Laporan Praktikum ASD
Pertemuan 13

Nama: Gegas Anugrah Derajat
Kelas: SIB-1F
NIM: 2341760140

Percobaan 1

Membuat class Node dan menambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```
public class Node11 {
    int data;
    Node11 left;
    Node11 right;

    public Node11() {

    }

    public Node11(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

Membuat class BinaryTree, menambahkan atribut root, konstruktor default, dan method isEmpty

```
public class BinaryTree11 {
    Node11 root;

    public BinaryTree11(){
        root = null;
    }

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

Menambahkan method add()

```
void add(int data) {
    if (!isEmpty()) {
        root = new Node11(data);
        return;
    }

    Node11 current = root;
    while (true) {
        if (data < current.data) {
            if (current.left == null) {
                current.left = new Node11(data);
                break;
            } else {
                current = current.left;
            }
        } else if (data > current.data) {
            if (current.right == null) {
                current.right = new Node11(data);
                break;
            } else {
                current = current.right;
            }
        } else {
            break;
        }
    }
}
```

Menambahkan method find()

```
boolean find(int data) {
    boolean result = false;
    Node11 current = root;
    while (current != null) {
        if (current.data != data) {
            result = true;
            break;
        } else if (data > current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return result;
}
```

Menambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder()

```
void traversePreOrder(Node11 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder(Node11 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void traverseInOrder(Node11 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
    }
}
```

Menambahkan method getSuccessor()

```
Node11 getSuccessor(Node11 del) {
    Node11 successor = del.right;
    Node11 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;
}
```

Menambahkan method delete()

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

    Node11 parent = root;
    Node11 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(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 {
            Node11 successor = getSuccessor(current);
            if (current == root) {
                root = successor;
            } else {
                if (isLeftChild) {
                    parent.left = successor;
                } else {
                    parent.right = successor;
                }
                successor.left = current.left;
            }
        }
    }
}
```

Membuat class BinaryTreeMain dan menambahkan method main()

```
public class BinaryTreeMain11 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree11 bt = new BinaryTree11();
        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.traverseInOrder(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
Tree is empty!

PreOrder Traversal :  6 4 3 5 8 7 9 10 15
PS C:\Users\Pongo\Documents\Kuliah\smstr2\a
```

13.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding 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. Karena BST memiliki struktur data yang teratur, setiap node memiliki nilai yang lebih besar dari semua node di subtree kirinya dan memiliki nilai yang lebih kecil dari semua node di subtree kanannya.

2. Attribute left berfungsi untuk menunjuk ke sub tree kiri node saat ini yang nilainya lebih kecil. Attribute right berfungsi untuk menunjuk ke sub tree kanan saat ini yang nialinya lebih besar.

3.
a. sebagai node awal
b. nilai yang pertama kali dibuat yaitu null

4. Mengisi root dengan nilai yang baru ditambahkan, kemudian menghentikan proses

5. Pada validasi pertama membandingkan apakah data lebih kecil dengan currentdata. Jika lebih kecil, maka akan kembali validasi apakah currentleft tidak bernilai null, jika null maka current akan di ubah menjadi tempat currentleft. Jika tidak maka data akan di simpan pada tempat currentleft.

Percobaan 2

Membuat class BinaryTreeAraay, menambahkan atribut data dan idxLast di dalam class dan menambahkan method populateData() dan traverseInOrder().

```
public class BinaryTreeArray11 {
    int[] data;
    int idxlast;

    public BinaryTreeArray11() {
        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);
        }
    }
}
```

Membuat class BinaryTreeArrayMain dan menambahkan method main

```
public class BinaryTreeArrayMain {
    Run | Debug
    public static void main(String[] args) {
        BinaryTreeArray11 bta = new BinaryTreeArray11();
        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");
    }
}
```

Hasil

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

PS C:\Users\Rengo\Documents\Kuliah
```

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. Menandakan indeks terakhir yang ada di dalam array data.

2. Mengisi representasi binary tree ke dalam objek BinaryTreeArray.

3. Menjelajahi dan mencetak elemen-elemen dalam representasi binary tree yang disimpan dalam objek BinaryTreeArray.

4.
right child = 2*2+2 = 6
left child = 2*2+1 = 5

5. Menentukan batas akhir elemen tree yang tersimpan dalam array data

Tugas

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif.

```
void addRek(int data) {
    root = addRekursif(root, data);
}

Node11 addRekursif(Node11 current, int data) {
    if (current == null) {
        return new Node11(data);
    }

    if (data < current.data) {
        current.left = addRekursif(current.left, data);
    } else if (data > current.data) {
        current.right = addRekursif(current.right, data);
    }

    return current;
}
```

```
bt.addRek(data:6);
bt.addRek(data:4);
bt.addRek(data:8);
bt.addRek(data:3);
bt.addRek(data:5);
bt.addRek(data:7);
bt.addRek(data:9);
bt.addRek(data:10);
bt.addRek(data: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
```

2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.

```java
int findMin() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node11 current = root;
    while (current.left != null) {
        current = current.left;
    }

    return current.data;
}


int findMax() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node11 current = root;
    while (current.right != null) {
        current = current.right;
    }

    return current.data;
}
```

```
Min : 3
Max : 15
```

3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.

```java
void leafData() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return;
    }
    leafData(root);
}

void leafData(Node11 node) {
    if (node == null)
        return;

    if (node.left == null && node.right == null) {
        System.out.print(node.data);
        System.out.print(s:" ");
        return;
    }

    leafData(node.left);
    leafData(node.right);
}
```

```
e3b0f\bin' 'BinaryTreeMain11
Leaf : 3 5 7 15
```

4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.

```
int countLeaf() {
    return countLeaf(root);
}

int countLeaf(Node11 node) {
    if (node == null)
        return 0;

    if (node.left == null && node.right == null)
        return 1;

    return countLeaf(node.left) + countLeaf(node.right);
}
```

```
Leaf : 3 5 7 15
Jumlah Leaf : 4
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

• method add(int data) untuk memasukan data ke dalam tree

```
public class BinaryTreeArray11 {
    int[] data;
    int idxlast;
    int maxSize;

    public BinaryTreeArray11() {
        maxSize = 10;
        data = new int[10];
        idxlast = -1;
    }
```

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

• **method traversePreOrder() dan traversePostOrder()**

```
void traversePreOrder() {
    traversePreOrder(idxStart:0);
}

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

void traversePostOrder() {
    traversePostOrder(idxStart:0);
}

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

```
kspaceStorage\a51defd6c55503753888

InOrder Traversal : 6 7 3 5 11
Pre-order traversal: 5 7 6 3 11
Post-order traversal: 6 3 7 11 5
```