https://course.acciojob.com/idle?question=36827380-5496-4eb7-a316-d683ea4b3e7e

- MEDIUM
- Max Score: 40 Points

Unreachable Pairs of Nodes in an Undirected Graph

You are given an undirected graph consisting of `n` nodes numbered from 0 to n-1. You are also given a 2D matrix `edges` where edges[i] = [ai, bi] denotes that there exists an undirected edge connecting nodes ai and bi.

Your task is to return the number of pairs of different nodes that are unreachable from each other.

Input Format

In example input, the first line contains two integers `n` and `k` representing the number of nodes and size of matrix edges.

The next `n` lines contain two space-separated integers denoting the undirected edges of the graph.

Output Format

Each test case returns the number of pairs of different nodes that are unreachable from each other.

Example 1

Input

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

```
0 2
```

Output

```
0
```

Explanation

```
The given graph is a complete graph of size 3. We can reach any node from any node.
Therefore answer is 0.
```

Example 2

Input

```
7 5
0 2
0 5
2 4
1 6
5 4
```

Output

```
14
```

Explanation

```
There are 14 pairs of nodes that are unreachable from each other:
[0,1], [0,3], [0,6], [1,2], [1,3], [1,4], [1,5], [2,3], [2,6], [3,4], [3,5], [3,6],
[4,6], [5,6]].
Therefore, the answer is 14.
```

Constraints

1 <= `n` <= 10^5

1 <= `k` <= 2*10^5

0 <= `edges[i][j]` < n

Topic Tags

- Graphs
- BFS
- DFS

My code

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

class Main {

	public static void main(String[] args)
	{
		int n,k;
	Scanner sc =new Scanner(System.in);
	n = sc.nextInt();
	k = sc.nextInt();
	int[][] arr = new int [k][2];
	for(int i=0;i<k;i++){
	    arr[i][0]=sc.nextInt();
	    arr[i][1]=sc.nextInt();
	}
		System.out.println(new Solution().countPairs(n,arr));
	}
}

// class Solution {
```

```
//     public long countPairs(int n, int[][] edges) {
//         //Write code here
//     }
// }
class Solution {

        public void dfs(ArrayList<ArrayList<Integer>> graph, int x,
boolean[] vis) {
                if(vis[x] == true) return;

                vis[x] = true;

                System.out.println("Entered at " + x);

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

                System.out.println("Exit from " + x);
        }

        public long bfs(ArrayList<ArrayList<Integer>> graph, int x,
boolean[] vis) {
                LinkedList<Integer> q = new LinkedList<Integer>();

                q.add(x);
                vis[x] = true;

                long ans = 0;
```

```
            while(q.size() > 0) {
                int f = q.poll();
                // System.out.print(f + " ");
                ans++;

                for(int i = 0; i < graph.get(f).size(); i++) {
                    int y = graph.get(f).get(i);

                    if(vis[y] == false) {
                        q.add(y);
                        vis[y] = true;
                    }
                }
            }

            return ans;
        }

    public long countPairs(int n, int[][] edges) {
        //Write code here

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

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

```
        for(int i = 0; i < edges.length; i++) {
            int x = edges[i][0], y = edges[i][1];
            graph.get(x).add(y);
            graph.get(y).add(x);
        }

        boolean[] vis = new boolean[n];

        ArrayList<Long> comps = new ArrayList<Long>();

        for(int i = 0; i < n; i++) {
            if(vis[i] == false) {
                long c = bfs(graph, i, vis);
                comps.add(c);
            }
        }

        long ans = 0;

        for(int i = 0; i < comps.size(); i++) {
            for(int j = i+1; j < comps.size(); j++) {
                ans += (comps.get(i) * comps.get(j));
            }
        }

        return ans;
    }
}
```