CSCI B505 - Fall 2018

Written Assignment 5 - Graph Algorithms

1. Our goal is to find no. of trees in a forest.

We can find out with the help of below algorithm.

① Start from one node and initialize one counter variable to count no. of tree as c.

② We can implement DFS (Depth First Search) and traverse through all the nodes connected each other until we encounter an leaf node. A leaf node is a node with no children, & we backtrack as little as possible and find another undiscovered nodes.

③ We can use color notations for the above process of discovering nodes.

w = white = undiscovered node

g = grey = discovered but not complete

b = black = Completed

④ We will look for white nodes in every grey node discovered. Once every node turned black means we have traversed through the tree, we will increase the counter variable by 1.

⑤ We again repeat the above process and start from a different undiscovered white node.

⑥ When all the nodes are discovered.

and if Complete, we will get the no. of trees as well.

Time complexity:-

Since we are using DFS, the time complexity will be O(V+E). ~~Some constant time~~ Where V = no. of vertices

E = no. of edges.

2. We are given a tree with directed tree edges. We need to sort it topologically without using DFS.

We can solve this with the following algorithm.

Topological sort means we need to arrange all the vertices such that there are no back edges.

① First we need to find out incoming edges or indegree to each node.

② Then append all the nodes and their indegree in some array.

③ Remove the node with indegree as '0' and append it to an array called 'B', simultaneously reduce the indegree

④ Of the nodes by '1', if their edges came from the recently removed node.

⑤ Repeat step 3 till we get all the nodes appended to array B.

⑥ Return array 'B'.

→ The time complexity of this sorting will be $O(V+E)$ which is same as a DFS. Here V = vertices, E = edges.

3. We are provided a graph G. We have to write an algorithm that correctly determines if the graph is a DAG or not.

1st approach:

① First we need to find out the parent and child nodes.

② We can assign different colors to parents and it's children.

③ We then append the colors along with the nodes into a dictionary.

④ If a node contains 2 different colors then we can say that it is not a DAG that means there exist a cycle.

2nd approach :-

① We can check each node and save the nodes in an array called visited.

② If there exist a node which we encounter in the process who has all the children that are in visited array already, then we say the graph is not a DAG.

→ Time complexity of this algorithm is $O(V+E)$ where V = vertices and E = edges in the worst case.

→ The 1st approach will fail if there is even no. of nodes in a cycle.

→ The 2nd approach will fail if one node has 2 incoming nodes from 2 different parent nodes.

3rd Approach:-

→ We can find the cycles by doing a topological sort on the graph (which does n't have back edges).

→ We will find indegree as the incoming edges to a node.

→ Then we will append in an array called B, the node who has 0 indegree. Then decrease the neighbouring nodes indegree by 1 and remove the node.

→ Repeat the above steps until we get all the nodes.

→ This will give us the DAG (as we can't topologically sort a cycle)

Running time = O(V+E)