Daffodil International University

Assignment II

Algorithms

Submitted To:

Subroto Nag Pinku

Dept. Of CSE

Submitted By:

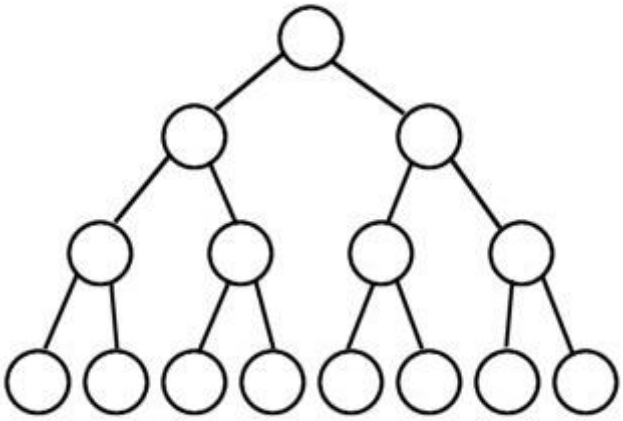
Raduan Ahamad

ID: 191-15-12943

Sec: O14

Dep. Of CSE

1.Full Tre Traversal



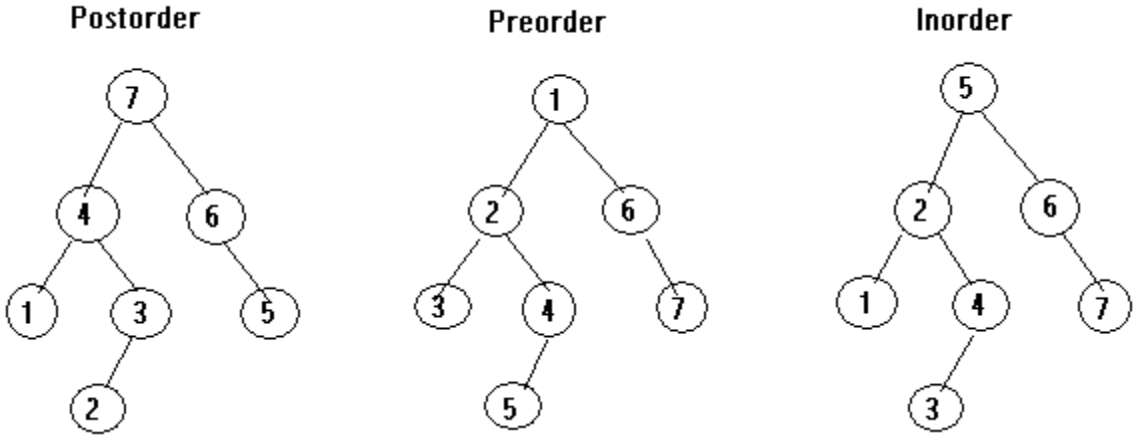
A Full Tree

Source: Google

In tree traversal is a process of visiting all the nodes and may show their values too. A full binary tree is a tree in which every node other than the leaves has two children.

There are three ways of traverse a full tree

i. In order
ii. Pre order
iii. Post Order



In order

1. Traverse the left subtree by recursively calling the in-order function.
2. Access the data part of the current node.
3. Traverse the right subtree by recursively calling the in-order function.

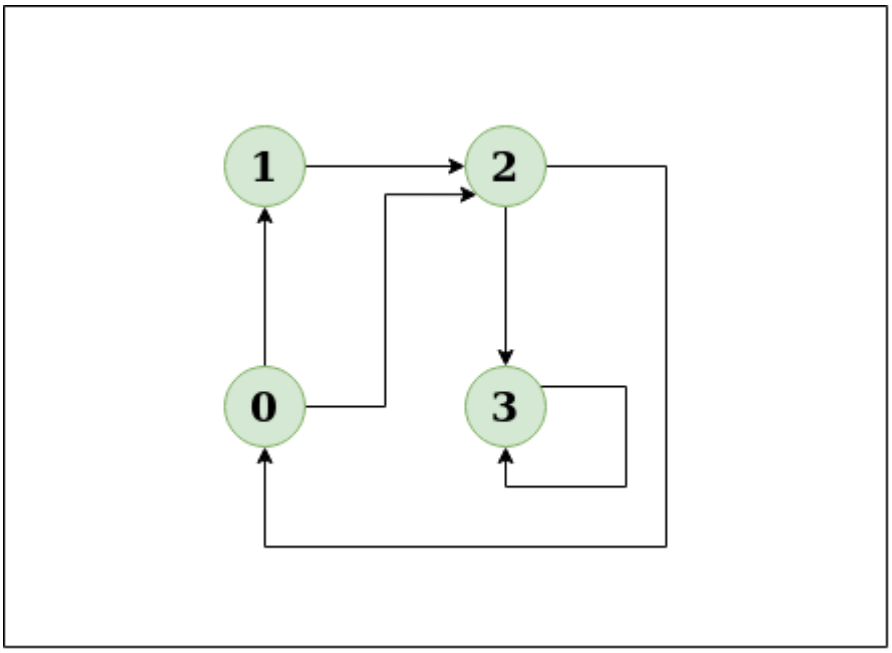
Pre order

1. Visit root node.
2. Recursively traverse left subtree.
3. Recursively traverse right subtree.

Post order

1. Recursively traverse left subtree.
2. Recursively traverse right subtree.
3. Visit root node.

2.Cycle Finding



A directed Graph with Cycle

Source: Google

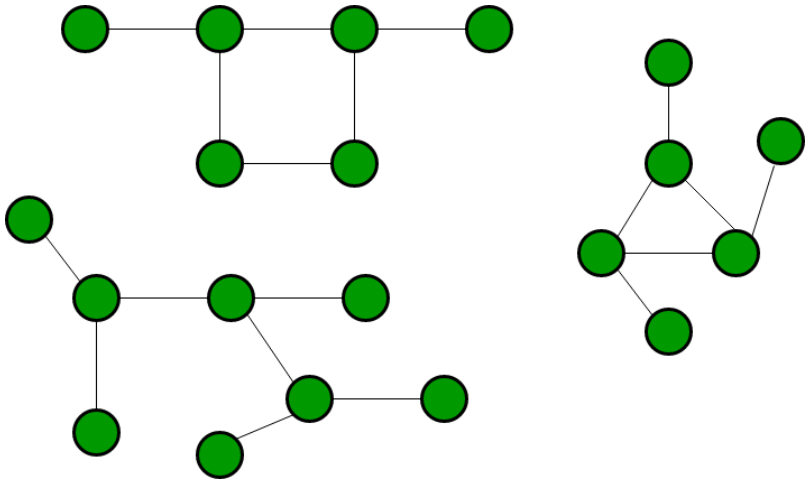
There are also directed and undirected graph. And in graph theory a cycle in a graph is a non-empty trail in which the only repeated vertices are the first and last vertices. A graph without cycle is called an acyclic graph.

To find cycle in a graph algorithm like DFS can be used.

Algorithm

1. Start by putting any one of the graph's vertices on top of a stack.
2. Take the top item of the stack and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add the ones which aren't in the visited list to the top of the stack.
4. Keep repeating steps 2 and 3 until the stack is empty.

3.Component Finding



Connected Components in an Undirected Graph

Source: GeeksforGeeks

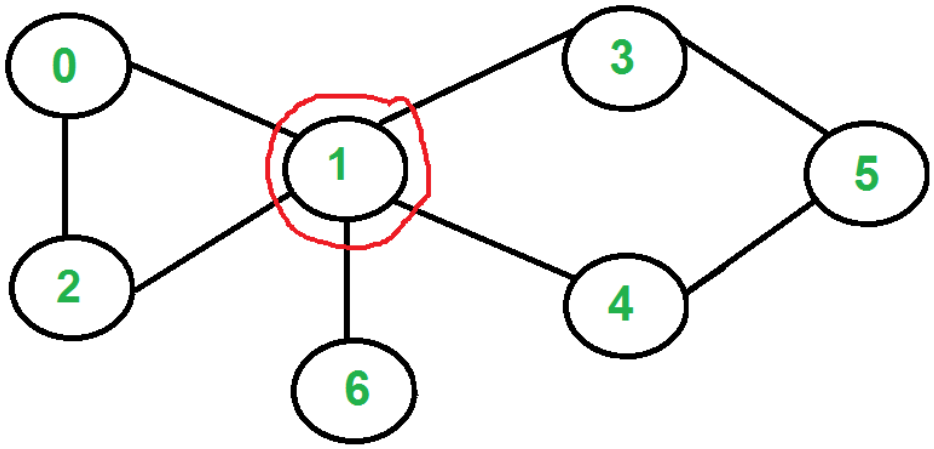
In graph theory, a component, sometimes called a connected component, of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the super graph.

Traversal algorithms like BFS or DFS can be used to find components is a graph.

DFS based Algorithm

1. Initialize all vertices as not visited.
2. Do following for every vertex ‘v’.
3. If v is not visited before call DFSUtil(v).
4. DFSUtil(v):
5. i. Mark v as visited
6. Print v.
7. Do following for every adjacent u of v. if u is not visited then recursively call DFSUtil(u).

4.Articulation Point Finding



Articulation Point is 1

Source: GeeksforGeeks

A vertex in an undirected connected graph is an articulation point (or cut vertex) if removing it (and edges through it) disconnects the graph.

Algorithms like **BFS** or **DFS** can be used.

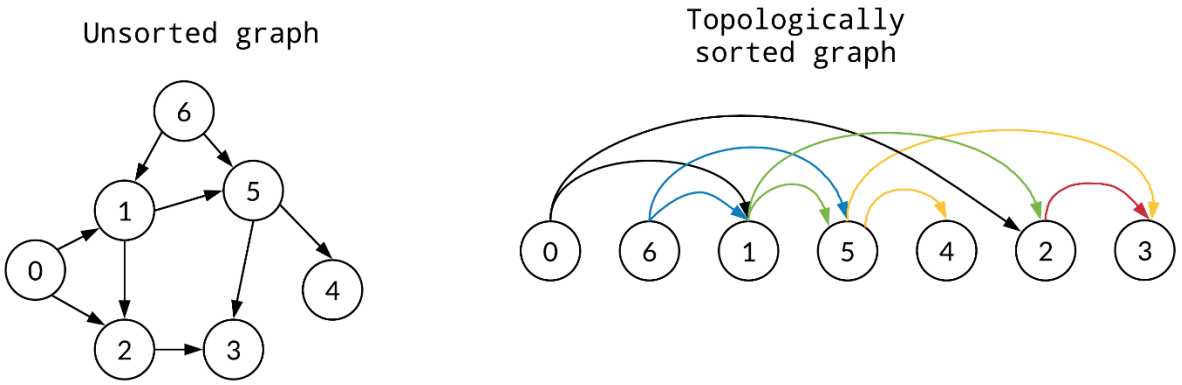
1.For every vertex v do following

a. Remove v from graph.

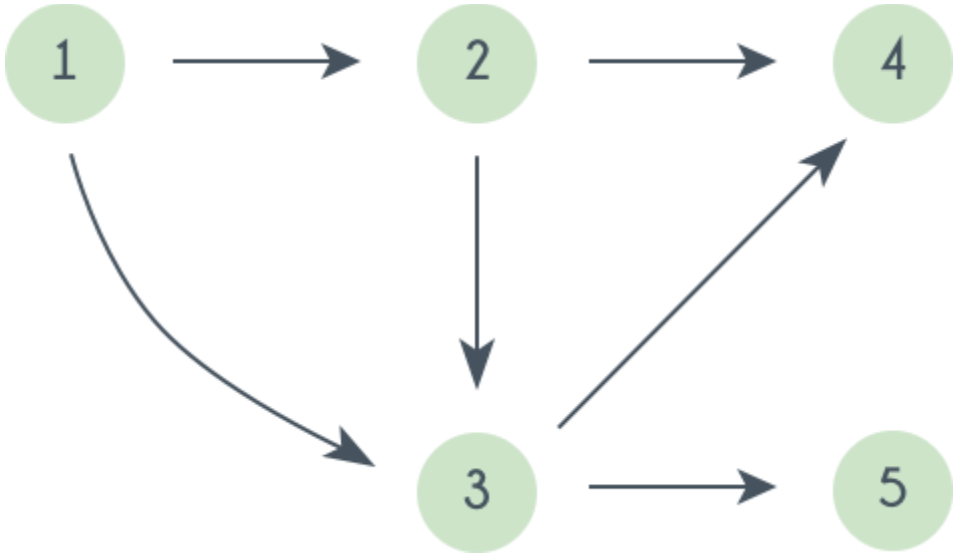
b. see if the graph remains connected (BFS or DFS can be used.)

c. Add v back to the graph.

5.Topological Sort



A topological sort or topological ordering of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.



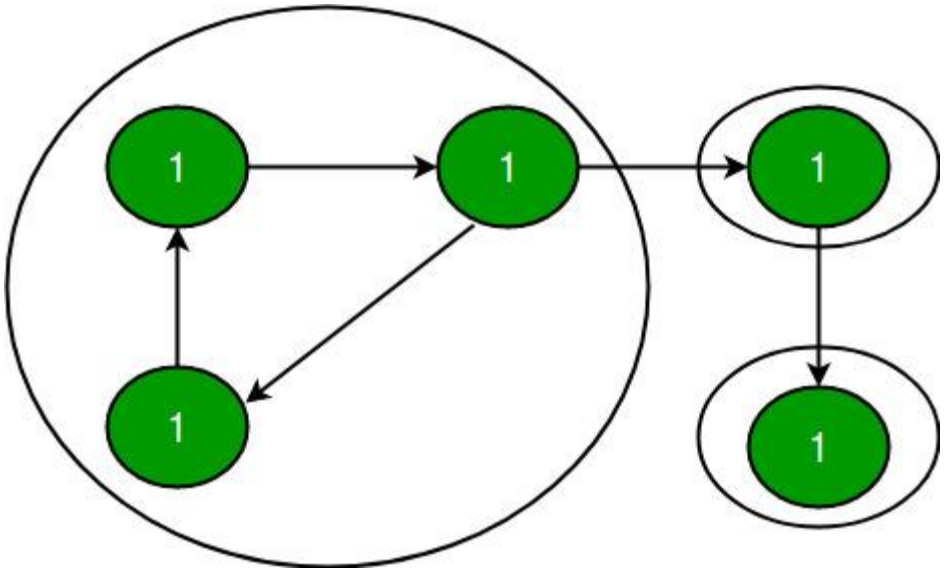
Topological Sort of This Graph is 1 2 3 4 5

Topological Sort can be also done using DFS

Algorithm

1. Step 1: Create the graph by calling addEdge(a,b).
2. Step 2: Call the topologicalSort( )
    a. Step 2.1: Create a stack and a boolean array named as visited[ ];
    b. Step 2.2: Mark all the vertices as not visited i.e. initialize visited[ ] with 'false' value.
    c. Step 2.3: Call the recursive helper function topologicalSortUtil() to store Topological Sort starting from all vertices one by one.
3. Step 3: def topologicalSortUtil(int v, bool visited[],stack<int> &Stack):
    a. Step 3.1: Mark the current node as visited.
    b. Step 3.2: Recur for all the vertices adjacent to this vertex.
    c. Step 3.3: Push current vertex to stack which stores result.
4. Step 4: Atlast after return from the utility function, print contents of stack.

6.Storngly Connected Components



A directed graph, a graph is said to be strongly connected if every vertex is reachable from every other vertex.

All Strongly connected components can be found using **Kosaraju's Algorithm.**

1.Do a DFS on the original graph.

2. Reverse the original graph.

3. Do DFS again.