CS 180 Discussion

Outline

- Union-find related questions
  - Detect cycle in an undirected graph [geeksforgeeks]
  - Kruskal’s minimum spanning tree [geeksforgeeks]
  - Prim’s minimum spanning tree [geeksforgeeks]
  - Friends Circles [547]

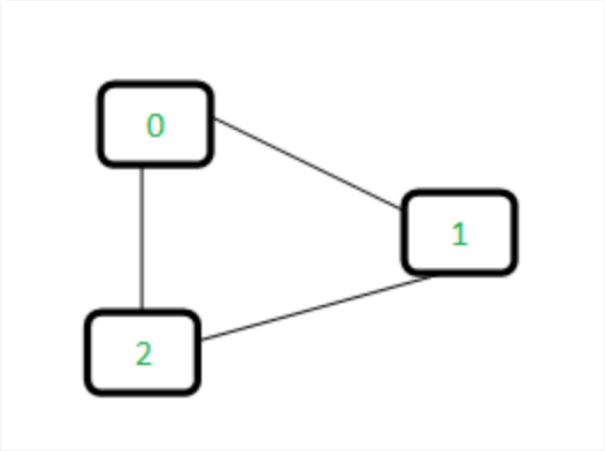
Detect cycle in an undirected graph

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

Union: Join two subsets into a single subset.

We can keep track of the subsets in a 1D array, let's call it parent[].

Let us consider the following graph:



Detect cycle in an undirected graph

Initially, all slots of parent array are initialized to -1 (means there is only one item in every subset).

```
0   1   2
-1 -1  -1
```

Now process all edges one by one.

Edge 0-1: Find the subsets in which vertices 0 and 1 are. Since they are in different subsets, we take the union of them. For taking the union, either make node 0 as parent of node 1 or vice-versa.

```
0   1   2    <----- 1 is made parent of 0 (1 is now representative of subset {0, 1})
1  -1  -1
```

Edge 1-2: 1 is in subset 1 and 2 is in subset 2. So, take union.

```
0   1   2    <----- 2 is made parent of 1 (2 is now representative of subset {0, 1, 
1   2  -1
```

Edge 0-2: 0 is in subset 2 and 2 is also in subset 2. Hence, including this edge forms a cycle.

How subset of 0 is same as 2?
0->1->2 // 1 is parent of 0 and 2 is parent of 1

MST by Kruskal's algorithm

What is Minimum Spanning Tree?

Given a connected and undirected graph, a *spanning tree* of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A *minimum spanning tree (MST)* or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

Below are the steps for finding MST using Kruskal's algorithm

1. *Sort all the edges in non-decreasing order of their weight.*

2. *Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.*

3. *Repeat step#2 until there are (V-1) edges in the spanning tree.*

MST by Prim's algorithm

Algorithm

1) Create a set *mstSet* that keeps track of vertices already included in MST.

2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.

3) While mstSet doesn't include all vertices

....**a)** Pick a vertex *u* which is not there in *mstSet* and has minimum key value.

....**b)** Include *u* to mstSet.

....**c)** Update key value of all adjacent vertices of *u*. To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u-v* is less than the previous key value of *v*, update the key value as weight of *u-v*

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 circle.
The 2nd student himself is in a friend circle. So return 2.
```

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 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 direct friends,
so the 0th and 2nd students are indirect friends. All of them are in the same
friend circle, so return 1.
```

Friend Circles

```
class Solution {
public:
    int findCircleNum(vector<vector<int>>& M) {
        int n = M.size(), res = n;
        vector<int> root(n);
        for (int i = 0; i < n; ++i) root[i] = i;
        for (int i = 0; i < n; ++i) {
            for (int j = i + 1; j < n; ++j) {
                if (M[i][j] == 1) {
                    int p1 = getRoot(root, i);
                    int p2 = getRoot(root, j);
                    if (p1 != p2) {
                        --res;
                        root[p2] = p1;
                    }
                }
            }
        }
        return res;
    }
    int getRoot(vector<int>& root, int i) {
        while (i != root[i]) {
            root[i] = root[root[i]];
            i = root[i];
        }
        return i;
    }
};
```