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 0_{th} and 1_{st} students are direct friends, so they are in a friend circle.
The 2_{nd} 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 0_{th} and 1_{st} students are direct friends, the 1_{st} and 2_{nd} students are direct friends,
so the 0_{th} and 2_{nd} students are indirect friends. All of them are in the same friend circle, 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

From some source, we can visit every connected node to it with a simple DFS. As is the case with DFS's, **seen** will keep track of nodes that have been visited.

For every node, we can visit every node connected to it with this DFS, and increment our answer as that represents one friend circle (connected component.)

```python
def findCircleNum(self, A):
    N = len(A)
    seen = set()
    def dfs(node):
        for nei, adj in enumerate(A[node]):
            if adj and nei not in seen:
                seen.add(nei)
                dfs(nei)

    ans = 0
    for i in xrange(N):
        if i not in seen:
            dfs(i)
            ans += 1
    return ans
```

written by awice 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

```cpp
class Solution {
public:
    int findCircleNum(vector<vector<int>>& M) {
        if (M.empty()) return 0;
        int n = M.size();
        vector<bool> visited(n, false);
        int groups = 0;
        for (int i = 0; i < visited.size(); i++) {
            groups += dfs(i, M, visited) > 0;
        }
        return groups;
    }

private:
    int dfs(int i, vector<vector<int>>& M, vector<bool>& visited) {
        if (visited[i]) {
            return 0;
        }

        int ppl = 1;
        visited[i] = true;
        for (int j = 0; j < visited.size(); j++) {
            if (i != j && M[i][j]) {
                ppl += dfs(j, M, visited);
            }
        }

        return ppl;
    }
};
```

Could be shorter. If the dfs() doesn't care how many people in each group;

```cpp
class Solution {
public:
    int findCircleNum(vector<vector<int>>& M) {
        if (M.empty()) return 0;
        int n = M.size();
        vector<bool> visited(n, false);
        int groups = 0;
        for (int i = 0; i < visited.size(); i++) {
            groups += !visited[i] ? dfs(i, M, visited), 1 : 0;
        }
        return groups;
    }

private:
    void dfs(int i, vector<vector<int>>& M, vector<bool>& visited) {
        visited[i] = true;
        for (int j = 0; j < visited.size(); j++) {
            if (i != j && M[i][j] && !visited[j]) {
                dfs(j, M, visited);
            }
        }
    }
};
```

written by alexander original link here