**Experiment 7:**

**Aim:** To write a python program to implement a Birth First Search (BFS) program.

**Algorithm:**

1. Represent the graph as an adjacency list or matrix.
2. Initialize BFS with a queue, starting from the source node, marking it visited, and enqueueing it.
3. Dequeue a node, process it, and enqueue unvisited neighbors, marking them visited.
4. Stop when the queue is empty, indicating all reachable nodes are visited.
5. Display the order of nodes visited.

**Program:**

from collections import deque

def build\_graph():

graph = {}

num\_edges = int(input("Enter the number of edges in the graph: "))

print("Enter each edge as a pair of space-separated nodes (e.g., A B):")

for \_ in range(num\_edges):

node1, node2 = input().split()

if node1 not in graph:

graph[node1] = []

if node2 not in graph:

graph[node2] = []

graph[node1].append(node2)

graph[node2].append(node1)

return graph

def bfs(graph, start\_node):

visited = set()

queue = deque([start\_node])

traversal\_order = []

while queue:

node = queue.popleft()

if node not in visited:

visited.add(node)

traversal\_order.append(node)

for neighbor in graph.get(node, []):

if neighbor not in visited:

queue.append(neighbor)

return traversal\_order

def main():

print("Build the graph:")

graph = build\_graph()

start\_node = input("Enter the start node for BFS: ")

if start\_node not in graph:

print("The start node is not in the graph.")

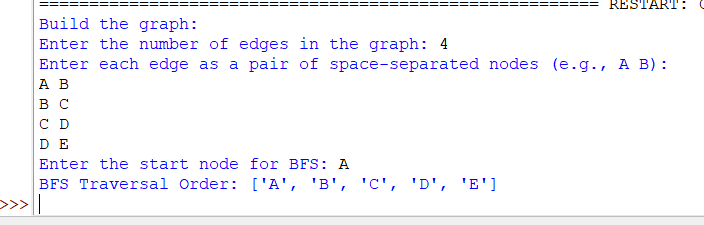
return

bfs\_result = bfs(graph, start\_node)

print("BFS Traversal Order:", bfs\_result)

main()

**Output:**

****

**Result:** Thus, the program was successfully completed using python programming.