Let’s look at an example!

Problem

You are given pairs of integers between 0 to n-1. Each pair represent that each number belong to same set. Now you have to find the number of ways of choosing a pair such that no two number in the pair belongs to the same set.

link: https://www.hackerrank.com/challenges/journey-to-the-moon

How to solve?

This question can be solved using dfs :) You are getting a number of pairs, where each number in the pair belongs to the same set. So, make an undirected graph from these pairs i.e. pair (a,b) will mean that we have two vertices a and b and there is an undirected edge between them.
Basically, the idea is to make disjoint components from the incoming pairs. Once we have the disjoint components, we can easily calculate the answer.

Step 1: Create components

```
First of all create 'n' vertices from 0 to n-1.
```

```
Now for each pair (a,b) create an undirected edge between
 vertx 'a' to vertex 'b'.
We have our grpah ready and divided into disjoint compone
nts.
```

Step 2: Calculating the number of ways

```
We need to calculate the final answer which is the number
 of ways of selecting a pair (a,b) such that a and b does
 not belong to the same set/component.

Let the total number of components = M
Number of element i-th component = Mi
Totol number of vertices = N

Total number of ways of selecting a pair from n vertices
= nC2
Total number of ways of selecting a pair from i-th compon
ent = (Mi)C2
Total number of components = M

final answer = nC2 - (∑MiC2) for i = 1 to M
```

Code

```
#include<bits/stdc++.h>

using namespace std; // }}}
```

```
void dfs(int v, bool *vis, int &vertices, vector<vector<i
nt> > adj){
    vis[v] = true;
    vertices = vertices + 1;
    //for(int i = 0;i<adj[v].size();i++)
    for(vector<int>::iterator it = adj[v].begin();it != a
dj[v].end();it++){
        if(!vis[*it]){
            dfs(*it,vis,vertices,adj);
        }
    }
}

int main()
{
    int N, P, a,b;
    cin >> N >> P;

    ///graph definition --> Adjacency list
    vector<vector<int> > adj(N);

    ///Component definition
    int no_of_comp = 0;
    vector<int> component(N);

    ///vis array
    bool *vis = new bool[N];
    memset(vis,0,sizeof vis);

    for(int i = 0;i<P;i++){
        cin>>a>>b;
        adj[a].push_back(b);
        adj[b].push_back(a);
    }

    ///Dfs to find the number of components and number of
 vertices in each component
    for(int i = 0;i<N;i++){
        if(!vis[i]){
            int vertices = 0;
            dfs(i,vis,vertices,adj);
            component[no_of_comp] = vertices;
            no_of_comp++;
        }
    }

    long long total_ways = 1LL*N*(N-1)/2;
    long long wrong_ways = 0;

    for(int i = 0;i < no_of_comp;i++){
        wrong_ways += 1LL*component[i]*(component[i]-1)/2
;
    }

    long long res = total_ways - wrong_ways;
    cout<<res<<endl;

    return 0;
}
```

Time Complexity: O(N + P)

Created By ***Vishal Panwar***