

**K. K. Wagh Institute of Engineering Education and Research, Nashik.**

**Department of Computer Engineering**

**Academic Year 2021-22**

**Course:** Laboratory Practice II                                                             **Course Code:** 410247

**Name of Student:**Prathamesh Dinkar Sujgure **Class:**TE

**Roll No.:** 71 **Div.:**A

**Name of Faculty:** Prof**.** Reshma Dhurjad

**Date of Completion**:

**Title:** Breadth First Search and Depth First Search Algorithms

**Objective:** To understand and implement BFS and DFS

**Problem Statement:** Implement depth first search algorithm and breadth first search algorithm. Use an undirected graph and develop s recursive algorithm for searching all the vertices of a graph or tree data structure.

**Outcomes:**

**software and Hardware requirements:**

**Theory-**

1. Concept in brief:

BFS: **Breadth-first search** (**BFS**) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for searching a [tree](https://en.wikipedia.org/wiki/Tree_(data_structure)) data structure for a node that satisfies a given property. It starts at the [tree root](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology) and explores all nodes at the present [depth](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology) prior to moving on to the nodes at the next depth level. Extra memory, usually a [queue](https://en.wikipedia.org/wiki/Queue_(data_structure)), is needed to keep track of the child nodes that were encountered but not yet explored.

DFS: **Depth-first search** (**DFS**) is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) for traversing or searching [tree](https://en.wikipedia.org/wiki/Tree_data_structure) or [graph](https://en.wikipedia.org/wiki/Graph_(data_structure)) data structures. The algorithm starts at the [root node](https://en.wikipedia.org/wiki/Tree_(data_structure)#Terminology) (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking.

1. Algorithm/Database design:

BFS:

**Step 1:** SET STATUS = 1 (ready state) for each node in G

**Step 2:** Enqueue the starting node A and set its STATUS = 2 (waiting state)

**Step 3:** Repeat Steps 4 and 5 until QUEUE is empty

**Step 4:** Dequeue a node N. Process it and set its STATUS = 3 (processed state).

**Step 5:** Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set

their STATUS = 2

(waiting state)

[END OF LOOP]

**Step 6:** EXIT

DFS:

* **Step 1:** SET STATUS = 1 (ready state) for each node in G
* **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
* **Step 3:** Repeat Steps 4 and 5 until STACK is empty
* **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
* **Step 5:** Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their  
  STATUS = 2 (waiting state)  
  [END OF LOOP]
* **Step 6:** EXIT

            c)test cases:

2

1

3

4

5

            d)conclusion/analysis.: BFS and DFS can be used for searching but DFS is not always optimal. BFS is more suitable for searching vertices which are closer to the given source. Whereas DFS is more suitable when there are solutions away from source.

**Screen shots:**

   Program codes with sample output