Lab Session –BFS and DFS

## Learning Outcomes

Implementation and thorough reading of following will make you capable of:

* Define and Implement the concept of Breadth First Search and Depth First Search algorithms
* Identifying applications of these searching algorithms.

## Breadth-First Search

**Breadth-first search (BFS)** is an algorithm used for tree traversal on graphs or tree data structures. BFS can be easily implemented using recursion and data structures like dictionaries and lists. It is a recursive algorithm to search all the vertices of a graph or a tree.

## Algorithm

The steps of the algorithm are as follow:

1. Start by putting any one of the graph’s vertices at the back of the queue.
2. Now take the front item of the queue and add it to the visited list.
3. Create a list of that vertex's adjacent nodes. Add those which are not within the visited list to the rear of the queue.
4. Keep continuing steps two and three till the queue is empty.

visited = [] # List for visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node): #function for BFS

visited.append(node)

queue.append(node)

while queue: # Creating loop to visit each node

m = queue.pop(0)

print (m, end = " ")

for neighbour in graph[m]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

Here, visited is a list that is used to keep track of visited nodes. And queue is a list that is used to keep track of nodes currently in the queue.

1. It checks and appends the starting node to the visited list and the queue.
2. Then, while the queue contains elements, it keeps taking out nodes from the queue, appends the neighbors of that node to the queue if they are unvisited, and marks them as visited.
3. This continues until the queue is empty.

## Time Complexity

The time complexity of the Breadth first Search algorithm is in the form of O(V+E), where V is the representation of the number of nodes and E is the number of edges. Also, the space complexity of the BFS algorithm is O(V).

## Depth-First Search

**Depth-first search** (DFS) is an algorithm for tree traversal on graph or tree data structures. It can be implemented easily using recursion and data structures like dictionaries and sets.

A standard DFS implementation puts each vertex of the graph into one of two categories:

1. Visited
2. Not Visited

The purpose of the algorithm is to mark each vertex as visited while avoiding cycles.

## Algorithm

1. Pick any node. If it is unvisited, mark it as visited and recur on all its adjacent nodes.
2. Repeat until all the nodes are visited, or the node to be searched is found.

def dfs(visited, graph, node):

    if node not in visited:

        print (node)

        visited.add(node)

        for neighbour in graph[node]:

            dfs(visited, graph, neighbour)

The DFS function is called and is passed the visited set, the graph in the form of a dictionary, and A, which is the starting node.

DFS follows the algorithm described above:

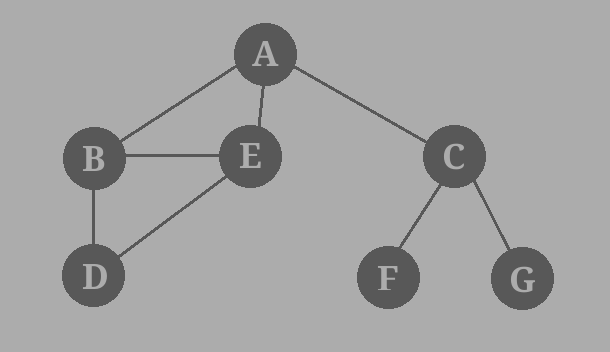
* It first checks if the current node is unvisited. If yes, it is appended in the visited set.
* Then for each neighbor of the current node, the DFS function is invoked again.
* The base case is invoked when all the nodes are visited. The function then returns.

## Time Complexity

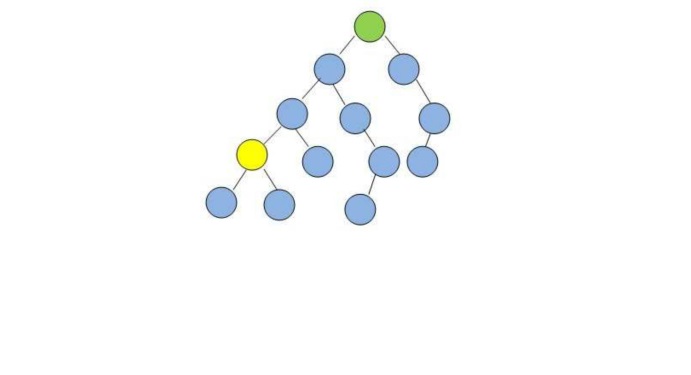
The time complexity of DFS if the entire tree is traversed is O(V) where V is the number of nodes.

## Exercise

1. Consider the following graph. If there is ever a decision between multiple neighbor nodes in the BFS or DFS algorithms, assume we always choose the letter closest to the beginning of the alphabet first. A [connected graph](http://mathworld.wolfram.com/ConnectedGraph.html) with 7 nodes and 7 edges. The edges are [undirected](http://mathinsight.org/definition/undirected_graph) and unweight.  Distance between two nodes will be measured based on the number of edges separating two vertices.

Represent a graph with [adjacency list](https://en.wikipedia.org/wiki/Adjacency_list) using dictionaries. The keys of the dictionary represent nodes; the values have a list of neighbors.

Define function name ‘connected\_component’, this function keep track of all the visited nodes with BFS, is as simple as implementing the steps of the algorithm and assign ‘queue’ variable  already has a node to be checked, i.e., the starting vertex that is used as an entry point to explore the graph. The next step is to implement a loop that keeps cycling until queue is empty. At each iteration of the loop, a node is checked.  If this wasn’t visited already, its neighbours are added to queue. Once the loop is exited, the function (connected\_component) returns all of the visited nodes.

1. Considering Q1, traverse the graph using DFS and find the shortest path between two nodes.  The function connected\_component’ should be able to accept a graph as argument, a starting node (e.g., ‘G’) and a node goal (e.g., ‘D’). If the algorithm is able to connect the start and the goal nodes, it has to return the path.
2. In the following graph, assume that if there is ever a choice amongst multiple nodes, both the BFS and DFS algorithms will choose the left-most node first. Starting from the green node at the top, which algorithm will visit the least number of nodes before visiting the yellow goal node? Answer via code.