Name:- Sufiyan Khan
Branch:- TE-A(AI &DS) , SEM-5 , A2
Roll No:- A12
Subject:- Artificial Intelligence(AI) Lab

Experiment No.4

Aim: -

To implement Breadth First Search algorithm in AI with python code

Objective: -

To traverse through a graph in the smallest number of iterations.

Theory: -

Breadth-First Search (BFS) is an algorithm used for traversing graphs or trees. Traversing means visiting each node of the graph. Breadth-First Search is a recursive algorithm to search all the vertices of a graph or a tree. BFS in python can be implemented by using data structures like a dictionary and lists. Breadth-First Search in tree and graph is almost the same. The only difference is that the graph may contain cycles, so we may traverse to the same node again.

The architecture of BFS algorithm:-

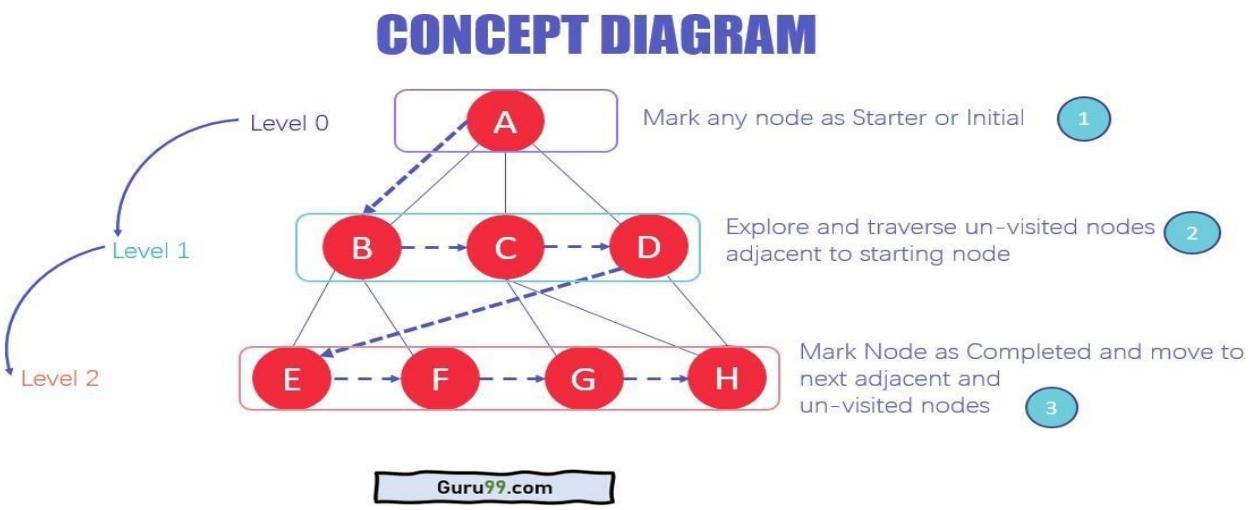

Fig. Concept Diagram of BFS

In the various levels of the data, you can mark any node as the starting or initial node to begin traversing. The BFS will visit the node and mark it as visited and places it in the queue.

Now the BFS will visit the nearest and un-visited nodes and marks them. These values are also added to the queue. The queue works on the FIFO model.

In a similar manner, the remaining nearest and un-visited nodes on the graph are analysed marked and added to the queue. These items are deleted from the queue as receive and printed as the result.

Need OF BFS Algorithm

There are numerous reasons to utilize the BFS Algorithm to use as searching for your dataset. Some of the most vital aspects that make this algorithm your first choice are:

- BFS is useful for analysing the nodes in a graph and constructing the shortest path of traversing through these.
- BFS can traverse through a graph in the smallest number of iterations.
- The architecture of the BFS algorithm is simple and robust.

- The result of the BFS algorithm holds a high level of accuracy in comparison to other algorithms.
- BFS iterations are seamless, and there is no possibility of this algorithm getting caught up in an infinite loop problem.

BFS used in

- In GPS navigation, it helps in finding the shortest path available from one point to another.
- In pathfinding algorithms
- Cycle detection in an undirected graph
- In minimum spanning tree
- To build index by search index
- In Ford-Fulkerson algorithm to find maximum flow in a network.

Example:-

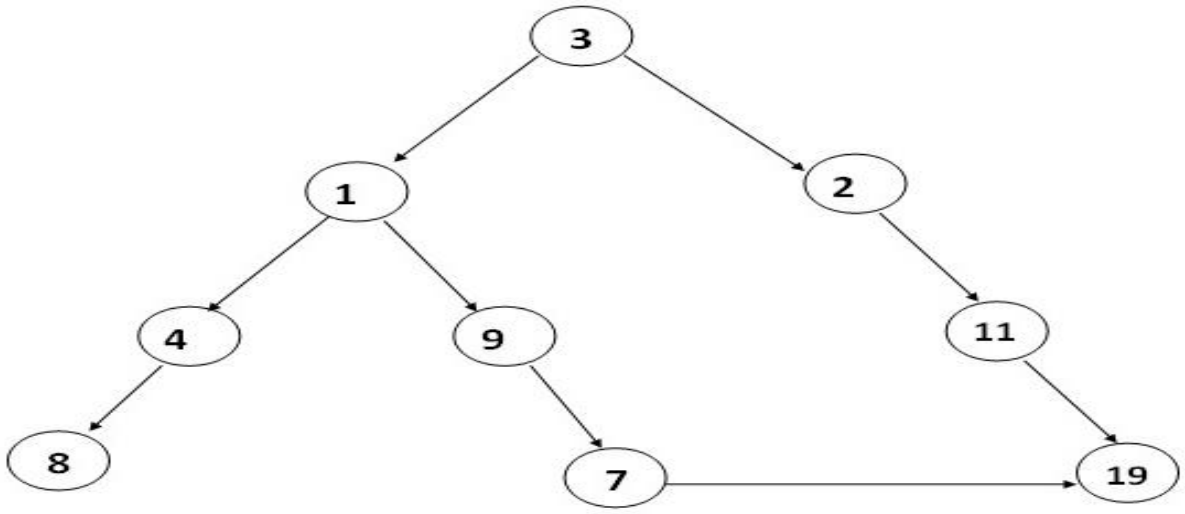

Fig:- BFS Diagram

Starting node = “3”

Here's how the BFS traversal works:

1. Initially, visited is an empty list, and queue is an empty queue.
2. The BFS traversal starts from node '3' (as specified by bfs(visited, graph, '3')).
3. Node '3' is marked as visited and added to the queue (visited = ['3'], queue = ['3']).
4. The while loop begins:
    - The current node '3' is dequeued from the queue (queue = []).
    - Node '3' is printed.
5. The neighbors of '3' are '1' and '2'. Since they are not visited yet, they are marked as visited and added to the queue (visited = ['3', '1', '2'], queue = ['1', '2']).
6. The while loop continues:
    - The current node '1' is dequeued from the queue (queue = ['2']).
    - Node '1' is printed.
7. The neighbors of '1' are '4' and '9'. They are added to the queue because they are not visited yet (visited = ['3', '1', '2', '4', '9'], queue = ['2', '4', '9']).
8. The while loop continues:
    - The current node '2' is dequeued from the queue (queue = ['4', '9']).
    - Node '2' is printed.
9. The neighbor of '2' is '11'. It is added to the queue because it's not visited yet (visited = ['3', '1', '2', '4', '9', '11'], queue = ['4', '9', '11']).
10. The while loop continues:
    - The current node '4' is dequeued from the queue (queue = ['9', '11']).
    - Node '4' is printed.
11. The neighbor of '4' is '8'. It is added to the queue because it's not visited yet (visited = ['3', '1', '2', '4', '9', '11', '8'], queue = ['9', '11', '8']).
12. The while loop continues:

- The current node '9' is dequeued from the queue (queue = ['11', '8']).
- Node '9' is printed.

13. The neighbor of '9' is '7'. It is added to the queue because it's not visited yet (visited = ['3', '1', '2', '4', '9', '11', '8', '7'], queue = ['11', '8', '7']).
14. The while loop continues:
    - The current node '11' is dequeued from the queue (queue = ['8', '7']).
    - Node '11' is printed.
15. The neighbor of '11' is '19'. It is added to the queue because it's not visited yet (visited = ['3', '1', '2', '4', '9', '11', '8', '7', '19'], queue = ['8', '7', '19']).
16. The while loop continues:
    - The current node '8' is dequeued from the queue (queue = ['7', '19']).
    - Node '8' is printed.
17. Node '8' has no unvisited neighbors, so we move on to the next node in the queue.
18. The current node '7' is dequeued from the queue (queue = ['19']).
    - Node '7' is printed.
19. The neighbor of '7' is '19', but it has already been visited, so it's not added to the queue.
20. The while loop continues:
    - The current node '19' is dequeued from the queue (queue = []).
21. Node '19' has no unvisited neighbors, so the while loop exits.

The corrected result of the BFS traversal starting from node '3' is the following sequence of nodes: '3', '1', '2', '4', '9', '11', '8', '7', '19'. This sequence represents the order in which nodes are visited during the BFS traversal of the graph.

Program:

```
graph = {

'3' : ['1','2'],
'1' : ['4','9'],
'2' : ['11'],
'4' : ['8'],
'9' : ['7'],
'11' : ['19'],
'7':['19'],
'8':[],
'19':[]
}

visited = []
queue = []

def bfs(visited,graph,node):
    visited.append(node)
    queue.append(node)

    while queue:

        m = queue.pop(0)
        print(m,end=" ")

        for neighbour in graph[m]:
            if neighbour not in  visited:
                visited.append(neighbour)
                queue.append(neighbour)

print("Following is the Breadth-First Search")
bfs(visited,graph, '9')
```

Output: -

```
[Running] python -u "c:\Users\Sufiyan\exp4.py"
Following is the Breadth-First Search
3 1 2 4 9 11 8 7 19
[Done] exited with code=0 in 0.186 seconds
```

Conclusion:-

The successful implementation of the Breadth First Search Algorithm in Ai using python provides a clear understanding of BFS'S working.