Experiment-1.3

Student Name: Vikash Kumar
UID: 21BCS9899
Branch: BE-CSE
Section/Group: IoT 640 – 'B'
Semester: 5
Date of Performance: /08/2022
Subject Name: AI&ML with Lab
Subject Code: 21CSH-316

1. Aim:

Implement the BFS algorithm and analyze its performance and characteristics

2. Objective:

The objective of this experiment is to implement the Breadth-First Search (BFS) algorithm and analyze its performance and characteristics.

3. Input/Apparatus Used:

The input for this experiment is a graph represented as an adjacency list or matrix. The graph can be either directed or undirected.

4. Theory:

The **Breadth First Search (BFS)** algorithm is used to search a graph data structure for a node that meets a set of criteria. It starts at the root of the graph and visits all nodes at the current depth level before moving on to the nodes at the next depth level.

Starting from the root, all the nodes at a particular level are visited first and then the nodes of the next level are traversed till all the nodes are visited.
To do this a queue is used. All the adjacent unvisited nodes of the current level are pushed into the queue and the nodes of the current level are marked visited and popped from the queue.

5. Algorithm:

1. Create a queue (usually implemented using a list or a deque) and enqueue the starting node.
2. Create a set or a list to keep track of visited nodes and add the starting node to it.
3. While the queue is not empty:
   a. Dequeue a node from the front of the queue.
   b. Process the node (print it, save it in a result list, etc.).
   c. Enqueue all unvisited neighbors of the dequeued node and mark them as visited.
4. Repeat step 3 until the queue becomes empty.

6. Code:

```
 from collections import deque

def bfs(graph, start):
    visited = set()         # Set to keep track of visited nodes
    queue = deque([start]) # Queue for BFS
    result = []             # List to store the traversal order

    while queue:
        node = queue.popleft()
        if node not in visited:
            result.append(node)  # Process the node (you can customize this part)
            visited.add(node)
            queue.extend(graph[node] - visited)  # Add unvisited neighbors to the queue

    return result

# Example graph represented as an adjacency list
graph = {
    1: {2, 3},
    2: {1, 4, 5},
    3: {1, 6},
    4: {2},
    5: {2},
    6: {3}
}

start_node = 1
bfs_result = bfs(graph, start_node)
print("BFS traversal starting from node", start_node, ":", bfs_result)
print("UID : 21BCS9899")
    print("Name : Vikash Kumar")
```

7. Result:

```
BFS traversal starting from node 1 : [1, 2, 3, 4, 5, 6]
UID : 21BCS9899
Name : Vikash Kumar


...Program finished with exit code 0
Press ENTER to exit console.
```

8. Output Explanation:

Starting from node 1, BFS explores nodes in layers, visiting neighbors before moving to their neighbors. The traversal sequence is: 1, 2, 3, 4, 5, and 6. BFS ensures that closer nodes are visited before distant ones, resulting in [1, 2, 3, 4, 5, 6].

a. Start at node 1.
b. Add node 1 to the result list and mark it as visited.
c. Enqueue nodes 2 and 3 (neighbors of 1).
d. Dequeue node 2, add it to the result list, mark it as visited.
e. Enqueue nodes 4 and 5 (neighbors of 2). Note that 1 is not added again since it's already visited.
f. Dequeue node 3, add it to the result list, mark it as visited.
g. Enqueue node 6 (neighbor of 3).
h. Dequeue node 4, add it to the result list, mark it as visited. No more unvisited neighbors.
i. Dequeue node 5, add it to the result list, mark it as visited. No more unvisited neighbors.
j. Dequeue node 6, add it to the result list, mark it as visited. No more unvisited neighbors.