Friend Circles

There are **N** students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a **direct** friend of B, and B is a **direct** friend of C, then A is an **indirect** friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a **N*N** matrix **M** representing the friend relationship between students in the class. If M[i][j] = 1, then the i_{th} and j_{th} students are **direct** friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

Example 1:

```
Input:
[[1,1,0],
 [1,1,0],
 [0,0,1]]
Output: 2
Explanation:The 0th and 1st students are direct friends, so they are in a friend circl
e.
The 2nd student himself is in a friend circle. So return 2.
```

Example 2:

```
Input:
[[1,1,0],
 [1,1,1],
 [0,1,1]]
Output: 1
Explanation:The 0th and 1st students are direct friends, the 1st and 2nd students are d
irect friends,
so the 0th and 2nd students are indirect friends. All of them are in the same friend c
ircle, so return 1.
```

Note:

1. N is in range [1,200].
2. M[i][i] = 1 for all students.
3. If M[i][j] = 1, then M[j][i] = 1.

Solution 1

This is a typical `Union Find` problem. I abstracted it as a standalone class. Remember the template, you will be able to use it later.

```java
public class Solution {
    class UnionFind {
        private int count = 0;
        private int[] parent, rank;

        public UnionFind(int n) {
            count = n;
            parent = new int[n];
            rank = new int[n];
            for (int i = 0; i < n; i++) {
                parent[i] = i;
            }
        }

        public int find(int p) {
         while (p != parent[p]) {
                parent[p] = parent[parent[p]];    // path compression by halving
                p = parent[p];
            }
            return p;
        }

        public void union(int p, int q) {
            int rootP = find(p);
            int rootQ = find(q);
            if (rootP == rootQ) return;
            if (rank[rootQ] > rank[rootP]) {
                parent[rootP] = rootQ;
            }
            else {
                parent[rootQ] = rootP;
                if (rank[rootP] == rank[rootQ]) {
                    rank[rootP]++;
                }
            }
            count--;
        }

        public int count() {
            return count;
        }
    }

    public int findCircleNum(int[][] M) {
        int n = M.length;
        UnionFind uf = new UnionFind(n);
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
                if (M[i][j] == 1) uf.union(i, j);
            }
        }
        return uf.count();
    }
}
```

written by shawngao original link here

Solution 2

```java
public class Solution {
    public void dfs(int[][] M, int[] visited, int i) {
        for (int j = 0; j < M.length; j++) {
            if (M[i][j] == 1 && visited[j] == 0) {
                visited[j] = 1;
                dfs(M, visited, j);
            }
        }
    }
    public int findCircleNum(int[][] M) {
        int[] visited = new int[M.length];
        int count = 0;
        for (int i = 0; i < M.length; i++) {
            if (visited[i] == 0) {
                dfs(M, visited, i);
                count++;
            }
        }
        return count;
    }
}
```

written by vinod23 original link here

Solution 3

Solution 1, using a SciPy function:

```
import scipy.sparse

class Solution(object):
    def findCircleNum(self, M):
        return scipy.sparse.csgraph.connected_components(M)[0]
```

Solution 2, compute the transitive closure of the (boolean) matrix and count the number of different rows:

```
import numpy as np

class Solution(object):
    def findCircleNum(self, M):
        return len(set(map(tuple, (np.matrix(M, dtype='bool')**len(M)).A)))
```

written by StefanPochmann original link here