题目描述：

某通信网络中有 N 个网络结点，用 1 到 N 进行标识。网络中的结点互联互通，且结点之间的消息传递有时延，相连结点的时延均为一个时间单位。

现给定网络结点的连接关系 link[i]={u,v}，其中 u 和 v 表示网络结点。

当指定一个结点向其他结点进行广播，所有被广播结点收到消息后都会在原路径上回复一条响应消息，请计算发送结点至少需要等待几个时间单位才能收到所有被广播结点的响应消息。

注：

1、N 的取值范围为[1,100]；

2、连接关系 link 的长度不超过 3000，且 1 <= u,v <= N；

3、网络中任意结点间均是可达的；

输入描述：

输入的第一行为两个正整数，分别表示网络结点的个数 N，以及时延列表的长度 l；

接下来的 l 行输入，表示结点间的连接关系列表；

最后一行的输入为一个正整数，表示指定的广播结点序号；

输出描述：

输出一个整数，表示发送结点接收到所有响应消息至少需要等待的时长。

示例 1

输入：

```
5 7
2 1
1 4
2 4
2 3
3 4
3 5
4 5
2
```

输出：

```
4
```

说明：

2 到 5 的最小时延是 2 个时间单位，而 2 到其他结点的最小时延是 1 个时间单位，所以 2 收到所有结点的最大响应时间为 2*2=4。

```
import java.util.*;

public class Main {
    static class Edge {
        final int x;
        final int y;

        Edge(int x, int y) {
            this.x = x;
            this.y = y;
        }
    }
    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int l = scanner.nextInt();
        List<Edge> edgeList = new ArrayList<>();
        for (int i = 0; i < l; i++) {
            int x = scanner.nextInt();
            int y = scanner.nextInt();
            Edge edge = new Edge(x, y);
            edgeList.add(edge);
        }
        int v = scanner.nextInt();

        int r = getResult(n, l, edgeList, v);
        System.out.println(r);
    }

    private static int getResult(int n, int l, List<Edge> edgeList, int v) {
        if (n == 1) {
            return 0;
        }
        Map<Integer, LinkedHashSet<Integer>> edgeMap = new HashMap<>();
        for (Edge edge : edgeList) {
            int x = edge.x;
            int y = edge.y;
            if (!edgeMap.containsKey(x)) {
```

```
                edgeMap.put(x, new LinkedHashSet<>());
            }
            if (!edgeMap.containsKey(y)) {
                edgeMap.put(y, new LinkedHashSet<>());
            }
            edgeMap.get(x).add(y);
            edgeMap.get(y).add(x);
        }

        HashSet<Integer> reached = new HashSet<>();
        HashSet<Integer> lastTurn = new HashSet<>();

        reached.add(v);
        lastTurn.add(v);

        for (int i = 1; i < n; i++) {
            HashSet<Integer> thisTurn = new HashSet<>();
            for (int t : lastTurn) {
                LinkedHashSet<Integer> vs = edgeMap.get(t);
                if (vs == null) {
                    vs = new LinkedHashSet<>();
                }
                for (int k : vs) {
                    if (!reached.contains(k)) {
                        reached.add(k);
                        thisTurn.add(k);
                    }
                }
            }
            if (reached.size() == n) {
                return i * 2;
            }
            lastTurn = thisTurn;
        }
        throw new RuntimeException();
    }
}
```