

Day-17 : Binary Tree - 1

Problem Statement: Given a Binary Tree. Find and print

1. the inorder traversal of Binary Tree.
2. the preorder traversal of Binary Tree
3. the postorder traversal of Binary Tree

class Node:

```
def __init__(self, value):
```

```
    self.value = value
```

```
    self.left = None
```

```
    self.right = None
```

```
def inorder_traversal(node):
```

```
    if node:
```

```
        inorder_traversal(node.left)
```

```
        print(node.value, end=" ")
```

```
        inorder_traversal(node.right)
```

```
def preorder_traversal(node):
```

```
    if node:
```

```
        print(node.value, end=" ")
```

```
        preorder_traversal(node.left)
```

```
        preorder_traversal(node.right)
```

```
def postorder_traversal(node):
```

```
    if node:
```

```
        postorder_traversal(node.left)
```

```
    postorder_traversal(node.right)
```

```
    print(node.value, end=" ")
```

```
root = Node(1)
```

```
root.left = Node(2)
```

```
root.right = Node(3)
```

```
root.left.left = Node(4)
```

```
root.left.right = Node(5)
```

```
print("Inorder traversal:")
```

```
inorder_traversal(root)
```

```
print()
```

```
print("Preorder traversal:")
```

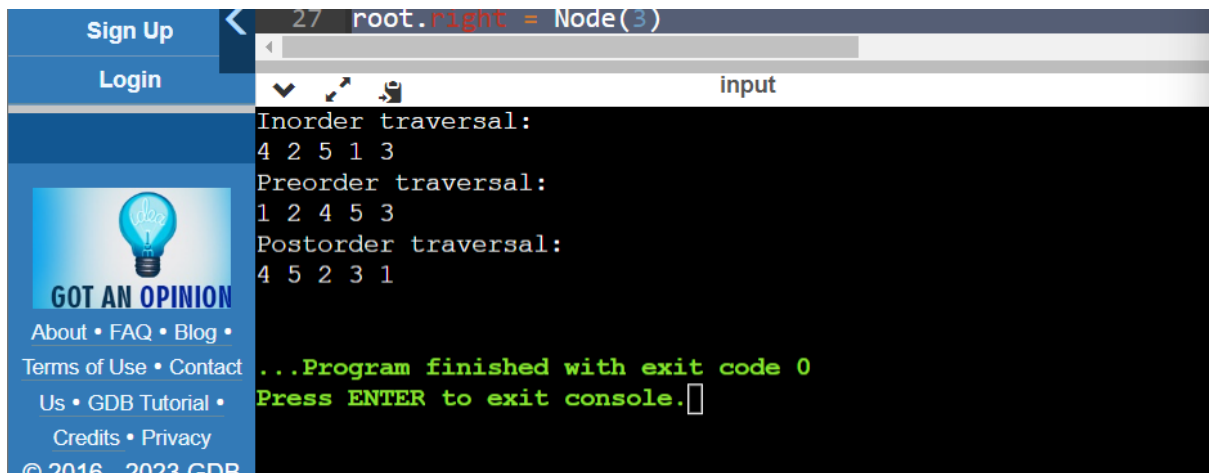
```
preorder_traversal(root)
```

```
print()
```

```
print("Postorder traversal:")
```

```
postorder_traversal(root)
```

```
print()
```



```
27 root.right = Node(3)

Inorder traversal:
4 2 5 1 3
Preorder traversal:
1 2 4 5 3
Postorder traversal:
4 5 2 3 1

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Write a program for

1. Morris Inorder Traversal of a Binary Tree.
2. Morris Preorder Traversal of a Binary Tree.

```
class TreeNode:
```

```
    def __init__(self, val):
```

```
        self.val = val
```

```
        self.left = None
```

```
        self.right = None
```

```
def morris_inorder(root):
```

```
    curr = root
```

```
    while curr:
```

```
        if not curr.left:
```

```
            print(curr.val, end=" ")
```

```
            curr = curr.right
```

```
        else:
```

```
            predecessor = curr.left
```

```
            while predecessor.right and predecessor.right != curr:
```

```
                predecessor = predecessor.right
```

```
    if not predecessor.right:
        predecessor.right = curr
        curr = curr.left
    else:
        predecessor.right = None
        print(curr.val, end=" ")
        curr = curr.right
```

```
def morris_preorder(root):
    curr = root
    while curr:
        if not curr.left:
            print(curr.val, end=" ")
            curr = curr.right
        else:
            predecessor = curr.left
            while predecessor.right and predecessor.right != curr:
                predecessor = predecessor.right

            if not predecessor.right:
                predecessor.right = curr
                print(curr.val, end=" ")
                curr = curr.left
            else:
                predecessor.right = None
                curr = curr.right
```

```
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)
```

```

root.left.left = TreeNode(4)
root.left.right = TreeNode(5)
root.right.left = TreeNode(6)
root.right.right = TreeNode(7)

```

```

print("Morris Inorder Traversal:")
morris_inorder(root)
print()

```

```

print("Morris Preorder Traversal:")
morris_preorder(root)
print()

```

The screenshot shows a web interface with a sidebar containing links like 'Learn Programming', 'Programming Questions', 'Jobs', 'Sign Up', and 'Login'. The main area displays a code editor with Python code for Morris Inorder and Preorder Traversal. Below the code, a terminal window shows the output of the program.

```

12         print(curr.val, end=" ")
13         curr = curr.right
14     else:
15         predecessor = curr.left
16         while predecessor.right and predecessor.right == curr:
17             predecessor = predecessor.right
18         print(curr.val, end=" ")
19         curr = predecessor
20     curr = curr.right
21
22 print("Morris Inorder Traversal:")
23 morris_inorder(root)
24 print()
25
26 print("Morris Preorder Traversal:")
27 morris_preorder(root)
28 print()

```

input

```

Morris Inorder Traversal:
4 2 5 1 6 3 7
Morris Preorder Traversal:
1 2 4 5 3 6 7
...Program finished with exit code 0
Press ENTER to exit console.

```

GOT AN OPINION
[About](#) • [FAQ](#) • [Blog](#) • [Terms of Use](#) • [Contact Us](#) • [GDB Tutorial](#) • [Credits](#) • [Privacy](#)
 © 2016 - 2023 GDB Online

Problem Statement: Given a Binary Tree, find the

1. Right view
2. Left view
3. Top view
4. Bottom view

class Node:

```

def __init__(self, data):

```

```
self.data = data  
self.left = None  
self.right = None
```

```
def printRightView(root):
```

```
    if root is None:
```

```
        return
```

```
    queue = [root]
```

```
    while queue:
```

```
        size = len(queue)
```

```
        for i in range(size):
```

```
            node = queue.pop(0)
```

```
            if i == size - 1:
```

```
                print(node.data, end=" ")
```

```
            if node.left:
```

```
                queue.append(node.left)
```

```
            if node.right:
```

```
                queue.append(node.right)
```

```
    print()
```

```
def printLeftView(root):
```

```
    if root is None:
```

```
        return
```

```
    queue = [root]
```

```
    while queue:
```

```
        size = len(queue)
```

```
        for i in range(size):
```

```
            node = queue.pop(0)
```

```
            if i == 0:
```

```
        print(node.data, end=" ")
    if node.left:
        queue.append(node.left)
    if node.right:
        queue.append(node.right)
    print()
```

```
def printBottomView(root):
```

```
    if root is None:
        return
```

```
    queue = [(root, 0)]
    bottom_view = {}
    while queue:
        node, hd = queue.pop(0)
        bottom_view[hd] = node.data
        if node.left:
            queue.append((node.left, hd - 1))
        if node.right:
            queue.append((node.right, hd + 1))
```

```
    for hd in sorted(bottom_view.keys()):
        print(bottom_view[hd], end=" ")
    print()
```

```
def printTopView(root):
```

```
    if root is None:
        return
```

```
    queue = [(root, 0)]
    top_view = {}
```

```
while queue:
    node, hd = queue.pop(0)
    if hd not in top_view:
        top_view[hd] = node.data
    if node.left:
        queue.append((node.left, hd - 1))
    if node.right:
        queue.append((node.right, hd + 1))
```

```
for hd in sorted(top_view.keys()):
    print(top_view[hd], end=" ")
print()
```

```
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.right = Node(4)
root.right.left = Node(5)
root.right.right = Node(6)
root.right.left.left = Node(7)
root.right.left.right = Node(8)
```

```
print("Right view of the binary tree:")
printRightView(root)
```

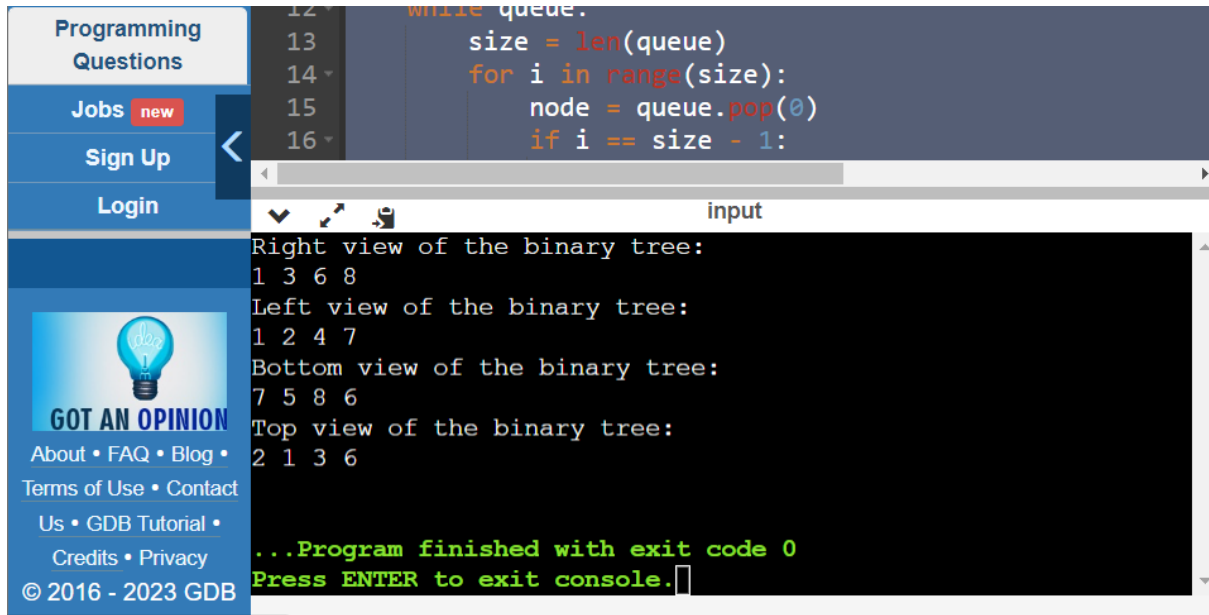
```
print("Left view of the binary tree:")
printLeftView(root)
```

```
print("Bottom view of the binary tree:")
printBottomView(root)
```



```
print("Top view of the binary tree:")
```

```
printTopView(root)
```



The screenshot shows a web application with a sidebar on the left containing links: "Programming Questions", "Jobs new", "Sign Up", "Login", and a "GOT AN OPINION" section with links to "About", "FAQ", "Blog", "Terms of Use", "Contact", "Us", "GDB Tutorial", "Credits", and "Privacy". The main content area displays a terminal window with the following output:

```
Right view of the binary tree:
1 3 6 8
Left view of the binary tree:
1 2 4 7
Bottom view of the binary tree:
7 5 8 6
Top view of the binary tree:
2 1 3 6

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Vertical Order Traversal Of A Binary Tree. Write a program for Vertical Order Traversal order of a Binary Tree.

```
from collections import defaultdict
```

```
class TreeNode:
```

```
    def __init__(self, key):
```

```
        self.key = key
```

```
        self.left = None
```

```
        self.right = None
```

```
def vertical_order_traversal(root):
```

```
    if not root:
```

```
        return []
```

```
vertical_levels = defaultdict(list)
```

```
queue = [(root, 0)]
```

```
while queue:
```

```
    node, level = queue.pop(0)
```

```
    vertical_levels[level].append(node.key)
```

```
    if node.left:
```

```
        queue.append((node.left, level - 1))
```

```
    if node.right:
```

```
        queue.append((node.right, level + 1))
```

```
sorted_levels = sorted(vertical_levels.keys())
```

```
for level in sorted_levels:
```

```
    print(*vertical_levels[level])
```

```
root = TreeNode(1)
```

```
root.left = TreeNode(2)
```

```
root.right = TreeNode(3)
```

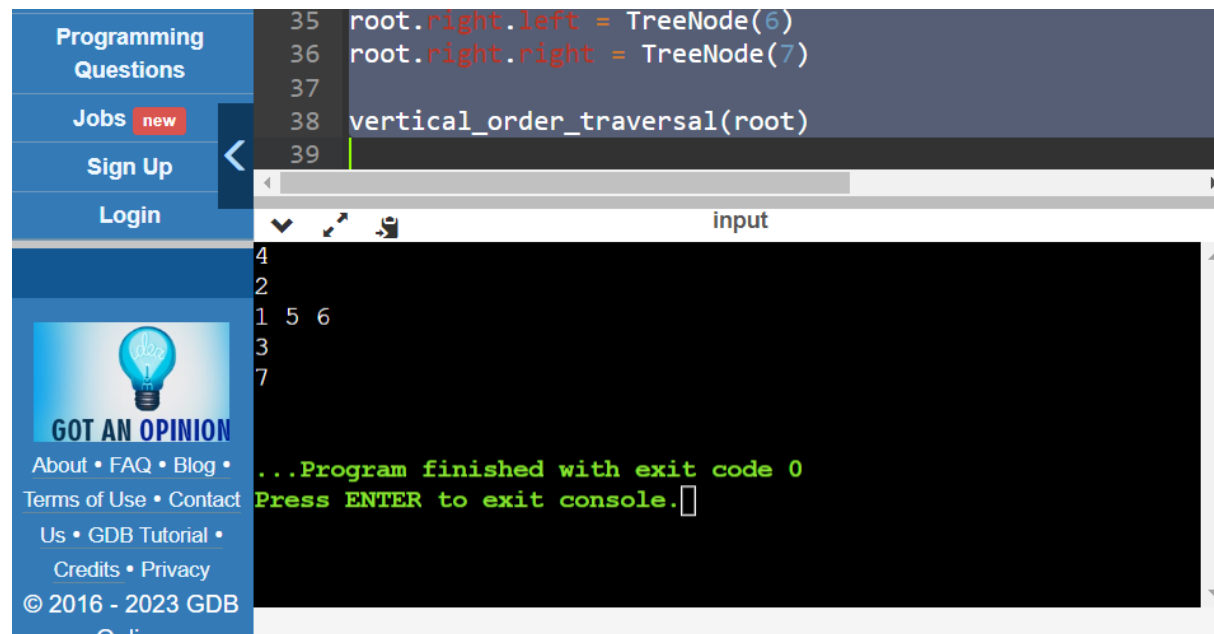
```
root.left.left = TreeNode(4)
```

```
root.left.right = TreeNode(5)
```

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

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

```
vertical_order_traversal(root)
```



The screenshot shows a web application interface. On the left is a sidebar with navigation links: 'Programming Questions', 'Jobs new', 'Sign Up', and 'Login'. Below these is a section titled 'GOT AN OPINION' with links to 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', and 'Privacy'. At the bottom of the sidebar is the copyright notice '© 2016 - 2023 GDB Online'. The main area contains a code editor with the following code:

```
35 root.right.left = TreeNode(6)
36 root.right.right = TreeNode(7)
37
38 vertical_order_traversal(root)
39
```

Below the code editor is a terminal window titled 'input' showing the output of the program:

```
4
2
1 5 6
3
7
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Print Root to Node Path In A Binary Tree. Write a program to print path from root to a given node in a binary tree.

```
class Node:
```

```
    def __init__(self, data):
```

```
        self.data = data
```

```
        self.left = None
```

```
        self.right = None
```

```
def print_path_to_node(root, target_node):
```

```
path = path_to_node_helper(root, target_node, [])

if path is not None:

    print("Path from root to node", target_node.data, ": ", end="")

    for i in range(len(path) - 1):

        print(path[i].data, "-> ", end="")

    print(path[-1].data)

else:

    print("Node", target_node.data, "not found in the tree")
```

```
def path_to_node_helper(node, target_node, path):

    if node is None:

        return None

    if node.data == target_node.data:

        path.append(node)

        return path

    left_path = path_to_node_helper(node.left, target_node, path)

    if left_path is not None:

        left_path.append(node)

        return left_path

    right_path = path_to_node_helper(node.right, target_node, path)
```

```
if right_path is not None:
    right_path.append(node)
    return right_path
return None
```

```
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)
root.right.left = Node(6)
root.right.right = Node(7)
```

```
target_node = root.left.right
print_path_to_node(root, target_node)
```

```
target_node = Node(8)
print_path_to_node(root, target_node)
```


Learn Programming

Programming Questions

Jobs new

Sign Up

Login


GOT AN OPINION

[About](#) • [FAQ](#) • [Blog](#) • [Terms of Use](#) • [Contact](#)

[Us](#) • [GDB Tutorial](#) • [Credits](#) • [Privacy](#)

© 2019 - 2020 GDB

```
11 for i in range(len(path) - 1):
12     print(path[i].data, "-> ", end="")
13     print(path[-1].data)
14 else:
15     print("Node", target_node.data, "not found in the tree")
16
```

input

```
Path from root to node 5 : 5 -> 2 -> 1
Node 8 not found in the tree

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem Statement: Write a program to find the **Maximum Width of A Binary Tree**.

class Node:

def __init__(self, data):

self.data = data

self.left = None

self.right = None

def get_tree_width(root):

if root is None:

return 0

max_width = 0

queue = []

```
queue.append((root, 1))
```

```
while len(queue) > 0:
```

```
    count = len(queue)
```

```
    max_width = max(max_width, count)
```

```
    while count > 0:
```

```
        node, index = queue.pop(0)
```

```
        if node.left is not None:
```

```
            queue.append((node.left, 2 * index))
```

```
        if node.right is not None:
```

```
            queue.append((node.right, 2 * index + 1))
```

```
        count -= 1
```

```
    return max_width
```

```
root = Node(1)
```

```
root.left = Node(2)
```

```
root.right = Node(3)
```

```
root.left.left = Node(4)
```

```
root.left.right = Node(5)
```

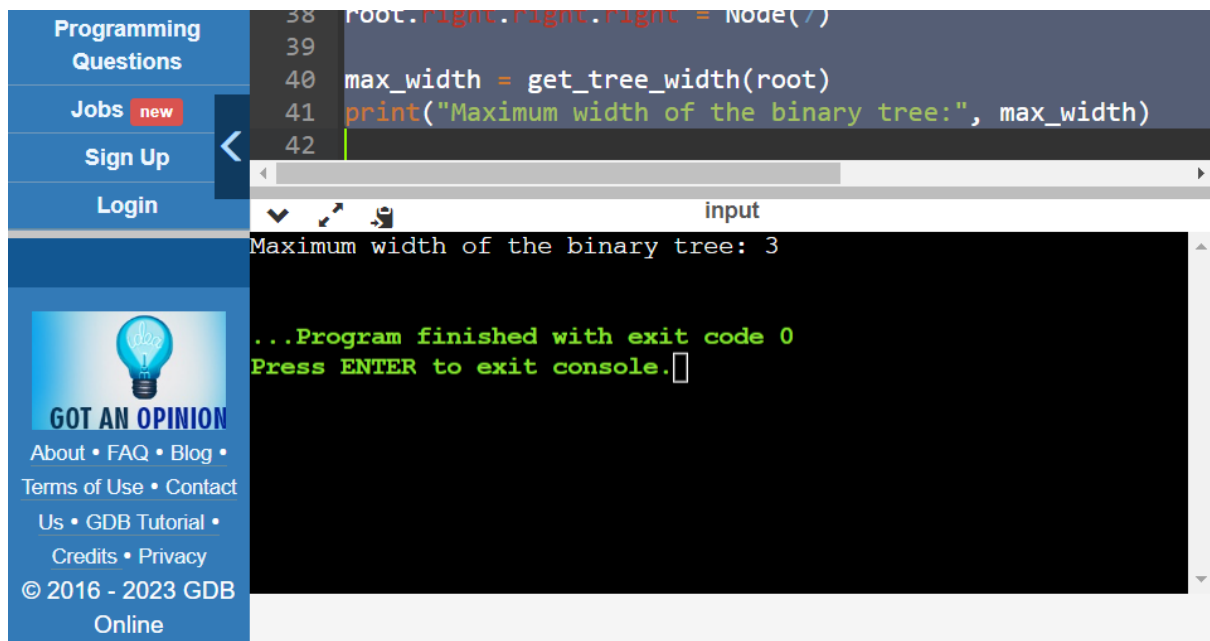
```
root.right.right = Node(8)
```

```
root.right.right.left = Node(6)
```

```
root.right.right.right = Node(7)
```

```
max_width = get_tree_width(root)
```

```
print("Maximum width of the binary tree:", max_width)
```



The screenshot shows a web application interface. On the left is a blue sidebar with navigation links: 'Programming Questions', 'Jobs' (with a 'new' badge), 'Sign Up', and 'Login'. Below these is a section titled 'GOT AN OPINION' with a lightbulb icon, followed by links for 'About', 'FAQ', 'Blog', 'Terms of Use', 'Contact Us', 'GDB Tutorial', 'Credits', 'Privacy', and a copyright notice '© 2016 - 2023 GDB Online'. The main area displays a code editor with Python code for a binary tree. The code includes comments in red and function calls in green. The output of the program is shown in a black console window, displaying the maximum width of the binary tree as 3. The console also shows the program finished with exit code 0 and a prompt to press ENTER to exit.

```
38 root.right.right.right = Node(7)
39
40 max_width = get_tree_width(root)
41 print("Maximum width of the binary tree:", max_width)
42
```

input

Maximum width of the binary tree: 3

...Program finished with exit code 0
Press ENTER to exit console.