湖南大學

数据结构

课程实验报告

题　　目：　通信网络（CCF201709-4）

学生姓名　　杨兰馨

学生学号　　201708020305

专业班级　　通信 1703 班

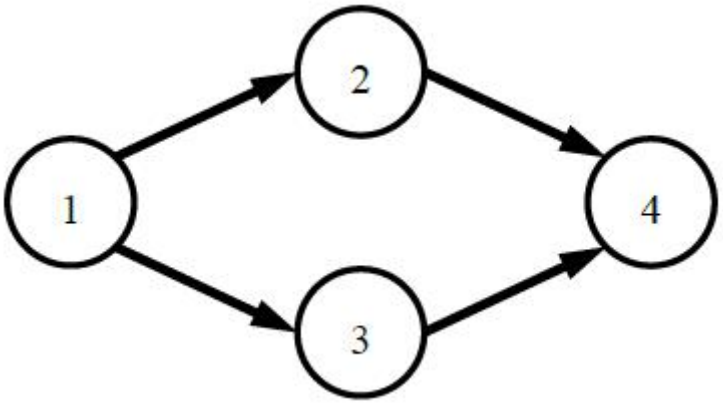
完成日期　　2019/5/30

一、 需求分析

问题描述

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。

1. 问题分析

(1) 用邻接矩阵的方式构建无权有向图

(2) 判断某一部门是否知道所有部门的存在的图论语言为：判断通过某一顶点的所有路径中是否包含了图中的所有顶点。

(3) 实现图的深度优先遍历得到图的周游路径。

(4) 在以 v1 顶点为起点的 DFS 后，通过 mark 数组的顶点 v2 是否被标记来判断两个顶点之间存不存在存不存在“联系”。

2. 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

3. 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

4. 测试样例设计

CCF 题目测试结果如图：

姓名	试题名称	提交时间	代码长度	编程语言	评测结果	得分	时间使用	空间使用
〈杨兰馨〉	通信网络	05-30 21:41	1.179KB	C++	正确	100	187ms	4.480MB

二、概要设计

1. 抽象数据类型

为实现上述程序的功能，创建一个无权有向图，图中各个顶点的数据类型都为整型。

抽象数据类型设计：

数据对象：一个顶点（顶点的元素为 int 型）

数据关系：每个顶点与其他顶点之间存在着有向路径

基本操作：

1. 返回顶点个数，返回当前顶点的第一个邻接顶点，返回当前顶点的邻接顶点中 w 顶点后的第一个邻接顶点；
2. 在顶点 v1 和 v2 之间设置边；
3. 返回顶点 v 的 mark 数组的值；设置 mark 数组中的顶点 v 的值，重置 mark 数组；
4. DFS 深度优先遍历

MatrixGraph {

数据对象：D = { $<vt1_i, vt2_i>$ | 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(); //返回顶点个数
int first(int v); //返回顶点 v 的第一个邻接顶点
int next(int v, int w);//返回 v 的邻接顶点中 w 顶点后的第一个邻接顶点
void setEdge(int v1, int v2);//在 v1 和 v2 之间设置边
int getMark(int v);//返回顶点 v 是否被标记过
void setMark(int v, int val);//设置顶点 v 的 mark 数组中的值
void resetMark();//充值 mark 数组
void DFSTraverse(MatrixGraph*, int);//对一个图进行深度优先搜索
```

}

2. 算法的基本思想

1. **建图的算法思想：**只要确立了一个图的顶点和边的信息，就可以建成一个图。该实验中，图的顶点为自然数 12...n，因此只需要输入顶点个数就可以自动匹配顶点信息。在

该实验中的边需要手动输入，用到 setEdge 函数。函数算法如下：如果图的邻接矩阵中行为 v1，列为 v2 的数值等于零，则说明原本这两个顶点之间没有边，numEdge++，并且将矩阵对应的值改为 1，如果为无向图，需要在 v2 和 v1 之间也赋值，如果为有向图则不必赋值。

2. **DFS（深度优先遍历）的算法思想：**传递参数图 ma，和顶点 v。首先设置顶点 v 为已被访问过，然后按顺序遍历顶点 v 的邻接顶点，将第一个没被标记的顶点传进 DFS 函数中进行递归操作。递归的终止条件为，所有的顶点都被遍历过。

3. **判断某一点是否能与所有顶点“互相通信”的算法思想：**通过 mark 数组的标记来看。创建一个二维数组 vt[N][N],数组的第 i 行表示，以顶点 i 为起点进行 DFS 时的 mark 数组标记情况，数组的第 j 列表示，分别以 0-n 顶点为起点进行 DFS 时，顶点 j 的被标记的情况。则对于判断顶点 i 是否能与所有顶点之间有通路，只需要看此矩阵的定 i 行和第 i 列的被标记情况，如果 vt[i][j]或者 vt[j][i]至少有一个等于 1，则可以判断顶点 i 与顶点 j 之间互相通信。用二重循环对二维数组进行遍历，如果对于顶点 i，可以互相通信的顶点数目达到 numVertex，则可以判断这个顶点符合要求，count++。

3.程序的流程

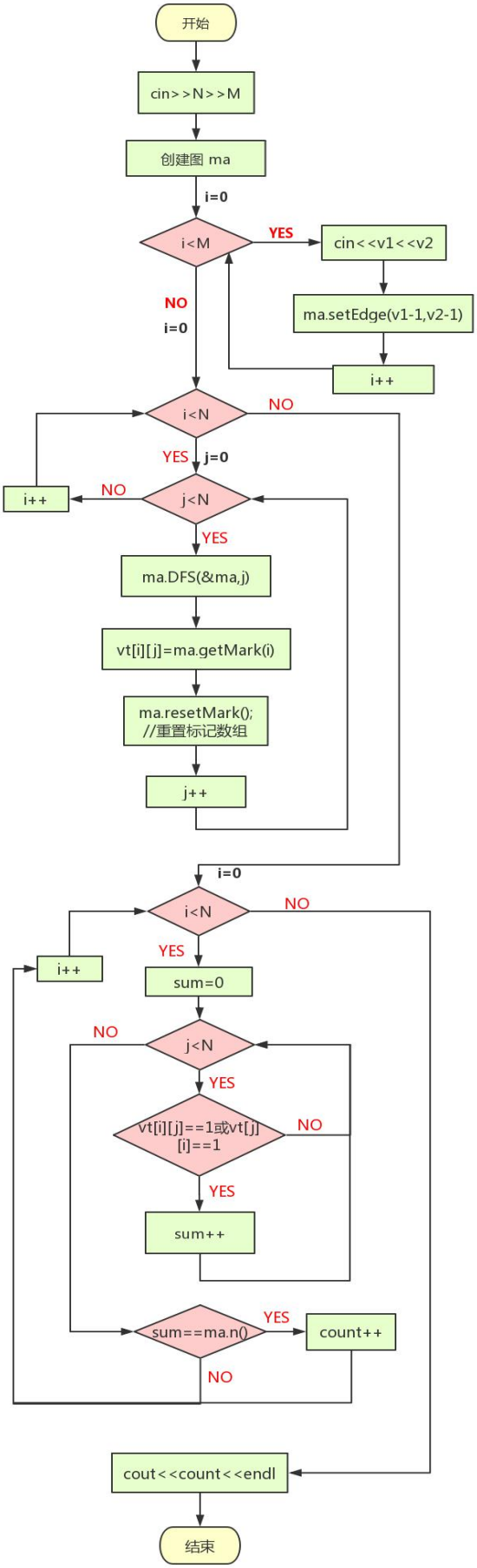
程序由三个模块组成：

（1） 输入模块：无提示语句，分别输入顶点数目 N 和边的数目 M，然后 M 行分别输入边的两个顶点信息。

（2） 处理模块：以邻接矩阵的形式创建有向无权图。以图的每个顶点作为起点进行 DFS，得到的 mark 数组信息存储在二维数组中 vt 中，通过 vt 二维数组来计算满足互相通信的顶点的数目。

（3） 输出模块：对结果进行格式化输出。

程序流程图如下：


开始
cin>>N>>M
创建图 ma
i=0
i<M
YES
cin<<v1<<v2
ma.setEdge(v1-1,v2-1)
i++
NO
i=0
i<N
NO
YES
j=0
i++
NO
j<N
YES
ma.DFS(&ma,j)
vt[i][j]=ma.getMark(i)
ma.resetMark();
//重置标记数组
j++
i=0
i<N
NO
YES
i++
sum=0
NO
j<N
YES
vt[i][j]==1或vt[j][i]==1
NO
YES
sum++
sum==ma.n()
YES
count++
NO
cout<<count<<endl
结束

三、详细设计

1.物理数据类型

本实验采用邻接矩阵的方式存储有向无权图，图中各个顶点的数据类型为整型 int 型，各个顶点之间用有向无权路径连接。

定义的图的 ADT 的伪代码：

```
class MatrixGraph :
private:
    int numVertex;//定义顶点个数
    int numEdge;//定义边个数
    int** matrix;//定义二维数组
    int* mark;//定义顶点数组
    bool undirected;//true 表示无向图  false 表示有向图
    void Init(int n) ;//初始化图
public:
    explicit MatrixGraph(int n);
    ~MatrixGraph();
    int n();//返回顶点个数
    int e();//返回边的个数
    int first(int v);
    int next(int v, int w);
    void setType(bool flag);//设置图的类型
    bool getType();//获取图的类型
    void setEdge(int v1, int v2);
    void deleteEdge(int v1, int v2);
    bool isEdge(int i, int j);
    int getMark(int v);
    void setMark(int v, int val);
    void resetMark();
    void DFSTraverse(MatrixGraph*, int);
};
```

2.输入和输出的格式

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。输出结果一行，包含一个整数。

3.算法的具体步骤

1. 建立无权有向图

只要确立了一个图的顶点和边的信息，就可以建成一个图。该实验中，图的顶点为自然数 12...n，因此只需要输入顶点个数就可以自动匹配顶点信息。在该实验中的边需要手动输入，用到 setEdge 函数。函数算法如下：如果图的邻接矩阵中行为 v1，列为 v2 的数值等

于零，则说明原本这两个顶点之间没有边，numEdge++，并且将矩阵对应的值改为 1，如果为无向图，需要在 v2 和 v1 之间也赋值，如果为有向图则不必赋值。

设置边的伪代码如下：

```
void MatrixGraph::setEdge(int v1, int v2) {
        if (matrix[v1][v2] == 0) {
            numEdge++;
        }
        matrix[v1][v2] = 1;
        if(undirected){
            matrix[v2][v1] = 1;
        }
}
```

建图的伪代码如下：

```
bool flag = 0;//确定该图为有向图
int N,M;
cin>>N>>M;
MatrixGraph ma(N);
ma.setType(flag);
int v1,v2,w;
for(int i=0;i<M;i++)
{
    cin>>v1>>v2;
    ma.setEdge(v1-1,v2-1);
}
```

2. DFS 深度优先搜索：

传递参数图&ma 和顶点 v。首先设置顶点 v 为已被访问过，然后按顺序遍历顶点 v 的邻接顶点，将第一个没被标记的顶点传进 DFS 函数中进行递归操作。递归的终止条件为，所有的顶点都被遍历过。注意细节，在传递参数时一定要加索引号&。

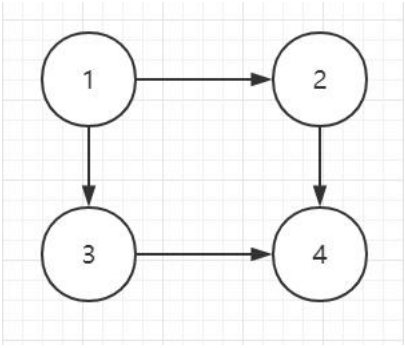
伪代码如下：

```
void MatrixGraph::DFSTraverse(MatrixGraph* m, int v) {
    m->setMark(v, VISITED);
    for (int w = m->first(v); w < m->numVertex; w = m->next(v, w)) {
        if (m->getMark(w) == UNVISITED) {
            DFSTraverse(m, w);
        }
    }
```

3. 判断某一点是否能与所有顶点“互相通信”的算法思想：

通过 mark 数组的标记来看。创建一个二维数组 vt[N][N],数组的第 i 行表示，以顶点 i 为起点进行 DFS 时的 mark 数组标记情况，数组的第 j 列表示，分别以 0-n 顶点为起点进行 DFS 时，顶点 j 的被标记的情况。则对于判断顶点 i 是否能与所有顶点之间有通路，只需要看此矩阵的定 i 行和第 i 列的被标记情况，如果 vt[i][j]或者 vt[j][i]至少有一个等于 1，则可以判断顶点 i 与顶点 j 之间互相通信。用二重循环对二维数组进行遍历，如果对于顶点 i，可以互相通信的顶点数目达到 numVertex，则可以判断这个顶点符合要求，count++。

举例说明：



该图的 vt 矩阵如下：

1 1 1 1
0 1 0 1
0 0 1 1
0 0 0 1

观察可以见得，第一行和第四列全为 1，及可以和所有的顶点进行互相通信。

伪代码如下：

```
int count=0;
  for(int i=0;i<N;i++)
  {
      for(int j=0;j<N;j++)
      {
          ma.DFSTraverse(&ma,j);
          vt[i][j]=ma.getMark(i);
          ma.resetMark();
      }
  }
  for(int i=0;i<N;i++)
  {
      int sum=0;
      for(int j=0;j<N;j++)
      {
          if(vt[i][j]==1||vt[j][i]==1) sum++;
      }
      if(sum==ma.n()) count++;
  }
  cout<<count<<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：

测试程序是否可运行，发现代码的语法错误、链接错误、逻辑错误、和运算错误，是否有不严谨之处。

测试样例：

4 4

1 2

1 3

2 3

3 4

调试计划：

设定好断点，在输入之前，单步执行，观察在 if 语句判断时的跳转是否正确。并 add watch 合适的变量值，观察程序执行的步骤。

设置断点：

在第一次输入命令前设置断点，之后单步执行，调用函数，跳转到对应函数中执行命令，观察对命令的执行情况（删除，插入）是否正确，及时进行修改，对代码的不完善之处进行修改。

2. 调试过程和结果，及分析

设置断点位置：

```
11      int N,M;
12      cin>>N>>M;
13      MatrixGraph ma(N);
14      ma.setType(flag);
```

单步进入，执行插入操作，首先输入 4 4. 设置顶点和边的数目。创建图。

执行图的构造函数，对图的邻接表和邻接矩阵进行初始化。

```
25  MatrixGraph::MatrixGraph(int n) {
26      Init(n);
27  }
```

设置图的类型为有向图：

```
void MatrixGraph::setType(bool flag){
    undirected = flag;
}
```

输入边信息，设置边

```
16      for(int i=0;i<M;i++)
17      {
18          cin>>v1>>v2;
19          ma.setEdge(v1-1,v2-1);
20      }
```

边的数目加一：

```
void MatrixGraph::setEdge(int v1, int v2) {
    if (matrix[v1][v2] == 0) {
        numEdge++;
```

更新邻接矩阵：

```
81 void MatrixGraph::setEdge(int v1, int v2) {
82         if (matrix[v1][v2] == 0) {
83             numEdge++;
84         }
85         matrix[v1][v2] = 1;
```

然后进行 DFS 深度优先搜索：

对已经访问过的进行标记，然后对顶点的邻接顶点进行遍历，如果找到没被标记过的顶点，则进行递归搜索。

```
void MatrixGraph::DFSTraverse(MatrixGraph* m, int v) {
    //cout << v << ' ';
    m->setMark(v, VISITED);
```

```
    for (int w = m->first(v); w < m->numVertex; w = m->next(v,
        if (m->getMark(w) == UNVISITED) {
            DFSTraverse(m, w);
        }
```

对 vt 数据进行赋值，每进行一次 DFS 之后都要重置一下 mark 数组：

```
26                 ma.DFSTraverse(&ma,j);
27                 vt[i][j]=ma.getMark(i);
28                 ma.resetMark();
29             }
```

重置 mark 数组将每一个顶点都初始化为未被访问

```
void MatrixGraph::resetMark(){
    for(int i = 0; i < numVertex; i++){
        mark[i]=UNVISITED;
    }
}
```

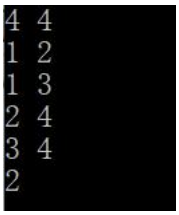
对 vt 数组进行遍历，并计算某顶点的互相通信的数目

```
31     for(int i=0;i<N;i++)
32     {
33         int sum=0;
34         for(int j=0;j<N;j++)
35         {
36             if(vt[i][j]==1||vt[j][i]==1) sum++;
37         }
38         if(sum==ma.n()) count++;
39     }
```

最后输出满足要求的顶点数目。

```
40     cout<<count<<endl;
41     return 0;
```

输出结果如图：



五、测试结果

CCF 测试通过。

六、实验日志

2019 年 5 月 30 日

开始看 CCF 的题目，2017 年的三个题目中，发现只有 9 月的题目不需要用到最短路径算法，因为我的实验五在实现图的时候并没有实现最短路径功能，所以我也就选择了这个我觉得最简单的一道题目。感觉解这个题需要思维技巧，比如说可以通过看 mark 数组中是否被标记过，来判断两个顶点之间存在通信与否。刚开始我做的时候是只考虑了这个顶点作为起始点或者终点的情况，没有考虑到顶点在路径中间也可能的情况，所以第一次提交只得了 25 分。

2019 年 5 月 31 日

开始该算法思想，如何将中间顶点也考虑进去。我发现我在 debug 输出中间的变量时，发现我所得到的 mark 数组如果按行输出的话是一个矩阵，那如何根据这个矩阵判断这个顶点是否和图中每一个顶点互相通信呢，比如说我要判断顶点 i，则就要判断 i 行和 i 列中相对应的位置至少有一个为 1，用通俗的话来讲就是 i 行和 i 列的互补一下只要达到 n 个 1 就可以了。然后改对了提交 60 分，错误原因是运行时间超时。

原来我做 CCF 的模拟题的时候，二维数组放在主函数内部，并且 100*100 这个维度的就已经为运行超时，当初的解决办法是将二维数组声明为全局变量。这里将二维数组声明为全局变量，1000*1000 的维度，即使是全局变量，也依旧运行超时。后经人点播，我把二维数组改成了 vector 容器，动态分配内存，会节省不少空间。果然改了之后就对了。

下午开始写实验报告，画流程图的时候依旧非常自闭，今天老师讲了将主流程序放在主流线上，所以我把这个流程图改得特别长哈哈，而且感觉双重 for 循环在写流程图的时候很麻烦。

这个实验两天完工，图的 ADT 创建地成功做起题来也比较顺利，果然学了数据结构之后解程序题都变得有技术含量了，算法是个利器。

继续加油！