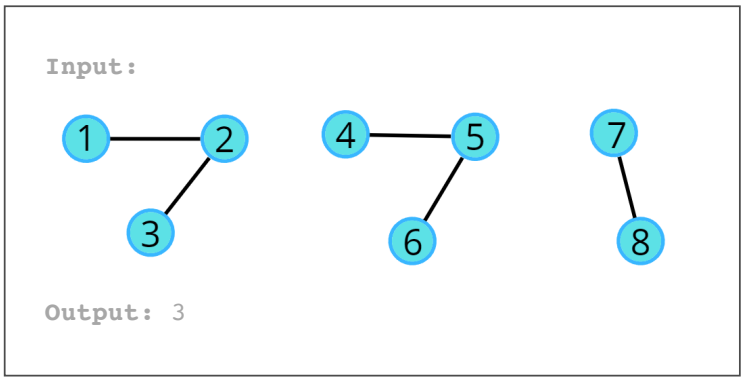
August 10, 2022 ▪ Data Structure / Graph

Number of Provinces

Problem Statement: Given an undirected graph with V vertices. We say two vertices u and v belong to a single province if there is a path from u to v or v to u. Your task is to find the number of provinces.

Pre-req: Connected Components, Graph traversal techniques

Examples:



Solution

***Disclaimer**. Don't jump directly to the solution, try it out yourself first.*

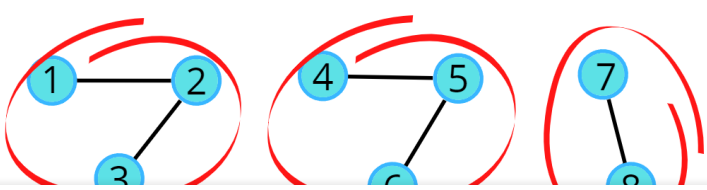
Approach:

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

is not a province. We know about both the traversals, Breadth First Search (BFS) and Depth First Search (DFS). We can use any of the traversals to solve this problem because a traversal algorithm visits all the nodes in a graph. In any traversal technique, we have one starting node and it traverses all the nodes in the graph. Suppose there is an 'N' number of provinces so we need to call the traversal algorithm 'N' times, i.e., there will be 'N' starting nodes. So, we just need to figure out the number of starting nodes.

The algorithm steps are as follows:

- We need a visited array initialized to 0, representing the nodes that are not visited.
- Run the for loop looping from 0 to N, and call the DFS for the first unvisited node.
- DFS function call will make sure that it starts the DFS call from that unvisited node, and visits all the nodes that are in that province, and at the same time, it will also mark them as visited.
- Since the nodes traveled in a traversal will be marked as visited, they will no further be called for any further DFS traversal.
- Keep repeating these steps, for every node that you find unvisited, and visit the entire province.
- Add a counter variable to count the number of times the DFS function is called, as in this way we can count the total number of starting nodes, which will give us the number of provinces.



Recent Posts

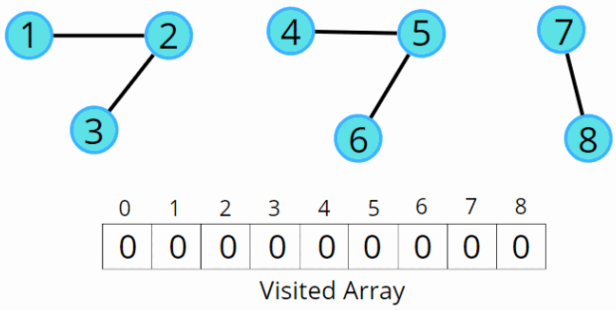
Print Nodes at Distance K in a Binary Tree

LCA in Binary Search Tree

Check if a tree is a Binary Search Tree or Binary Tree

Delete a Node in Binary Search Tree

Insert a Given Node in Binary Search Tree



0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	0

Visited Array

```
for ( i = 1; i <= V; i++ )
{
    if(visited[i] == 0)
    {
        dfs(i); // bfs(i)
    }
}
```

Code:

C++ Code

```
#include <bits/stdc++.h>
using namespace std;

class Solution {
  private:
    // dfs traversal function
    void dfs(int node, vector<int> adjLs[], i
        // mark the more as visited
        vis[node] = 1;
        for(auto it: adjLs[node]) {
            if(!vis[it]) {
                dfs(it, adjLs, vis);
            }
        }
    }
  public:
    int numProvinces(vector<vector<int>> adj,
        vector<int> adjLs[V];

        // to change adjacency matrix to list
        for(int i = 0;i<V;i++) {
            for(int j = 0;j<V;j++) {
                // self nodes are not conside
                if(adj[i][j] == 1 && i != j)
                    adjLs[i].push_back(j);
                    adjLs[j].push_back(i);
                }
```

```
            // if the node is not visited
            if(!vis[i]) {
                // counter to count the numbe
                cnt++;
               dfs(i, adjLs, vis);
            }
        }
        return cnt;

    }
};

int main() {

    vector<vector<int>> adj
    {
        {1, 0, 1},
        {0, 1, 0},
        {1, 0, 1}
    };


    Solution ob;
    cout << ob.numProvinces(adj,3) << endl;

    return 0;
}
```

Output: 2

Time Complexity: O(N) + O(V+2E), Where O(N) is for outer loop and inner loop runs in total a single DFS over entire graph, and we know DFS takes a time of O(V+2E).

Space Complexity: O(N) + O(N),Space for recursion stack space and visited array.

Java Code

```
import java.util.*;

class Solution {
    // dfs traversal function
    private static void dfs(int node
```

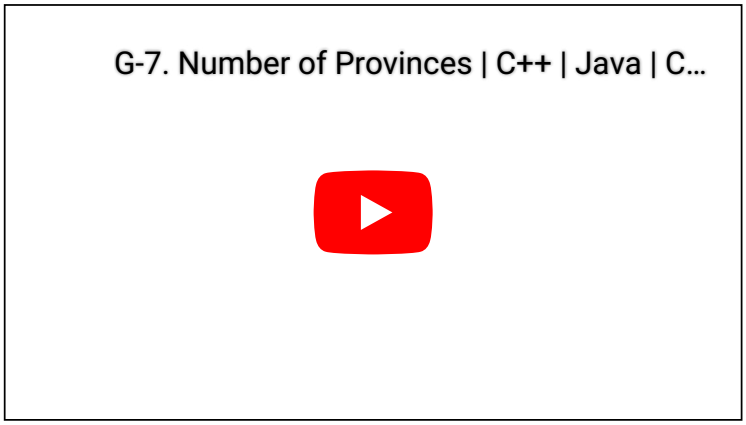
```
                dfs(it, adjLs, vis);
            }
        }
    }
    static int numProvinces(ArrayList<ArrayLi
        ArrayList<ArrayList<Integer>> adjLs =
        for(int i = 0;i<V;i++) {
            adjLs.add(new ArrayList<Integer>(
        }

        // to change adjacency matrix to list
        for(int i = 0;i<V;i++) {
            for(int j = 0;j<V;j++) {
                // self nodes are not conside
                if(adj.get(i).get(j) == 1 &&
                    adjLs.get(i).add(j);
                    adjLs.get(j).add(i);
                }
            }
        }
        int vis[] = new int[V];
        int cnt = 0;
        for(int i = 0;i<V;i++) {
            if(vis[i] == 0) {
               cnt++;
               dfs(i, adjLs, vis);
            }
        }
        return cnt;
    }
    public static void main(String[] args)
    {

        // adjacency matrix
        ArrayList<ArrayList<Integer> > adj =

        adj.add(new ArrayList<Integer>());
        adj.get(0).add(0, 1);
        adj.get(0).add(1, 0);
        adj.get(0).add(2, 1);
        adj.add(new ArrayList<Integer>());
        adj.get(1).add(0, 0);
        adj.get(1).add(1, 1);
        adj.get(1).add(2, 0);
        adj.add(new ArrayList<Integer>());
        adj.get(2).add(0, 1);
        adj.get(2).add(1, 0);
        adj.get(2).add(2, 1);
```

Output: 2

Time Complexity: O(N) + O(V+2E), Where O(N) is for outer loop and inner loop runs in total a single DFS over entire graph, and we know DFS takes a time of O(V+2E).

Space Complexity: O(N) + O(N),Space for recursion stack space and visited array.

> Special thanks to **Vanshika Singh Gour** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article. If you want to suggest any improvement/correction in this article please mail us at write4tuf@gmail.com



« Previous Post
Number of Islands

Next Post »
Flood Fill Algorithm – Graphs

Load Comments

Follow Us

DSA Playlist

Array Series

Tree Series

Graph Series

DP Series

DSA Sheets

Striver's SDE Sheet

Striver's A2Z DSA Sheet

SDE Core Sheet

Striver's CP Sheet

Contribute

Write an Article