CSCI 115 Lab

Week 14- DFS and BFS

- Professor: Dr. Matin Pirouz
  Email: mpirouz@csufresno.edu

- TA: Shreeja Miyyar
  Email:shreejarao12@mail.fresnostate.edu

Table of Contents

Depth First Search

- DFS is an algorithm to traverse the tree or graph data structure.
- The algorithm starts with an arbitrary node and explore as deep as possible before backtracking to that node.
- The approach is as follows:
  a) Select an unvisited node and make it as a current node and add it to the visitors list.
  b) For its unvisited neighbor, make the neighbor as the current node and add it to the visitors list.
  c) If the current node has no unvisited neighbors, backtrack to the parent node and make that as the current node.
  d) Repeat (b) and (c) until no more nodes can be visited.

DFS Algorithm

```
DFS_Util(visited, Adj, u) {
  visited[u] = true
  //Print the traversed node here
  for each v ∈ Adj[u] {
    if visited[v] == false
       DFS_Util(visited, Adj, v)
  }
}
DFS (Adj, N) {
//Create G : the list of nodes, example: If number of nodes is 4,
then G is [1,2,3,4]
//Define Visited Vector of length equal to number of nodes
  For each values in Visited Vector{
    visited[u] = false
  }

  For each u ∈ G {
    if visited[u]==false
       DFS_Util(visited, Adj, u)
   }
}
```

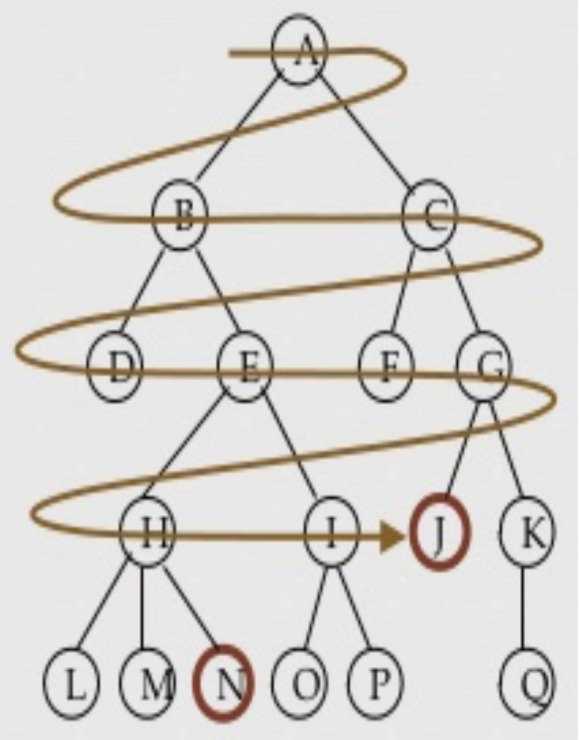
```
Main()
{
//Take user input number of nodes
// Create an adjacency list (can be a vector or linked
list).
DFS(Adj, N))
}
```

Time Complexity:
O(V+E) where V is a number of vertices in the graph and E is a number of edges in the graph.

Breadth First Search

- BFS is an algorithm to traverse the tree or graph data structure.
- It uses Queue data structure for implementation.
- The algorithm starts with an arbitrary node and travels level by level until all the nodes are visited.

- The approach is as follows:
  a) Select an unvisited node and Enqueue it and add it to the visitors list.
  b) Iterate the queue until it is empty, pop the node from the queue and add its neighbors to the Queue and visitors list.
  c) Repeat (b) until Queue is empty.

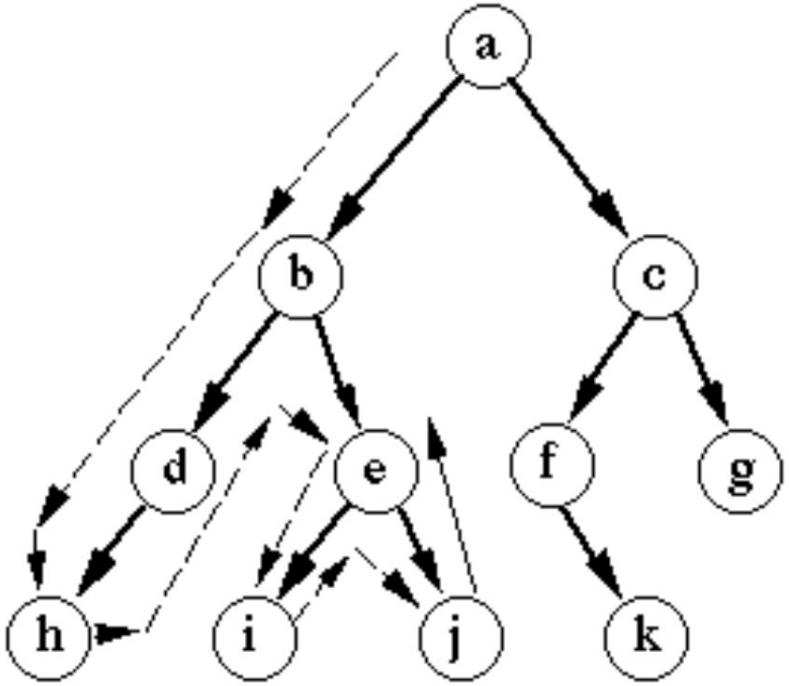


BFS Algorithm

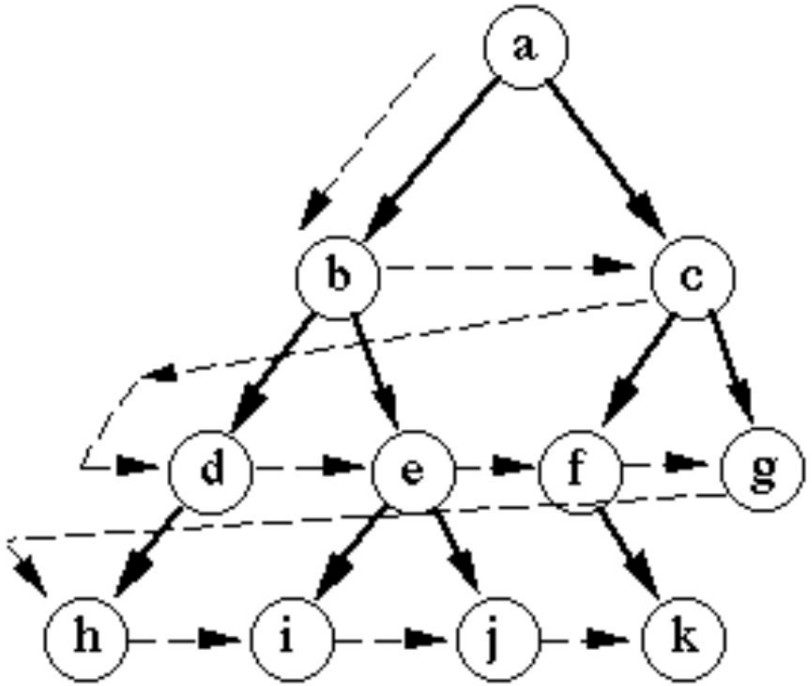
```
BFS(N, Adj, u) {
  create a queue Q
  visited[u] = true
  add u to the queue Q
  while Q is non-empty {
       remove the head u of Q
      //Print u required for the traversal.
       visited[u] = true
       for each v ∈ Adj[u] {
            if visited[v] == false {
                visited[v]=true
                 Enqueue the vertex v
            }
       }
  }
}

main () {
  BFS(N, Adj, startNode)
}
```

Time Complexity:
O(V+E) where V is a number of vertices in the graph and E is a number of edges in the graph.

BFS vs DFS traversal



Depth-first search



Breadth-first search

Lab Assignment

Hints and Coding Guidelines:

Note: Graph is an undirected graph.

DFS:

- Create a main function which accepts number of nodes and adjacency list.
- Adjacency list can be a vector, linked list or matrix.
- To create adjacency list, use a function addEdge that adds edges in both direction since we are considering undirected graph.
- Create a function DFS which accepts these two inputs as arguments. Create a Boolean array/vector to track the visited nodes.
- You can implement the DFS algorithm using Recursive method or by using a Stack to store the current nodes.

BFS:

- Create a main function which accepts number of nodes, adjacency list and start node as inputs.
- Adjacency list can be a vector or linked list or matrix.
- Create a function BFS which accepts these three inputs as arguments. Create a Boolean array/vector to track the visited nodes.
- Create a Queue (you are allowed to use built-in Queue. Do not implement Queue from scratch) which stores the current nodes and its unvisited neighbors.

Questions?