LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

Dosen Pengajar : Triana Fatmawati, S.T, M.T

Jobsheet-14 Binary Tree

Nama : Surya Rahmat Fatahillah

NIM : 2341760020

Prodi : Sistem Informasi Bisnis

JURUSAN TEKNOLOGI INFORMASI POLITEKNIK NEGERI MALANG 2023/2024

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

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

```
package Jobsheet13;

public class Node27 {
    int data;
    Node27 left;
    Node27 right;

    public Node27() {
    }

    public Node27(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
package Jobsheet13;

public class BinaryTree27 {
    Node27 root;
```

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

```
    public BinaryTree27() {
        root = null;
    }

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

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.

```
void add(int data) {
    if (isEmpty()) {
        root = new Node27(data);
    } else {
        Node27 current = root;
        while (true) {
            if (data < current.data) {
                if (current.left == null) {
                    current.left = new Node27(data);
                    break;
                } else {
                    current = current.left;
                }
            } else if (data > current.data) {
                if (current.right == null) {
                    current.right = new Node27(data);
                    break;
                } else {
                    current = current.right;
                }
            } else { // data already exists, do nothing
                break;
            }
        }
    }
}
```

6. Tambahkan method find()

```
boolean find(int data) {
    Node27 current = root;
    while (current != null) {
        if (current.data == data) {
            return true;
        } else if (data < current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return false;
}
```

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.

```
void traversePreOrder(Node27 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder(Node27 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void transverseInOrder(Node27 node) {
    if (node != null) {
        transverseInOrder(node.left);
        System.out.print(" " + node.data);
        transverseInOrder(node.right);
    }
}
```

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

```
  Node27 getSuccessor(Node27 del) {
      Node27 successor = del.right;
      Node27 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;
  }
```

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

```
void delete(int data) {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return;
    }
    Node27 parent = root;
    Node27 current = root;
    boolean isLeftChild = false;
    while (current != null) {
        if (current.data == data) {
            break;
        } else if (data < current.data) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else if (data > current.data) {
            parent = current;
            current = current.right;
            isLeftChild = false;

        }

    }
```

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

```
    if (current == null) {
        System.out.println(x:"Couldn't find data!");
        return;
    } else {
        if (current.left == null && current.right == null) {
            if (current == root) {
                root = null;
            } else {
                if (isLeftChild) {
                    parent.left = null;
                } else {
                    parent.right = null;
                }
            }
        } else if (current.left == null) {
            if (current == root) {
                root = current.right;
            } else {
                if (isLeftChild) {
                    parent.left = current.right;
                } else {
                    parent.right = current.right;
                }
            }
        } else if (current.right == null) {
            if (current == root) {
                root = current.left;
            } else {
                if (isLeftChild) {
                    parent.left = current.left;
                } else {
                    parent.right = current.left;
                }
            }
```

```
        } else {
            Node27 successor = getSuccessor(current);
            if (current == root) {
                root = successor;
            } else {
                if (isLeftChild) {
                    parent.left = successor;
                } else {
                    parent right successor;
                }
                successor.left = current.left;
            }
        }
    }
}
```

Node27 left

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

```
package Jobsheet13;

public class BinaryTreeMain27 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree27 bt = new BinaryTree27();
        bt.add(data:6);
        bt.add(data:4);
        bt.add(data:8);
        bt.add(data:3);
        bt.add(data:5);
        bt.add(data:7);
        bt.add(data:9);
        bt.add(data:10);
        bt.add(data:15);
        System.out.print(s:"PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"inOrder Traversal : ");
        bt.transverseInOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"PostOrder Traversal : ");
        bt.traversePostOrder(bt.root);
        System.out.println(x:"");
        System.out.println("Find Node : " + bt.find(data:5));
        System.out.println(x:"Delete Node 8");
        bt.delete(data:8);
        System.out.println(x:"");
        System.out.print(s:"PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
    }
}
```

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

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
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

13.2.2 Pertanyaan Percobaan

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

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

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

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

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

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;
 }
}
```

Jawaban!

1. Proses pencarian data dalam Binary Search Tree (BST) bisa lebih efektif dilakukan dibandingkan dengan binary tree biasa karena BST memiliki sifat terurut dimana setiap node di sub tree kiri lebih kecil dan setiap node di sub tree kanan lebih besar dari node saat ini, sehingga memungkinkan pencarian dilakukan dengan kompleksitas waktu logaritmik \(O(\log n)\) pada pohon yang seimbang, dibandingkan dengan pencarian pada binary tree biasa yang tidak memiliki struktur terurut dan kompleksitas waktu bisa mencapai \(O(n)\).

2. Di dalam class Node, atribut `left` dan `right` digunakan untuk merepresentasikan hubungan antara node dalam struktur data binary tree, di mana `left` digunakan untuk menyimpan referensi ke anak kiri yang memiliki nilai lebih kecil dan `right` menyimpan referensi ke anak kanan yang memiliki nilai lebih besar, sehingga memungkinkan traversal, pencarian, penambahan, dan penghapusan node berdasarkan nilai data yang dibandingkan, dan mendukung pembentukan struktur yang kompleks.

3. A. Atribut root di dalam class BinaryTree digunakan untuk menyimpan referensi ke node akar dari pohon biner, yang merupakan titik awal untuk semua operasi traversal (pre-order, in-order, post-order), pencarian, penambahan, dan penghapusan node, serta pengelolaan keseluruhan struktur pohon dengan memastikan bahwa semua operasi tersebut dimulai dari node akar dan memungkinkan kita untuk mengosongkan pohon atau memeriksa apakah pohon kosong.

   B. Ketika objek tree pertama kali dibuat, nilai dari root adalah null. Hal ini menandakan bahwa tree tersebut awalnya kosong dan tidak memiliki node apa pun.

4. Ketika tree masih kosong dan akan ditambahkan sebuah node baru, proses yang terjadi adalah sebagai berikut:

   - Pemeriksaan Apakah Tree Kosong:

   Pertama, sistem akan memeriksa apakah tree kosong dengan mengecek nilai dari atribut root. Jika root adalah null, maka tree dianggap kosong.

   - Pembuatan Node Baru:

   Sebuah node baru akan dibuat dengan nilai data yang akan ditambahkan.

   - Penempatan Node Baru sebagai Root:

   Karena tree kosong (nilai root adalah null), node baru ini akan ditempatkan sebagai root dari tree. Ini dilakukan dengan mengatur referensi root ke node baru yang baru saja dibuat.

5. Baris program berikut merupakan bagian dari metode add() dalam class BinaryTree27 yang berfungsi untuk menambahkan node baru ke dalam tree sesuai dengan aturan Binary Search Tree (BST).

   Penjelasan detil:

1. **if (data < current.data) {**:
   Bagian ini berfungsi untuk memeriksa apakah nilai data yang akan ditambahkan lebih kecil dari nilai data pada node saat ini (current.data).
   Dalam BST, jika nilai data lebih kecil dari nilai node saat ini, node baru harus ditempatkan di sub tree kiri.
2. **if (current.left != null) {**:
   Jika sub tree kiri dari node saat ini (current.left) tidak null, berarti ada node di sebelah kiri node saat ini.
   Kita perlu bergerak ke node kiri tersebut untuk melanjutkan pencarian posisi yang tepat untuk node baru.
3. **current = current.left;**:
   Mengubah referensi current menjadi current.left, artinya sekarang node saat ini adalah anak kiri dari node sebelumnya.
   Ini mempersiapkan iterasi berikutnya dari loop untuk memeriksa posisi lebih lanjut dalam sub tree kiri.
4. **else {**:
   Bagian else dijalankan jika current.left adalah null, yang berarti tidak ada node di sebelah kiri node saat ini.
   Ini berarti kita telah menemukan posisi yang tepat untuk menempatkan node baru di sub tree kiri dari node saat ini.
5. **current.left = new Node27(data);**:
   Membuat node baru dengan nilai data dan menetapkannya sebagai anak kiri dari node saat ini (current.left).
   Node baru ini sekarang menjadi anak kiri dari node saat ini, sesuai dengan aturan BST.
6. **break;**:
   Setelah menambahkan node baru, loop dihentikan dengan break.
   break digunakan untuk keluar dari loop while(true), karena kita telah menyelesaikan penambahan node baru ke dalam tree.

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()

```
package Jobsheet13;

public class BinaryTreeArray27 {
    int[] data;
    int idxLast;

    public BinaryTreeArray27(){
        data = new int[10];
    }

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

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

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

```
package Jobsheet13;

public class BinaryTreeArrayMain27 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray27 bta = new BinaryTreeArray27();

        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        System.out.print(s:"\nInOrder Traversal : ");
        bta.transverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}
```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
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?

2. Apakah kegunaan dari method populateData()?

3. Apakah kegunaan dari method traverseInOrder()?

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

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

Jawaban!

1. Atribut `data` dalam class `BinaryTreeArray` digunakan untuk menyimpan elemen-elemen node dari Binary Tree dalam bentuk array, sedangkan `idxLast` digunakan untuk menyimpan indeks terakhir dari node yang valid dalam array tersebut.

2. Method populateData() digunakan untuk mengisi atribut data dengan elemen-elemen node dari Binary Tree dalam bentuk array dan mengatur idxLast untuk menunjukkan indeks terakhir dari node yang valid dalam array tersebut.

3. Method `traverseInOrder()` digunakan untuk melakukan traversal in-order pada pohon Binary Tree disimpan dalam array, dimulai dari indeks tertentu, dengan mencetak elemen-elemen node sesuai urutan in-order yaitu mengunjungi node kiri terlebih dahulu, kemudian node saat ini, dan terakhir node kanan.

4. Jika suatu node binary tree disimpan dalam array pada indeks 2, maka posisi left child dan right child dari node tersebut dapat dihitung menggunakan rumus berikut:

   - **Left Child**:

     a. Indeks left child adalah $2 \times \text{idx} + 1$ $2 \times idx + 1$
     b. Jika $idx = 2$, maka indeks left child adalah $2 \times 2 + 1 = 5$ $2 \times 2 + 1 = 5$

   - **Right Child**:

     c. Indeks right child adalah $2 \times \text{idx} + 2$ $2 \times idx + 2$
     d. Jika $idx = 2$, maka indeks right child adalah $2 \times 2 + 2 = 6$ $2 \times 2 + 2 = 6$

5. Statement `int idxLast = 6` digunakan untuk menunjukkan indeks terakhir dari node yang valid dalam array yang merepresentasikan Binary Tree yang berarti bahwa elemen-elemen array dari indeks 0 hingga 6 (inklusif) berisi node-node yang valid dari Binary Tree, sedangkan indeks-indeks setelah 6 tidak digunakan atau tidak berisi node yang valid. Dengan demikian, `idxLast` dapat membantu metode traversal seperti `traverseInOrder()` untuk menentukan batas atas dari indeks yang perlu diproses.

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()

Jawaban!

1. Berikut method untuk menambahkan node dengan cara rekursif

```
    // Metode untuk menambahkan node dengan cara rekursif
    void addRecursive(int data) {
        root = addRecursiveHelper(root, data);
    }

    // Helper method rekursif untuk menambahkan node
    private Node27 addRecursiveHelper(Node27 current, int data) {
        if (current == null) {
            return new Node27(data);
        }

        if (data < current.data) {
            current.left = addRecursiveHelper(current.left, data);
        } else if (data > current.data) {
            current.right = addRecursiveHelper(current.right, data);
        } // else, data sudah ada, tidak perlu menambahkan apa-apa

        return current;
    }
}
```

Implementasi di Class Main:

```
public class BinaryTreeMain27 {
    public static void main(String[] args) {
        bt.addRecursive(data:6);
        bt.addRecursive(data:4);
        bt.addRecursive(data:8);
        bt.addRecursive(data:3);
        bt.addRecursive(data:5);
        bt.addRecursive(data:7);
        bt.addRecursive(data:9);
        bt.addRecursive(data:10);
        bt.addRecursive(data:15);
        System.out.print(s:"PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"inOrder Traversal : ");
        bt.transverseInOrder(bt.root);
        System.out.println(x:"");
        System.out.print(s:"PostOrder Traversal : ");
        bt.traversePostOrder(bt.root);
        System.out.println(x:"");
        System.out.println("Find Node : " + bt.find(data:5));
        System.out.println(x:"Delete Node 8");
        bt.delete(data:8);
        System.out.println(x:"");
        System.out.print(s:"PreOrder Traversal : ");
        bt.traversePreOrder(bt.root);
        System.out.println(x:"");
```

Hasil:

```
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
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

2. Berikut method untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

- Method untuk menampilkan nilai paling kecil

```
// Metode untuk mencari nilai paling kecil dalam pohon biner
int findMinValue() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty");
        return Integer.MIN_VALUE;
    }
    Node27 current = root;
    while (current.left != null) {
        current = current.left;
    }
    return current.data;
}
```

- Method untuk menampilkan nilai paling besar

```
// Metode untuk mencari nilai paling besar dalam pohon biner
int findMaxValue() {
    if (isEmpty()) {
        System.out.println(x:"Tree is empty");
        return Integer.MAX_VALUE;
    }
    Node27 current = root;
    while (current.right != null) {
        current = current.right;
    }
    return current.data;
}
```

- Implementasi di Class Main

```
// Memanggil metode untuk menemukan nilai paling kecil dan paling besar
int minValue = bt.findMinValue();
int maxValue = bt.findMaxValue();

System.out.println("Nilai paling kecil dalam pohon: " + minValue);
System.out.println("Nilai paling besar dalam pohon: " + maxValue);
}
```

- Hasil:

```
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
Nilai paling kecil dalam pohon: 3
Nilai paling besar dalam pohon: 15
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

3. Berikut method untuk untuk menampilkan data yang ada di leaf

```java
// Method untuk menampilkan data di leaf nodes
void printLeafNodes(Node27 node) {
    if (node == null) {
        return;
    }
    if (node.left == null && node.right == null) {
        System.out.print(node.data + " ");
    } else {
        printLeafNodes(node.left);
        printLeafNodes(node.right);
    }
}
```

- Implementasi di Class Main

```java
        // Memanggil metode untuk menampilkan leaf nodes
        System.out.print(s:"Leaf Nodes: ");
        bt.printLeafNodes(bt.root);
        System.out.println(x:"");
    }
```

- Hasil:

```
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
Nilai paling kecil dalam pohon: 3
Nilai paling besar dalam pohon: 15
Leaf Nodes: 3 5 7 15
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

4. Berikut method untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```java
// Method untuk menghitung jumlah leaf nodes
int countLeafNodes(Node27 node) {
  if (node == null) {
      return 0;
  }
  if (node.left == null && node.right == null) {
      return 1;
  } else {
      return countLeafNodes(node.left) + countLeafNodes(node.right);
  }
}
```

- Implementasi di Class Main

```java
// Memanggil metode untuk menghitung jumlah leaf nodes
int leafCount = bt.countLeafNodes(bt.root);
System.out.println("Jumlah Leaf Nodes: " + leafCount);
}
```

- Hasil:

```
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
Nilai paling kecil dalam pohon: 3
Nilai paling besar dalam pohon: 15
Leaf Nodes: 3 5 7 15
Jumlah Leaf Nodes: 4
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

5. Berikut Modifikasi class BinaryTreeArray, dan menambahkan :
   • method add(int data) untuk memasukan data ke dalam tree

```java
//Jawaban Tugas Praktikum Nomer 5
package Jobsheet13;

public class BinaryTreeArray27 {
    int[] data;
    int idxLast;

    public BinaryTreeArray27() {
        data = new int[10]; // Default size
        idxLast = -1;
    }

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

    void add(int data) {
        if (idxLast == this.data.length - 1) {
            System.out.println(x:"Tree is full!");
            return;
        }
        this.data[++idxLast] = data;
    }
```

   • method traversePreOrder() dan traversePostOrder()

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

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

- Implementasi di Class Main

```java
package Jobsheet13;

public class BinaryTreeArrayMain27 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray27 bta = new BinaryTreeArray27();

        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(data, idxLast);

        System.out.print(s:"InOrder Traversal: ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();

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

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

        // Adding new data to the tree
        bta.add(data:10);
        bta.add(data:2);

        System.out.println(x:"");
        System.out.print(s:"InOrder Traversal Setelah adding: ");
        bta.traverseInOrder(idxStart:0);
        System.out.println();

        System.out.print(s:"PreOrder Traversal Setelah adding: ");
        bta.traversePreOrder(idxStart:0);
        System.out.println();

        System.out.print(s:"PostOrder Traversal Setelah adding: ");
        bta.traversePostOrder(idxStart:0);
        System.out.println();
```

- Hasil:

```
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

InOrder Traversal Setelah adding: 10 3 2 4 5 6 7 8 9
PreOrder Traversal Setelah adding: 6 4 3 10 2 5 8 7 9
PostOrder Traversal Setelah adding: 10 2 3 5 4 7 9 8 6
PS D:\COLLEGE\SEMESTER 2\P.STRUKTUR DATA>
```

Commit dan Push Ke Github:

https://github.com/SuryaRf/27_Surya_ASD/tree/main/Jobsheet13