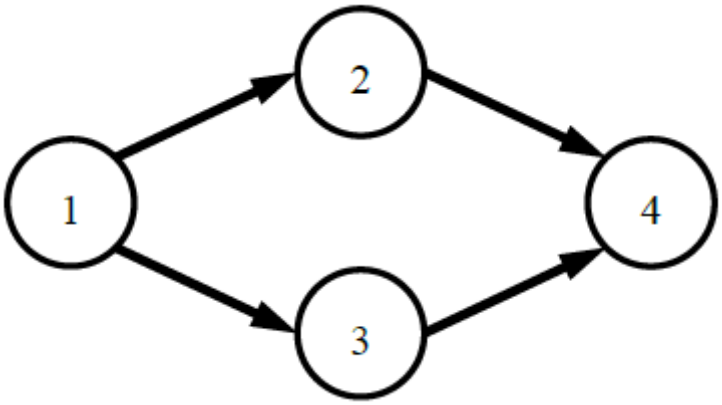
实验六- 图的应用（通信网络）

【问题描述】

> 应用图的ADT的物理实现来解决图的应用问题。
>
> 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*。

【输入形式】

> 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

【输出形式】

> 输出一行，包含一个整数，表示答案。

样例分析

> 输入
>
> - 4 4
> - 1 2
> - 1 3
> - 2 4
> - 3 4
>
> 输出

> - 2

在1 2 3 4，四个节点中，只有1和4，两个节点，可以连接所有的节点

- 1可以发给2，3，再给4
- 2、3可以给4，从1接受
- 4可以从2、3接受，再从1接受

即我们是分析了每个节点的所有情况，那么我们针对每个节点进行一次处理，相应的，我们可以用的处理方法是DFS（深度优先搜索）和BFS（广度优先搜素）

- **DFS深度优先搜索**

针对每一个节点，进行它的下一个节点的遍历，如果下一个连接的节点尚未被访问，则对该节点进行DFS搜索，若该节点已经被访问完毕，则进行下下一个连接的节点的判断和搜索。直到该节点的相应的所有连接节点（子树）访问完毕。

针对DFS搜索的处理，我们每次访问到直接相邻的节点，或者间接相邻的节点，就将我们的连通图（自建）的相应节点之间二点可访问性改为可访问。

- **BFS广度优先搜索**

针对每一个节点，进行他下一层节点的遍历，再根据每一层节点放入队列，依次进行其更下一层节点的遍历，直至所有节点访问完毕。

BFS搜索的处理和DFS搜索的处理是一样的。

在这里我们考虑BFS和DFS的实现难度，考虑到BFS应用了队列，进行队列依次的遍历，我们采用DFS来实现这个代码。

下面针对DFS思想进行简单的样例分解

首先是节点1，我们首先对节点1进行（1，1）自己的可连通性，首先检测的是1，2的连通性，连通，则（1，2）（2，1）为1，然后检测 2 的子节点，4没有被访问过，（4，1）的可连通性，则（4，1）（1，4）为1

其余节点同理

需要注意的是，我们每次访问完一个节点，需要重置每个节点的访问状态

该部分代码如下

```
        void dfs(int v1,int v2)
        {
            int w;
            setMark(v1,1);
            flag[v1][v2]=1;
            flag[v2][v1]=1;
            for(w=first(v1);w<n();w=next(v1,w))
            {
                if(getMark(w)==0)
                dfs(w,v2);
            }
        }
```

求解部分

我们已经获得了每个节点的可访问目录，则该节点的所有访问的可能情况我们都已经知道了，即如果该节点和所有其他节点的可访问性都是可访问，则该节点是符合要求的一个节点

```
        int ans()
        {
            int ans=0,j=0;
            for(int i=0;i<numVertex;i++)
            {
                for( j=0;j<numVertex&&flag[i][j]==1;j++)
                {

                }
                if(j==numVertex)ans++;
            }
            return ans;
        }
```

调用两个函数，做main函数的编写

需要注意的是x和y的0和1的问题，做自减1

```
int main(int argc, char** argv) {
    int n,m;
    cin >> n>>m;
    Graphm a(n);
    for(int i=0;i<m;i++)
    {
        int x,y;
        cin >> x>> y;
        a.setEdge(x-1,y-1,1);
    }
    for(int i=0;i<n;i++)
    {
        a.dfs(i,i);
        a.clearmark();
    }
    cout<<a.ans();
    return 0;
}
```

完成实验

实验总结

合理利用BFS和DFS，而不是warshell算法（时间复杂度太高，循环次数太多）3

我们在做实验时，应当首先考虑算法，想一个实现的方法，再手动去写代码实现，这样比每次写了代码再debug有效率的多