**DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING**

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

AI & Decision Support Systems

Lab Report #4

**Student Name: Nawab Aarij Imam**

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

**Task1:**

Implement a LIFO data structure in python and using it implement DFS algorithm recursively and iteratively in python.

**Code:**

class Stack:

def \_\_init\_\_(self):

self.items = []

def push(self, item):

self.items.append(item)

def pop(self):

if not self.is\_empty():

return self.items.pop()

def is\_empty(self):

return len(self.items) == 0

graph = {

'A': ['B', 'C'],

'B': ['D', 'E'],

'C': ['F'],

'D': [],

'E': ['F'],

'F': []

}

def dfs\_recursive(graph, node, visited=None):

if visited is None:

visited = set()

visited.add(node)

print(node, end=' ')

for neighbor in graph[node]:

if neighbor not in visited:

dfs\_recursive(graph, neighbor, visited)

def dfs\_iterative(graph, start):

visited = set()

stack = Stack()

stack.push(start)

while not stack.is\_empty():

node = stack.pop()

if node not in visited:

print(node, end=' ')

visited.add(node)

for neighbor in reversed(graph[node]):

if neighbor not in visited:

stack.push(neighbor)

if \_\_name\_\_ == '\_\_main\_\_':

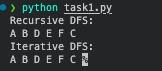
print("Recursive DFS:")

dfs\_recursive(graph, 'A')

print("\nIterative DFS:")

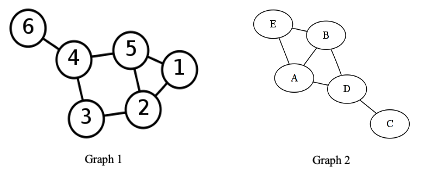
dfs\_iterative(graph, 'A')

**Output:**

****

**Task2:**

Traverse Graph 1 and 2 through implemented DFS algorithm. The starting node is ‘6’ for Graph 1 while the starting node is ‘E’ for Graph 2.

****

**Code:**

from task1 import \*

if \_\_name\_\_ == '\_\_main\_\_':

graph1 = {

1: [2, 5],

2: [1, 3, 5],

3: [2, 4],

4: [3, 5, 6],

5: [1, 2, 4],

6: [4]

}

graph2 = {

'A': ['D', 'F'],

'B': ['E', 'F'],

'C': ['D'],

'D': ['A', 'C'],

'E': ['B'],

'F': ['A', 'B']

}

print("Graph 1 DFS (starting from node 6):")

print("Recursive:")

dfs\_recursive(graph1, 6)

print("\nIterative:")

dfs\_iterative(graph1, 6)

print("\n\nGraph 2 DFS (starting from node 'E'):")

print("Recursive:")

dfs\_recursive(graph2, 'E')

print("\nIterative:")

dfs\_iterative(graph2, 'E')

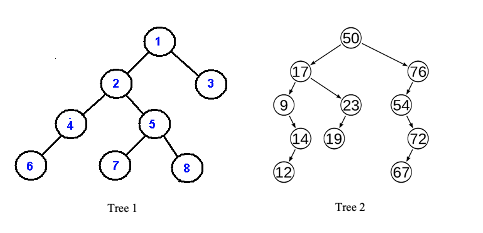
**Output:**

**A screenshot of a computer program

Description automatically generated**

**Task3:**

Traverse Tree 1 and 2 using Pre-Order, In-Order and Post-Order DFS traversals. The starting node is ‘1’ for Tree 1 while the starting node is ‘50’ for Tree 2.

****

**Code:**

class TreeNode:

def \_\_init\_\_(self, value):

self.value = value

self.left = None

self.right = None

def create\_tree1():

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

root.left.left.left = TreeNode(6)

root.left.right.left = TreeNode(7)

root.left.right.right = TreeNode(8)

return root

def create\_tree2():

root = TreeNode(50)

root.left = TreeNode(17)

root.right = TreeNode(76)

root.left.left = TreeNode(9)

root.left.right = TreeNode(23)

root.left.left.left = TreeNode(14)

root.left.left.left.left = TreeNode(12)

root.left.right.left = TreeNode(19)

root.right.left = TreeNode(54)

root.right.left.right = TreeNode(72)

root.right.left.right.left = TreeNode(67)

return root

def pre\_order(node):

if node:

print(node.value, end=' ')

pre\_order(node.left)

pre\_order(node.right)

def in\_order(node):

if node:

in\_order(node.left)

print(node.value, end=' ')

in\_order(node.right)

def post\_order(node):

if node:

post\_order(node.left)

post\_order(node.right)

print(node.value, end=' ')

if \_\_name\_\_ == '\_\_main\_\_':

tree1 = create\_tree1()

tree2 = create\_tree2()

print("Tree 1 Traversals:")

print("Pre-Order:", end=' ')

pre\_order(tree1)

print("\nIn-Order:", end=' ')

in\_order(tree1)

print("\nPost-Order:", end=' ')

post\_order(tree1)

print("\n\nTree 2 Traversals:")

print("Pre-Order:", end=' ')

pre\_order(tree2)

print("\nIn-Order:", end=' ')

in\_order(tree2)

print("\nPost-Order:", end=' ')

post\_order(tree2)

**Output:**

**A screenshot of a computer

Description automatically generated**

**Task4:**

Write a script to decompose the given image into an undirected graph where the pixel represents the vertices and adjacent vertices are connected to each other via 4-connectivity. Use DFS algorithm to traversal decomposed image starting from pixel 150.

|  |  |  |
| --- | --- | --- |
| 150 | 2 | 5 |
| 80 | 145 | 45 |
| 74 | 102 | 165 |

**Code:**

image = [

[150, 2, 5],

[80, 145, 45],

[74, 102, 165]

]

def create\_graph(image):

graph = {}

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

for i in range(rows):

for j in range(cols):

pixel = image[i][j]

neighbors = []

*# Check 4-connectivity: up, down, left, right*

for di, dj in [(-1, 0), (1, 0), (0, -1), (0, 1)]:

ni, nj = i + di, j + dj

if 0 <= ni < rows and 0 <= nj < cols:

neighbors.append(image[ni][nj])

graph[pixel] = neighbors

return graph

def dfs(graph, start, visited=None):

if visited is None:

visited = set()

visited.add(start)

print(start, end=' ')

for neighbor in graph[start]:

if neighbor not in visited:

dfs(graph, neighbor, visited)

graph = create\_graph(image)

print("DFS traversal starting from pixel 150:")

dfs(graph, 150)

**Output:**

****