**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

**COLLEGE OF E&ME, NUST, RAWALPINDI**

AI & Decision Support Systems

Lab Report #3

**Student Name: Nawab Aarij Imam**

**Degree/ Syndicate: 43 CE - A**

**Task1:**

**Code:**

def bfs(self, start):

visited = set()

queue = deque([start])

order = []

while queue:

vertex = queue.popleft()

if vertex not in visited:

visited.add(vertex)

order.append(vertex)

for neighbor in self.graph[vertex]:

if neighbor not in visited:

queue.append(neighbor)

return order

graph = UndirectedGraph()

for vertex in ['A', 'B', 'C', 'D', 'E']:

graph.add\_vertex(vertex)

edges = [

('A', 'B'), ('A', 'D'), ('A', 'E'),

('B', 'E'), ('B', 'D'),

('D', 'C')

]

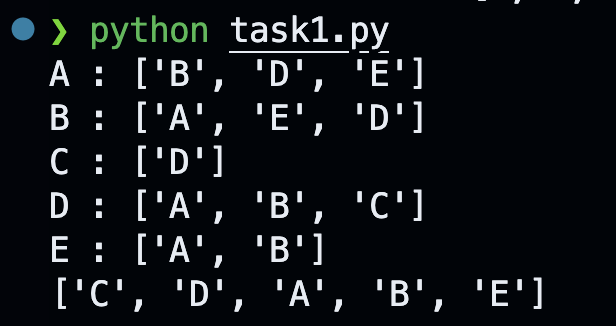
for edge in edges:

graph.add\_edge(edge[0], edge[1])

graph.print\_graph()

print(graph.bfs('C'))

**Output:**

****

**Task2:**

**Code:**

from collections import deque

class Tree:

def \_\_init\_\_(self, data=None):

self.data = data

self.left = None

self.middle = None

self.right = None

def bfs(root):

if root is None:

return []

queue = deque([root])

order = []

while queue:

current\_node = queue.popleft()

order.append(current\_node.data)

if current\_node.left:

queue.append(current\_node.left)

if current\_node.middle:

queue.append(current\_node.middle)

if current\_node.right:

queue.append(current\_node.right)

return order

root = Tree(1)

root.left = Tree(2)

root.middle = Tree(3)

root.right = Tree(4)

root.left.left = Tree(5)

root.left.right = Tree(6)

root.right.left = Tree(7)

root.right.right = Tree(8)

root.left.left.left = Tree(9)

root.left.left.right = Tree(10)

root.right.left.left = Tree(11)

root.right.left.right = Tree(12)

bfs\_result = bfs(root)

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

**Output:**

****

**Task3:**

**Code:**

from collections import deque

def shortest\_path\_bfs(maze):

if not maze or not maze[0]:

return -1

rows, cols = len(maze), len(maze[0])

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

queue = deque([(0, 0, 0)])

visited = set((0, 0))

while queue:

row, col, distance = queue.popleft()

if row == rows - 1 and col == cols - 1:

return distance

for dr, dc in directions:

new\_row, new\_col = row + dr, col + dc

if 0 <= new\_row < rows and 0 <= new\_col < cols and maze[new\_row][new\_col] == 0 and (new\_row, new\_col) not in visited:

visited.add((new\_row, new\_col))

queue.append((new\_row, new\_col, distance + 1))

return -1

if \_\_name\_\_ == "\_\_main\_\_":

maze = [

[0, 1, 0, 0, 0],

[0, 1, 0, 1, 0],

[0, 0, 0, 1, 0],

[0, 1, 1, 0, 0],

[0, 0, 0, 0, 0]

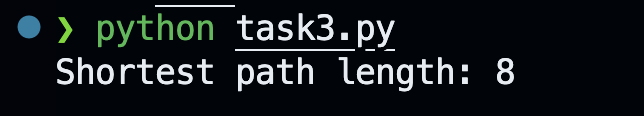
]

# Find the shortest path

result = shortest\_path\_bfs(maze)

print("Shortest path length:", result)

**Output:**

****