https://course.acciojob.com/idle?question=98332447-d28d-42ee-9843-3d4d9d5ea20d

- MEDIUM
- Max Score: 40 Points
-
-
-

Course Schedule

You are given a number N, the number of courses you have to take labeled from 0 t N-1. You are given an array `prerequisites` where `prerequisites[i] = [ai, bi]` indicates that you must take course `bi` first if you want to take course `ai`.

eg: [2,4] means take course `4` before you can take course `2`.

Input Format

The First line of input contain two integers N denoting number of people and M denoting size of prerequisites array.

Next line contains two integer each denoting `ai` and `bi`.

Output Format

print 1 if it is possible to finish all the courses else print 0.

Example 1

Input

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

Output

```
1
```

Explanation

We need to take course `2,3` **and** `0` **before taking course** `1`**. Since no conflict is there, we can take it.**

Example 2

Input

```
4 3
1 2
2 3
3 1
```

Output

```
0
```

Explanation

The prerequisites for each course can be mapped as

- 1 -> 2
- 2 -> 3
- 3 -> 1

Any two of these courses can be done without any issue, but when the third course comes in, the prerequisite of the prerequisite becomes one the course you need to take and it's impossible to take all of it.

Constraints

- **1 <= N <=2000**
- **0 <= M <= 5000**
- **1 <= prerequisite[i][j] < N**

Topic Tags

- Graphs
- BFS

My code

```
// n java
import java.util.*;

class Solution {

    @SuppressWarnings("unchecked")
    static void dfs(ArrayList<ArrayList<Integer>> adj, int x, boolean[]
vis, Stack<Integer> s) {
        if(vis[x] == true) return;

        vis[x] = true;

        for(int i = 0; i < adj.get(x).size(); i++) {
            dfs(adj, adj.get(x).get(i), vis, s);
        }

        s.push(x);
    }

        public boolean check(ArrayList<ArrayList<Integer>> g,
Stack<Integer> s) {
```

```
                Map<Integer, Integer> mp = new HashMap<Integer,
Integer>();

                int n = g.size();

                int counter = 0;
                while(s.isEmpty() == false) {
                        mp.put(s.peek(), counter);

                        s.pop();
                        counter++;
                }

                for(int i = 0; i < n; i++) {
                        for(int j = 0; j < g.get(i).size(); j++) {
                                int y = g.get(i).get(j);
                                if(mp.get(i) > mp.get(y)) return false;
                        }
                }

                return true;
        }

    public int canFinish(int n, int[][] prerequisites) {
        //write your code here

                ArrayList<ArrayList<Integer>> adj = new ArrayList<>();

                for(int i = 0; i < n; i++) {
                        adj.add(new ArrayList<Integer>());
                }

                for(int i = 0; i < prerequisites.length; i++) {
                        int x = prerequisites[i][0], y = prerequisites[i][1];

                        adj.get(y).add(x);
                }

                boolean[] vis = new boolean[n];
        Stack<Integer> s = new Stack<>();

        for(int i = 0; i < n; i++) {
            if(vis[i] == false) {
                dfs(adj, i, vis, s);
            }
        }

        boolean possible = check(adj, s);

                if(possible) return 1;
                return 0;
    }

    }

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N= sc.nextInt();
        int M= sc.nextInt();
        int prerequisites[][] = new int[M][2];

        for(int i=0; i<M; i++){
            for(int j=0; j<2; j++)
                prerequisites[i][j]= sc.nextInt();
        }

        Solution Obj = new Solution();
        System.out.println(Obj.canFinish(N,prerequisites));

    }
}
```