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:

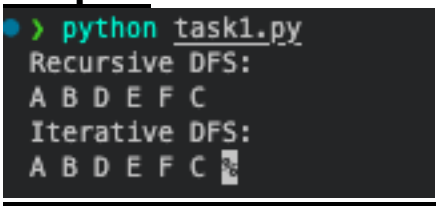
Implement a LIFO data structure in python and using it implement DFS algorithm recursively and iteratively in python.

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=' ')
```

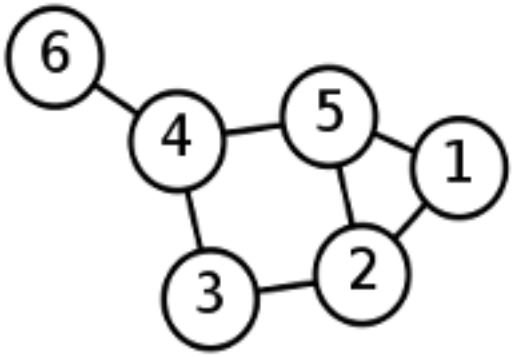
```
    for neighbor in graph[node]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor, visited)


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:

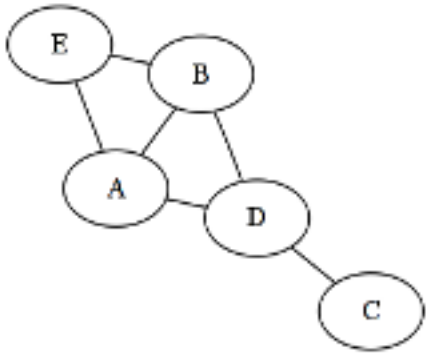


Task2:

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.



Graph 1



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']
  }
```

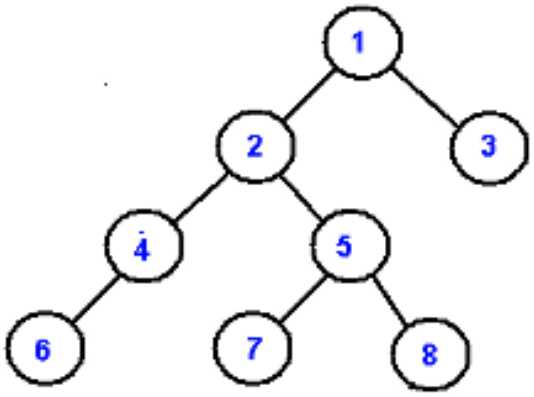
```
print("Graph 1 DFS (starting from node 6):")
print("Recursive:")
dfs_recursive(graph1, 6)
print("\nIterative:")
dfs_iterative(graph1, 6)

print("\n\nGraph 2 DFS (starting from node 'E'):")
print("Recursive:")
dfs_recursive(graph2, 'E')
print("\nIterative:")
dfs_iterative(graph2, 'E')
```

Output:

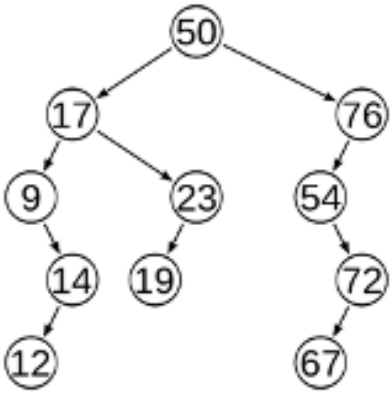
```
❯ python task2.py
Graph 1 DFS (starting from node 6):
Recursive:
6 4 3 2 1 5
Iterative:
6 4 3 2 1 5

Graph 2 DFS (starting from node 'E'):
Recursive:
E B F A D C
Iterative:
E B F A D C
```

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.



Tree 1



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:

```
❯ python task3.py
Tree 1 Traversals:
Pre-Order: 1 2 4 6 5 7 8 3
In-Order: 6 4 2 7 5 8 1 3
Post-Order: 6 4 7 8 5 2 3 1

Tree 2 Traversals:
Pre-Order: 50 17 9 14 12 23 19 76 54 72 67
In-Order: 12 14 9 17 19 23 50 54 67 72 76
Post-Order: 12 14 9 19 23 17 67 72 54 76 50
```

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:

```
❯ python task4.py
DFS traversal starting from pixel 150:
150 80 74 102 145 2 5 45 165
```