Shubham Sharma

RA1911003010649

AI- Lab 4

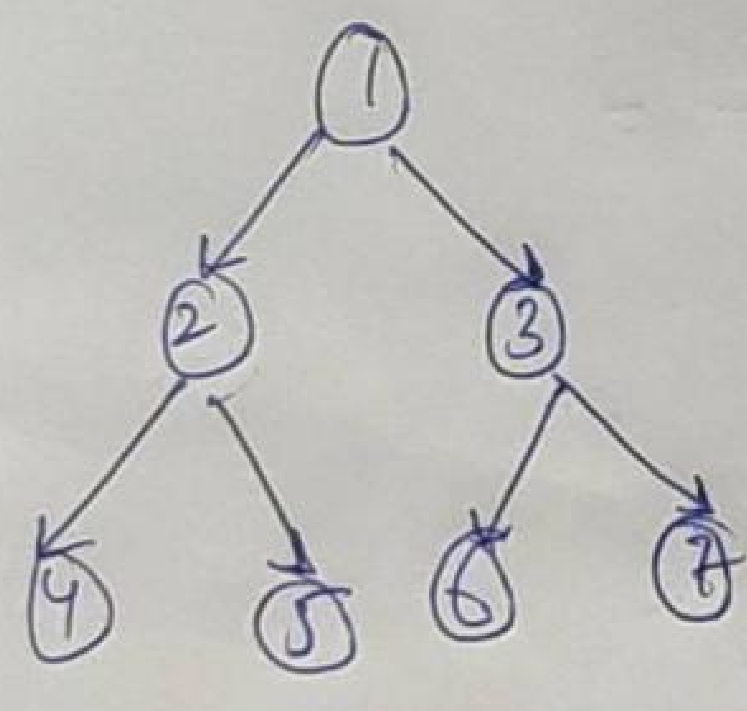
Aim - Implement & analysis of BFS and DFS for an application.

i) Implementation and Analysis of BFS using Level order Traversal.

Problem formulation -

Given a binary Tree, prints it node level by level from left to right level. Print Nodes of any level

Initial State



Final State

LOT = [1, 2, 3, 4, 5, 6, 7, 8]

For each given directed graph, LOT order would be empty array

Problem solving -

Here, we visit every node on a level before going to lower level.

Print all Nodes present in a level be modifying preorder traversal on tree

T.C = O(n^2)

S.C = O(h)

(ii) Implement and analysis of DFS using Flood Fill Algorithm

Problem formulation –

It is an algorithm that determines the area connected to a given node in a multi-dimensional array.

initial state

$$\begin{bmatrix} Y & Y & Y & G & G & G & G & G & G & G \\ Y & Y & Y & Y & Y & Y & G & X & X & X \\ G & G & G & G & G & G & G & X & X & X \\ W & W & W & W & W & G & G & G & G & X \\ W & R & R & R & R & R & G & X & X & X \\ W & W & W & R & R & G & G & X & X & X \\ W & B & W & R & R & R & R & R & R & X \\ W & B & B & B & B & B & R & X & X & X \\ W & B & B & X & B & B & B & B & X & X \\ W & B & B & X & X & X & X & X & X & X \end{bmatrix}$$

final state

$$\begin{bmatrix} Y & Y & Y & G & G & G & G & G & G & G \\ Y & Y & Y & Y & Y & Y & G & C & C & C \\ G & G & G & G & G & G & G & C & C & C \\ W & W & W & W & W & G & G & G & G & C \\ W & R & R & R & R & R & G & C & C & C \\ W & B & W & R & R & G & G & C & C & C \\ W & B & W & R & R & R & R & R & R & C \\ W & B & B & B & B & R & R & C & C & C \\ W & B & B & C & B & B & B & B & C & C \\ W & B & B & C & C & C & C & C & C & C \end{bmatrix}$$

Problem solving –

⇒ The time is to start from the source in the matrix, with the replacement color and recursively replace its color with the explore all its valid eight adjacent pixels and replace the Color.

T.C ⇒ $O(M \times N)$

S.C ⇒ $O(M \times N)$

NAME – SHUBHAM SHARMA

RA1911003010649

AI LAB 4

Algorithm

Step 1- Start

Step 2- Make a class to store binary tree node.

Step 3- Make function to print all nodes of a given level from left to right.

Step 4- Return true if at least one node is present at the given level.

Step 5- Call function to print level order traversal and start from 1 to height h of tree.

Step 6- Run till function returns false.

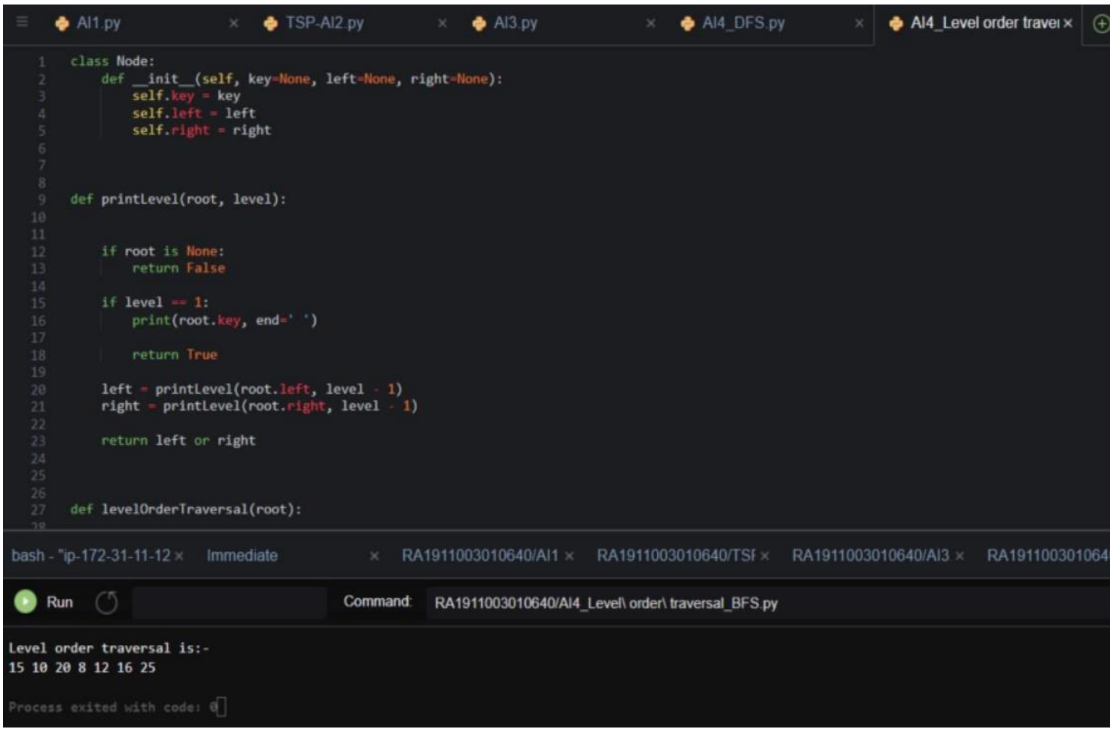
Step 7- End

Source Code

```
class Node:
def __init__(self, key=None, left=None, right=None):
self.key = key
self.left = left
self.right = right

def printLevel(root, level):


if root is None:
return False


if level == 1:


print(root.key, end=' ')


return True
```

```
left = printLevel(root.left, level - 1)
right = printLevel(root.right, level - 1)

return left or right

def levelOrderTraversal(root):

level = 1

while printLevel(root, level):
level = level + 1

if __name__ == '__main__':

root = Node(15)
root.left = Node(10)
root.right = Node(20)
root.left.left = Node(8)
root.left.right = Node(12)
root.right.left = Node(16)
root.right.right = Node(25)

print("Level order traversal is:- ")

levelOrderTraversal(root)
```

OUTPUT –



```
class Node:
    def __init__(self, key=None, left=None, right=None):
        self.key = key
        self.left = left
        self.right = right



def printLevel(root, level):


    if root is None:
        return False

    if level == 1:
        print(root.key, end=' ')

        return True

    left = printLevel(root.left, level - 1)
    right = printLevel(root.right, level - 1)

    return left or right



def levelOrderTraversal(root):
```

```
RA1911003010640/AI4_Level\ order\ traversal_BFS.py
```

```
Level order traversal is:-
15 10 20 8 12 16 25

Process exited with code: 0
```

Result

Hence level order traversal using BFS is successfully executed.

DFS - Flood Fill Algorithm

Algorithm

Step 1- Start

Step 2- Initialize roe and column array.

Step 3- Check if it is possible to go to pixel (x,y) from the current pixel. Return false if it has different color.

Step 4- Call the function, if if has same color, returns else replace that color with replacement color.

Step 5- Print colors after replacement.

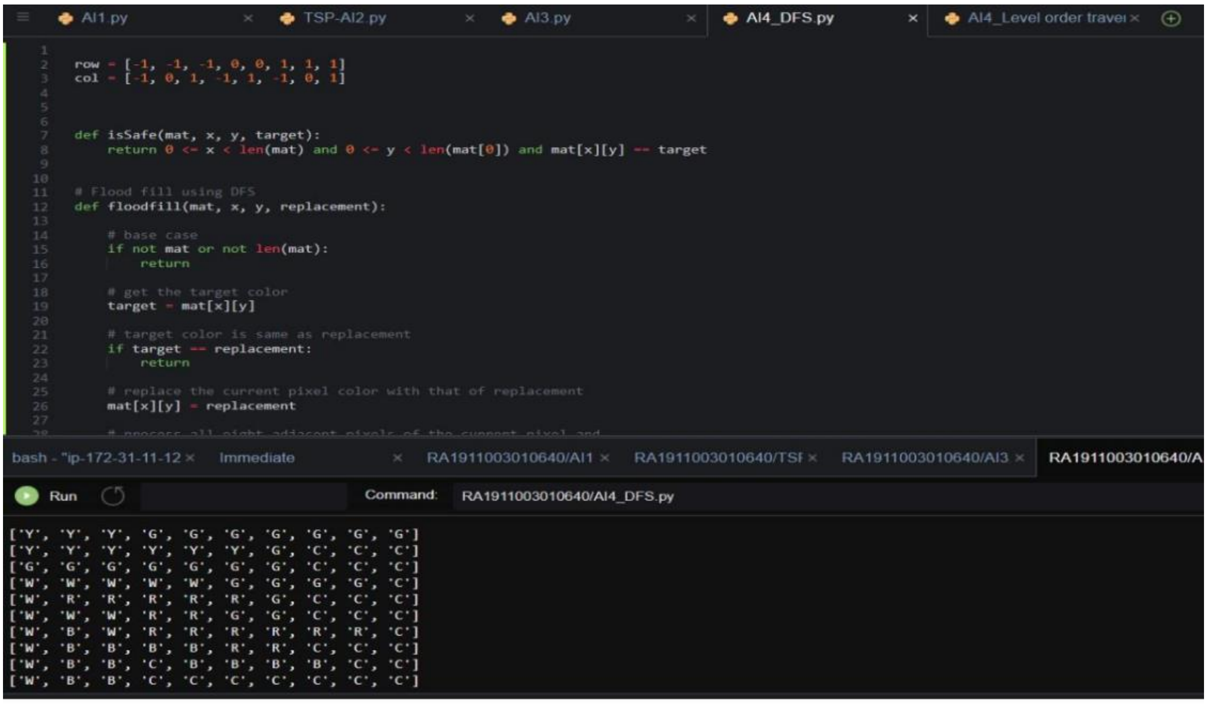
Step 6- End

Source Code

ow = [-1, -1, -1, 0, 0, 1, 1, 1]

```
col = [-1, 0, 1, -1, 1, -1, 0, 1]

def isSafe(mat, x, y, target):
return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and mat[x][y] == target

def floodfill(mat, x, y, replacement):

# base case
if not mat or not len(mat):
return

target = mat[x][y]

if target == replacement:
return

mat[x][y] = replacement

for k in range(len(row)):

if isSafe(mat, x + row[k], y + col[k], target):
floodfill(mat, x + row[k], y + col[k], replacement)

if __name__ == '__main__':

mat = [
['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G'],
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'X', 'X', 'X'],
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'X', 'X', 'X'],
['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'X'],
['W', 'R', 'R', 'R', 'R', 'R', 'G', 'X', 'X', 'X'],
['W', 'W', 'W', 'R', 'R', 'G', 'G', 'X', 'X', 'X'],

['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'X'],

['W', 'B', 'B', 'B', 'B', 'R', 'R', 'X', 'X', 'X'],

['W', 'B', 'B', 'X', 'B', 'B', 'B', 'B', 'X', 'X'],

['W', 'B', 'B', 'X', 'X', 'X', 'X', 'X', 'X', 'X']

]


x, y = (3, 9)


replacement = 'C'


floodfill(mat, x, y, replacement)


for r in mat:

print(r)
```

Output-



```

row = [-1, -1, -1, 0, 0, 1, 1, 1]
col = [-1, 0, 1, -1, 1, -1, 0, 1]



def isSafe(mat, x, y, target):
    return 0 <= x < len(mat) and 0 <= y < len(mat[0]) and mat[x][y] == target


# Flood fill using DFS
def floodfill(mat, x, y, replacement):

    # base case
    if not mat or not len(mat):
        return

    # get the target color
    target = mat[x][y]

    # target color is same as replacement
    if target == replacement:
        return

    # replace the current pixel color with that of replacement
    mat[x][y] = replacement

    # process all eight adjacent pixels of the current pixel and
```

RA1911003010640/AI4_DFS.py

```
['Y', 'Y', 'Y', 'G', 'G', 'G', 'G', 'G', 'G', 'G']
['Y', 'Y', 'Y', 'Y', 'Y', 'Y', 'G', 'C', 'C', 'C']
['G', 'G', 'G', 'G', 'G', 'G', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'W', 'W', 'G', 'G', 'G', 'G', 'C']
['W', 'R', 'R', 'R', 'R', 'R', 'G', 'C', 'C', 'C']
['W', 'W', 'W', 'R', 'R', 'G', 'G', 'C', 'C', 'C']
['W', 'B', 'W', 'R', 'R', 'R', 'R', 'R', 'R', 'C']
['W', 'B', 'B', 'B', 'B', 'R', 'R', 'C', 'C', 'C']
['W', 'B', 'B', 'C', 'B', 'B', 'B', 'B', 'C', 'C']
['W', 'B', 'B', 'C', 'C', 'C', 'C', 'C', 'C', 'C']
```

Result

Hence Flood Fill Algorithm using DFS is successfully executed.