快速开租建站

知识点 BFS 搜索拓扑排序
时间限制：1s 空间限制：256MB 限定语言：不限

题目描述：

当前IT部门支撑了子公司颗粒化业务，该部门需要实现为子公司快速开租建站的能力，建站是指在一个全新的环境部署一套IT服务。每个站点开站会由一系列部署任务项构成，每个任务项部署完成时间都是固定和相等的，设为1。部署任务项之间可能存在依赖，假如任务2依赖任务1，那么等任务1部署完，任务2才能部署。任务有多个依赖任务则需要等所有依赖任务都部署完该任务才能部署。没有依赖的任务可以并行部署，优秀的员工们会做到完全并行无等待的部署。给定一个站点部署任务项和它们之间的依赖关系，请给出一个站点的最短开站时间。

输入描述：

第一行是任务数taskNum,第二行是任务的依赖关系数relationsNum
接下来 relationsNum 行，每行包含两个id，描述一个依赖关系，格式为：IDi IDj，表示部署任务i部署完成了，部署任务j才能部署，IDi 和 IDj 值的范围为：[0, taskNum)
注：输入保证部署任务之间的依赖不会存在环。

输出描述：

一个整数，表示一个站点的最短开站时间。

补充说明：

1<taskNum<=100
1=<relationsNum<=5000

示例1

输入：

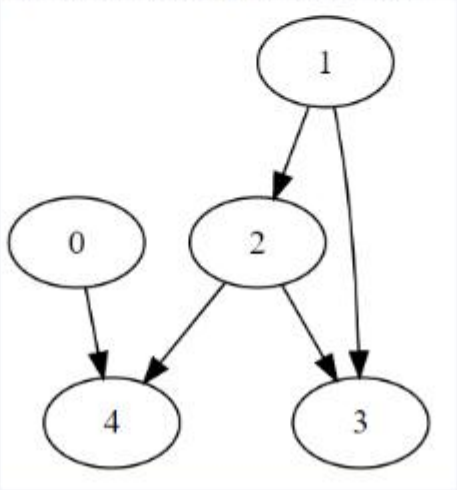
```
5
5
0 4
1 2
1 3
2 3
2 4
```

输出：

```
3
```

说明：

有5个部署任务项，5个依赖关系，如下图所示。我们可以先同时部署任务项0和任务项1，然后部署任务项2，最后同时部署任务项3和任务项4。最短开站时间为3。



示例2

输入：

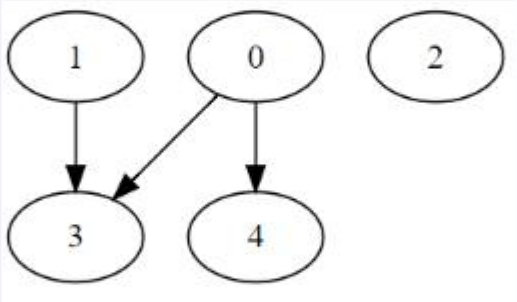
5
3
0 3
0 4
1 3

输出：

2

说明：

有5个部署任务项，3个依赖关系，如下图所示。我们可以先同时部署任务项0，任务项1，任务项2。然后再同时部署任务项3和任务项4。最短开站时间为2。



解题思路：

以示例1为例：

1. **根据输入的value值可知，0和1不需要依赖，直接部署**
2. **求出需要依赖步骤1中部署的0和1的服务有4,2,3；再求出剩下的需要依赖的还有3,4；两者合并，则只有2可以部署**
3. **求出需要步骤2中部署的2的服务有3和4；再求出剩下的需要依赖的服务没有了；两者合并，3和4可以部署**
4. **所有服务部署完毕需要3个时间**

```
public class Main{

    /**
     * key：服务器
     * value：需要依赖 key 的服务器
     */
    public static Map<Integer, Set<Integer>> map = new HashMap<>();
    public static int res = 1;
```

```
public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int taskNum = sc.nextInt();
        int relationsNum = sc.nextInt();

        int[] taskInts = new int[taskNum];    //用来记录需要依赖的服务

        for(int i=0; i<relationsNum; i++){
            Set<Integer> inputList;
            int k = sc.nextInt();
            int v = sc.nextInt();
            if(map.containsKey(k)){
                inputList = map.get(k);
            }else {
                inputList = new HashSet<>();
            }
            inputList.add(v);
            map.put( k, inputList);
            taskInts[v]++;
        }

        Set<Integer> taskStart = new HashSet<>();      //已经部署的服务
        Set<Integer> taskWait = new HashSet<>();       //等待部署的服务
        for(int i=0; i<taskNum; i++){
            if(taskInts[i] == 0){
                taskStart.add(i);      //不需要依赖的服务直接部署
            }else {
                taskWait.add(i);       //需要依赖的服务等待部署
            }
        }

        handle( taskStart, taskWait);
        System.out.println(res);
}

/**
 * 1、先求出依赖已经部署的服务的服务
 * 2、从步骤 1 中的集合移除还需要其他依赖的服务
 * 3、根据步骤 2 求得的集合更新  taskStart 和 taskWait
 * @param taskStart         已经部署的服务
 * @param taskWait          等待部署的服务
 */
```

```
    public static void handle( Set<Integer> taskStart, Set<Integer> taskWait){

        if(taskWait.size() != 0){     //还有服务等待部署
            Set<Integer> temp = new HashSet<>();       //接下来可以部署的服务
            for(int i : taskStart){
                if(map.containsKey(i)){
                    temp.addAll(map.get(i));       //依赖已经部署的服务的服务
                }
            }

            for(int i : map.keySet()){
                if(!taskStart.contains(i)){
                    temp.removeAll(map.get(i));        //移除那些还需要依赖的服务
                }
            }

            taskStart.addAll(temp);
            taskWait.removeAll(temp);
            res ++;         //部署过一次  ++
            handle( taskStart, taskWait);
        }
    }
}
```