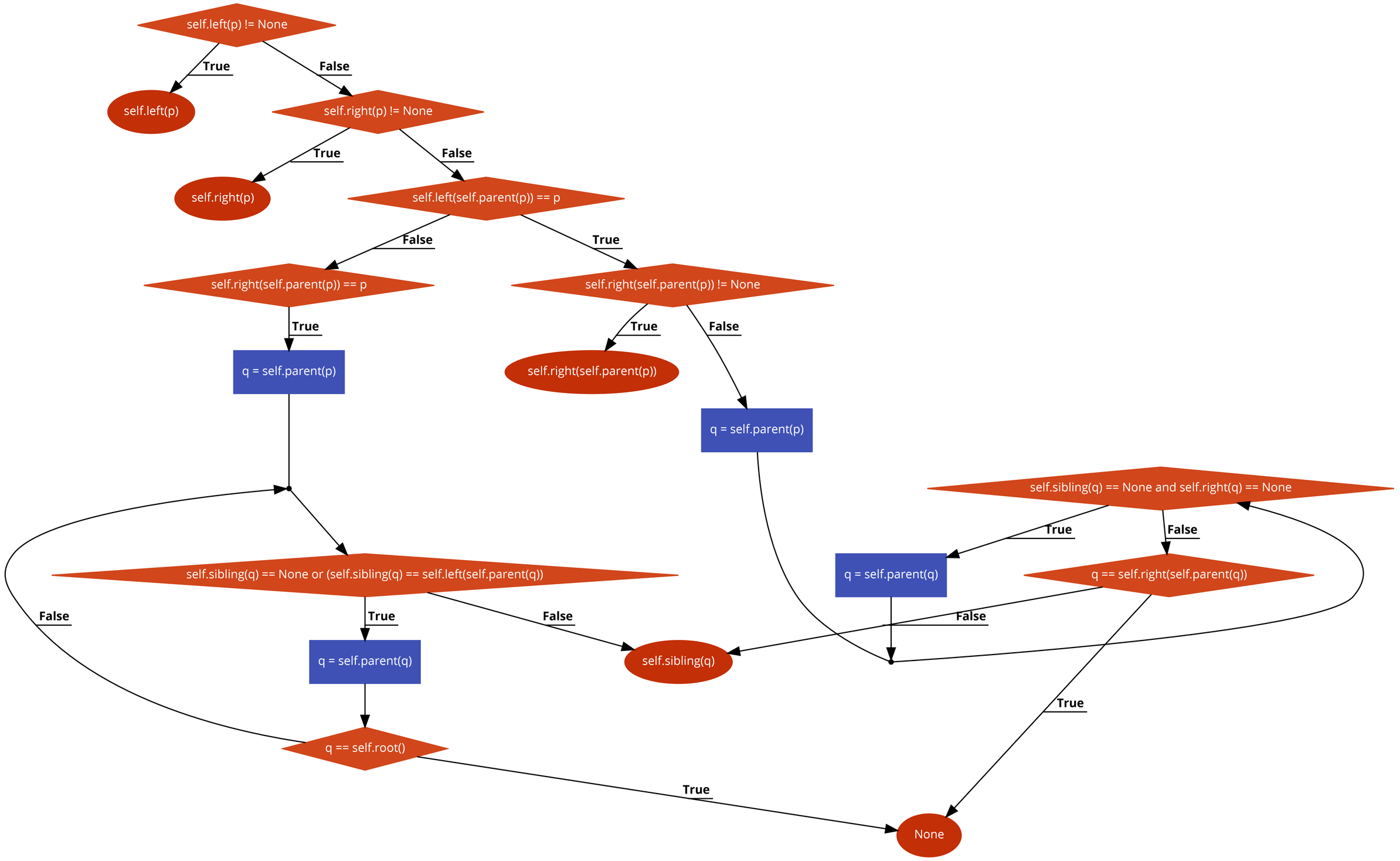
preorder_next
(self, p)



inorder_next
(self, p)

p == self.root()

True

d = self.right(p)

self.left(self.parent(p)) == p

True

self.num_children(p) == 0

True

self.left(d) != None

True

d = self.left(d)

False

d

False

self.right(p) != None

True

self.parent(p)

False

True

a = self.left(self.right(p))

False

self.right(p) != None

True

q = self.right(p)

self.left(q) != None

True

c = self.left(q)

False

q

False

b = self.parent(p)

a != None

False

self.right(p)

True

self.left(a) != None

False

a

True

a = self.left(a)

self.left(c) != None

True

c = self.left(c)

False

c

self.right(self.parent(b)) == b

False

b == self.root()

True

b = self.parent(b)

True

self.left(self.parent(b)) == b

False

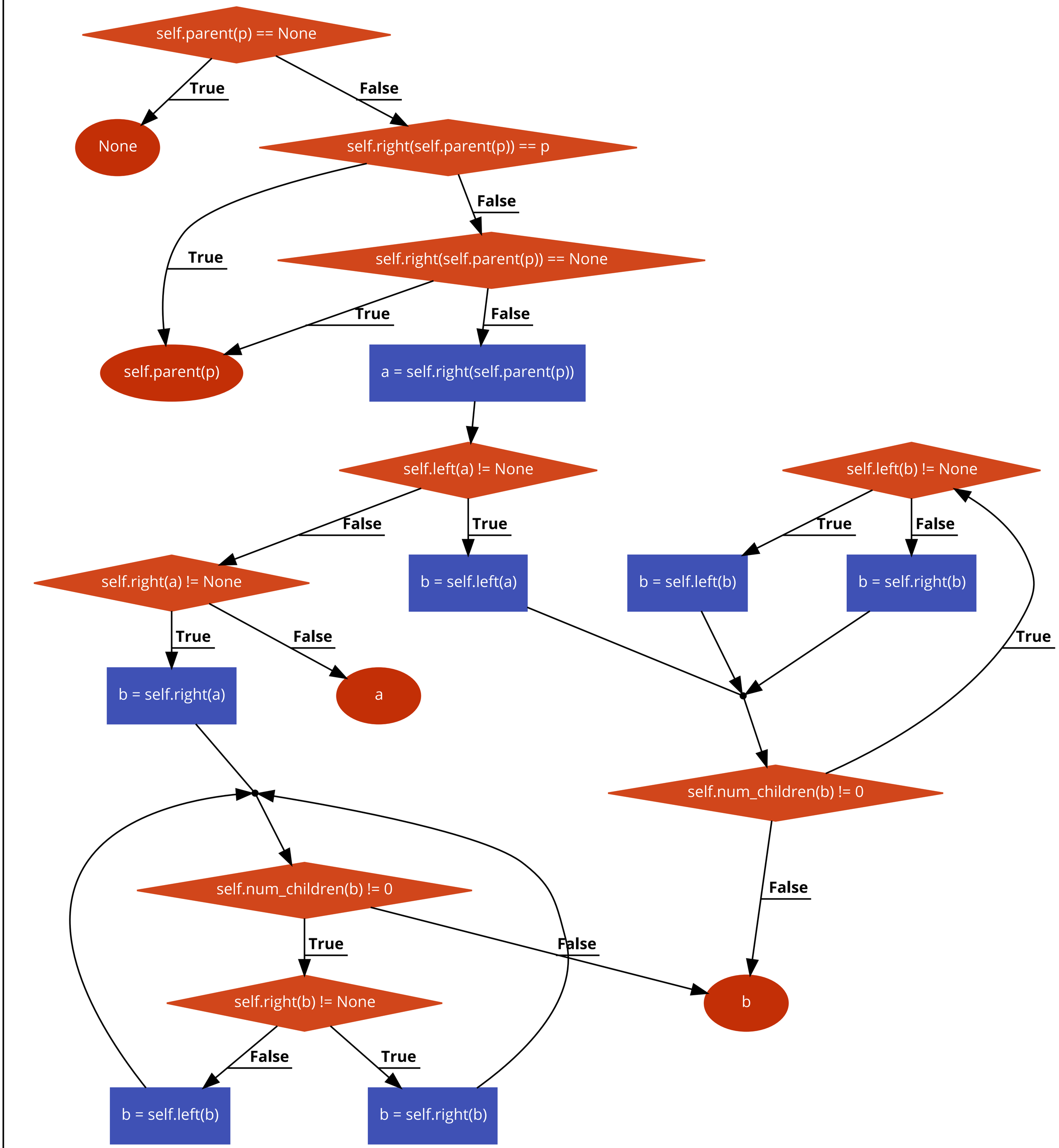
self.parent(b)

True

None

False

postorder_next
(self, p)



delete_subtree_recur
(self, p)

self.left(p) != None

True

a = self.left(p)

self.num_children(a) != 0

True

temp = self.left(p)

self.delete_subtree_recur(temp)

False

self._delete(a)

False

self.right(p) != None

True

a = self.right(p)

self.num_children(a) != 0

True

temp = self.right(p)

self.delete_subtree_recur(temp)

False

self._delete(a)

delete_subtree
(self, p)

self.delete_subtree_recur(p)

self._delete(p)

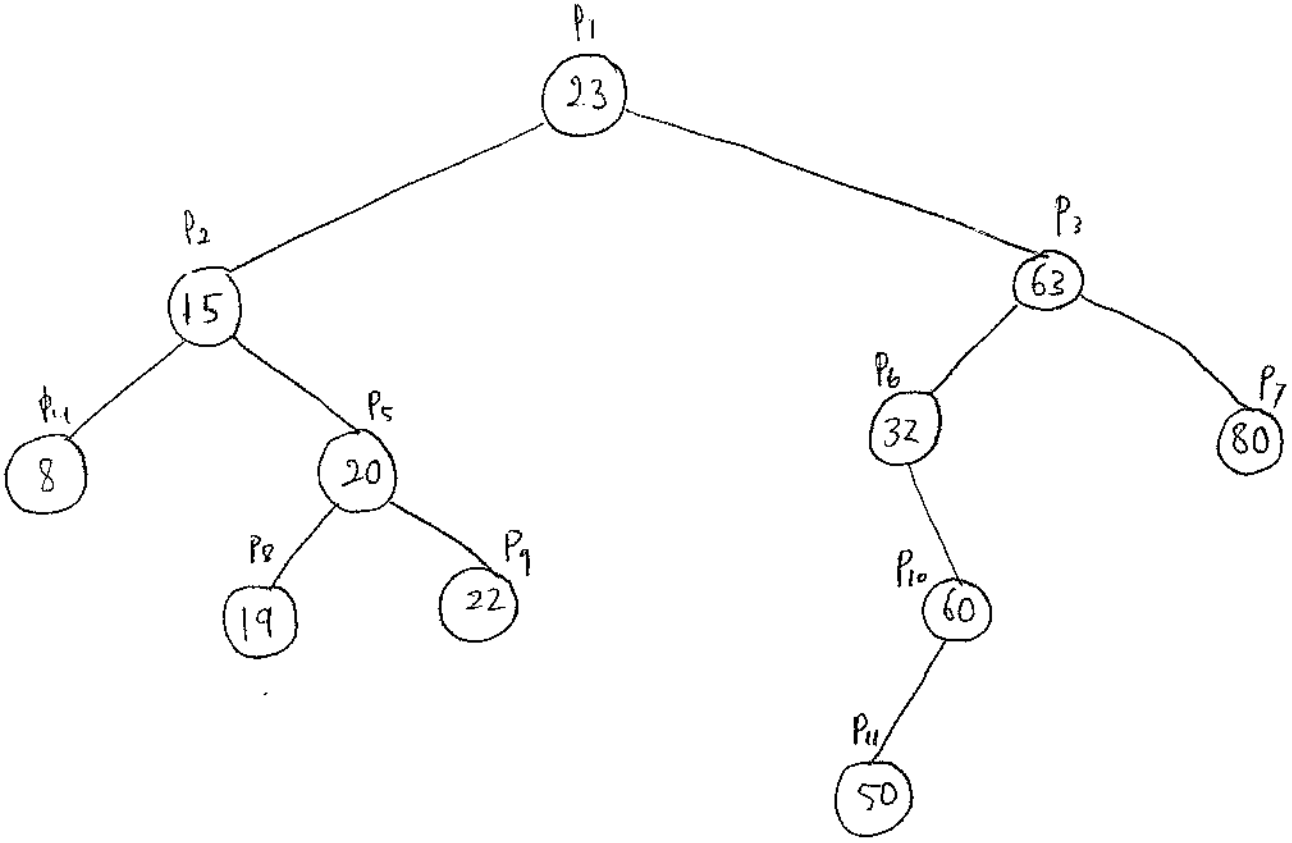


Figure 1

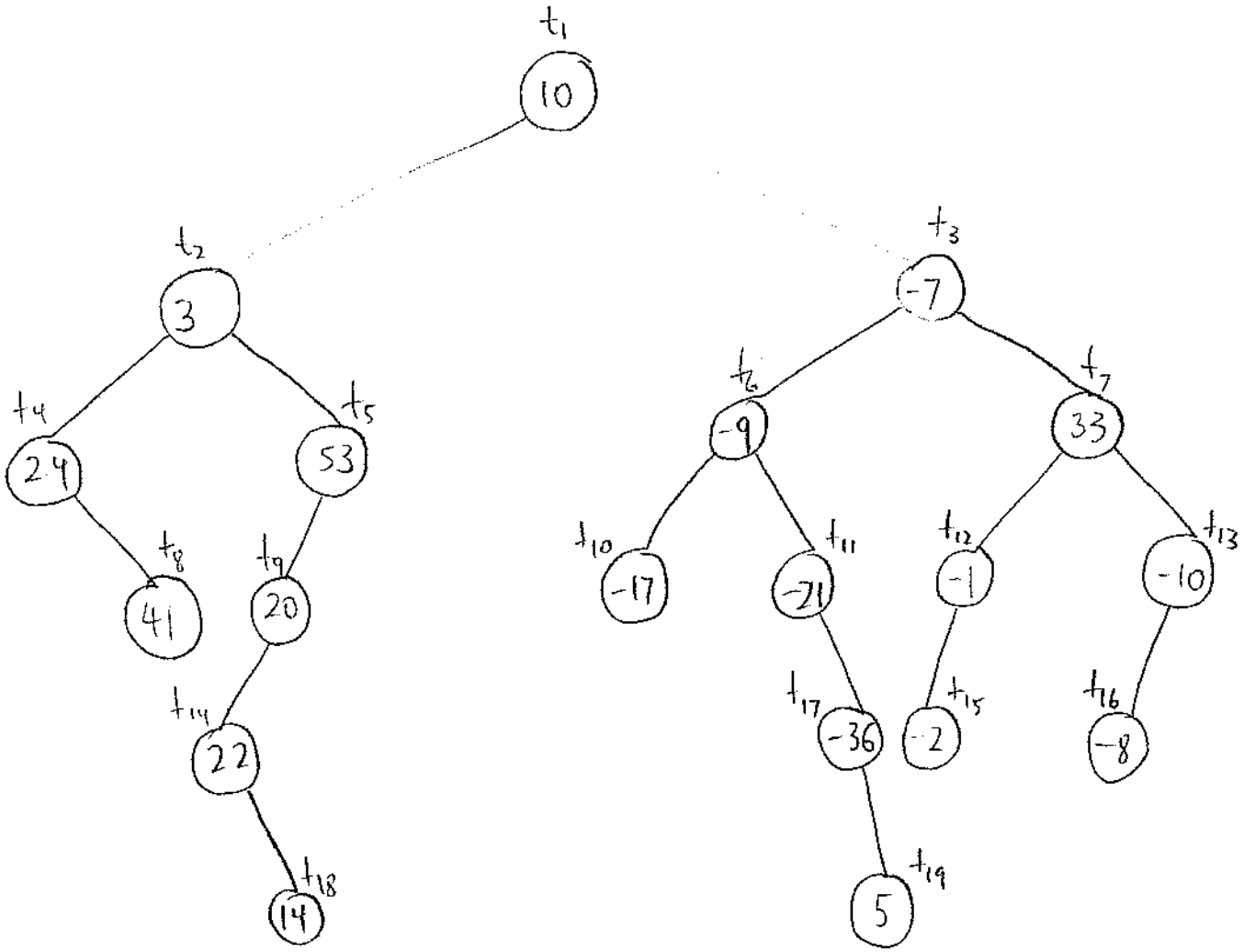

t_1
10
t_2
3
t_3
-7
t_4
24
t_5
53
t_6
-9
t_7
33
t_8
41
t_9
20
t_{10}
-17
t_{11}
-21
t_{12}
-1
t_{13}
-10
t_{14}
22
t_{15}
-2
t_{16}
-8
t_{17}
-36
t_{18}
14
t_{19}
5

Figure 2

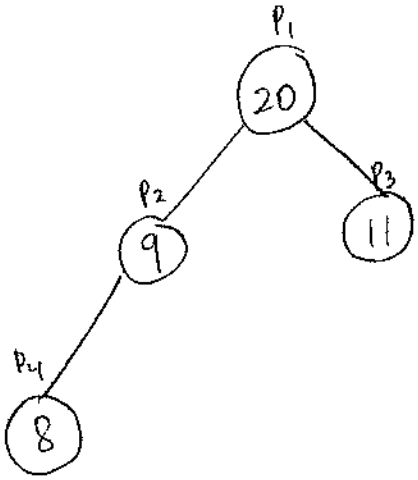


Figure 3.1

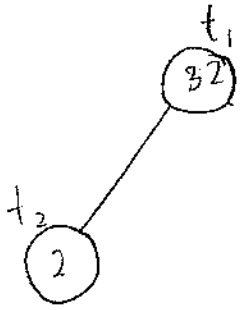


Figure 3.2

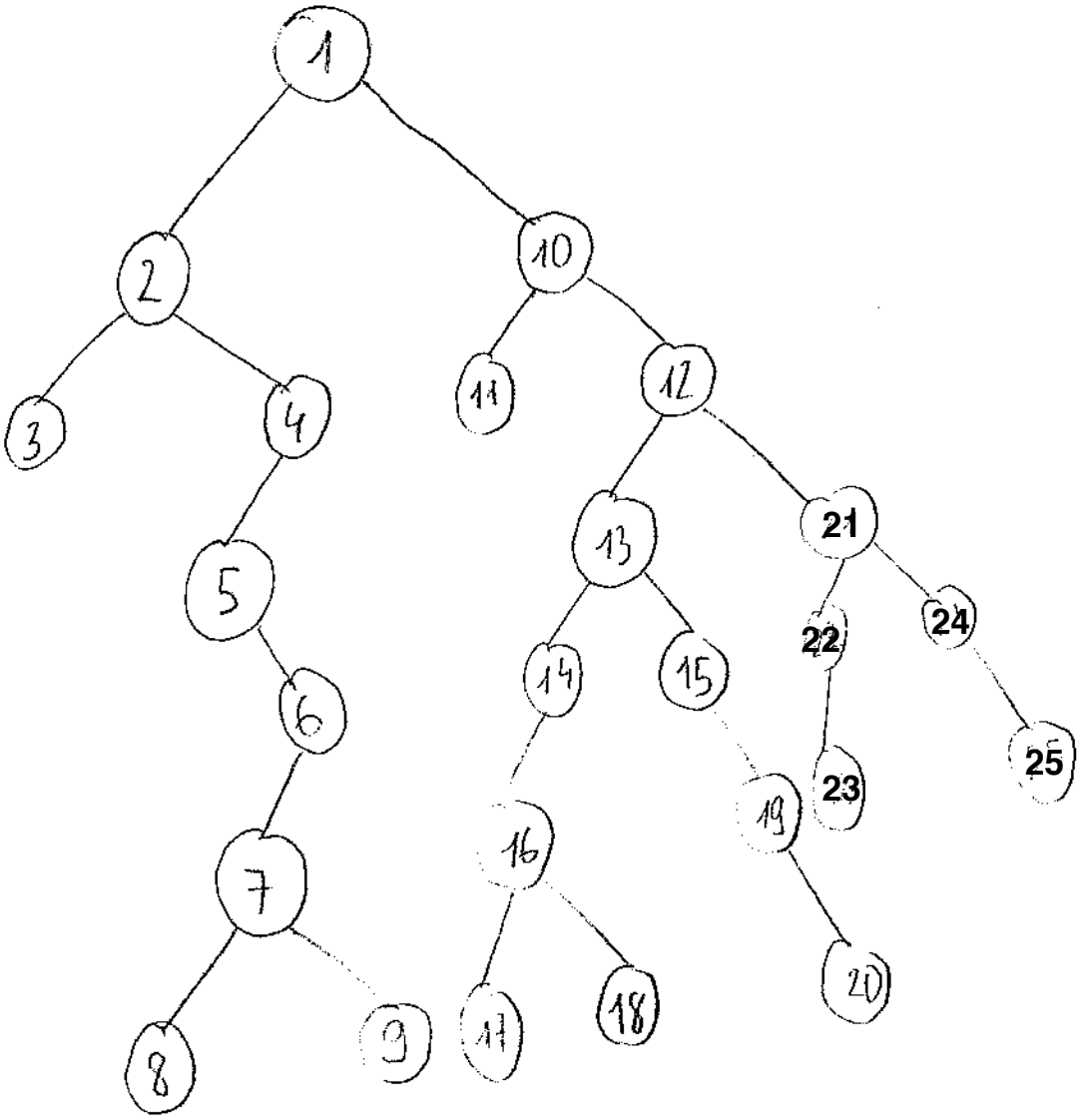


Figure 4

Test for linked binary tree in Figure 1				
Test Case #	Test Scenario	Test Steps	Expected Result	Pass/ Fail
1.1	Check preorder_next(p) with valid data (Figure 1)	1. Create an instance of ExtendedLinkedBinaryTree 2. Assign positions p1, p2, p3,…, p11 to distinct number elements as shown in Figure 1 3. Call preorder() and print the elements stored in the tree in preorder traversal 4. Call preorder_next(p) and print the elements stored in the tree, and compare them to the results of preorder()	Both preorder() and preorder_next(p) print: **23 15 85 20 19 22 63 32 60 50 80**	Pass
1.2	Check inorder_next(p) with valid data (Figure 1)	1. Call inorder() and print the elements stored in the tree in preorder traversal 2. Call inorder_next(p) and print the elements stored in the tree, and compare them to the results of inorder()	Both inorder() and inorder_next(p) print: **8 15 19 20 22 23 32 50 60 63 80**	Pass
1.3	Check postorder_next(p) with valid data (Figure 1)	1. Call postorder() and print the elements stored in the tree in preorder traversal 2. Call postorder_next(p) and print the elements stored in the tree, and compare them to the results of postorder()	Both postorder() and postorder_next(p) print: **8 19 22 20 15 50 60 32 80 63 23**	Pass
1.4	Check delete_subtree(p) with valid data (Figure 1)	1. Call delete_subtree(p2) 2. Call inorder() and print the elements of the remaining tree 3. Call delete_subtree(p2) again	**23 32 50 60 63 80** When delete_subtree(p2) is called again, a ValueError is raised	Pass
Test for linked binary tree in Figure 2				
Test Case #	Test Scenario	Test Steps	Expected Result	Pass/ Fail
2.1	Check preorder_next(p) with valid data (Figure 2)	1. Create an instance of ExtendedLinkedBinaryTree 2. Assign positions t1, t2, t3,…, t19 to distinct number elements as shown in Figure 2 3. Call preorder() and print the elements stored in the tree in preorder traversal 4. Call preorder_next(p) and print the elements stored in the tree, and compare them to the results of preorder()	Both preorder() and preorder_next(p) print: **10 3 24 53 20 22 14 -7 -9 -17 -21 -36 5 33 -1 -2 -10 -8**	Pass

2.2	Check inorder_next(p) with valid data (Figure 2)	1. Call inorder() and print the elements stored in the tree in preorder traversal 2. Call inorder_next(p) and print the elements stored in the tree, and compare them to the results of inorder()	Both inorder() and inorder_next(p) print: **24 41 3 22 14 20 53 10 -17 -9 -21 -36 5 -7 -2 -1 33 -8 -10**	Pass
2.3	Check postorder_next(p) with valid data (Figure 2)	1. Call postorder() and print the elements stored in the tree in preorder traversal 2. Call postorder_next(p) and print the elements stored in the tree, and compare them to the results of postorder()	Both postorder() and postorder_next(p) print: **41 24 14 22 20 53 3 -17 5 -36 -21 -9 -2 -1 -8 -10 33 -7 10**	Pass
2.4	Check delete_subtree(p) with valid data (Figure 2)	1. Call delete_subtree(t3) 2. Call inorder() and print the elements of the remaining tree 3. Call delete_subtree(t3) again	**24 41 3 22 14 20 53 10** When delete_subtree(t3) is called again, a ValueError is raised	Pass
Test for ValueError and TypeError raised (Figure 3)				
Test Case #	Test Scenario	Test Steps	Expected Result	Pass/ Fail
3.1	Check preorder_next(p) with invalid data type (Figure 3)	1. Create an instance of ExtendedLinkedBinaryTree 2. Assign positions p1, p2, p3, p4 to distinct number elements as shown in Figure 3.1 3. Create another instance of ExtendedLinkedBinaryTree 4. Assign positions t1, t2 to distinct number elements as shown in Figure 3.2 5. Call preorder_next(p) for linked binary tree in Figure 3.1 with p = 5	A TypeError is raised	Pass
3.2	Check preorder_next(p) with invalid data type (Figure 3)	Call preorder_next(p) for linked binary tree in Figure 3.1 with p = 'p3'	A TypeError is raised	Pass
3.3	Check preorder_next(p) with invalid data value (Figure 3)	Call preorder_next(p) for linked binary tree in Figure 3.2 with p = p2	Since position p3 does not belong in linked binary tree in Figure 4.2, a ValueError is raised	Pass
3.4	Check inorder_next(p) with invalid data type (Figure 3)	Call inorder_next(p) for linked binary tree in Figure 3.1 with p = 5	A TypeError is raised	Pass

3.5	Check inorder_next(p) with invalid data type (Figure 3)	Call inorder_next(p) for linked binary tree in Figure 3.1 with p = 5	A TypeError is raised	Pass
3.6	Check inorder_next(p) with invalid data value (Figure 3)	Call inorder_next(p) for linked binary tree in Figure 3.2 with p = p2	Since position p3 does not belong in linked binary tree in Figure 4.2, a ValueError is raised	Pass
3.7	Check postorder_next(p) with invalid data type (Figure 3)	Call postorder_next(p) for linked binary tree in Figure 3.1 with p = 5	A TypeError is raised	Pass
3.8	Check postorder_next(p) with invalid data type (Figure 3)	Call postorder_next(p) for linked binary tree in Figure 3.1 with p = 'p3'	A TypeError is raised	Pass
3.9	Check postorder_next(p) with invalid data value (Figure 3)	Call postorder_next(p) for linked binary tree in Figure 3.2 with p = p2	Since position p3 does not belong in linked binary tree in Figure 4.2, a ValueError is raised	Pass
3.10	Check delete_subtree(p) with invalid data type (Figure 3)	Call delete_subtree(p) for linked binary tree in Figure 3.1 with p = 5	A TypeError is raised	Pass
3.11	Check delete_subtree(p) with invalid data type (Figure 3)	Call delete_subtree(p) for linked binary tree in Figure 3.1 with p = 'p3'	A TypeError is raised	Pass
3.12	Check delete_subtree(p) with invalid data value (Figure 3)	Call delete_subtree(p) for linked binary tree in Figure 3.2 with p = p2	Since position p3 does not belong in linked binary tree in Figure 4.2, a ValueError is raised	Pass
Test for linked binary tree in Figure 4				
Test Case #	Test Scenario	Test Steps	Expected Result	Pass/ Fail
4.1	Check preorder_next(p) with valid data (Figure 4)	1. Create an instance of ExtendedLinkedBinaryTree 2. Assign positions l1, l2, l3,…, l25 to distinct number elements as shown in Figure 4	Both preorder() and preorder_next(p) print: **1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25**	Pass

		3. Call preorder() and print the elements stored in the tree in preorder traversal 4. Call preorder_next(p) and print the elements stored in the tree, and compare them to the results of preorder()		
4.2	Check inorder_next(p) with valid data (Figure 4)	1. Call inorder() and print the elements stored in the tree in preorder traversal 2. Call inorder_next(p) and print the elements stored in the tree, and compare them to the results of inorder()	Both inorder() and inorder_next(p) print: **3 2 5 8 7 9 6 4 1 11 10 17 16 18 14 13 15 19 20 12 23 22 21 24 25**	Pass
4.3	Check postorder_next(p) with valid data (Figure 4)	1. Call postorder() and print the elements stored in the tree in preorder traversal 2. Call postorder_next(p) and print the elements stored in the tree, and compare them to the results of postorder()	Both postorder() and postorder_next(p) print: **3 8 9 7 6 5 4 2 11 17 18 16 14 20 19 15 13 23 22 25 24 21 12 10 1**	Pass
4.4	Check delete_subtree(p) with valid data (Figure 4)	1. Call delete_subtree(l10) 2. Call inorder() and print the elements of the remaining tree 3. Call delete_subtree(l10) again	**3 2 5 8 7 9 6 4 1** When delete_subtree(l10) is called again, a ValueError is raised	Pass

Responsibility
The four methods: Mainly Anh and Lam (a little bit less than Anh)
Test code: Huong, Lam
Binary tree drawings: Huong, Lam
Table, flowchart: Huong
Typescript: Anh

Problem we encountered: debugging methods and unittest, exploring many cases of binary trees, time constraints

What we learn: The use and exploration of binary tree data structure, the different traversals through all the positions of binary tree