Experiment No. 2

Aim: To write a program of BFS (Breadth First Search).

Requirements: Windows/MAC/Linux O.S, Compatible version of Python.

Theory:

BFS is an algorithm to traverse each node of **Graph** at least once. It goes level by level that means it will only explore nodes of next level if all the nodes of previous level are explored or it first traverse all neighbouring nodes and then move forwards. This algorithm implements **Queue Data Structure** to store intermediate nodes and terminates when Queue is exhausted or empty (depends on how it is used) and visited array is used to keep tracks of already visited nodes so they don't get visited again. Normal **Set Data Structure** can also be used for maintaining visited nodes if number of nodes are unknown, below code uses **Set** to store visited nodes.

Algorithm:

Step 1: Take an Empty Queue.

Step 2: Select a starting node (visiting a node) and insert it into the Queue.

Step 3: Provided that the Queue is not empty, extract the node from the Queue and insert its child nodes (exploring a node) into the Queue.

Step 4: Print the extracted node.

Breadth-First Search Algorithm Pseudocode

```
1. Input: s as the source node

2. BFS (G, s)
```

```
3. Let Q be queue.
4. Q.enqueue(s)
5. Mark s as visited
6. While(Q is not empty)
7. v = Q.dequeue( )
8. For all neighbours w of v in Graph G
9. If w is not visited
10. Q.enqueue(w)
11. Mark was visited
```

In the above code, the following steps are executed:

1. (G, s) is input, here G is the graph and s is the root node

2. A queue Q is created and initialized with the source node s

3. All child nodes of s are marked.

4. Extract s from queue and visit the child nodes

5. Process all the child nodes of v

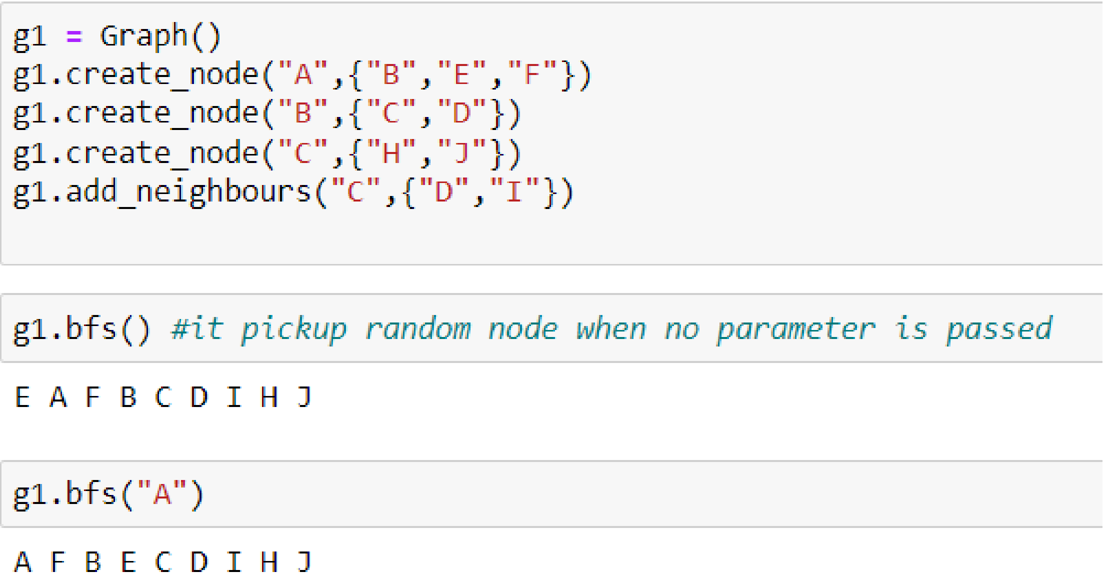
6. Stores w (child nodes) in Q to further visit its child nodes

7. Continue till Q is empty

<u>Code</u>:

```
#For Creating Nodes
class Node:
    '''Creating a new node and adding all its neighbour'''
    def __init__(self,value,Node_map,neighbours = set()):
        self.value = value
        self.neighbours = set()
        self.add_neighbours(Node_map,neighbours)

    '''Adding Neighbours to a node(bidirectional edges) by checking neighbour node exist or not,
        if doesn't creating one
    '''
    def add_neighbours(self,Node_map,neighbours=set()):
        for neighbour in neighbours:
            if neighbour not in Node_map:
                temp = Node(neighbour,Node_map)
                Node_map[neighbour] = temp
                temp.neighbours.add(self.value)
                self.neighbours.add(temp.value)

            else:
                self.neighbours.add(neighbour)
                Node_map[neighbour].neighbours.add(self.value)

class Graph:

    def __init__(self):
        self.Node_map = {} #Keeping accounting of nodes added till now with their value and references

    '''Simply create a Node if doesn't exist using Node class
        if Node exist only add edges
    '''
    def create_node(self,value,neighbours = set()):
        if value in self.Node_map:
            self.add_neighbours(value,neighbours)
        else:
            node = Node(value,self.Node_map,neighbours)
            self.Node_map[value] = node

    def add_neighbours(self,value, neighbhours):
        #Before adding neigbhour to node first checking if node exist or node, if not raise error
        assert value in self.Node_map, "Given node doesn't exist"
        node = self.Node_map[value]
        node.add_neighbours(self.Node_map,neighbhours)

    def bfs(self,value =None):
        visited = set()
        que = []
        if value == None:
            #picking up random node since no Node is passed
            que.append(list(self.Node_map.keys())[0])
        else:
            #Cheking if passed Node exist or not, if not raise error
            assert value in self.Node_map, "Node doesn't exist"
            que.append(value)

        visited.add(que[0])
        for value in(que):
            for neighbour in self.Node_map[value].neighbours:
                if neighbour not in visited:
                    que.append(neighbour)
                    visited.add(neighbour)
            print(value,end=" ")
```

<u>Output</u>:



```
g1 = Graph()
g1.create_node("A",{"B","E","F"})
g1.create_node("B",{"C","D"})
g1.create_node("C",{"H","J"})
g1.add_neighbours("C",{"D","I"})
```

```
g1.bfs() #it pickup random node when no parameter is passed
```

```
E A F B C D I H J
```

```
g1.bfs("A")
```

```
A F B E C D I H J
```

<u>Conclusion</u>: We have successfully implemented BFS program using Python.