Depth First Search (DFS) Algorithm

# Introduction

DFS algorithm provides a way to traverse a graph systematically. It also help to give some information about the graph structure. To be able to travel as deep as possible, the implemented algorithm can

* Allow the ability to travel along the neighboring nodes before backtracking
* Prevent visiting a node twice except when backtracking
* Maintain the state of the visited node

The ability to backtrack along the visited nodes is provided by the following

* Ability to track the visited nodes
* Ability to track the state of the nodes

The algorithm described here is a recursive algorithm which classifies each vertex and its corresponding edge as a tree. Each node in the spanning tree can have 1 on the following state

* Unvisited - White
* In-Process – Grey (has unvisited connected node)
* Processed – Black (All connected nodes visited)

# Code

## Pseudocode

Described below is the pseudocode for the proposed algorithm

* Initially set all vertices as unvisited and mark as white
* Start from vertex **A** and mark vertex **A** as visited
* For each connected edge (**B, A**), if **B** is unvisited mark **B** as visited and set **A** as the parent of **B** Take  **B** as the new vertex and run depth-first search for **B** recursively for all connected nodes.
* Mark node **B** as black and backtrack to the **A**

## Implementation

This implementation is written in C#.

Considering a basic Quick-Union tree where each node maintains a reference to its parent as shown below

1 2 3 4 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 2 | 3 | 1 |

The following code below encapsulates the recursive algorithm

public class DepthFirstSearch

{

enum NodeState

{

White=1, Gray=2, Black=3

}

int[] nodeParent = {1, 1, 2, 3, 1};

int nodeCount = 5;

public void StartDepthFirstSearch()

{

int[] state = new int[nodeCount];

**for (int i = 0; i < nodeCount; i++)**

Running Time = **O(V)**

**state[i] = (int)NodeState.White;**

runDFS(0, state);

}

public void runDFS(int u, int[] state)

{

state[u] = (int)NodeState.Gray;

**for (int v = 0; v < nodeCount; v++)**

**if (IsConnected(u, v) && state[v] == (int)NodeState.White)**

Running Time = **O(E)**

**runDFS(v, state);**

state[u] = (int)NodeState.Black;

}

private bool IsConnected(int u, int v)

{

return nodeParent[u] == nodeParent[v];

}

}

## DFS Running Time

As shown above the loop in the StartDepthFirstSearch() method have a runtime of O(V) irrespective of the time it takes to excute the runDFS( ) call since each node in the arrayis accessed only once.

Also the method runDFS(u, int[] states) is called only once for each white(unvisited) vertex v. This is guaranteed by marking the vertex state as visited (Gray) once the runDFS is called (Cormen T, 2009). The runtime is therefore O(E) since each vertex is visited only once.

The overall runtime is therefore O(V+E);

# References

Techie Me, 'Depth First Traversal - Techie Me' (2015) <http://techieme.in/depth-first-traversal/> accessed 8 November 2015

Cormen T, Introduction To Algorithms (MIT Press 2009)