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 cir
cle.
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 ar
e direct friends,
so the 0th and 2nd 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

```
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;

```
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

Solution 2

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.)

```
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 3

```
public int findCircleNum(int[][] M) {
    int count = 0;
    for (int i=0; i<M.length; i++)
        if (M[i][i] == 1) { count++; BFS(i, M); }
    return count;
}

public void BFS(int student, int[][] M) {
    Queue<Integer> queue = new LinkedList<>();
    queue.add(student);
    while (queue.size() > 0) {
        int queueSize = queue.size();
        for (int i=0;i<queueSize;i++) {
            int j = queue.poll();
            M[j][j] = 2; // marks as visited
            for (int k=0;k<M[0].length;k++)
                if (M[j][k] == 1 && M[k][k] == 1) queue.add(k);
        }
    }
}
```

written by compton_scatter original link here