



Applied Artificial Intelligence

Practical # 4

Name	Ninad Karlekar	Roll Number	22306A1012
Subject/Course:	Applied Artificial Intelligence	Class	M.Sc. IT – Sem III
Topic	Implement DFS and BFS algorithm	Batch	1

A program to implement Rule Based System.

a) **AIM: Write an application to implement DFS algorithm.**

DESCRIPTION:

DFS explores a graph by starting at a source node, visiting all of its neighbors, and then recursively moving to an unvisited neighbor until no more unvisited nodes are reachable from the current path.

It uses a stack or recursion to track nodes and can be used for both tree and graph traversal, where it effectively explores the deepest unvisited nodes first.

DFS is suitable for topological sorting, cycle detection, and solving problems like finding connected components in a graph.

It doesn't guarantee the shortest path, making it less suitable for pathfinding problems, but it is efficient for exploring and analyzing graph structures.

Care should be taken to avoid infinite loops in cyclic graphs, and it's important to mark and handle visited nodes to prevent revisiting them.

Code:

```
graph = {
    '5' : ['3', '7'],
    '3' : ['2', '4'],
    '7' : ['8'],
```

```

'2' : [],
'4' : ['8'],
'8' : []
}

visited = [] # List for visited nodes.
queue = []    #Initialize a queue

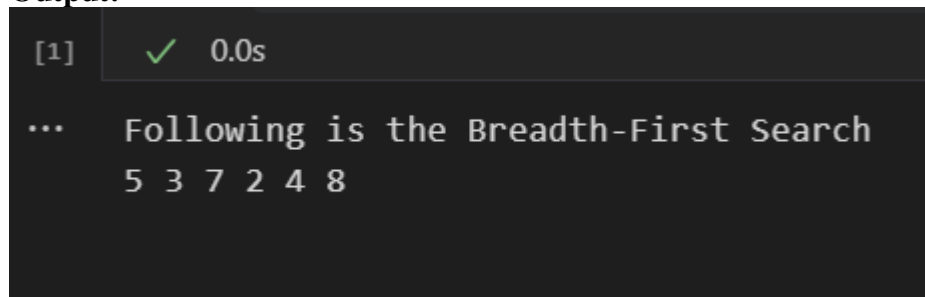
def bfs(visited, graph, node): #function for BFS
    visited.append(node)
    queue.append(node)

    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)

# Driver Code
print("Following is the Breadth-First Search")
bfs(visited, graph, '5')# function calling

```

Output:


```

[1] ✓ 0.0s
... Following is the Breadth-First Search
   5 3 7 2 4 8

```

b) AIM: Write an application to implement BFS algorithm.

DESCRIPTION:

BFS explores a graph by starting at a source node, visiting all its neighbors, and then moving to their unvisited neighbors in a systematic manner, level by level.

It uses a queue to maintain the order of node exploration, ensuring that nodes at the same level are visited before moving deeper into the graph.

BFS is suitable for finding the shortest path in unweighted graphs, as it guarantees the shortest path when used for traversal.

It's often used for solving problems like shortest path, connected components, and puzzle solving (e.g., maze-solving) where the focus is on finding the solution in the fewest steps.

BFS is less efficient for exploring deep or heavily branched graphs compared to Depth-First Search (DFS) but is a valuable tool for various graph-related tasks.

Code:

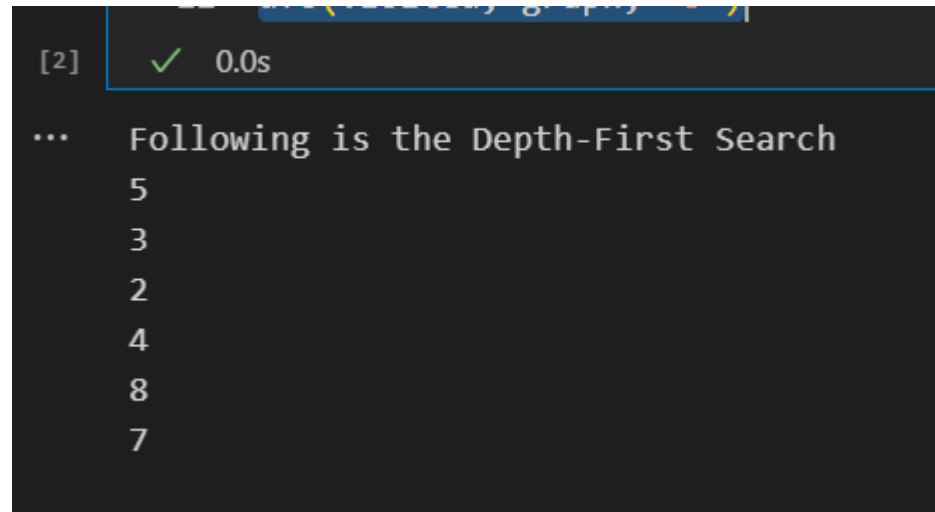
```
# Using a Python dictionary to act as an adjacency list
graph = {
    '5' : ['3','7'],
    '3' : ['2', '4'],
    '7' : ['8'],
    '2' : [],
    '4' : ['8'],
    '8' : []
}

visited = set() # Set to keep track of visited nodes of graph.

def dfs(visited, graph, node): #function for dfs
    if node not in visited:
        print (node)
        visited.add(node)
        for neighbour in graph[node]:
            dfs(visited, graph, neighbour)
```

```
# Driver Code
print("Following is the Depth-First Search")
dfs(visited, graph, '5')
```

Output:

A screenshot of a Jupyter Notebook output cell. The cell is labeled [2] and shows a green checkmark and 0.0s execution time. The output text is: "... Following is the Depth-First Search", followed by a vertical list of numbers: 5, 3, 2, 4, 8, and 7.

```
[2] ✓ 0.0s
... Following is the Depth-First Search
    5
    3
    2
    4
    8
    7
```