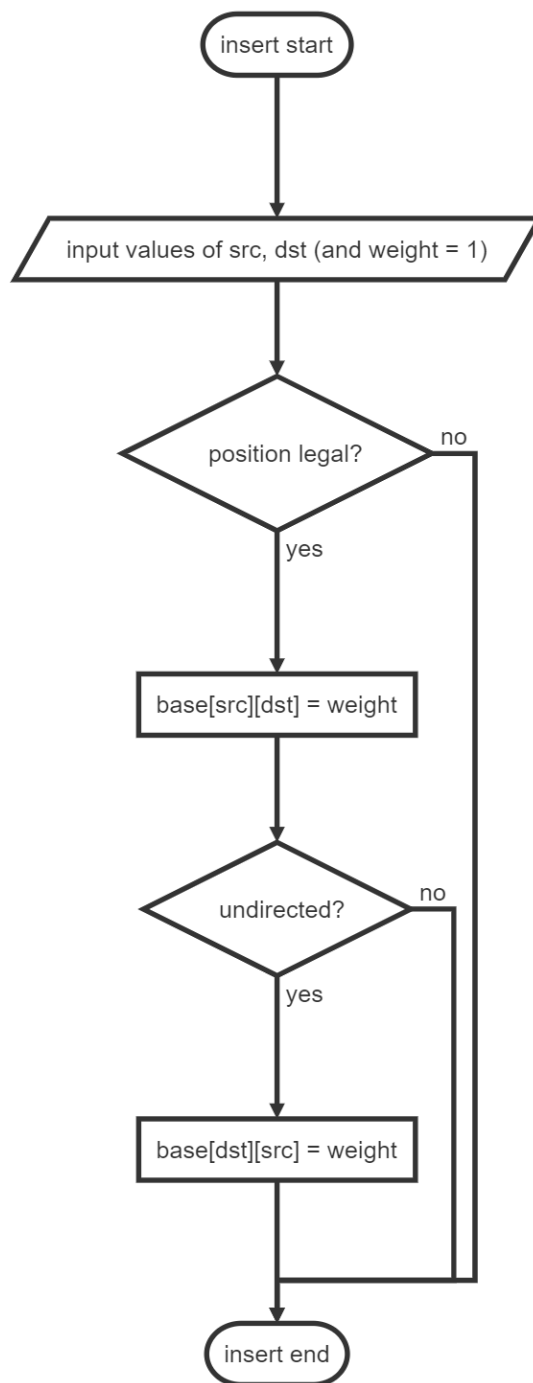
心得体会

## 一、 实验总结

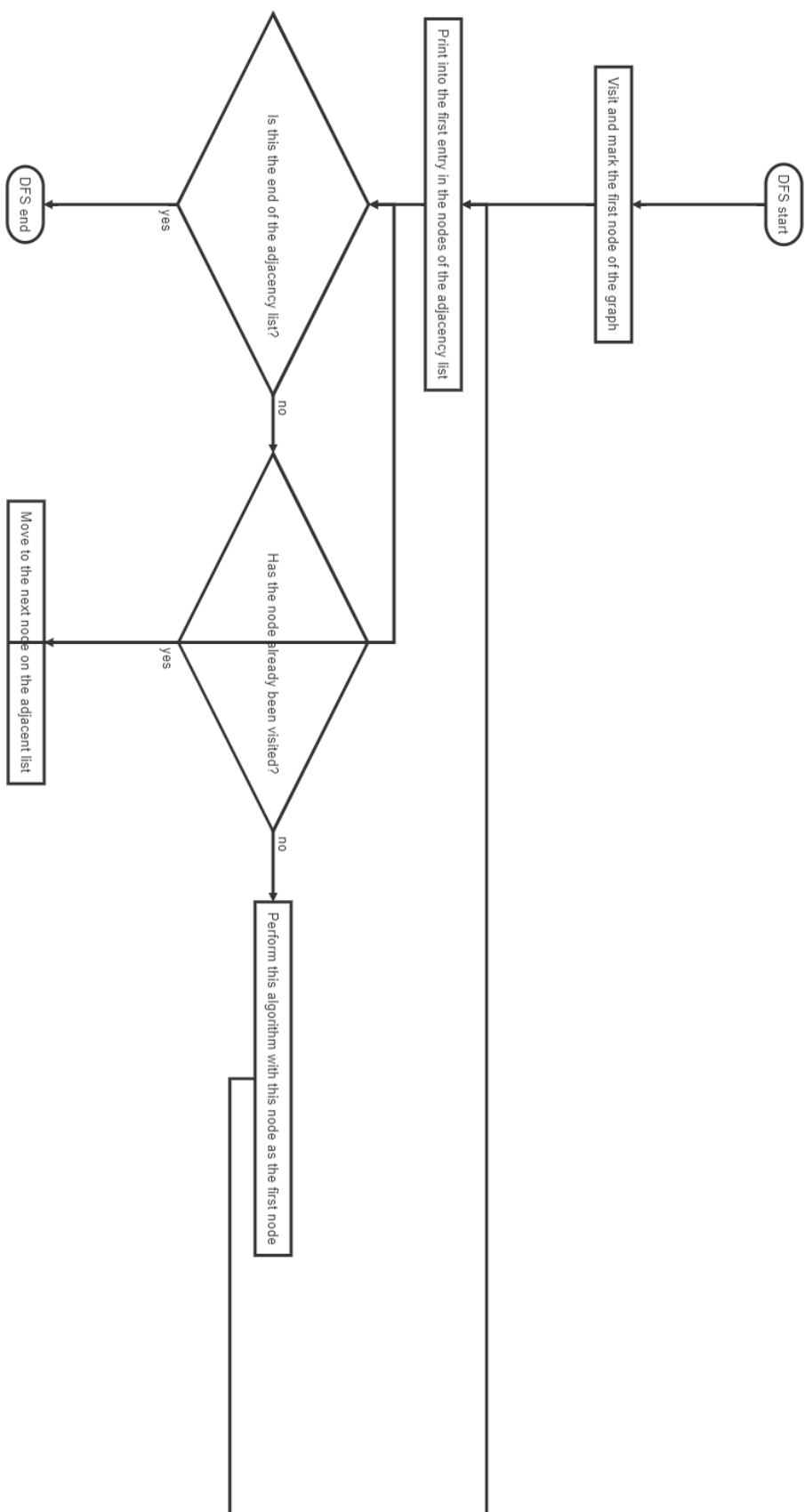
在本次上机实验中，我复习了理论课上学到的图的定义和作用，将课本上图的矩阵存储结构和功能封装成了 `class MGraph`，并利用 DFS 找出所有一笔画路径。

## 二、 性能分析

- (1) 图的边的插入。  
时间复杂度： $O(1)$ 。



(2) 图的深度优先搜索 (DFS)。  
时间复杂度:  $O(n^2)$ 。



## 源代码

```
1. #if defined(__GNUC__)
2. #include <bits/stdc++.h>
3. #elif defined(_MSC_VER)
4. #define _CRT_SECURE_NO_WARNINGS
5. #include <iostream>
6. #include <cstdio>
7. #include <cstdlib>
8. #include <cstring>
9. #include <climits>
10. #include <ctime>
11. #include <iomanip>
12. #include <queue>
13. #endif
14.
15. #define inf INT_MAX
16. #define MAX_N 2000
17.
18. using namespace std;
19.
20. enum GraphType
21. {
22.     UNDEFINED, // Undefined
23.     DG,         // Directed Graph
24.     DN,         // Directed Network
25.     UDG,        // Undirected Graph
26.     UDN         // Undirected Network
27. };
28.
29. class MGraph
30. {
31.
32. protected:
33.     int **base = nullptr; // pointer to the matrix
34.     GraphType type = UNDEFINED;
35.     int vexNum = 0;
36.     int arcNum = 0;
37.     int *vexs = nullptr;
38.     int *path = nullptr;
39.     inline bool isNetwork() const { return type == DN || type == UDN; }
40.     inline bool isUndirected() const { return type == UDG || type == UDN; }
41.
42.     int locate(const int v) const
43.     {
44.         for (int i = 0; i < vexNum; i++)
```

```

45.     {
46.         if (v == vexs[i])
47.         {
48.             return i;
49.         }
50.     }
51.     return inf;
52. }
53.
54. int printPath(const int *path, const int edgeCount) const
55. {
56.     for (int i = 0; i <= edgeCount; i++)
57.     {
58.         if (i)
59.         {
60.             cout << "->";
61.         }
62.         cout << vexs[path[i]];
63.     }
64.     cout << "\n";
65.     return 1;
66. }
67.
68. public:
69.     MGraph(const GraphType t, const int vn, const int an)
70.         : type(t), vexNum(vn), arcNum(an),
71.         vexs(new int[vexNum]), path(new int[MAX_N])
72.     {
73.         for (int i = 0; i < vexNum; i++)
74.         {
75.             vexs[i] = i + 1;
76.         }
77.
78.         base = new int *[vexNum];
79.         for (int i = 0; i < vexNum; i++)
80.         {
81.             base[i] = new int[vexNum];
82.             memset(base[i], 0, sizeof(int) * vexNum);
83.         }
84.
85.         cout << "Please input src and dst (and weight, "
86.             << "if needed) of each arc: \n";
87.
88.         // build the graph

```

```

89.         for (int c = 0; c < arcNum; c++)
90.         {
91.             int src_v, dst_v;
92.             int w = 0;
93.             cin >> src_v >> dst_v;
94.             if (isNetwork())
95.             {
96.                 cin >> w;
97.             }
98.             int i_src_v = locate(src_v);
99.             int j_dst_v = locate(dst_v);
100.            base[i_src_v][j_dst_v] = isNetwork() ? w : 1;
101.            if (isUndirected())
102.            {
103.                base[j_dst_v][i_src_v] = base[i_src_v][j_dst_v];
104.            }
105.        }
106.
107.        memset(path, 0, sizeof(int) * MAX_N);
108.    }
109.
110.    ~MGraph()
111.    {
112.        for (int i = 0; i < vexNum; i++)
113.        {
114.            delete[] base[i];
115.            base[i] = nullptr;
116.        }
117.        delete[] base;
118.        delete[] vxs;
119.        delete[] path;
120.    }
121.
122.    int DFS_OnePath(
123.        const int currentArc = 0,
124.        const int edgeCount = 0,
125.        int ansNum = 0)
126.    {
127.        path[edgeCount] = currentArc;
128.
129.        if (edgeCount == arcNum)
130.        {
131.            cout << "Path No." << ++ansNum << ": ";
132.            printPath(path, edgeCount);

```



```

133.     }
134.
135.     for (int i = 0; i < vexNum; i++)
136.     {
137.         if (base[currentArc][i])
138.         {
139.             int temp = base[currentArc][i];
140.             base[currentArc][i] = 0;
141.             if (isUndirected())
142.             {
143.                 base[i][currentArc] = 0;
144.             }
145.
146.             ansNum = DFS_OnePath(i, edgeCount + 1, ansNum);
147.             base[currentArc][i] = temp;
148.             if (isUndirected())
149.             {
150.                 base[i][currentArc] = temp;
151.             }
152.         }
153.     }
154.     return ansNum;
155. }
156.
157. }; // class MGraph
158.
159. int main()
160. {
161.     (void)freopen(".\\HW7_6_input.txt", "rb", stdin);
162.     int vexnum, arcnum, count = 0;
163.     cout << "Please input the numbers of "
164.           "vertices and arcs respectively: \n";
165.     cin >> vexnum >> arcnum;
166.     MGraph mg(UDG, vexnum, arcnum);
167.
168.     cout << "Found accessible paths are: \n";
169.     if (!(count = mg.DFS_OnePath()))
170.     {
171.         cout << "Not found!\n";
172.     }
173.     else
174.     {
175.         cout << "Done! There are "
176.               << count << " paths found! \n";

```

	177. }
	178.
	179. <b>return</b> 0;
	180. }