Title: DFS & BFS

**Ex. No.:** 4 **Reg. No.:** RA2011051010006

**Date:** 09/02/23 **Name:** Vittesh Raghavan

# Aim:

To implement and analyze DFS (Depth First Search) and BFS(Breadth First Search) for same application. In this case the application will be a graph.

**Procedure/Algorithm:**

**For BFS :**

* Declare a queue and insert the starting vertex.
* Initialize a visited array and mark the starting vertex as visited.
* Follow the below process till the queue becomes empty:
* Remove the first vertex of the queue.
* Mark that vertex as visited.
* Insert all the unvisited neighbors of the vertex into the queue.

**For DFS :**

* Create a recursive function that takes the index of the node and a visited array.
* Mark the current node as visited and print the node.
* Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

**Program:**

# FOR BFS : FOR DFS :

# 

# Manual Output: Manual calculation for the example you have taken:

# Example-of-dfs-algorithm.

# Step 1: Mark vertex A as a visited source node by selecting it as a source node.

# You should push vertex A to the top of the stack.

# Step 2: Any nearby unvisited vertex of vertex A, say B, should be visited.

# You should push vertex B to the top of the stack.

# Step 3: From vertex C and D, visit any adjacent unvisited vertices of vertex B. Imagine you have chosen vertex C, and you want to make C a visited vertex.

# Vertex C is pushed to the top of the stack.

# Step 4: You can visit any nearby unvisited vertices of vertex C, you need to select vertex D and designate it as a visited vertex.

# Vertex D is pushed to the top of the stack.

# Step 5: Vertex E is the lone unvisited adjacent vertex of vertex D, thus marking it as visited.

# Vertex E should be pushed to the top of the stack.

# Step 6: Vertex E's nearby vertices, namely vertex C and D have been visited, pop vertex E from the stack.

# Step 7: Now that all of vertex D's nearby vertices, namely vertex B and C, have been visited, pop vertex D from the stack.

# Step 8: Similarly, vertex C's adjacent vertices have already been visited; therefore, pop it from the stack.

# Step 9: There is no more unvisited adjacent vertex of b, thus pop it from the stack.

# Step 10: All of the nearby vertices of Vertex A, B, and C, have already been visited, so pop vertex A from the stack as well.

# Now, examine the pseudocode for the depth-first search algorithm in this.

# FOR BFS :

# Step 1: In the graph, every vertex or node is known. First, initialize a queue.

# Step 2: In the graph, start from source node A and mark it as visited.

# Step 3: Then you can observe B and E, which are unvisited nearby nodes from A. You have two nodes in this example, but here choose B, mark it as visited, and enqueue it alphabetically.

# Step 4: Node E is the next unvisited neighboring node from A. You enqueue it after marking it as visited.

# Step 5: A now has no unvisited nodes in its immediate vicinity. As a result, you dequeue and locate A.

# Step 6: Node C is an unvisited neighboring node from B. You enqueue it after marking it as visited.

# Step 7: Node D is an unvisited neighboring node from C. You enqueue it after marking it as visited.

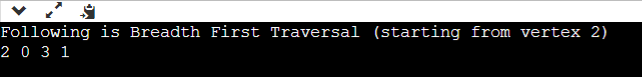
# Step 8: If all of D's adjacent nodes have already been visited, remove D from the queue.

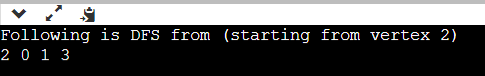
# Step 9: Similarly, all nodes near E, B, and C nodes have already been visited; therefore, you must remove them from the queue.

# Step 10: Because the queue is now empty, the bfs traversal has ended.

# In the next section of this tutorial, you will look at the breadth-first search algorithm's complexity.

# Screenshot of output: Actual Output you get after executing your program:

****

****

**Result:**

DFS and BFS implementation on a graph was successfully executed.