55、跳格子游戏，考点 or 实现——图论/拓扑排序

题目描述

地上共有N个格子，你需要跳完地上所有的格子，但是格子间是有强依赖关系的，跳完前一个格子后，后续的格子才会被开启，格子间的依赖关系由多组steps数组给出，steps[0]表示前一个格子，steps[1]表示steps[0]可以开启的格子:

比如[0,1]表示从跳完第0个格子以后第1个格子就开启了，比如[2,1]，[2,3]表示跳完第2个格子后第1个格子和第3个格子就被开启了。

请你计算是否能由给出的steps数组跳完所有的格子，如果可以输出yes，否则输出no。

说明:

1.你可以从一个格子跳到任意一个开启的格子

2.没有前置依赖条件的格子默认就是开启的

3.如果总数是N，则所有的格子编号为[0,1,2,3...N-1]连续的数组

输入描述

输入一个整数N表示总共有多少个格子，接着输入多组二维数组steps表示所有格子之间的依赖关系。

输出描述

如果能按照steps给定的依赖顺序跳完所有的格子输出yes，

否则输出no。

备注

- 1 ≤ N ＜ 500
- step[i].length = 2
- 0 ≤ step[i][0]，step[i][1] ＜ N

用例

输入	3 0 1 0 2
输出	yes
说明	总共有三个格子[0,1,2]，跳完0个格子后第1个格子就开启了，跳到第0个格子后第2个格子也被开启了，按照0->1->2或者0->2->1的顺序都可以跳完所有的格子

输入	2 1 0 0 1
输出	no
说明	总共有2个格子，第1个格子可以开启第0格子，但是第1个格子又需要第0个格子才能开启，相互依赖，因此无法完成

输入	6 0 1 0 2 0 3 0 4 0 5
输出	yes
说明	总共有6个格子，第0个格子可以开启第1,2,3,4,5个格子，所以跳完第0个格子之后其他格子都被开启了，之后按任何顺序可以跳完剩余的格子

输入	5 4 3 0 4 2 1 3 2
输出	yes
说明	跳完第0个格子可以开启格子4，跳完格子4可以开启格子3，跳完格子3可以开启格子2，跳完格子2可以开启格子1，按照0->4->3->2->1这样就跳完所有的格子

输入	4 1 2 1 0
输出	yes
说明	总共4个格子[0,1,2,3]，格子1和格子3没有前置条件所以默认开启，格子1可以开启格子0和格子2，所以跳到格子1之后就可以开启所有的格子，因此可以跳完所有格子。

题目解析

另外，本题输入描述中未给出输入结束条件，因此，我这里判断如果输入是一个空串，则判定为输入结束。

Java算法源码

```java
import java.util.*;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int n = Integer.parseInt(sc.nextLine());

    ArrayList<Integer[]> relations = new ArrayList<>();
    while (sc.hasNextLine()) {
      String line = sc.nextLine();
      if ("".equals(line)) break; // 题目没有说输入终止条件，因此我这里将输入空行作为终止条件
      relations.add(Arrays.stream(line.split(" ")).map(Integer::parseInt).toArray(Integer[]::new));
    }

    System.out.println(getResult(n, relations));
  }
```

```java

  public static String getResult(int n, ArrayList<Integer[]> relations) {
    int[] inDegree = new int[n];
    HashMap<Integer, ArrayList<Integer>> next = new HashMap<>();

    for (Integer[] relation : relations) {
      int a = relation[0], b = relation[1];
      inDegree[b]++;
      next.putIfAbsent(a, new ArrayList<>());
      next.get(a).add(b);
    }

    LinkedList<Integer> stack = new LinkedList<>();
    for (int i = 0; i < n; i++) {
      if (inDegree[i] == 0) {
        stack.add(i);
      }
    }

    int count = 0;
    while (stack.size() > 0) {
      int a = stack.removeLast();
      count++;

      if (next.containsKey(a)) {
        for (int b : next.get(a)) {
          if (--inDegree[b] == 0) {
              stack.add(b);
            }
          }
        }
      }

      return count == n ? "yes" : "no";
    }
}
```

JS算法源码

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
rl.on("line", (line) => {
  if (line === "") {
    const n = parseInt(lines.shift());
    const relations = lines.map((line) => line.split(" ").map(Number));

    console.log(getResult(n, relations));

    lines.length = 0;
  } else {
    lines.push(line);
  }
});

function getResult(n, relations) {
```

```
  const inDegree = new Array(n).fill(0); // 每个顶点的入度
  const next = {}; // 每个顶点的后继

  for (let i = 0; i < relations.length; i++) {
    const [a, b] = relations[i]; // 学完a，才能学b，即 a → b

    inDegree[b]++; // b顶点入度+1
    next[a] ? next[a].push(b) : (next[a] = [b]); // a顶点的后继加入b
  }

  const stack = [];
  for (let i = 0; i < n; i++) {
    if (inDegree[i] === 0) {
      stack.push(i);
    }
  }

  let count = 0;
  while (stack.length) {
    const a = stack.pop();
    count++;

    next[a]?.forEach((b) => {
      if (--inDegree[b] === 0) stack.push(b);
    });
  }

  return count === n ? "yes" : "no";
}
```

Python算法源码

```python
# 输入获取
n = int(input())

relations = []
while True:
    line = input()

    # 本题没有给输入终止条件，因此我已输入空串为结束条件
    if "" == line:
        break

    relations.append(list(map(int, line.split())))


# 算法入口
def getResult():
    inDegree = [0] * n
    next = {}

    for a, b in relations:
        inDegree[b] += 1
        next.setdefault(a, [])
        next[a].append(b)

    stack = []
    for i in range(n):
        if inDegree[i] == 0:
```

```
            stack.append(i)

    count = 0
    while len(stack) > 0:
        a = stack.pop()
        count += 1

        if next.get(a) is not None:
            for b in next[a]:
                inDegree[b] -= 1
                if inDegree[b] == 0:
                    stack.append(b)

    return "yes" if count == n else "no"


# 算法调用
print(getResult())
```