***Muhammad Kaleem***

***56614***

***Lab Task no 7***

***Q1***

***Code:***

def dfs(node, graph, visited, component):

component.append(node)

visited[node] = True

for child in graph[node]:

if not visited[child]:

dfs(child, graph, visited, component) # Use child for recursion

# Function to perform DFS and print the traversal

def perform\_dfs(graph, start\_node):

visited = [False] \* len(graph) # Initialize visited list

component = []

dfs(start\_node, graph, visited, component)

return component

# Initial graph

graph = {

0: [2],

1: [2, 3],

2: [0, 1, 4],

3: [1, 4],

4: [2, 3]

}

# Perform DFS on the initial graph

print("Initial graph DFS starting from node 0:")

initial\_dfs = perform\_dfs(graph, 0)

print("Following is the Depth First Search:", initial\_dfs)

# Adding an edge between 3 and 0

graph[3].append(0)

graph[0].append(3)

# Removing an edge between 1 and 2

graph[1].remove(2)

graph[2].remove(1) # Undirected graph: remove edge in both directions

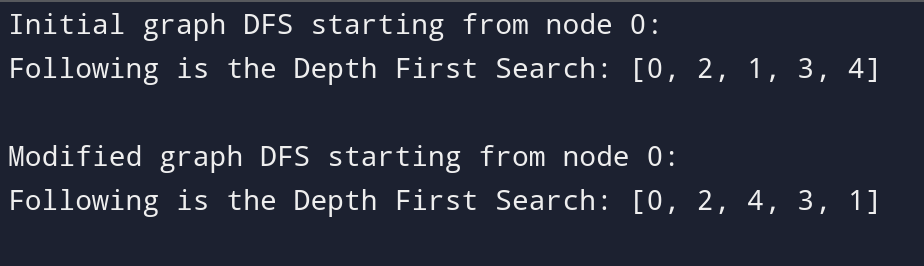
# Perform DFS on the modified graph

print("\nModified graph DFS starting from node 0:")

modified\_dfs = perform\_dfs(graph, 0)

print("Following is the Depth First Search:", modified\_dfs)

***Output:***

******

***Q2***

***Code:***

def dfs\_iterative(start\_node, graph):

visited = [False] \* len(graph) # Initialize visited list

component = []

stack = [start\_node] # Initialize stack with the starting node

while stack:

node = stack.pop() # Get the last node added to the stack

if not visited[node]:

visited[node] = True # Mark it as visited

component.append(node) # Add to the component list

# Add all unvisited neighbors to the stack

for child in reversed(graph[node]): # Reverse to maintain order

if not visited[child]:

stack.append(child)

return component

# Main

graph = {

0: [2],

1: [2, 3],

2: [0, 1, 4],

3: [1, 4],

4: [2, 3]

}

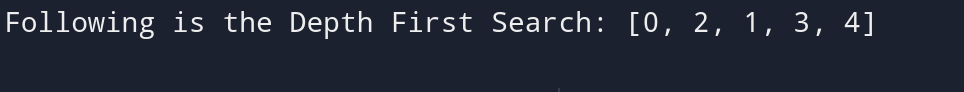
# Perform DFS iteratively

node = 0

result = dfs\_iterative(node, graph)

print("Following is the Depth First Search:", result)

***Output:***

******

***Q3***

***Code:***

def dfs(node, graph, visited, component):

component.append(node)

visited[node] = True

for child in graph[node]:

if not visited[child]:

dfs(child, graph, visited, component) # Use child for recursion

def count\_connected\_components(graph):

visited = [False] \* len(graph) # Initialize visited list

components = []

for node in graph:

if not visited[node]:

component = []

dfs(node, graph, visited, component)

components.append(component)

return len(components), components

# Initial graph

graph = {

0: [2],

1: [2, 3],

2: [0, 1, 4],

3: [1, 4],

4: [2, 3]

}

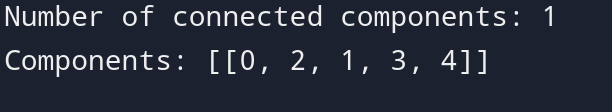
# Counting connected components in the initial graph

num\_components, components = count\_connected\_components(graph)

print(f"Number of connected components: {num\_components}")

print("Components:", components)

***Output:***

***Q4***

***Code:***

def dfs(node, graph, visited):

# Mark the node as visited

visited[node] = True

print(f"Node {node} -> Adjacent: {graph[node]}")

# Traverse through all adjacent nodes (children)

for child in graph[node]:

if not visited[child]:

dfs(child, graph, visited)

def perform\_dfs(graph):

visited = [False] \* len(graph) # Initialize visited list for all nodes

# Start DFS from the first node (assuming node 0)

for node in graph:

if not visited[node]:

print(f"\nStarting DFS from node {node}")

dfs(node, graph, visited)

# Initial graph (based on the image you uploaded)

graph = {

0: [1, 2],

1: [0],

2: [0, 3, 4],

3: [2],

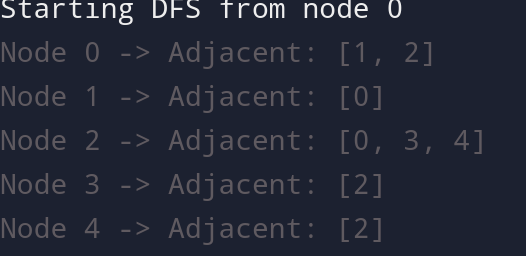
4: [2]

}

# Perform DFS and print traversal details

perform\_dfs(graph)

***Output:***



***Q5***

***Code:***

def bfs\_recursive(graph, level, visited, bfs\_result):

next\_level = []

# Visit all nodes in the current level

for node in level:

if not visited[node]:

visited[node] = True

bfs\_result.append(node)

# Collect all unvisited neighbors for the next level

for neighbor in graph[node]:

if not visited[neighbor]:

next\_level.append(neighbor)

# If the next level has nodes to visit, call bfs\_recursive for the next level

if next\_level:

bfs\_recursive(graph, next\_level, visited, bfs\_result)

def bfs(start\_node, graph):

visited = [False] \* len(graph) # Initialize visited list

bfs\_result = [] # To store the BFS traversal order

# Start BFS from the start node

bfs\_recursive(graph, [start\_node], visited, bfs\_result)

return bfs\_result

# Initial graph

graph = {

0: [2],

1: [2, 3],

2: [0, 1, 4],

3: [1, 4],

4: [2, 3]

}

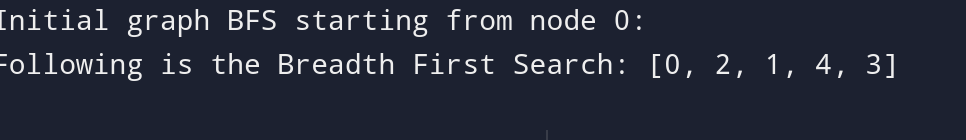
# Perform BFS on the graph

print("Initial graph BFS starting from node 0:")

initial\_bfs = bfs(0, graph)

print("Following is the Breadth First Search:", initial\_bfs)

***Output:***

******