Experiment No. 3

Name: Sakshi Jadhav **Roll no: 49**

Class: BE IT B **Batch: B3**

A) Breadth First Search

AIM: To study Uninformed search strategy BFS

Theory: Graph traversals

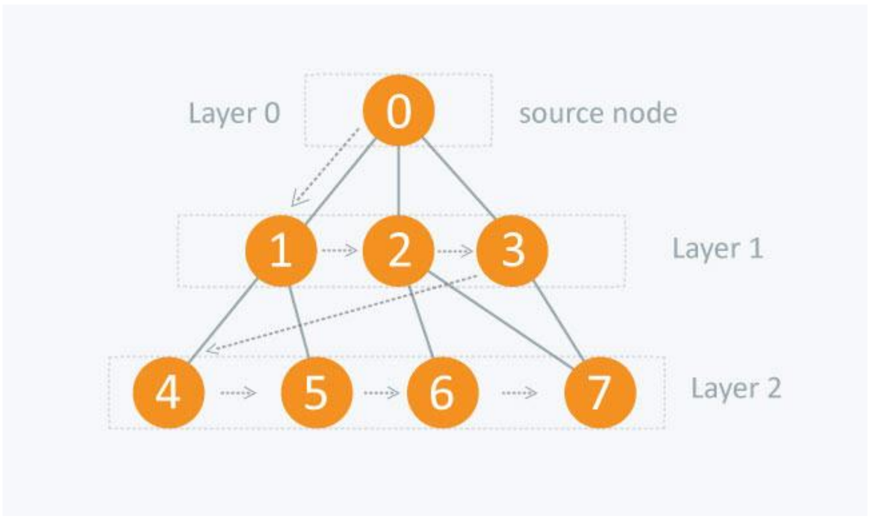
Graph traversal means visiting every vertex and edge exactly once in a well-defined order. While using certain graph algorithms, you must ensure that each vertex of the graph is visited exactly once. The order in which the vertices are visited are important and may depend upon the algorithm or question that you are solving.

During a traversal, it is important that you track which vertices have been visited. The most common way of tracking vertices is to mark them.
Breadth First Search (BFS)

There are many ways to traverse graphs. BFS is the most commonly used approach.
BFS is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layerwise thus exploring the neighbour nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbour nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:
1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

Consider the following diagram.



The distance between the nodes in layer 1 is comparatively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

Traversing child nodes

A graph can contain cycles, which may bring you to the same node again while traversing the graph. To avoid processing of same node again, use a boolean array which marks the node after it is processed. While visiting the nodes in the layer of a graph, store them in a manner such that you can traverse the corresponding child nodes in a similar order.

In the earlier diagram, start traversing from 0 and visit its child nodes 1, 2, and 3. Store them in the order in which they are visited. This will allow you to visit the child nodes of 1 first (i.e. 4 and 5), then of 2 (i.e. 6 and 7), and then of 3 (i.e. 7) etc.

To make this process easy, use a queue to store the node and mark it as 'visited' until all its neighbours (vertices that are directly connected to it) are marked. The queue follows the First In First Out (FIFO) queuing method, and therefore, the neighbors of the node will be visited in the order in which they were inserted in the node i.e. the node that was inserted first will be visited first, and so on.

Algorithm:

Step1: SET STATUS = 1 (ready state) for each node in G

Step 2: Enqueue the starting node A and set its STATUS = 2 (waiting state)

Step 3: Repeat Steps 4 and 5 until QUEUE is empty

Step 4: Dequeue a node N. Process it and set its STATUS = 3 (processed state).

Step 5: Enqueue all the neighbours of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)
[END OF LOOP]

Step 6: EXIT

Pseudocode:

```
BFS (G, s)    //Where G is the graph and s is the source node
    let Q be queue.
    Q.enqueue( s ) //Inserting s in queue until all its neighbour vertices are marked.
    mark s as visited.
    while ( Q is not empty)
        //Removing that vertex from queue, whose neighbour will be visited now
        v  =  Q.dequeue( )
        //processing all the neighbours of v
        for all neighbours w of v in Graph G
            if w is not visited
                Q.enqueue( w )          //Stores w in Q to further visit its neighbour
                mark w as visited.
```

Python code:

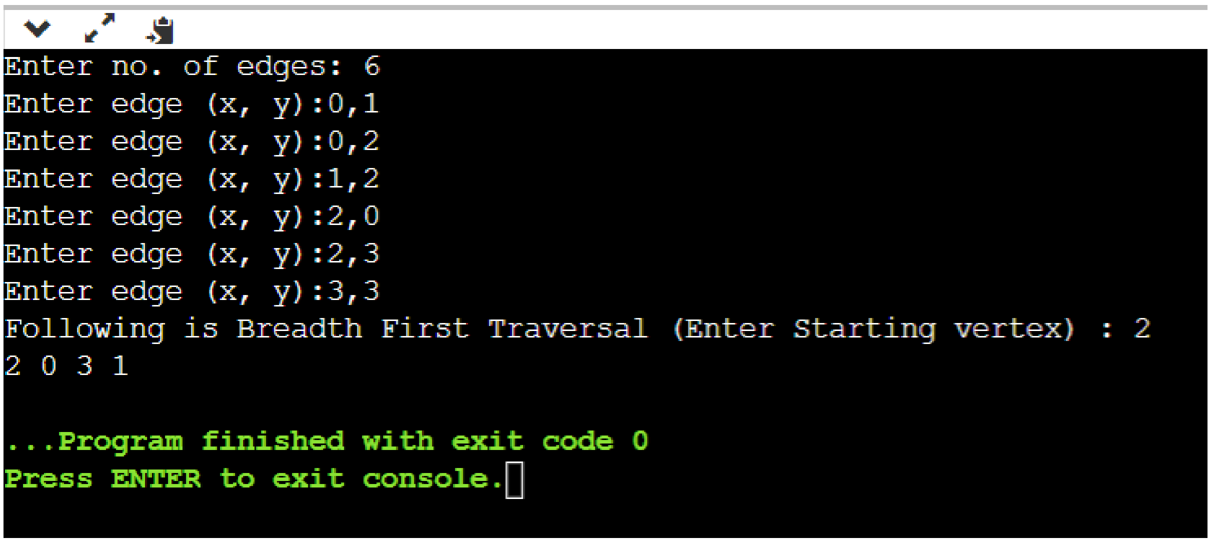
```
from collections import defaultdict
class Graph:
    def __init__(self):
        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def BFS(self, s):
        # Mark all the vertices as not visited
        visited = [False] * (max(self.graph) + 1)
        # Create a queue for BFS
        queue = [s]
        visited[s] = True
        while queue:
            # Dequeue a vertex from queue and print
            s = queue.pop(0)
            print (s, end = " ")
            # Get all adjacent vertices s. If a adjacent has not been visited,

            # then mark it visited and enqueue
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True

    # Create a Graph
g = Graph()
n = int(input("Enter no. of edges: "))
for _ in range(n):
    edges = list(map(int, input("Enter edge (x, y):").rstrip().split(',')))
    g.addEdge(edges[0], edges[1])
# Call BFS
v = int(input("Following is Breadth First Traversal (Enter Starting vertex) : "))
g.BFS(v)
```



```
Enter no. of edges: 6
Enter edge (x, y):0,1
Enter edge (x, y):0,2
Enter edge (x, y):1,2
Enter edge (x, y):2,0
Enter edge (x, y):2,3
Enter edge (x, y):3,3
Following is Breadth First Traversal (Enter Starting vertex) : 2
2 0 3 1

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

Conclusion: Thus, we have successfully studied Breadth first Search.