- HARD
- Max Score: 60 Points
- https://course.acciojob.com/idle?question=717ac001-d919-4e01-ae4e-db1cbc4b8cba

Possible Bipartition

We want to split a group of `n` people (labeled from 1 to `n`) into two groups of any size. Each person may dislike some other people, and they should not go into the same group.

Given the integer `n` and the array dislikes where `dislikes[i] = [ai, bi]` indicates that the person labeled `ai` does not like the person labeled `bi`, return true if it is possible to split everyone into two groups in this way.

Input Format

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

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

Output Format

print 1 if it is possible to split everyone into two groups else print 0.

Example 1

Input

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

Output

```
1
```

Explanation

We can divide the array into two parts. The two groups are {1} and {2,3,4}.

Example 2

Input

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

Output

```
0
```

Explanation

Since 1 hates 2 and 3, it can't be grouped with them, and since 2 hates 3, 2 and 3 can't be together either. Hence we need at least three groups to keep all the people who don't like each other separately, but we need to divide only in 2.

Constraints

- **1 <= N <=2000**
- **0 <= M <= 10^4**
- **1 <= dislikes[i][j] <=2**
- **ai < bi**
- **0 <= |arr[i]| <= 2**

Topic Tags

- **Graphs**
- **BFS**
- **DFS**

My code

```
import java.util.*;

// class Solution {

//     public int possibleBipartition(int n, int[][] dislikes) {
//         // Write your code here
//     }
// }
class Solution {
  public boolean dfs(ArrayList<ArrayList<Integer>> g, int x, int col, int[] vis) {
          if(vis[x] != -1) {
                  if(vis[x] == col) return true;
                  else return false;
          }

          vis[x] = col;
                boolean ans = true;
          for(int i = 0; i < g.get(x).size(); i++) {
                        boolean temp = dfs(g, g.get(x).get(i), 1 - col, vis);
                  ans = (ans & temp);
          }

          return ans;
  }

   public int possibleBipartition(int n, int[][] dislikes) {
      // Write your code here
                ArrayList<ArrayList<Integer>> g = new ArrayList<>();

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

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

                        g.get(x).add(y);
                        g.get(y).add(x);
                }
```

```
                    int[] vis = new int[n];
                    Arrays.fill(vis, -1);

                    boolean ans = true;
                    for(int i = 0; i < n; i++) {
                              if(vis[i] == -1) {
                                        boolean temp = dfs(g, i, 0, vis);
                                        ans = (ans & temp);
                              }
                    }

                    if(ans) return 1;
                    else 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 dislike[][] = new int[M][2];

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

      Solution Obj = new Solution();
      System.out.println(Obj.possibleBipartition(N,dislike));

   }
}
```