**Experiment-No.5**

**Objective: Write a program to implement breadth first search**

| **Scheduled Date:** | **Compiled Date:** | **Submitted Date:** |
| --- | --- | --- |
| 24 Sep 2024 | 24 Sep 2024 | 26- Sep 2024 |

### Description of DFS (Depth-First Search)

**Depth-First Search (DFS)** is a fundamental graph traversal algorithm that explores vertices in a depthward motion, meaning it visits a node and then explores as far as possible along each branch before backtracking. It operates on both trees and graphs.

#### 1. General Overview:

* **Type**: DFS is a graph traversal algorithm used on both trees and graphs.
* **Purpose**: DFS explores a graph's nodes by diving deep into each branch before backtracking.
* **Strategy**: DFS explores as deep as possible from a given node, processing one neighbor and moving deeper before exploring the next neighbor.
* **Stack-based (or recursive)**: DFS can be implemented either with an explicit stack (in an iterative approach) or using the system’s call stack (recursion).

#### 2. How DFS Works:

* **Start**: The algorithm starts from a source node and explores it.
* **Visit Neighbors**: For each unvisited neighbor of the current node, DFS dives deep, recursively exploring that neighbor and its neighbors.
* **Mark as Visited**: Once a node is visited, it is marked as "visited" to avoid revisiting.
* **Repeat**: The process continues, exploring all unvisited nodes recursively, until the entire graph (or connected component) has been traversed.

#### 3. Key Features:

* **Deep Exploration**: DFS explores nodes as far as possible along a branch before backtracking to visit other neighbors.
* **Path Exploration**: DFS is useful for finding paths in graphs, especially when trying to reach a particular node or explore entire connected components.
* **Graph Types**: DFS works for both directed and undirected graphs, and can be used in scenarios such as detecting cycles, connected components, and solving maze problems.

#### 4. Time and Space Complexity:

* **Time Complexity**: O(V + E), where V is the number of vertices and E is the number of edges.
* **Space Complexity**: O(V), due to the space required for the recursive call stack (or an explicit stack in the iterative approach) and the visited list.

### Algorithm of DFS (Depth-First Search)

#### Initialization:

* Create an empty list visited to store the nodes that have been visited.

#### Start DFS:

* Add the starting node to the visited list.

#### Process the Node:

* For each neighbor of the current node:
  + If the neighbor has not been visited:
    - Recursively call DFS on that neighbor.

#### End:

* When all nodes have been visited, the DFS traversal is complete.

**Algorithm of DFS (Depth-First Search)**

graph={

'A':['B','C'],

'B':['D','E'],

'C':['F'],

'D':[],

'E':['F'],

'F':[]

}

visited=[]

queue=[]

def bfs(visited,graph,node):

visited.append(node)

queue.append(node)

while queue:

m=queue.pop()

print(m,end=" ")

for n in graph[m]:

if n not in visited:

visited.append(n)

queue.append(n)

bfs(visited,graph,'A')