

Nama : Romy Mahardika Pangestu Lazuardi
No : 25
Kelas : 1H/D4TI

JOBSHEET 13

Tree

13.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

13.2.1 Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan array (praktikum 2) dan linked list (praktikum 1). Sebelumnya, akan dibuat class Node, dan Class BinaryTree

Node		
data: int left: Node right: Node		
Node(left:	Node,	data:int,
right:Node)		

BinaryTree	
root: Node	size : int
DoubleLinkedLists() add(data: int): void find(data: int) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) add(item: int, index:int): void delete(data: int): void	

1. Buatlah class NodeNoAbsen, BinaryTreeNoAbsen dan BinaryTreeMainNoAbsen
2. Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

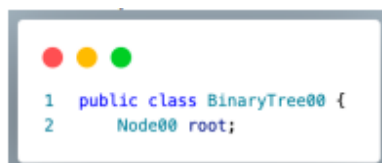


```

1 public class Node00 {
2     int data;
3     Node00 left;
4     Node00 right;
5
6     public Node00(){
7     }
8     public Node00(int data){
9         this.left = null;
10        this.data = data;
11        this.right = null;
12    }
13 }
14

```

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.



```

1 public class BinaryTree00 {
2     Node00 root;
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

4. Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTreeNoAbsen

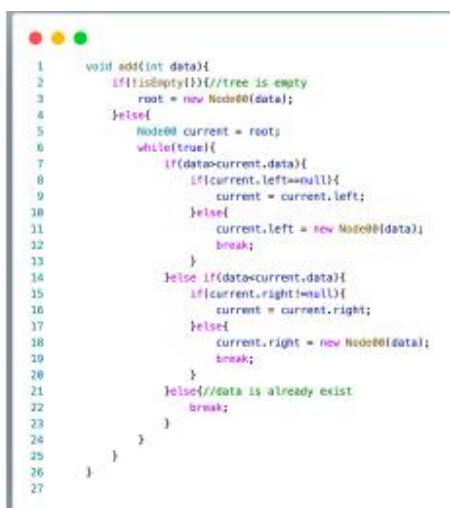


```

1 public BinaryTree00(){
2     root = null;
3 }
4 boolean isEmpty(){
5     return root==null;
6 }
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

5. Tambahkan method add() di dalam class BinaryTreeNoAbsen. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya, jika dilakukan dengan proses rekursif, penulisan kode akan lebih efisien.



```

1 void add(int data){
2     if(isEmpty()){tree is empty
3         root = new Node00(data);
4     }else{
5         Node00 current = root;
6         while(true){
7             if(data<current.data){
8                 if(current.left==null){
9                     current = current.left;
10                }else{
11                    current.left = new Node00(data);
12                    break;
13                }
14            }else if(data>current.data){
15                if(current.right==null){
16                    current = current.right;
17                }else{
18                    current.right = new Node00(data);
19                    break;
20                }
21            }else{//data is already exist
22                break;
23            }
24        }
25    }
26 }
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99

```

6. Tambahkan method find()

```

1  boolean find(int data){
2      boolean result = false;
3      Node00 current = root;
4      while(current!=null){
5          if(current.data!=data){
6              result = true;
7              break;
8          }else if(data>current.data){
9              current = current.left;
10         }else{
11             current = current.right;
12         }
13     }
14     return result;
15 }

```

7. Tambahkan method `traversePreOrder()`, `traverseInOrder()` dan `traversePostOrder()`. Method `traverse` digunakan untuk mengunjungi dan menampilkan node-node dalam tree, baik dalam mode pre-order, in-order maupun post-order.

```

1  void traversePreOrder(Node00 node) {
2      if (node != null) {
3          System.out.print(" " + node.data);
4          traversePreOrder(node.left);
5          traversePreOrder(node.right);
6      }
7  }
8  void traversePostOrder(Node00 node) {
9      if (node != null) {
10         traversePostOrder(node.left);
11         traversePostOrder(node.right);
12         System.out.print(" " + node.data);
13     }
14 }
15 void traverseInOrder(Node00 node) {
16     if (node != null) {
17         traverseInOrder(node.left);
18         System.out.print(" " + node.data);
19         traverseInOrder(node.right);
20     }
21 }

```

8. Tambahkan method `getSuccessor()`. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child

```

1  Node00 getSuccessor(Node00 del){
2      Node00 successor = del.right;
3      Node00 successorParent = del;
4      while(successor.left!=null){
5          successorParent = successor;
6          successor = successor.left;
7      }
8      if(successor!=del.right){
9          successorParent.left = successor.right;
10         successor.right = del.right;
11     }
12     return successor;
13 }

```

9. Tambahkan method `delete()`. Di dalam method `delete` tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```

1 void delete(int data){
2     if(isEmpty()){
3         System.out.println("Tree is empty!");
4         return;
5     }
6     //find node (current) that will be deleted
7     Node00 parent = root;
8     Node00 current = root;
9     boolean isLeftChild = false;
10    while(current!=null){
11        if(current.data==data){
12            break;
13        }else if(data<current.data){
14            parent = current;
15            current = current.left;
16            isLeftChild = true;
17        }else if(data>current.data){
18            parent = current;
19            current = current.right;
20            isLeftChild = false;
21        }
22    }

```

10. Kemudian tambahkan proses penghapusan didalam method delete() terhadap node current yang telah ditemukan.

```

1 //deletion
2 if(current==null){
3     System.out.println("Couldn't find data!");
4     return;
5 }else{
6     //if there is no child, simply delete it
7     if(current.left==null&&current.right==null){
8         if(current==root){
9             root = null;
10        }else{
11            if(isLeftChild){
12                parent.left = null;
13            }else{
14                parent.right = null;
15            }
16        }
17    }else if(current.left==null){ //if there is 1 child (right)
18        if(current==root){
19            root = current.right;
20        }else{
21            if(isLeftChild){
22                parent.left = current.right;
23            }else{
24                parent.right = current.right;
25            }
26        }

```

```

27    }else if(current.right==null){ //if there is 1 child (left)
28        if(current==root){
29            root = current.left;
30        }else{
31            if(isLeftChild){
32                parent.left = current.left;
33            }else{
34                parent.right = current.left;
35            }
36        }
37    }else{//if there is 2 childs
38        Node00 successor = getSuccessor(current);
39        if(current==root){
40            root = successor;
41        }else{
42            if(isLeftChild){
43                parent.left = successor;
44            }else{
45                parent.right = successor;
46            }
47            successor.left = current.left;
48        }
49    }
50 }
51 }
52 }

```

11. Buka class BinaryTreeMainNoAbsen dan tambahkan method main() kemudian tambahkan kode berikut ini

```
1  BinaryTree08 bt = new BinaryTree08();
2  bt.add(6);
3  bt.add(4);
4  bt.add(8);
5  bt.add(3);
6  bt.add(5);
7  bt.add(7);
8  bt.add(9);
9  bt.add(10);
10 bt.add(15);
11 System.out.print("PreOrder Traversal : ");
12 bt.traversePreOrder(bt.root);
13 System.out.println("");
14 System.out.print("InOrder Traversal : ");
15 bt.traverseInOrder(bt.root);
16 System.out.println("");
17 System.out.print("PostOrder Traversal : ");
18 bt.traversePostOrder(bt.root);
19 System.out.println("");
20 System.out.println("Find Node : "+bt.find(5));
21 System.out.println("Delete Node 8 ");
22 bt.delete(8);
23 System.out.println("");
24 System.out.print("PreOrder Traversal : ");
25 bt.traversePreOrder(bt.root);
26 System.out.println("");
```

12. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.

```
public class Node25 {
    int data;
    Node25 left;
    Node25 right;

    public Node25() {
    }

    public Node25(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

```
public class BinaryTree25 {
    Node25 root;

    public BinaryTree25() {
        root = null;
    }

    boolean isEmpty() {
        return root == null;
    }
}
```

```

}

void add(int data) {
    if (isEmpty()) {
        root = new Node25(data);
    } else {
        Node25 current = root;
        while (true) {
            if (data < current.data) {
                if (current.left != null) {
                    current = current.left;
                } else {
                    current.left = new Node25(data);
                    break;
                }
            } else if (data > current.data) {
                if (current.right != null) {
                    current = current.right;
                } else {
                    current.right = new Node25(data);
                    break;
                }
            } else {
                break;
            }
        }
    }
}

boolean find(int data) {
    Node25 current = root;
    while (current != null) {
        if (data == current.data) {
            return true;
        } else if (data < current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return false;
}

void traversePreOrder(Node25 node) {
    if (node != null) {
        System.out.print(node.data + " ");
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

```

```

    }
}

void traverseInOrder(Node25 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(node.data + " ");
        traverseInOrder(node.right);
    }
}

void traversePostOrder(Node25 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(node.data + " ");
    }
}

Node25 getSuccessor(Node25 del) {
    Node25 successor = del.right;
    Node25 successorParent = del;
    while (successor.left != null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successor != del.right) {
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}

void delete(int data) {
    if (isEmpty()) {
        System.out.println("Tree is empty!");
        return;
    }

    Node25 parent = root;
    Node25 current = root;
    boolean isLeftChild = false;

    while (current != null) {
        if (data == current.data) {
            break;
        } else if (data < current.data) {
            parent = current;

```

```

        current = current.left;
        isLeftChild = true;
    } else {
        parent = current;
        current = current.right;
        isLeftChild = false;
    }
}

if (current == null) {
    System.out.println("Couldn't find data!");
    return;
}

// Case 1: no children
if (current.left == null && current.right == null) {
    if (current == root) {
        root = null;
    } else if (isLeftChild) {
        parent.left = null;
    } else {
        parent.right = null;
    }
} // Case 2: one child (right)
else if (current.left == null) {
    if (current == root) {
        root = current.right;
    } else if (isLeftChild) {
        parent.left = current.right;
    } else {
        parent.right = current.right;
    }
} // Case 3: one child (left)
else if (current.right == null) {
    if (current == root) {
        root = current.left;
    } else if (isLeftChild) {
        parent.left = current.left;
    } else {
        parent.right = current.left;
    }
} // Case 4: two children
else {
    Node25 successor = getSuccessor(current);
    if (current == root) {
        root = successor;
    } else if (isLeftChild) {
        parent.left = successor;
    }
}

```



```

        } else {
            parent.right = successor;
        }
        successor.left = current.left;
    }
}
}

```

```

public class BinaryTreeMain25 {
    public static void main(String[] args) {
        BinaryTree25 bt = new BinaryTree25();
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        System.out.print("PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println("");

        System.out.print("InOrder Traversal: ");
        bt.traverseInOrder(bt.root);
        System.out.println("");

        System.out.print("PostOrder Traversal: ");
        bt.traversePostOrder(bt.root);
        System.out.println("");

        System.out.println("Find Node: " + bt.find(5));
        System.out.println("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");

        System.out.print("PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}

```

13. Amati hasil running tersebut.

```

PreOrder Traversal : 6 4 3 5 8 7 9 10 15
InOrder Traversal : 3 4 5 6 7 8 9 10 15
PostOrder Traversal : 3 5 4 7 15 10 9 8 6
Find Node : true
Delete Node 8

```

```

PreOrder Traversal : 6 4 3 5 9 7 10 15

```

```

PreOrder Traversal: 6 4 3 5 8 7 9 10 15
InOrder Traversal: 3 4 5 6 7 8 9 10 15
PostOrder Traversal: 3 5 4 7 15 10 9 8 6
Find Node: true
Delete Node 8
PreOrder Traversal: 6 4 3 5 9 7 10 15

```

13.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?

Semua nilai di subtree kiri dari node tersebut lebih kecil dari nilai node itu sendiri.

2. Untuk apakah di class Node, kegunaan dari atribut left dan right?

left: Merujuk ke node di subtree kiri yang menyimpan nilai lebih kecil dari node saat ini.

right: Merujuk ke node di subtree kanan yang menyimpan nilai lebih besar dari node saat ini.

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?

Atribut root di dalam class BinaryTree digunakan untuk merujuk ke node paling atas atau node akar dari tree tersebut.

b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

Ketika objek tree pertama kali dibuat, nilai dari root biasanya adalah null atau None

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

Membuat instance baru dari node dengan data yang diberikan.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```

if(data<current.data){
    if(current.left!=null){
        current = current.left;
    }else{
        current.left = new Node(data);
        break;
    }
}

```

if(data < current.data): Memeriksa apakah nilai data yang akan ditambahkan lebih kecil daripada nilai pada node saat ini (current). Jika iya, maka data baru seharusnya ditempatkan di subtree kiri dari node saat ini.

13.3 Kegiatan Praktikum 2

Implementasi binary tree dengan array (45 Menit)

13.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.
2. Buatlah class BinaryTreeArrayNoAbsen dan BinaryTreeArrayMainNoAbsen
3. Buat atribut data dan idxLast di dalam class BinaryTreeArrayNoAbsen. Buat juga method populateData() dan traverseInOrder().

```
1 public class BinaryTreeArray00 {
2     int[] data;
3     int idxLast;
4
5     public BinaryTreeArray00(){
6         data = new int[10];
7     }
8     void populateData(int data[], int idxLast){
9         this.data = data;
10        this.idxLast = idxLast;
11    }
12    void traverseInOrder(int idxStart){
13        if(idxStart<=idxLast){
14            traverseInOrder(2*idxStart+1);
15            System.out.print(data[idxStart]+" ");
16            traverseInOrder(2*idxStart+2);
17        }
18    }
19 }
```

4. Kemudian dalam class BinaryTreeArrayMainNoAbsen buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method Main

```
1 BinaryTreeArray00 bta = new BinaryTreeArray00();
2 int[] data = {6,4,8,3,5,7,9,0,0,0};
3 int idxLast = 6;
4 bta.populateData(data, idxLast);
5 System.out.print("\nInOrder Traversal : ");
6 bta.traverseInOrder(0);
7 System.out.println("\n");
```

```
public class BinaryTreeArray25 {
    int[] data;
    int idxLast;

    public BinaryTreeArray25() {
        data = new int[10];
        idxLast = -1;
    }
}
```

```

void populateData(int[] data, int idxLast) {
    this.data = data;
    this.idxLast = idxLast;
}

void traverseInOrder(int idxStart) {
    if (idxStart <= idxLast && data[idxStart] != 0) {
        traverseInOrder(2 * idxStart + 1);
        System.out.print(data[idxStart] + " ");
        traverseInOrder(2 * idxStart + 2);
    }
}
}

```

```

public class BinaryTreeArrayMain25 {
    public static void main(String[] args) {
        BinaryTreeArray25 bta = new BinaryTreeArray25();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;

        bta.populateData(data, idxLast);
        System.out.print("InOrder Traversal: ");
        bta.traverseInOrder(0);
        System.out.println();
    }
}

```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

InOrder Traversal : 3 4 5 6 7 8 9

InOrder Traversal: 3 4 5 6 7 8 9

13.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?

Atribut data dalam BinaryTreeArray biasanya merupakan array yang digunakan untuk menyimpan elemen-elemen dari binary tree dalam bentuk array.

Atribut idxLast digunakan untuk melacak indeks terakhir yang diisi dalam array data.

2. Apakah kegunaan dari method populateData()?

Method populateData() digunakan untuk mengisi array data dengan nilai-nilai yang diperlukan untuk membentuk binary tree.

3. Apakah kegunaan dari method traverseInOrder()?

Method `traverseInOrder()` digunakan untuk melakukan traversal in-order pada binary tree yang disimpan dalam array.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan right child masing-masing?

Jika suatu node disimpan dalam array pada indeks 2, maka posisi left child dan right child ditentukan dengan aturan berikut:

Left child: Berada di indeks $2 * i + 1$, di mana i adalah indeks node saat ini.

Right child: Berada di indeks $2 * i + 2$.

Dengan $i = 2$:

Indeks left child = $2 * 2 + 1 = 5$

Indeks right child = $2 * 2 + 2 = 6$

Jadi, left child berada di indeks 5, dan right child berada di indeks 6.

5. Apa kegunaan statement `int idxLast = 6` pada praktikum 2 percobaan nomor 4?

Statement `int idxLast = 6` digunakan untuk menginisialisasi variabel `idxLast` dengan nilai 6.

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

1. Buat method di dalam class `BinaryTree` yang akan menambahkan node dengan cara rekursif.
2. Buat method di dalam class `BinaryTree` untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.
3. Buat method di dalam class `BinaryTree` untuk menampilkan data yang ada di leaf.
4. Buat method di dalam class `BinaryTree` untuk menampilkan berapa jumlah leaf yang ada di dalam tree.
5. Modifikasi class `BinaryTreeArray`, dan tambahkan :
 - method `add(int data)` untuk memasukan data ke dalam tree
 - method `traversePreOrder()` dan `traversePostOrder()`

```
public class BinaryTree25 {
    Node25 root;

    public BinaryTree25() {
        root = null;
    }
}
```

```

}

boolean isEmpty() {
    return root == null;
}

// Recursive add method
void add(int data) {
    root = addRecursive(root, data);
}

private Node25 addRecursive(Node25 current, int data) {
    if (current == null) {
        return new Node25(data);
    }
    if (data < current.data) {
        current.left = addRecursive(current.left, data);
    } else if (data > current.data) {
        current.right = addRecursive(current.right, data);
    }
    return current;
}

boolean find(int data) {
    Node25 current = root;
    while (current != null) {
        if (data == current.data) {
            return true;
        } else if (data < current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return false;
}

void traversePreOrder(Node25 node) {
    if (node != null) {
        System.out.print(node.data + " ");
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traverseInOrder(Node25 node) {
    if (node != null) {
        traverseInOrder(node.left);
    }
}

```

```

        System.out.print(node.data + " ");
        traverseInOrder(node.right);
    }
}

void traversePostOrder(Node25 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(node.data + " ");
    }
}

Node25 getSuccessor(Node25 del) {
    Node25 successor = del.right;
    Node25 successorParent = del;
    while (successor.left != null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successor != del.right) {
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}

void delete(int data) {
    if (isEmpty()) {
        System.out.println("Tree is empty!");
        return;
    }

    Node25 parent = root;
    Node25 current = root;
    boolean isLeftChild = false;

    while (current != null) {
        if (data == current.data) {
            break;
        } else if (data < current.data) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else {
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
}

```

```

    }
}

if (current == null) {
    System.out.println("Couldn't find data!");
    return;
}

// Case 1: no children
if (current.left == null && current.right == null) {
    if (current == root) {
        root = null;
    } else if (isLeftChild) {
        parent.left = null;
    } else {
        parent.right = null;
    }
} // Case 2: one child (right)
else if (current.left == null) {
    if (current == root) {
        root = current.right;
    } else if (isLeftChild) {
        parent.left = current.right;
    } else {
        parent.right = current.right;
    }
} // Case 3: one child (left)
else if (current.right == null) {
    if (current == root) {
        root = current.left;
    } else if (isLeftChild) {
        parent.left = current.left;
    } else {
        parent.right = current.left;
    }
} // Case 4: two children
else {
    Node25 successor = getSuccessor(current);
    if (current == root) {
        root = successor;
    } else if (isLeftChild) {
        parent.left = successor;
    } else {
        parent.right = successor;
    }
    successor.left = current.left;
}
}
}

```



```

// Method to find the minimum value
int findMin() {
    if (isEmpty()) {
        throw new IllegalStateException("Tree is empty!");
    }
    Node25 current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current.data;
}

// Method to find the maximum value
int findMax() {
    if (isEmpty()) {
        throw new IllegalStateException("Tree is empty!");
    }
    Node25 current = root;
    while (current.right != null) {
        current = current.right;
    }
    return current.data;
}

// Method to display leaf nodes
void displayLeaves(Node25 node) {
    if (node != null) {
        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        }
        displayLeaves(node.left);
        displayLeaves(node.right);
    }
}

// Method to count the number of leaf nodes
int countLeaves(Node25 node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    return countLeaves(node.left) + countLeaves(node.right);
}

// Method to start displaying leaf nodes

```

```

void displayLeaves() {
    displayLeaves(root);
}

// Method to get the number of leaf nodes
int countLeaves() {
    return countLeaves(root);
}
}

```

```

public class BinaryTreeMain25 {
    public static void main(String[] args) {
        BinaryTree25 bt = new BinaryTree25();
        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        System.out.print("PreOrder Traversal: ");
        bt.traversePreOrder(bt.root);
        System.out.println("");

        System.out.print("InOrder Traversal: ");
        bt.traverseInOrder(bt.root);
        System.out.println("");

        System.out.print("PostOrder Traversal: ");
        bt.traversePostOrder(bt.root);
        System.out.println("");

        System.out.println("Find Node 5: " + bt.find(5));

        System.out.println("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");

        System.out.print("PreOrder Traversal after deletion: ");
        bt.traversePreOrder(bt.root);
        System.out.println("");

        System.out.println("Minimum value in the tree: " + bt.findMin());
        System.out.println("Maximum value in the tree: " + bt.findMax());
    }
}

```

```

        System.out.print("Leaf nodes in the tree: ");
        bt.displayLeaves();
        System.out.println("");

        System.out.println("Number of leaf nodes in the tree: " +
bt.countLeaves());
    }
}

```

```

PreOrder Traversal: 6 4 3 5 8 7 9 10 15
InOrder Traversal: 3 4 5 6 7 8 9 10 15
PostOrder Traversal: 3 5 4 7 15 10 9 8 6
Find Node 5: true
Delete Node 8

PreOrder Traversal after deletion: 6 4 3 5 9 7 10 15
Minimum value in the tree: 3
Maximum value in the tree: 15
Leaf nodes in the tree: 3 5 7 15
Number of leaf nodes in the tree: 4

```

```

public class BinaryTreeArray25 {
    int[] data;
    int idxLast;

    public BinaryTreeArray25() {
        data = new int[10];
        idxLast = -1;
    }

    void populateData(int[] data, int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }

    void traverseInOrder(int idxStart) {
        if (idxStart <= idxLast && data[idxStart] != 0) {
            traverseInOrder(2 * idxStart + 1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder(2 * idxStart + 2);
        }
    }

    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast && data[idxStart] != 0) {
            System.out.print(data[idxStart] + " ");
            traversePreOrder(2 * idxStart + 1);
            traversePreOrder(2 * idxStart + 2);
        }
    }

    void traversePostOrder(int idxStart) {

```

```

        if (idxStart <= idxLast && data[idxStart] != 0) {
            traversePostOrder(2 * idxStart + 1);
            traversePostOrder(2 * idxStart + 2);
            System.out.print(data[idxStart] + " ");
        }
    }

    void add(int data) {
        if (idxLast == this.data.length - 1) {
            System.out.println("Tree is full. Cannot add more elements.");
            return;
        }
        this.data[++idxLast] = data;
    }
}

```

```

public class BinaryTreeArrayMain25 {
    public static void main(String[] args) {
        BinaryTreeArray25 bta = new BinaryTreeArray25();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;

        bta.populateData(data, idxLast);
        System.out.print("InOrder Traversal: ");
        bta.traverseInOrder(0);
        System.out.println();

        System.out.print("PreOrder Traversal: ");
        bta.traversePreOrder(0);
        System.out.println();

        System.out.print("PostOrder Traversal: ");
        bta.traversePostOrder(0);
        System.out.println();

        System.out.println("Adding new elements: ");
        bta.add(10);
        bta.add(11);
        System.out.print("InOrder Traversal after adding: ");
        bta.traverseInOrder(0);
        System.out.println();
    }
}

```

```
InOrder Traversal: 3 4 5 6 7 8 9
PreOrder Traversal: 6 4 3 5 8 7 9
PostOrder Traversal: 3 5 4 7 9 8 6
Adding new elements:
InOrder Traversal after adding: 10 3 11 4 5 6 7 8 9
```