

SOURCE CODE: PART A

```
1 from BinaryTree import BinaryTree
2 class LinkedBinaryTree(BinaryTree):
3     '''Linked representation of a binary tree structure.'''
4     class Node: #Lightweight, non public class for storing a node
5         __slots__ = '_element', '_parent', '_left', '_right'
6         def __init__(self, element, parent=None, left=None, right=None):
7             self._element = element
8             self._parent = parent
9             self._left = left
10            self._right = right
11
12    class Position(BinaryTree.Position):
13        '''An abstraction representing the location of a single element.'''
14
15        def __init__(self, container, node):
16            '''Constructor should not be invoked by the user.'''
17            self._container = container
18            self._node = node
19
20        @property
21        def element(self):
22            '''Return the element stored at this Position'''
23            return self._node._element
24
25        def __eq__(self, other):
26            '''Return True if other is a Position representing the same location.'''
27            return type(other) is type(self) and other._node is self._node
28
29    def _validate(self, p):
30        '''Return position's node or raise appropriate error if invalid'''
31        if not isinstance(p, self.Position):
32            raise TypeError('p must be proper Position type')
33        if p._container is not self:
34            raise ValueError('p does not belong to this container')
35        if p._node._parent is p._node: #convention for deprecated nodes
36            raise ValueError('p is no longer valid')
37        return p._node
38
39    def _make_position(self, node):
40        '''Return Position instance for given node (or None if sentinel).'''
41        return self.Position(self, node) if node is not None else None
42
43    #---binary tree constructor---
44    def __init__(self):
45        '''Create an empty binary tree.'''
46        self._root = None
47        self._size = 0
48
49    #---public accessors---
50    def __len__(self):
51        '''Return the total number of elements in the tree.'''
52        return self._size
53
54    def root(self):
55        '''Return the root Position of the tree (or None if tree is empty)'''
56        return self._make_position(self._root)
57
58    def parent(self, p):
59        '''Return the Position of p's parent (or None if p is root)'''
60        node = self._validate(p)
61        return self._make_position(node._parent)
62
63    def left(self, p):
64        '''Return the Position of p's left child (or None if p has no left child)'''
65        node = self._validate(p)
66        return self._make_position(node._left)
67
68    def right(self, p):
69        '''Return the Position of p's right child (or None if p has no right child)'''
70        node = self._validate(p)
71        return self._make_position(node._right)
72
73    def num_children(self, p):
74        '''Return the number of children of Position p.'''
75        node = self._validate(p)
76        return len([self.left(p), self.right(p)])
```

```
38 def _make_position(self, node):
39     '''Return Position instance for given node (or None if sentinel).'''
40     return self.Position(self, node) if node is not None else None
41
42 #---binary tree constructor---
43 def __init__(self):
44     '''Create an empty binary tree.'''
45     self._root = None
46     self._size = 0
47
48 #---public accessors---
49 def __len__(self):
50     '''Return the total number of elements in the tree.'''
51     return self._size
52
53 def root(self):
54     '''Return the root Position of the tree (or None if tree is empty)'''
55     return self._make_position(self._root)
56
57 def parent(self, p):
58     '''Return the Position of p's parent (or None if p is root)'''
59     node = self._validate(p)
60     return self._make_position(node._parent)
61
62 def left(self, p):
63     '''Return the Position of p's left child (or None if p has no left child)'''
64     node = self._validate(p)
65     return self._make_position(node._left)
66
67 def right(self, p):
68     '''Return the Position of p's right child (or None if p has no right child)'''
69     node = self._validate(p)
70     return self._make_position(node._right)
71
72 def num_children(self, p):
73     '''Return the number of children of Position p.'''
74     node = self._validate(p)
75     return len([self.left(p), self.right(p)])
```

```
DSALG01-IDB2 main
Current File
vck_Deque.py BinaryTree.py main.py Term Project#2 Part A.py Term Project#2 Part B.py LinkedBinaryTree.py x Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
72 """Return the number of children of Position p."""
73 node = self._validate(p)
74 count = 0
75 if node._left is not None: #left child exists
76     count += 1
77 if node._right is not None: #right child exists
78     count += 1
79 return count
80
11 usages
81 def _add_root(self, e):
82     """Place element e at the root of an empty tree and return new Position."""
83     """Raise ValueError if tree nonempty."""
84     if self._root is not None:
85         raise ValueError('Root exists')
86     self._size = 1
87     self._root = self.Node(e)
88     return self._make_position(self._root)
89
51 usages
90 def _add_left(self, p, e):
91     """Create a new left child for Position p, storing element e."""
92
93     """Return the position of new node.
94     Raise ValueError if Position p is invalid or p already has a left child"""
95     node = self._validate(p)
96     if node._left is not None:
97         raise ValueError('Left child exists')
98     self._size += 1
99     node._left = self.Node(e, node) #node is its parent
100     return self._make_position(node._left)
101
55 usages
102 def _add_right(self, p, e):
103     """Create a new right child for Position p, storing element e."""
104
105     """Return the Position of new node
106     Raise ValueError if Position p is invalid or p already has a right child"""
107     node = self._validate(p)
108     if node._right is not None:
109         raise ValueError('Right child exists')
110
LinkedBinaryTree > Node > __init__
DSALG01-IDB2 > FINALS > Trees_Activities > LinkedBinaryTree.py 8:34 LF UTF-8 4 spaces Python 3.12
Type here to search 3:46 pm 06/12/2024
```

```
DSALG01-IDB2 main
Current File
vck_Deque.py BinaryTree.py main.py Term Project#2 Part A.py Term Project#2 Part B.py LinkedBinaryTree.py x Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
102 def _add_right(self, p, e):
103     """Create a new right child for Position p, storing element e."""
104
105     """Return the Position of new node
106     Raise ValueError if Position p is invalid or p already has a right child"""
107     node = self._validate(p)
108     if node._right is not None:
109         raise ValueError('Right child exists')
110     self._size += 1
111     node._right = self.Node(e, node) #node is its parent
112     return self._make_position(node._right)
113
114 def _replace(self, p, e):
115     """Replace the element at position p with e, and return old element."""
116     node = self._validate(p)
117     old = node._element
118     node._element = e
119     return old
120
121 def _delete(self, p):
122     """Delete the node at Position p, and replace it with its child, if any."""
123     """Return the element that had been stored at Position p."""
124     """Raise ValueError if Position p is invalid or p has two children"""
125     node = self._validate(p)
126     if self._num_children(p) == 2:
127         raise ValueError('Position has two children')
128     child = node._left if node._left else node._right # might be None
129     if child is not None:
130         child._parent = node._parent #child's grandparent becomes parent
131     if node is self._root:
132         self._root = child # child becomes root
133     else:
134         parent = node._parent
135         if node is parent._left:
136             parent._left = child
137         else:
138             parent._right = child
139     self._size -= 1
140     node._parent = None # convention for deprecated node
141     return node._element
142
LinkedBinaryTree > Node > __init__
DSALG01-IDB2 > FINALS > Trees_Activities > LinkedBinaryTree.py 8:34 LF UTF-8 4 spaces Python 3.12
Type here to search 3:46 pm 06/12/2024
```

```
DSALGO1-IDB2 main
Current File
vck_Deque.py BinaryTree.py main.py Term Project#2 Part A.py Term Project#2 Part B.py LinkedBinaryTree.py x Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
125 node = self._validate(p)
126 if self.num_children(p) == 2:
127     raise ValueError('Position has two children')
128 child = node._left if node._left else node._right # might be None
129 if child is not None:
130     child._parent = node # child's grandparent becomes parent
131 if node is self._root:
132     self._root = child # child becomes root
133 else:
134     parent = node._parent
135     if node is parent._left:
136         parent._left = child
137     else:
138         parent._right = child
139 self._size += 1
140 node._parent = node # convention for deprecated node
141 return node._element
142
143 def _attach(self, p, t1, t2):
144     '''Attach tree t1 and t2 as left and right subtrees of external p.'''
145     node = self._validate(p)
146     if not self.is_leaf(p): raise ValueError('position must be leaf')
147     if not type(self) is type(t1) is type(t2): # all 3 trees must be same type
148         raise TypeError('Tree types must match')
149     self._size += len(t1) + len(t2)
150     if not t1.is_empty(): # attached t1 as left subtree of node
151         t1._root._parent = node
152         node._left = t1._root
153         t1._root = None
154         t1._size = 0
155     if not t2.is_empty(): # attached t2 as right subtree of node
156         t2._root._parent = node
157         node._right = t2._root
158         t2._root = None
159         t2._size = 0
160
LinkedBinaryTree Node _init_()
DSALGO1-IDB2 > FINALS > Trees_Activities > LinkedBinaryTree.py
8:34 LF UTF-8 4 spaces Python 3.12
Type here to search
```

```
DSALGO1-IDB2 main
Current File
vck_Deque.py BinaryTree.py main.py Term Project#2 Part A.py x Term Project#2 Part B.py LinkedBinaryTree.py Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
1 from LinkedBinaryTree import LinkedBinaryTree
2
3
4 # Create a tree for Equation 1
5 tree1 = LinkedBinaryTree()
6 root = tree1._add_root('-')
7 left = tree1._add_left(root, '@ *')
8 tree1._add_left(left, @ 3)
9 tree1._add_right(left, @ 5)
10 right = tree1._add_right(root, @ '+' )
11 left_right = tree1._add_left(right, @ '*' )
12 tree1._add_left(left_right, @ 4)
13 tree1._add_right(left_right, @ 5)
14 right_right = tree1._add_right(right, @ '-' )
15 tree1._add_left(right_right, @ 6)
16 tree1._add_right(right_right, @ 7)
17 print("Traversals for Tree 1")
18 #use the in order traversal to print the tree
19 print("Inorder traversal: ", end = " ")
20 for i in tree1.inorder():
21     print(i.element(), end = " ")
22 print()
23 #use the pre order traversal to print the tree
24 print("Preorder traversal: ", end = " ")
25 for i in tree1.positions():
26     print(i.element(), end = " ")
27 print()
28 #use the post order traversal to print the tree
29 print("Postorder traversal: ", end = " ")
30 for i in tree1.postorder():
31     print(i.element(), end = " ")
32 print()
33
34 # Create a tree for Equation 2
35 tree2 = LinkedBinaryTree()
36 root2 = tree2._add_root('-')
37 left2 = tree2._add_left(root2, @ '*' )
38 left_left2 = tree2._add_left(left2, @ '+' )
39 tree2._add_left(left_left2, @ 'a')
40 tree2._add_right(left_left2, @ 'b')
41 tree2._add_right(left2, @ 'c')
42
DSALGO1-IDB2 > FINALS > Trees_Activities > Term Project#2 Part A.py
10:36 CRLF UTF-8 4 spaces Python 3.12
Type here to search
```

```
DSALGO1-IDB2 main
tree2._add_right(left2.left2, @'b')
tree2._add_right(left2, @'c')
right2 = tree2._add_right(right2, @'--')
tree2._add_left(right2, @'d')
tree2._add_right(right2, @'e')
print("Traversals for Tree 2")
#use the in order traversal to print the tree
print("Inorder traversal: ", end=" ")
for i in tree2.inorder():
    print(i.element(), end=" ")
print()
#use the pre order traversal to print the tree
print("Preorder traversal: ", end=" ")
for i in tree2.positions():
    print(i.element(), end=" ")
print()
#use the post order traversal to print the tree
print("Postorder traversal: ", end=" ")
for i in tree2.postorder():
    print(i.element(), end=" ")
print()
# Create a tree for Equation 3
tree3 = LinkedBinaryTree()
root3 = tree3._add_root('/')
left3 = tree3._add_left(root3, @'+')
left_left3 = tree3._add_left(left3, @'A')
tree3._add_left(left_left3, @'a')
tree3._add_right(left_left3, @'b')
right_left3 = tree3._add_right(left3, @'+')
tree3._add_left(right_left3, @'c')
tree3._add_right(right_left3, @'d')
right3 = tree3._add_right(root3, @'/')
left_right3 = tree3._add_left(right3, @'+')
tree3._add_left(left_right3, @'e')
tree3._add_right(left_right3, @'f')
right_right3 = tree3._add_right(right3, @'+')
tree3._add_left(right_right3, @'g')
tree3._add_right(right_right3, @'h')
print("Traversals for Tree 3")
```

```
DSALGO1-IDB2 main
tree3._add_right(right_right3, @'b')
print("Traversals for Tree 3")
#use the in order traversal to print the tree
print("Inorder traversal: ", end=" ")
for i in tree3.inorder():
    print(i.element(), end=" ")
print()
#use the pre order traversal to print the tree
print("Preorder traversal: ", end=" ")
for i in tree3.positions():
    print(i.element(), end=" ")
print()
#use the post order traversal to print the tree
print("Postorder traversal: ", end=" ")
for i in tree3.postorder():
    print(i.element(), end=" ")
print()
# Create a tree for Equation 4
tree4 = LinkedBinaryTree()
root4 = tree4._add_root('/')
left4 = tree4._add_left(root4, @'+')
tree4._add_left(left4, @'a')
tree4._add_right(left4, @'b')
right4 = tree4._add_right(root4, @'+')
tree4._add_left(right4, @'c')
right_right4 = tree4._add_right(right4, @'--')
tree4._add_left(right_right4, @'d')
right_right_right4 = tree4._add_right(right_right4, @'A')
tree4._add_left(right_right_right4, @'e')
tree4._add_right(right_right_right4, @'f')
print("Traversals for Tree 4")
#use the in order traversal to print the tree
print("Inorder traversal: ", end=" ")
for i in tree4.inorder():
    print(i.element(), end=" ")
print()
#use the pre order traversal to print the tree
print("Preorder traversal: ", end=" ")
for i in tree4.positions():
```



```
DSALG01-IDB2 main
sckDeque.py BinaryTree.py main.py Term Project#2 Part A.py x Term Project#2 Part B.py LinkedBinaryTree.py Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
175 tree6._add_left(num_left, @ '+' )
176 tree6._add_right(num_left, @ '-' )
177 tree6._add_left(tree6.left(num_left), @ 5 )
178 tree6._add_right(tree6.left(num_left), @ 2 )
179 tree6._add_left(tree6.right(num_left), @ 2 )
180 tree6._add_right(tree6.right(num_left), @ 1 )
181
182 # Subtree for the denominator: (2 + 9) + ((7 - 2) - 1)
183 den_right = tree6._add_right(left_sub, @ '+' )
184 tree6._add_left(den_right, @ '+' )
185 tree6._add_right(den_right, @ '-' )
186 tree6._add_left(tree6.left(den_right), @ 2 )
187 tree6._add_right(tree6.left(den_right), @ 9 )
188 tree6._add_left(tree6.right(den_right), @ '-' )
189 tree6._add_right(tree6.right(den_right), @ 1 )
190 tree6._add_left(tree6.left(tree6.right(den_right)), @ 7 )
191 tree6._add_right(tree6.left(tree6.right(den_right)), @ 2 )
192
193
194 print("Traversals for Tree 6")
195 #use the in order traversal to print the tree
196 print("Inorder traversal: ", end=" ")
197 for i in tree6.inorder():
198     print(i.element(), end=" ")
199 print()
200 #use the pre_order traversal to print the tree
201 print("Preorder traversal: ", end=" ")
202 for i in tree6.positions():
203     print(i.element(), end=" ")
204 print()
205 #use the post_order traversal to print the tree
206 print("Postorder traversal: ", end=" ")
207 for i in tree6.postorder():
208     print(i.element(), end=" ")
209 print()
210
DSALG01-IDB2 > FINALS > Trees_Activities > Term Project#2 Part A.py 210:1 CRLF UTF-8 4 spaces Python 3.12
Type here to search 3:47 pm 06/12/2024
```

SOURCE CODE PART B:

```
DSALG01-IDB2 main
TeamProject#1 (2 Stack_Deque).py BinaryTree.py main.py Term Project#2 Part B.py x LinkedBinaryTree.py Tree.py TermProject#1 (Stack_Queue_Deque).py LinkedStack.py LinkedQueue.py
1 from LinkedBinaryTree import LinkedBinaryTree
2
3 tree1 = LinkedBinaryTree()
4 r = tree1._add_root('r')
5 a = tree1._add_left(r, @ 'a')
6 b = tree1._add_left(a, @ 'b')
7 d = tree1._add_right(b, @ 'd')
8 c = tree1._add_right(a, @ 'c')
9 e = tree1._add_left(c, @ 'e')
10 g = tree1._add_right(e, @ 'g')
11 h = tree1._add_right(g, @ 'h')
12 f = tree1._add_right(c, @ 'f')
13 print("Traversals for Tree 1")
14 #use the in order traversal to print the tree
15 print("Inorder traversal: ", end=" ")
16 for i in tree1.inorder():
17     print(i.element(), end=" ")
18 print()
19 #use the pre_order traversal to print the tree
20 print("Preorder traversal: ", end=" ")
21 for i in tree1.positions():
22     print(i.element(), end=" ")
23 print()
24 #use the post_order traversal to print the tree
25 print("Postorder traversal: ", end=" ")
26 for i in tree1.postorder():
27     print(i.element(), end=" ")
28 print()
29
30 # Create the tree for the second matrix
31 tree2 = LinkedBinaryTree()
32 r = tree2._add_root('r')
33 a = tree2._add_left(r, @ 'a')
34 b = tree2._add_right(r, @ 'b')
35 c = tree2._add_left(a, @ 'c')
36 d = tree2._add_right(a, @ 'd')
37 e = tree2._add_right(b, @ 'e')
38 f = tree2._add_right(e, @ 'f')
39 g = tree2._add_right(f, @ 'g')
40 print("Traversals for Tree 2")
41 #use the in order traversal to print the tree
DSALG01-IDB2 > FINALS > Trees_Activities > Term Project#2 Part B.py 94:26 CRLF UTF-8 4 spaces Python 3.12
Type here to search 3:48 pm 06/12/2024
```

The image shows a screenshot of a Visual Studio Code editor window. The top bar displays the file name 'DSALGO1-IDB2' and the current file 'main.py'. The editor is open to a file named 'Term Project#2 Part B.py', which contains Python code for binary tree traversals. The code is organized into three sections, each starting with a comment indicating the traversal type. The first section implements Inorder, Preorder, and Postorder traversals for a tree named 'tree2'. The second section implements the same three traversals for a tree named 'tree3'. The third section creates a new tree 'tree3' by adding nodes 'a' through 'f' and then prints the traversals for 'tree3'. The code uses a 'LinkedListBinaryTree' class and its methods 'add_root', 'add_left', and 'add_right'. The bottom status bar shows the file path 'DSALGO1-IDB2 > FINALS > Trees_Activities > Term Project#2 Part B.py' and various settings like '94:26', 'CLRF', 'UTF-8', '4 spaces', and 'Python 3.12'.

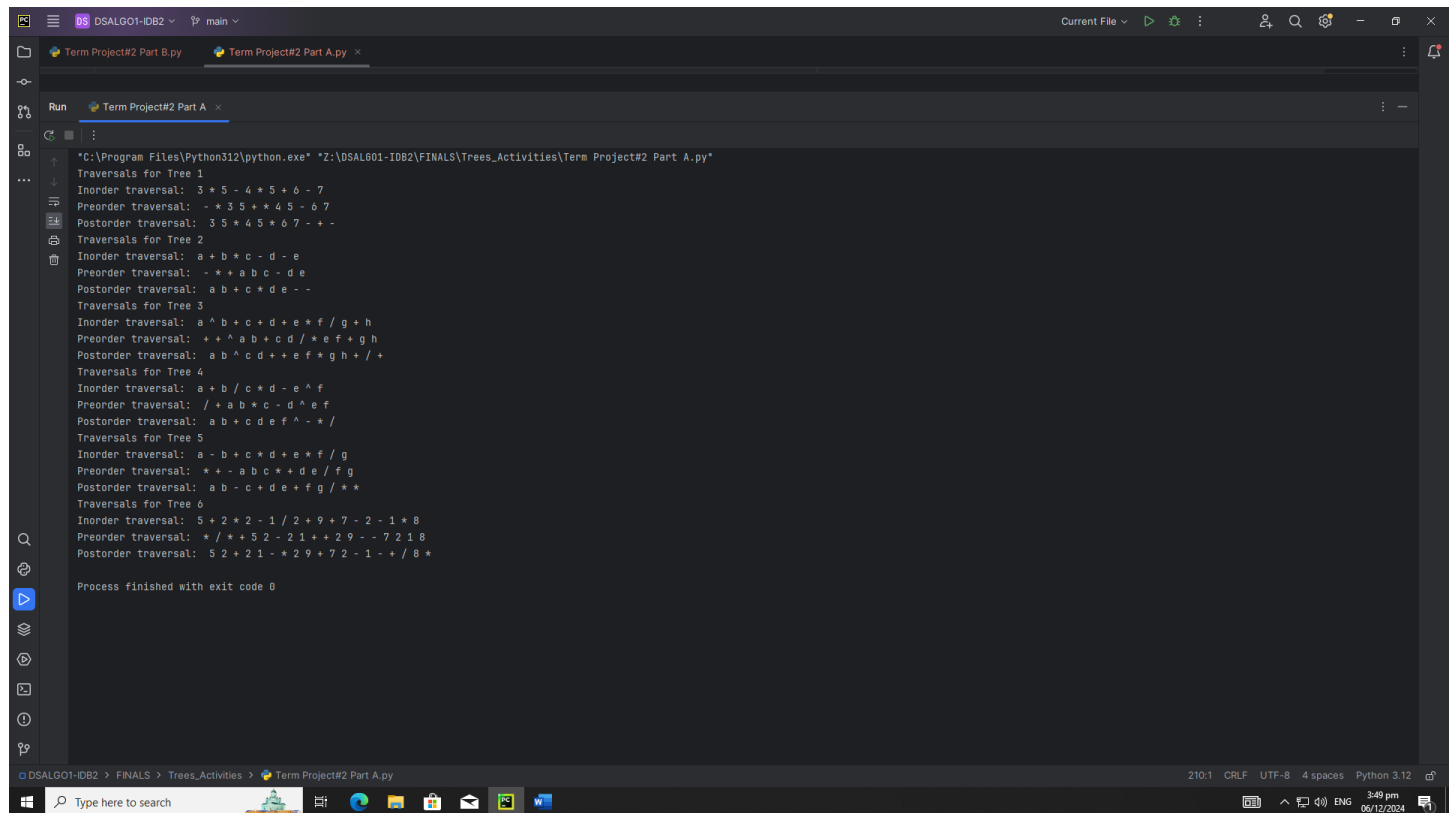
```
40 print("\nTraversals for Tree 2")
41 #Use the in order traversal to print the tree
42 print("\nInorder traversal: ", end=" ")
43 for i in tree2.inorder():
44     print(i.element(), end=" ")
45 print()
46 #Use the pre order traversal to print the tree
47 print("\nPreorder traversal: ", end=" ")
48 for i in tree2.positions():
49     print(i.element(), end=" ")
50 print()
51 #Use the post order traversal to print the tree
52 print("\nPostorder traversal: ", end=" ")
53 for i in tree2.postorder():
54     print(i.element(), end=" ")
55 print()
56
57 # Create the tree for the third matrix
58 tree3 = LinkedListBinaryTree()
59 r = tree3._add_root('r')
60 a = tree3._add_left(r, 'a')
61 b = tree3._add_right(r, 'b')
62 c = tree3._add_right(a, 'c')
63 d = tree3._add_left(b, 'd')
64 e = tree3._add_right(b, 'e')
65 f = tree3._add_left(c, 'f')
66 print("\nTraversals for Tree 3")
67 #Use the in order traversal to print the tree
68 print("\nInorder traversal: ", end=" ")
69 for i in tree3.inorder():
70     print(i.element(), end=" ")
71 print()
72 #Use the pre order traversal to print the tree
73 print("\nPreorder traversal: ", end=" ")
74 for i in tree3.positions():
75     print(i.element(), end=" ")
76 print()
77 #Use the post order traversal to print the tree
78 print("\nPostorder traversal: ", end=" ")
79 for i in tree3.postorder():
80     print(i.element(), end=" ")
```

The image shows a VS Code editor window with a Python script for a binary tree. The script defines a `BinaryTreeNode` class and a `BinaryTreeNode` class, then creates a tree structure and prints its postorder, inorder, and preorder traversals.

```
78 print("Postorder traversal: ", end=" ")
79 for i in tree3.postorder():
80     print(i.element(), end=" ")
81 print()
82
83 # Create the tree for the fourth matrix
84 tree4 = LinkedBinaryTree()
85 r = tree4._add_root('r')
86 a = tree4._add_left(r, 'a')
87 b = tree4._add_right(r, 'b')
88 c = tree4._add_left(a, 'c')
89 d = tree4._add_right(a, 'd')
90 e = tree4._add_left(b, 'e')
91 f = tree4._add_right(b, 'f')
92 g = tree4._add_left(c, 'g')
93 h = tree4._add_right(c, 'h')
94 i = tree4._add_left(e, 'i')
95 print("Traversals for Tree 4")
96 #use the in order traversal to print the tree
97 print("Inorder traversal: ", end=" ")
98 for i in tree4.inorder():
99     print(i.element(), end=" ")
100 print()
101 #use the pre order traversal to print the tree
102 print("Preorder traversal: ", end=" ")
103 for i in tree4.positions():
104     print(i.element(), end=" ")
105 print()
106 #use the post order traversal to print the tree
107 print("Postorder traversal: ", end=" ")
108 for i in tree4.postorder():
109     print(i.element(), end=" ")
110 print()
111
112
113
```

OUTPUT:

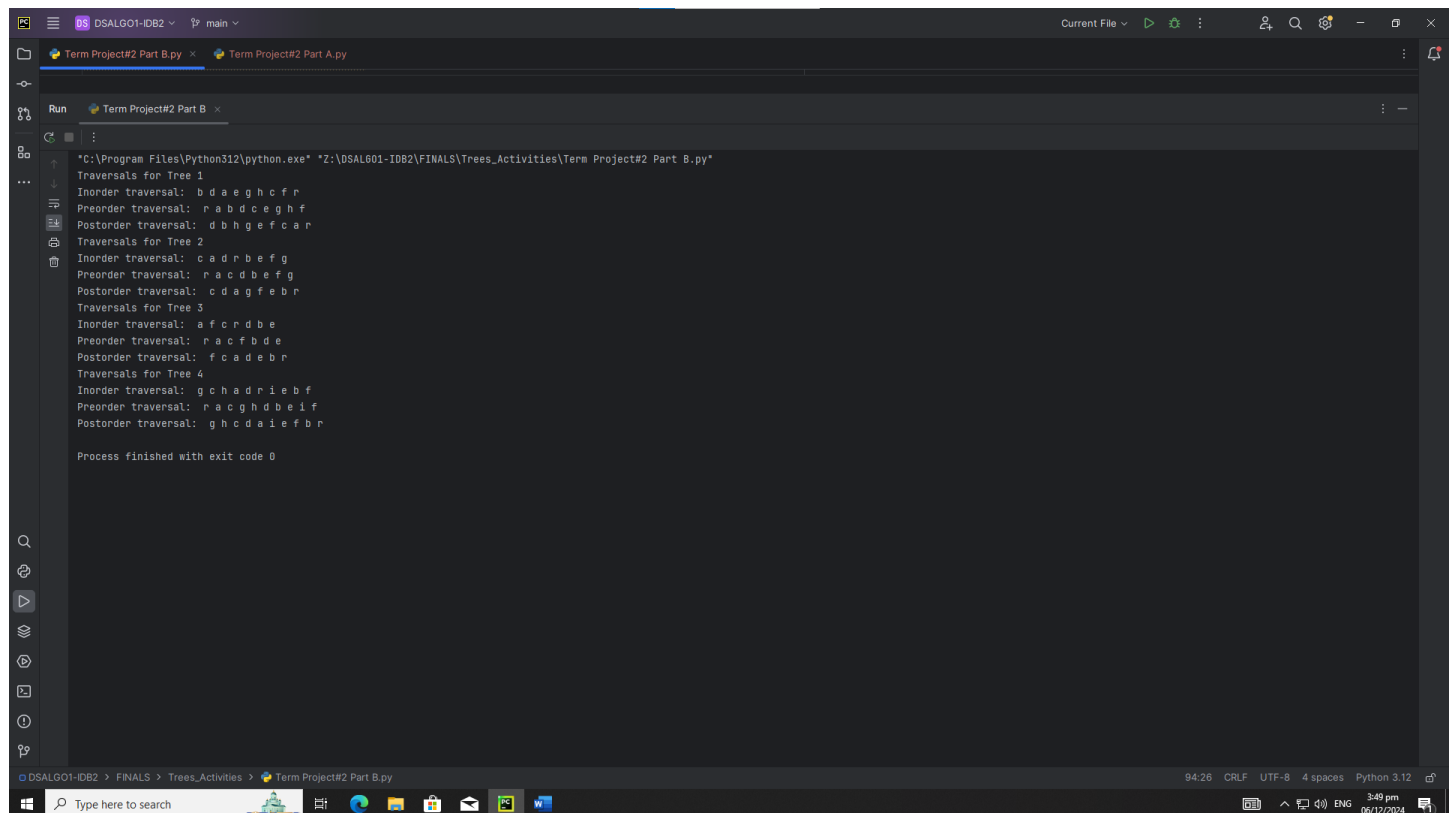
TERM PROJECT PART A



```
"C:\Program Files\Python312\python.exe" "Z:\DSALG01-IDB2\FINALS\Trees_Activities\Term Project#2 Part A.py"
Traversals for Tree 1
Inorder traversal: 3 * 5 - 4 * 5 + 6 - 7
Preorder traversal: - * 3 5 + * 4 5 - 6 7
Postorder traversal: 3 5 * 4 5 * 6 7 - + -
Traversals for Tree 2
Inorder traversal: a + b * c - d - e
Preorder traversal: - * + a b c - d e
Postorder traversal: a b + c * d e - -
Traversals for Tree 3
Inorder traversal: a ^ b + c + d + e * f / g + h
Preorder traversal: + + ^ a b + c d / + e f + g h
Postorder traversal: a b ^ c d + e f g h + / +
Traversals for Tree 4
Inorder traversal: a + b / c * d - e ^ f
Preorder traversal: / + a b * c - d ^ e f
Postorder traversal: a b + c d e f ^ - * /
Traversals for Tree 5
Inorder traversal: a - b + c * d + e * f / g
Preorder traversal: * + - a b c + * d e / f g
Postorder traversal: a b - c + d e + f g / * *
Traversals for Tree 6
Inorder traversal: 5 + 2 * 2 - 1 / 2 + 9 + 7 - 2 - 1 * 8
Preorder traversal: * / * + 5 2 - 2 1 + + 2 9 - - 7 2 1 8
Postorder traversal: 5 2 + 2 1 - * 2 9 + 7 2 - 1 - + / 8 *

Process finished with exit code 0
```

PART B:



```
"C:\Program Files\Python312\python.exe" "Z:\DSALG01-IDB2\FINALS\Trees_Activities\Term Project#2 Part B.py"
Traversals for Tree 1
Inorder traversal: b d a e g h c f r
Preorder traversal: r a b d c e g h f
Postorder traversal: d b h g e f c a r
Traversals for Tree 2
Inorder traversal: c a d r b e f g
Preorder traversal: r a c d b e f g
Postorder traversal: c d a g f e b r
Traversals for Tree 3
Inorder traversal: a f c r d b e
Preorder traversal: r a c f b d e
Postorder traversal: f c a d e b r
Traversals for Tree 4
Inorder traversal: g c h a d r i e b f
Preorder traversal: r a c g h d b e i f
Postorder traversal: g h c d a i e f b r

Process finished with exit code 0
```