湖南大學

数据结构

课程实验报告

题　　目：CCF201709-4 通信网络

学生姓名：魏子铖

学生学号：201726010308

专业班级：软件 1703

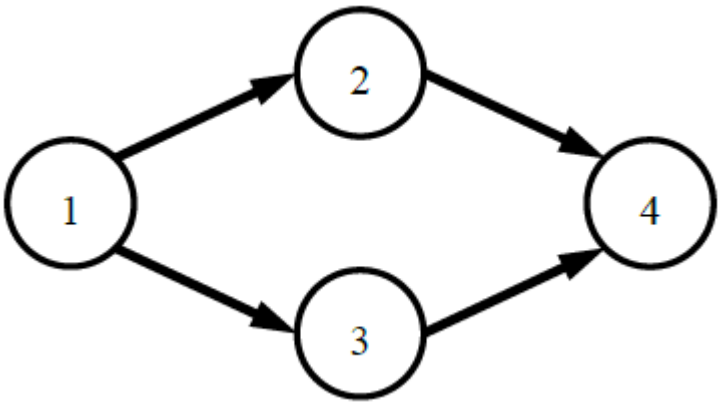
完成时间：2018.12.15

一、需求分析

0) 问题分析

某国的军队由 N 个部门组成，为了提高安全性，部门之间建立了 M 条通路，每条通路只能单向传递信息，即一条从部门 a 到部门 b 的通路只能由 a 向 b 传递信息。信息可以通过中转的方式进行传递，即如果 a 能将信息传递到 b，b 又能将信息传递到 c，则 a 能将信息传递到 c。一条信息可能通过多次中转最终到达目的地。

由于保密工作做得很好，并不是所有部门之间都互相知道彼此的存在。只有当两个部门之间可以直接或间接传递信息时，他们才彼此知道对方的存在。部门之间不会把自己知道哪些部门告诉其他部门。



上图中给了一个 4 个部门的例子，图中的单向边表示通路。部门 1 可以将消息发送给所有部门，部门 4 可以接收所有部门的消息，所以部门 1 和部门 4 知道所有其他部门的存在。部门 2 和部门 3 之间没有任何方式可以发送消息，所以部门 2 和部门 3 互相不知道彼此的存在。

现在请问，有多少个部门知道所有 N 个部门的存在。或者说，有多少个部门所知道的部门数量（包括自己）正好是 N。

对于上述问题，需要实现的功能有：

I) 将有向图存储在计算机的物理结构中

II) 实现有向图的深度优先搜索算法

III) 对每一次 DFS 得到的结点数组进行处理

1) 输入数据

【输入格式】

- 输入的第一行包含两个整数 N, M，分别表示部门的数量和单向通路的数量。所有部门从 1 到 N 标号。
- 接下来 M 行，每行两个整数 a, b，表示部门 a 到部门 b 有一条单向通路。

【输入样例】

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

2) 输出数据

【输出格式】

输出一行，包含一个整数，表示答案。

【输出样例】

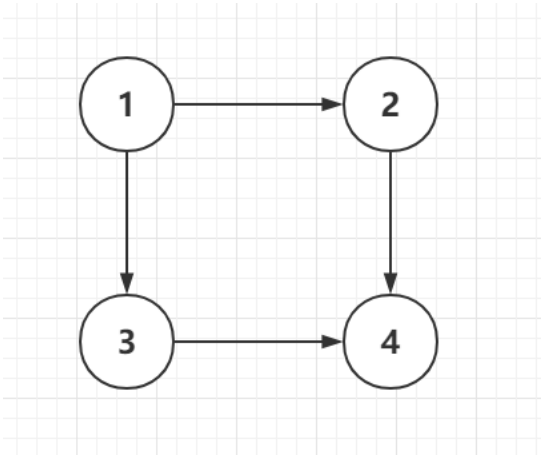
2

3) **测试样例设计**

样例一

样例输入

4 4
1 2
1 3
2 4
3 4



样例输出

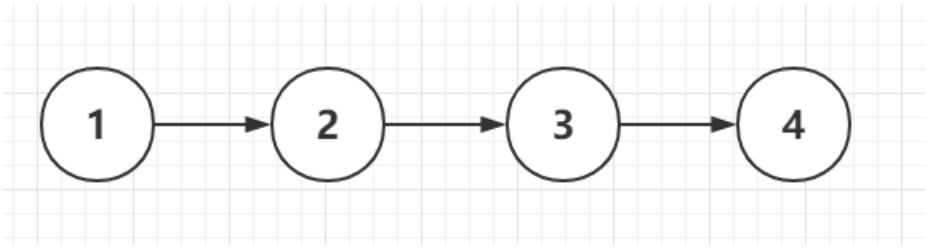
2

设计理由：一般情况

样例二

样例输入

4 3
1 2
2 3
3 4



样例输出

4

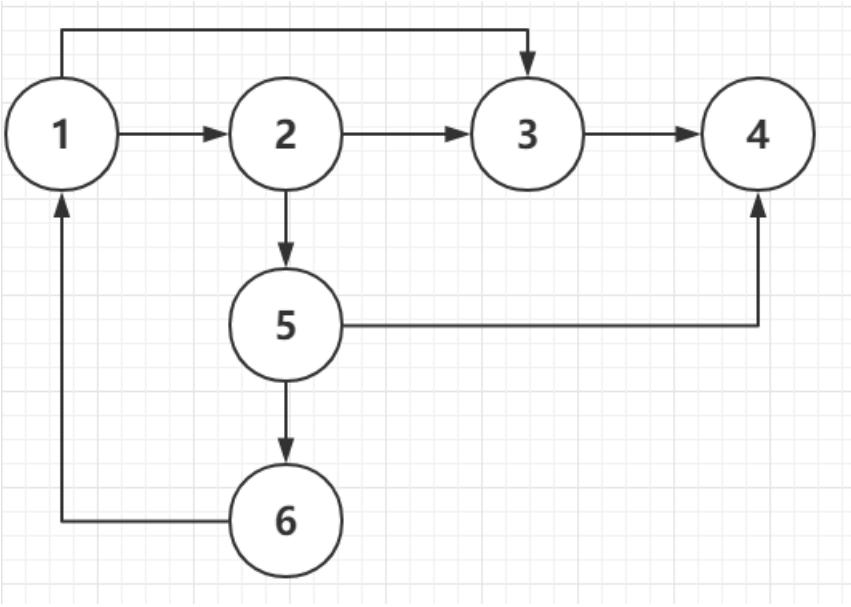
设计理由：有向图为链式结构

样例三

样例输入

6 8
1 2
2 3

3 4
1 3
2 5
5 4
6 1
5 6



样例输出

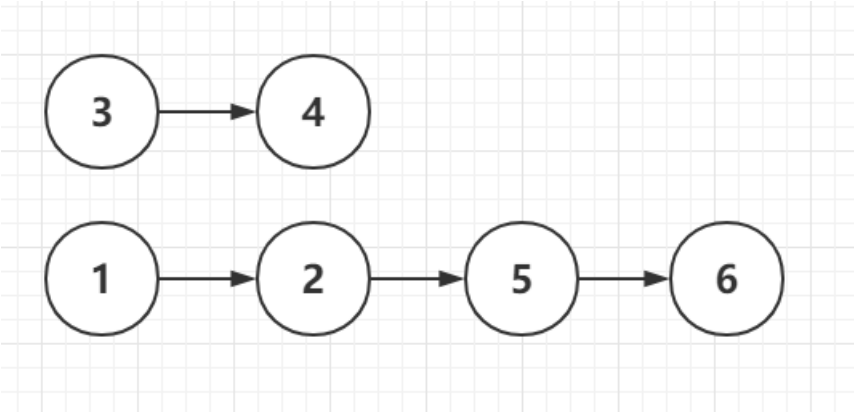
6

设计理由：有向图中存在回路

样例四

样例输入

6 4
3 4
1 2
2 5
5 6



样例输出

0

设计理由：有向图不是连通图

样例五

样例输入

1 0

样例输出

1

设计理由：有向图中只有一个点

二、概要设计

1. 抽象数据类型

为实现上述功能，由于题目规定输入数据均为整数，所以使用 int 类型来存储用户的输入，并将用户输入的值存储在有向图相应的顶点中。

抽象数据类型设计：

- 数据对象：两个相关的整数，表示节点之间的通路。
- 数据关系：第一个整数表示路径的起点，第二个整数表示路径的终点。
- 基本操作：在构建有向图时插入路径信息；判断有向图中的路径是否存在；获取静态查找表中元素的个数。
- **ADT：**

```
MatrixGraph {
数据对象：D = { <vt1ᵢ , vt2ᵢ >| vt1, vt2∈N, i = 1, 2, 3, ……, n, 1≤n≤1000 }
数据关系：R = { edge | edge∈Graph , <vt1, vt2> = edge}
基本操作：
    int n();
    //返回有向图中顶点的数目，时间复杂度 O（1）
    void setEgde(const int&, const int&, const int&);
    //向有向图中插入元素，时间复杂度 O（1）
    int getFirst(const int) const;
    //返回与该顶点相连的顶点编号最小的顶点编号，时间复杂度 O（n）
    int next(const int, const int);
    //返回下一个与该顶点相邻的顶点编号，时间复杂度 O（n）
    int getMark(const int);
    //返回被访问顶点的标记数组的值，时间复杂度 O（1）
    void setMark(const int, const int);
    //设置被访问顶点的标记数组的值，时间复杂度 O（1）
}
```

2. 算法的基本思想

对于各种功能：

1) 建立有向图

基于邻接矩阵来建立有向图，数组中为 0 表示顶点没有相连，为 1 表示顶点相连。

2) 实现有向图的查找功能

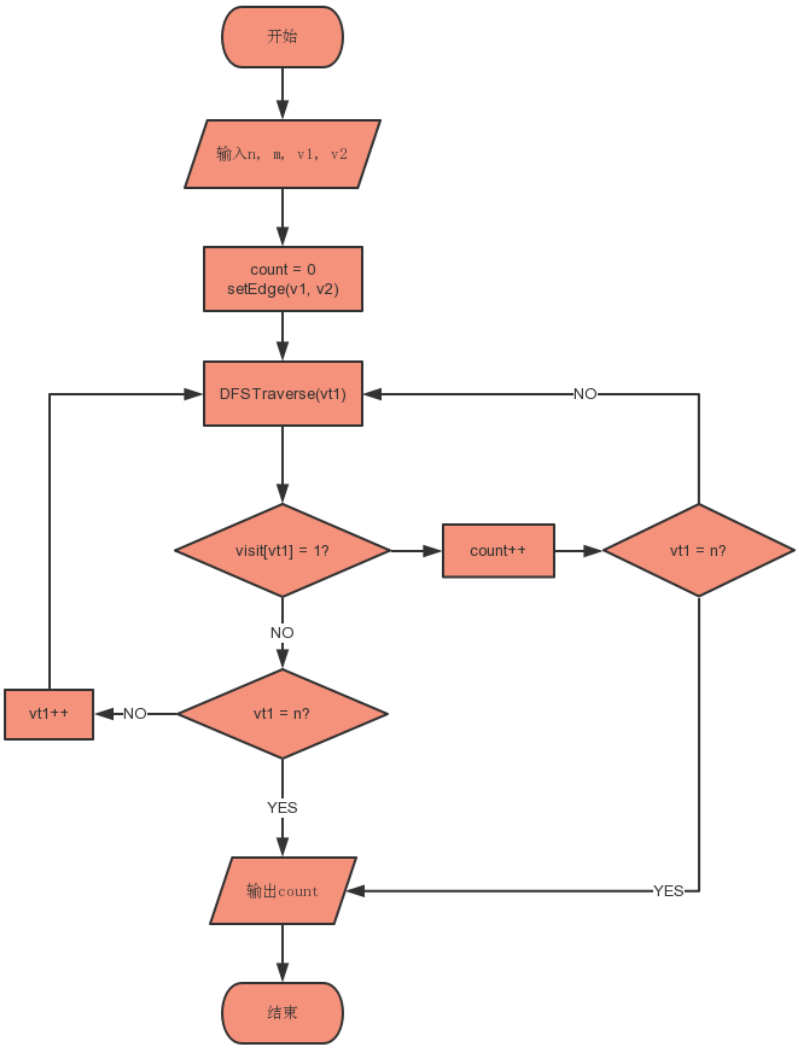
调用类中的 **getFirst()**与 **next()**方法，找出所有与被访问顶点相连的顶点。

3) 实现对顶点的深度优先搜索
沿着图的深度遍历图的顶点，尽可能深的搜索图的分支。当顶点 **vt** 的所在边都已被探寻过，搜索将回溯到发现顶点 **vt** 的那条边的起始顶点。这一过程一直进行到已发现从源顶点可达的所有顶点为止。如果还存在未被发现的顶点，则选择其中一个作为源顶点并重复以上过程，整个进程反复进行直到所有顶点都被访问为止。
4) 对结果进行格式化输出
输出一个整数，表示答案。

3. 程序的流程

程序由四个模块组成：
1) 输入模块：提示输入格式以及结束标志，输入分两个部分，一部分是有向图顶点数与边数的信息，一部分为有向边的据体信息。
2) 构建有向图：以邻接矩阵形式实现有向图的构建。
3) 查找模块：在有向图中执行深度优先搜索，返回值为标记数组。
4) 输出模块：格式化输出查找信息。

程序流程图如下：



三、详细设计

1. 物理数据类型

输入的数据为有向图的信息与有向边的信息，都是整数类型，由于有向图是基于邻接矩阵的，满足顺序特征，所以逻辑实现上可以采用数组的形式。

2. 输入和输出的格式

输入时有提示语句，说明输入的方法与结束条件，将每个元素存储在同一个有向图，将查找信息格式化输出。

3. 算法的具体步骤

1) 基于邻接矩阵建立有向图

邻接矩阵是表示一个图的常用存储表示。它用两个数组分别存储数据元素（顶点）的信息和数据元素之间的关系（边或弧）的信息。

阶为 n 的图 G 的邻接矩阵 A 是 n×n 的。将 G 的顶点标签为 v1, v2, ⋯, vn。若 (vi, vj) ∈ E(G)，Aij = 1，否则 Aij = 0。

```
void MatrixGraph::setEdge(int vt1, int vt2, int wt) {
    try {
        if (wt <= 0) {
        //如果边权不是正整数，则抛出非法输入的警告
            throw "Illegal weight value";
        }
        if (matrix[vt1][vt2] == 0) {
        //如果该边没有被设置过，则边数+1
            numEdge++;
        }
        //设置边权，表示从 vt1 指向 vt2 的边权为 wt
        matrix[vt1][vt2] = wt;
    }
    catch (const char* str) {
        //输出错误信息
        std::cerr << str << std::endl;
    }
}
```

2) 对每个顶点执行一次深度优先搜索

从一个顶点 v0 开始，沿着一条路一直走到底，如果到达底部，那就返回到上一个节点，然后从另一条路开始走到底，这种尽量往深处走的概念即是深度优先的概念。

```
void DFSTraverse(MatrixGraph* m, int vt) {
    m->setMark(vt, VISITED);
    //将访问过的顶点数组置为 VISITED
    for (int w = m->getFirst(vt); w <= m->n(); w = m->next(vt, w)) {
    //如果没有到达路径的最底部，则继续往深处遍历
        if (m->getMark(w) == UNVISITED) {
        //如果改顶点没有被访问过，则对该顶点继续执行深度优先搜索
            DFSTraverse(m, w);
        }
    }
```

```
}
```

3) **对深度优先搜索的结果即标记数组进行处理**

标记数组的每一行表示从该顶点出发可以访问到的所有顶点，若该行的值全为 1，则表明这个部门知道其他所有部门的存在，则 count 值加一。

```
int count = 0;
for (int i = 0; i < N; i++) {
//外层循环遍历每个部门
    for (int j = 0; j < N; j++) {
        //内层循环遍历该部门与其他部门的访问结果
        if (visit[i][j] == 0) {
        //若部门 i 不知道部门 j 的存在，则结束该次循环
            break;
        }
        if (j == N - 1) {
        //若该部门知道其他所有部门的存在，则 count++
            count++;
        }
    }
}
```

4) **格式化输出结果**

输出 count 值即为所求答案

```
std::cout << count << std::endl;
```

4. 算法的时空分析

1) 设置有向图中边的信息，时间复杂度 O（1）。
2) 对每个顶点的深度优先搜索，时间复杂度 O（n^2+e）。
3) 对标记数组进行处理，时间复杂度 O（n^2）。

四、调试分析

1.调试方案设计

调试目的：发现思维逻辑与代码实现上的区别，改进代码结构，排除语法逻辑上的错误。

样例：

样例输入

4 4
1 2
1 3
2 4
3 4

样例输出

2

调试计划：设置好断点，注意观察每一步时各个变量的变化情况，找出错误的地方，然后改正；单步调试，更能准确定位出现错误的代码区域。

2.调试过程和结果，及分析

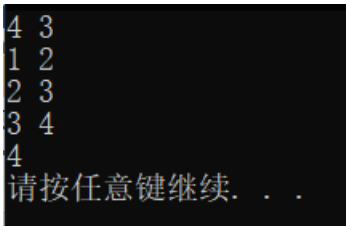
调试过程中由于出现数组越界情况，导致代码多次崩溃，发现是 next()方法出现了问题，排除错误后调试成功，输出了正确的结果。

五、测试结果

1. **样例一**

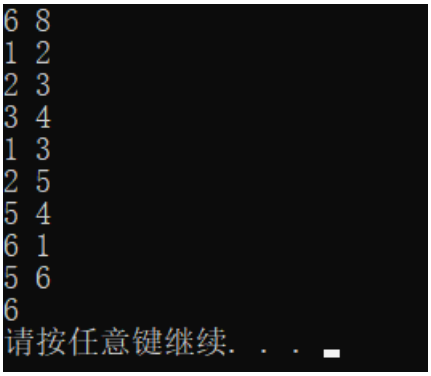
一般情况，输出结果正确

2. **样例二**



有向图为链式结构，所有部门都知道其他的所有部门，输出结果正确

3. **样例三**



有向图中存在回路，对深度优先搜索没有影响，输出结果正确

4. **样例四**

有向图不是连通图，则所有部门都不知道其他所有部门，输出结果正确

5. **样例五**

有向图中只有一个点，显然输出为 1，输出结果正确

六、实验日志

12/08

在历年的三个题目中选择了通信网络，对题目分析后确定用图以及图的算法可以解决，再设计输入输出格式与抽象数据类型还有算法思想。

阅读题目后，以前碰到 oj 题都直接看有没有思路，这次脑子里首先想到的是用什么数据结构存储，然后才是算法思想。

12/10

完成了主函数的实现，在主函数中分为输入、计算和输出三个模块。调用类中的函数先构建图，然后用深度优先搜索遍历所有的部门，再次遍历计算哪些部门知 道所有部门的存在。

用图 ADT 去解决问题时，尤其注意 ADT 里基本操作的使用。

12/12

在大概完成整个程序，编译运行后，对设计的测试样例进行调试测试，在这个过程中发现了错误，经程序调试后发现是逻辑错误导致指针指向崩溃，在调试后修改正确。

调试对于完善和找到程序的 bug 十分重要。