**Graph Theory**

Tree - Connected Acyclic Graph

Tree Traversals -

1. Preorder : Preorder: parent, left subtree, right subtree
2. PostOrder : Postorder: left subtree, right subtree, parent
3. Inorder : left subtree, parent, right subtree

Given all three orders, find out do they belong to same tree?

We can use below concepts.

# Construct Tree from given Inorder and Pre/Post order traversals.

Will always be Unique. Can be proved.

<https://www.geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal/>

<https://www.geeksforgeeks.org/construct-tree-from-given-inorder-and-preorder-traversal/>

For the traversal to uniquely identify a tree, we need In Order + (any other) always.

Complexity - O(n2)

Lowest Common Ancestor (LCA)

QED

Sparse Graphs - |E| << |V|2 -> Use Adjacency List

Dense Graphs - |E| ~ |V|2 -> Use Adjacency matrix

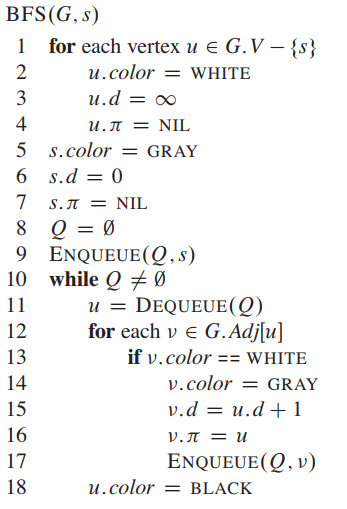
**BFS** -

Implemented using Queues.

Complexity - O(V + E) | V for no. of queue operations and E for searching adjacency list.

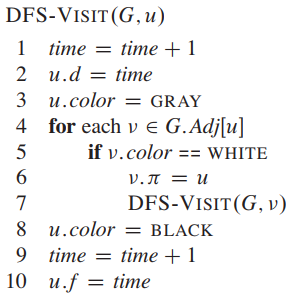
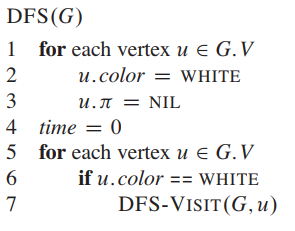
Properties -

Can be used to find **shortest** distance in an **undirected** graph.



**DFS -**

Pseudo Code -

****

Properties -

* Discovery and finishing times have parenthesis structure.
* An acyclic graph will never have back edges.
* If we sort the vertices according to finish time (v.f), we get **topological sorting** of the vertices.

**Strongly Connected Components** (Directed Graphs) (An application of DFS)

For each pair of vertices in that component, we have a path from u to v and from v to u. (u and v are reachable from each other)

Kosaraju Algorithm and Proof -

<http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/strongComponent.htm>

Note-Why do we start second dfs in reverse topo-order.

**IMPORTANT\*\***

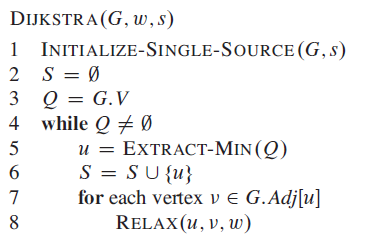
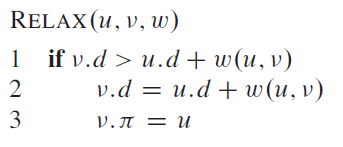
1. Always check if the graph is completely connected or disconnected.
2. BFS - Bipartite - 2Color
3. Every tree is 2-colorable
4. Longest path in a tree - 2DFS

**SHORTEST PATH ALGORITHMS**

**Dijkstra Algorithm**

* Single Source shortest path
* Directed Graph (can have cycles)
* Non Negative Edge Weights

\*\*\* Can we modify Dijkstra to find Shortest Path Tree with minimum total sum of weights? \*\*\* - YES



Initialize Single Source - For each v, set v.d = inf, v.parent = Null, s.d = 0

Greedy - because in each iteration, we choose the vertex having the least d value

Time - while loop - |V| times

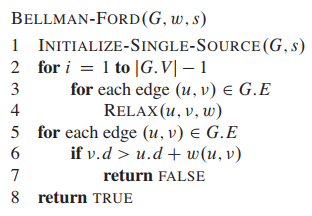
Extract-Min takes O(|V|), decrease key/insert - O(1)

Runtime - O(|V|2 + |E|)

The runtime is dependent on the implementation of the priority queue. And thus can be improved further using better PQ. (P Queue with binary min-heap or with Fibonacci heap).

**Bellman-Ford Algorithm**

* Single Source Shortest Path
* Edge Weights may be Negative
* Returns a Boolean Value indicating whether or not a negative-weights cycle exists.



Runtime - O( VE )

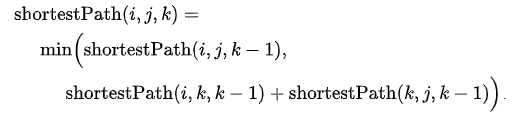
A shortest path from vertex s to v must be a simple path (never contains any cycles)

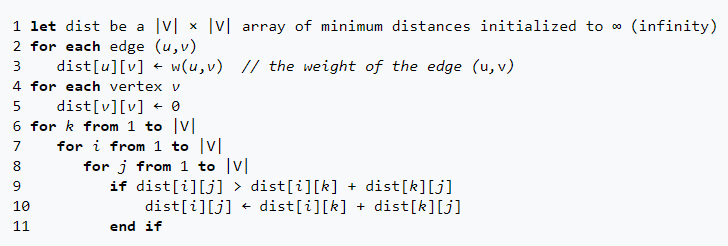
The number of edges in the shorted path from s to v must be at most V-1, where V denotes the total number of vertices

Thus we need at most V-1 iteration the discover the last vertex.

**Floyd Warshall Algorithm**

* All pair shortest path
* Negative weights may be present
* No Negative Cycles





Time Complexity - O(|V|3)