

## **PROGRAM TITLE 07**

### **BREADTH – FIRST SEARCH**

#### **AIM:**

To Write the python program to implement BFS.

#### **PROCEDURE:**

1. Initialize Graph Class: Define a class named Graph to represent a graph. The class initializes with an empty dictionary to store the adjacency list representation of the graph.
2. Add Edges: Implement the add\_edge method within the Graph class to add edges to the graph. If a vertex is not present in the graph, create a new list to store its neighbors and append the neighbor to the list.
3. Breadth-First Search (BFS): Implement the bfs method within the Graph class to perform the Breadth-First Search traversal starting from a given vertex. The method initializes a set to store visited vertices and a queue (implemented using deque) to perform the BFS.
4. Traverse Graph: In the bfs method, use a loop to traverse the graph in a breadth-first manner. Print each visited vertex as it is dequeued from the queue. Enqueue the unvisited neighbors of the current vertex and mark them as visited.
5. Usage: In the main section of the program, create an instance of the Graph class. Add edges to the graph using the add\_edge method. Finally, call the bfs method with the starting vertex to perform the BFS traversal and print the result.

#### **CODING:**

```
from collections import deque
```

```
class Graph:
```

```
    def __init__(self):  
        self.graph = {}
```

```
    def add_edge(self, u, v):  
        if u not in self.graph:  
            self.graph[u] = []  
        self.graph[u].append(v)
```

```

def bfs(self, start):
    visited = set()
    queue = deque([start])
    visited.add(start)

    while queue:
        vertex = queue.popleft()
        print(vertex, end=" ")

        if vertex in self.graph:
            for neighbor in self.graph[vertex]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append(neighbor)

if __name__ == "__main__":
    g = Graph()
    g.add_edge(0, 1)
    g.add_edge(0, 2)
    g.add_edge(1, 2)
    g.add_edge(2, 0)
    g.add_edge(2, 3)
    g.add_edge(3, 3)

    print("Breadth First Traversal (starting from vertex 2):")
    g.bfs(2)

```

**OUTPUT:**

Breadth First Traversal (starting from vertex 2):

2 0 3 1

**RESULT:**

Hence the program been successfully executed and verified.