# 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 - $O(n^2)$

## Lowest Common Ancestor (LCA)

QED

Sparse Graphs - $|E| << |V|^2$ -> Use Adjacency List
Dense Graphs - $|E| \sim |V|^2$ -> 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.

```
BFS(G, s)
for each vertex u ∈ G.V − {s}
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅
    u = DEQUEUE(Q)
    for each v ∈ G.Adj[u]
        if v.color == WHITE
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```

**DFS -**

Pseudo Code -

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

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](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

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = ∅
Q = G.V
while Q ≠ ∅
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

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.

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

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

$$\begin{aligned}&\mathrm{shortestPath}(i,j,k) = \\ &\quad \min\Big(\mathrm{shortestPath}(i,j,k-1), \\ &\qquad \mathrm{shortestPath}(i,k,k-1)+\mathrm{shortestPath}(k,j,k-1)\Big).\end{aligned}$$

```
let dist be a |V| × |V| array of minimum distances initialized to ∞ (infinity)
for each edge (u,v)
   dist[u][v] ← w(u,v)  // the weight of the edge (u,v)
for each vertex v
   dist[v][v] ← 0
for k from 1 to |V|
   for i from 1 to |V|
      for j from 1 to |V|
         if dist[i][j] > dist[i][k] + dist[k][j]
             dist[i][j] ← dist[i][k] + dist[k][j]
         end if
```

Time Complexity - $O(|V|^3)$