NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

Percobaan 1

```java
package P12;

public class Node15 {
    int data;
    Node15 left;
    Node15 right;

    Node15() {

    }

    Node15(int data) {
        this.data = data;
    }
}
```

```java
package P12;

public class BinaryTreeMain15 {
    Run | Debug
    public static void main(String[] args) {

        BinaryTree15 bt = new BinaryTree15();
        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.traverseIndOrder(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:"");
    }
}
```

NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

```java
package P12;

public class BinaryTree15 {

    Node15 root;

    BinaryTree15() {
        root = null;
    }

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

    void add(int data) {
        if (isEmpty()) {
            root = new Node15(data);
        } else {
            Node15 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left != null) {
                        current = current.left;
                    } else {
                        current.left = new Node15(data);
                        break;
                    }
                } else if (data > current.data) {
                    if (current.right != null) {
                        current = current.right;
                    } else {
                        current.right = new Node15(data);
                        break;
                    }
                } else {
                    break;
                }
            }
        }
    }

    boolean find(int data) {
        boolean result = false;
        Node15 current = root;
        while (current != null) {
            if (current.data == data) {
                result = true;
                break;
            } else if (current.data < data) {
                current = current.right;
            } else {
                current = current.left;
            }
        }
        return result;
    }

    void traversePreOrder(Node15 node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traversePostOrder(Node15 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.data);
        }
    }

    void traverseIndOrder(Node15 node) {
        if (node != null) {
            traverseIndOrder(node.left);
            System.out.print(" " + node.data);
            traverseIndOrder(node.right);
        }
    }

    Node15 getSuccessor(Node15 del) {
        Node15 successor = del.right;
        Node15 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;
        }
        Node15 parent = root;
        Node15 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;
            }
        }
        if (current == null) {
            System.out.println("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 {
                Node15 successor = getSuccessor(current);
                if (current == root) {
                    root = successor;
                } else {
                    if (isLeftChild) {
                        parent.left = successor;
                    } else {
                        parent.right = successor;
                    }
                }
                successor.left = current.left;
            }
        }
    }

}
```

NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

```
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
```

Pertanyaan :

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

   Jawab : Proses pencarian data dalam binary search tree (BST) lebih efektif dibandingkan dengan binary tree biasa karena struktur terurutnya, di mana setiap node memiliki nilai kiri lebih kecil dan kanan lebih besar, sehingga mempermudah navigasi dan pembagian ruang pencarian, yang mengarahkan pencarian ke subtree kiri atau kanan serta mengurangi ruang pencarian setengahnya setiap langkah, menghasilkan waktu pencarian lebih cepat dengan kompleksitas O(log n) dalam BST yang seimbang dibandingkan O(n) pada binary tree biasa.

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

   Jawab : Di class Node15, atribut left dan right digunakan untuk merepresentasikan hubungan hirarkis dalam struktur data binary search tree (BST), dimana atribut left ini menunjuk ke node anak kiri dari node saat ini, dan atribut right menunjuk ke node anak kanan dari node saat ini. Sehingga kedua atribut tersebut memungkinkan pembentukan struktur tree di mana setiap node bisa memiliki dua anak, yang pada gilirannya bisa memiliki anak mereka sendiri, sehingga membentuk struktur hierarkis dari binary search tree. Struktur ini memungkinkan operasi pencarian, penyisipan, dan penghapusan dilakukan secara efisien.

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

   Jawab : Atribut root dalam class BinaryTree15 digunakan untuk menunjukkan node pertama atau akar dari tree. Itu adalah titik awal dari tree dan dari sana, kita bisa mengakses seluruh struktur tree.

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

   Jawab : Ketika objek tree pertama kali dibuat, nilai dari root adalah null karena pada saat itu belum memiliki node apa pun. Itu menunjukkan bahwa tree kosong.

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

   Jawab : Dalam class BinaryTree15 ketika tree masih kosong maka sebuah node baru dengan data yang diberikan akan dibuat kemudian node baru tersebut ditetapkan sebagai root (akar) setelah node baru ditambahkan sebagai root, maka tree tidak lagi kosong

NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

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;
        }
}
```

Jawab : Baris program tersebut bertujuan untuk menemukan posisi yang tepat untuk menambahkan node baru dengan nilai data di sebelah kiri node saat ini (current) dalam tree. Tahapannya ialah sebagai berikut, pada baris if (data < current.data), kita memeriksa apakah nilai data kurang dari nilai current.data. Jika benar, maka kita masuk ke dalam blok if. Pada baris if (current.left != null), kita memeriksa apakah node kiri dari current tidak null. Jika tidak null, itu berarti ada node di sebelah kiri current, sehingga kita pindahkan current ke node kiri tersebut untuk melanjutkan pencarian tempat yang tepat untuk menyisipkan node baru. Jika node kiri current adalah null (artinya tidak ada node di sebelah kiri current), maka kita menemukan tempat yang tepat untuk menyisipkan node baru. Oleh karena itu, kita buat node baru dengan nilai data dan menghubungkannya ke node kiri current. Setelah itu, kita keluar dari loop while dengan menggunakan break, karena kita telah menemukan tempat yang tepat dan tidak perlu melanjutkan pencarian.

Percobaan 2

```java
package P12;

public class BinaryTreeArray15 {
    int[] data;
    int idxLast;

    BinaryTreeArray15() {
        data = new int[10];
    }

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

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

NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

```java
package P12;

public class BinaryTreeArrayMain15 {
    Run | Debug
    public static void main(String[] args) {

        BinaryTreeArray15 bta = new BinaryTreeArray15();
        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.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}

```

```
InOrder Traversal :  3 4 5 6 7 8 9
```

Pertanyaan :

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

   Jawab : Dalam class BinaryTreeArray, atribut data dan idxLast memiliki peran penting dalam mengelola representasi dan traversal dari tree biner yang diimplementasikan menggunakan array, dimana atribut data digunakan untuk menyimpan elemen-elemen pohon biner dalam bentuk array, sementara idxLast menunjukkan indeks elemen terakhir yang valid untuk menentukan batas traversal pohon.

2. Apakah kegunaan dari method populateData()?

   Jawab : Method populateData() pada kelas BinaryTreeArray15 digunakan untuk mengisi array data dan mengatur indeks terakhir (atau node terakhir yang valid) dari binary tree yang direpresentasikan dalam array tersebut.

3. Apakah kegunaan dari method traverseInOrder()?

   Jawab : Method traverseInOrder() pada kelas BinaryTreeArray15 digunakan untuk melakukan traversal in-order (kiri, root, kanan) pada pohon biner yang diimplementasikan menggunakan array.

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

   Jawab : Dalam sebuah binary tree yang disimpan dalam array, posisi left child (anak kiri) dan right child (anak kanan) dari suatu node yang berada di indeks i dapat ditentukan menggunakan rumus berikut:
   Left child node akan disimpan pada indeks 2*i + 1.

NAMA : HIZKIA ELSADANTA
NIM : 2341720253
KELAS : TI-1G
MATERI : TREE

Right child node akan disimpan pada indeks 2*i + 2.
Oleh karena itu jika suatu node binary tree disimpan dalam array indeks 2, maka left child-nya berada di indeks 5, dan right child-nya berada di indeks 6.

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

   Jawab : Statement idxLast = 6 digunakan sebagai parameter untuk menentukan indeks terakhir dari array yang akan dipopulasikan ke dalam objek BinaryTreeArray15 melalui metode populateData(). Dengan nilai 6, itu menunjukkan bahwa hanya elemen-elemen indeks 0 hingga 6 dari array data yang akan dimasukkan ke dalam objek bta, sementara elemen-elemen lainnya (indeks 7 hingga akhir) dianggap tidak ada atau kosong. Oleh karena itu statement int idxLast = 6; memberikan informasi kepada objek bta tentang seberapa banyak dari array data yang harus diambil menjadi bagian dari struktur pohon biner yang direpresentasikan oleh objek tersebut.

Link GitHub :

https://github.com/Kiaakk/Algoritma_Struktur_Data_1G_15.git