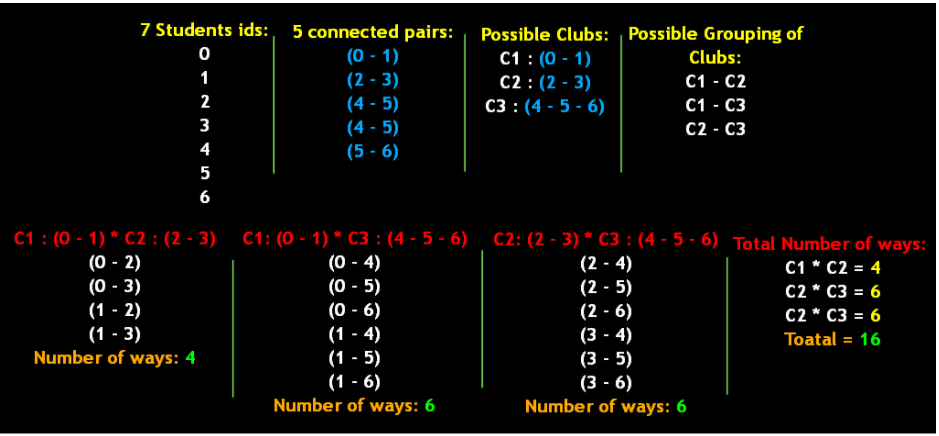
**1. Problem Discussion :**

In this problem you are given a number n (representing the number of students). Each student will have an id from 0 to n - 1. You are also given a number k (representing the number of pairs of students, clubbed together). In the next k lines, two numbers are given separated by a space. The numbers are ids of students belonging to the same club. You have to find in how many ways can we select a pair of students such that both students are from different clubs. Input is managed for you.

For example:

Sample Input -> 7 5 0 1 2 3 4 5 5 6 4 6 Sample Output -> 16

How 16?

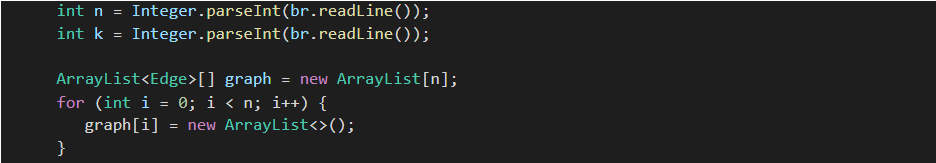
****

We are given 7 student's ids and 5 pairs of students from the same club. These 5 pairs can make possible 3 clubs, first being (0 - 1), second (2 - 3) and third, (4 - 5 - 6) because (4 - 5), (5 - 6) and (4 - 6) are pairs from the same club. Now we can group these 3 clubs in three ways: (C1-C2), (C1-C3) and (C2-C3). If one student is chosen from each of C1 and C2 clubs then there are 4 possible pairs. If one student is chosen from each of C1and C3 clubs then there are 6 possible pairs. If one student is chosen from each of C2 and C3 clubs then there are 6 possible pairs. Therefore total ways to pair students from different clubs are 16. For better understanding of the question, watch this part of the video lecture.

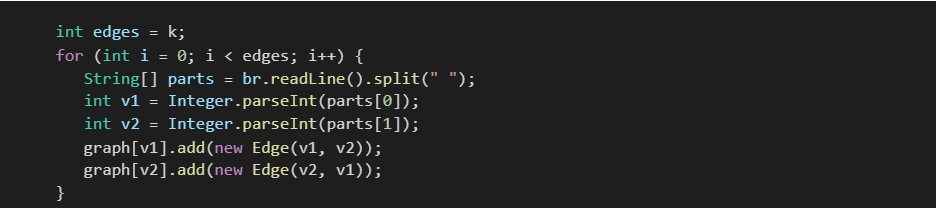
**2. Approach :**

We start with making a graph. For making a graph, we need to define the edge. In previous lectures of Graphs, we used to have 3 data members for edge, but here we can drop the idea of weight as the data member because it's not required. So, in the new edge, we will only keep vertices and neighbors. And then define a constructor for this edge class. Watch this part this of the video lecture for better understanding of this portion.

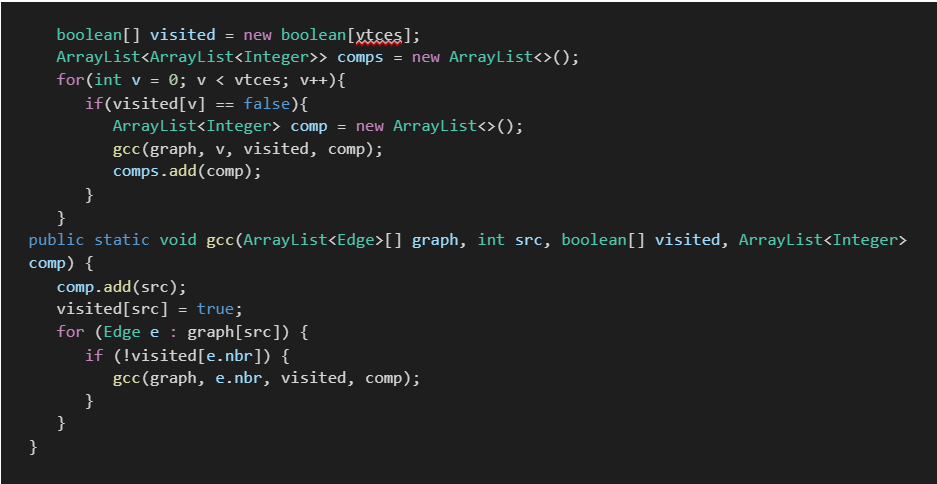
After this in main, n, which represents the number of students, also represents the number of vertices in the graph. We define an "n" length Array List of arrays (of type edge) and name it graph. Then we apply a for loop on the graph's length to put an array list at each of it's index. Watch this part this of the video lecture for better understanding of this portion.

****

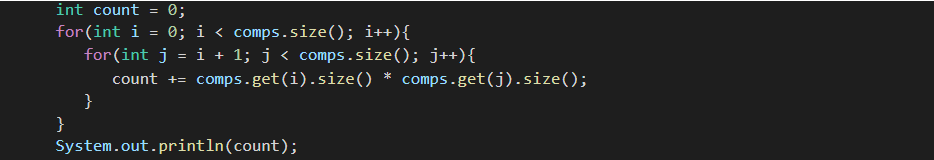
K, which represents the number of pairs, also represents the number of edges in the graph. So, to set these edges in the graph, we need to run a for loop k times. And in each iteration we collect and interpret the input from each line, which represents the pairs of students in the same club. To do so we define an array of strings, parts and using the readline function of buffer reader, we split the line when we find a space. Then we collect both the number representing vertices and which are now stored in the array parts, in some variables say v1 and v2. Then finally we add the edges at v1 and v2 indices of the graph. Watch this part this of the video lecture for better understanding of this portion.

****

After that we use the same logic that we discussed in Get Connected Components (GCC). So we will not be discussing the code of Get Connected Components. For any doubt, watch GCC's video lecture.

****

After applying gcc logic we will get the connected components of the graph stored in an array list of arraylists, comps. Each connected component represents each club. It means comps store all clubs and their members. Now to get our final answer, we need to do some simple math in main. Initialize a variable count to 0. Then we run a for loop on the array list storing connected components initializing i=0. And a nested for loop, initializing j with i+1. In each iteration, we update count with the product of, sizes of clubs stored at index i and j. Sizes of clubs are basically size of arraylist at ith index in components and size of arraylist at jth index in components.

****

**3. Code :**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

static class Edge {

int src;

int nbr;

Edge(int src, int nbr) {

this.src = src;

this.nbr = nbr;

}

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int k = Integer.parseInt(br.readLine());

int vtces = n;

ArrayList< Edge>[] graph = new ArrayList[vtces];

for (int i = 0; i < vtces; i++) {

graph[i] = new ArrayList< >();

}

int edges = k;

for (int i = 0; i < edges; i++) {

String[] parts = br.readLine().split(" ");

int v1 = Integer.parseInt(parts[0]);

int v2 = Integer.parseInt(parts[1]);

graph[v1].add(new Edge(v1, v2));

graph[v2].add(new Edge(v2, v1));

}

boolean[] visited = new boolean[vtces];

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

for (int v = 0; v < vtces; v++) {

if (visited[v] == false) {

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

gcc(graph, v, visited, comp);

comps.add(comp);

}

}

int count = 0;

for (int i = 0; i < comps.size(); i++) {

for (int j = i + 1; j < comps.size(); j++) {

count += comps.get(i).size() \* comps.get(j).size();

}

}

System.out.println(count);

}

public static void gcc(ArrayList< Edge>[] graph, int src, boolean[] visited, ArrayList< Integer>

comp) {

comp.add(src);

visited[src] = true;

for (Edge e : graph[src]) {

if (!visited[e.nbr]) {

gcc(graph, e.nbr, visited, comp);

}

}

}

}

**4. Analysis :**

Time Complexity:

O(V+E) Because the DFS approach has been used.

Space Complexity :

O(v) for visited array