

EXPERIMENT NO - 4:

IMPLEMENTATION AND ANALYSIS OF DFS AND BFS FOR AN APPLICATION

Nikith Kumar Seemakurthi
RA1911003020480

AIM:

To implement and analyze the DFS and BFS for an application.

ALGORITHM:

DFS:

1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

BFS:

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

SOURCE CODE:

Source Code for DFS:

DFS algorithm in Python

DFS algorithm

```
def dfs(graph, start, visited=None):
```

```
    if visited is None:
```

```
        visited = set()
```

```
    visited.add(start)
```

```
    print(start)
```

```
    for next in graph[start] - visited:
        dfs(graph, next, visited)
    return visited
```

```
graph = {'0': set(['1', '2']),
        '1': set(['0', '3', '4']),
        '2': set(['0']),
        '3': set(['1']),
        '4': set(['2', '3'])}

dfs(graph, '0')
```

Source Code for BFS:

BFS algorithm in Python

```
import collections
```

BFS algorithm

```
def bfs(graph, root):
    visited, queue = set(), collections.deque([root])
    visited.add(root)

    while queue:
        # Dequeue a vertex from queue
        vertex = queue.popleft()
        print(str(vertex) + " ", end=" ")

        # If not visited, mark it as visited, and
        # enqueue it
        for neighbour in graph[vertex]:
            if neighbour not in visited:
                visited.add(neighbour)
                queue.append(neighbour)
```

```

if __name__ == '__main__':

    graph = {0: [1, 2],

             1: [2],

             2: [3],

             3: [1, 2]}

    print("Following is Breadth First Traversal: ")

    bfs(graph, 0)

```

OUTPUT:

The image displays two screenshots of a code editor interface, likely PyCharm, showing the implementation of Depth-First Search (DFS) and Breadth-First Search (BFS) algorithms in Python.

Top Screenshot (DFS): The code defines a `dfs` function that takes a graph, a start vertex, and a visited set. It recursively explores the graph. The output in the shell shows the traversal sequence: 0, 1, 3, 4, 2, 2.

```

main.py  Run Shell Clear
1 # DFS algorithm in Python
2 # DFS algorithm
3 def dfs(graph, start, visited=None):
4     if visited is None:
5         visited = set()
6         visited.add(start)
7
8     print(start)
9
10    for next in graph[start] - visited:
11        dfs(graph, next, visited)
12    return visited
13
14 graph = {'0': set(['1', '2']),
15         '1': set(['0', '3', '4']),
16         '2': set(['0']),
17         '3': set(['1']),
18         '4': set(['2', '3'])}
19
20 dfs(graph, '0')
21

```

Bottom Screenshot (BFS): The code defines a `bfs` function that uses a queue to explore the graph level by level. The output in the shell shows the traversal sequence: 0, 1, 2, 3, >.

```

main.py  Run Shell Clear
1 # BFS algorithm in Python
2 import collections
3 # BFS algorithm
4 def bfs(graph, root):
5     visited, queue = set(), collections.deque([root])
6     visited.add(root)
7
8     while queue:
9         # Dequeue a vertex from queue
10        vertex = queue.popleft()
11        print(str(vertex) + " ", end=" ")
12        # If not visited, mark it as visited, and
13        # enqueue it
14        for neighbour in graph[vertex]:
15            if neighbour not in visited:
16                visited.add(neighbour)
17                queue.append(neighbour)
18
19 if __name__ == '__main__':
20     graph = {0: [1, 2],
21             1: [2],
22             2: [3],
23             3: [1, 2]}

```

RESULT:

Hence DFS and BFS for an application is implemented and analyzed.