Adinda Wahyu Luchmansyahvira

2141720096 / TI-1H / 05

JOBSHEET XIII

TREE

2.1 Implementasi Binary Search Tree menggunakan Linked List

2.1.1 Tahapan percobaan

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 Node, BinaryTree dan BinaryTreeMain
2. Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```
package js_13;
public class Node {
    int data;
    Node left;
    Node right;
    public Node(){
    }
    public Node(int data){
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

3. Di dalam class BinaryTree, tambahkan atribut root

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

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

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

5. Tambahkan method add() di dalam class BinaryTree. 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 Node(data);
    }else{
        Node current =root;
        while(true){
            if(data < current.data){
                if(current.left!= null){
                    current = current.left;
                }else{
                    current.left = new Node(data);
                    break;
                }
            }else if(data > current.data){
                if(current.right != null){
                current = current.right;
            }else{
                current.right = new Node(data);
                break;
            }
        }else{
            break;
            }
        }
    }
}
```

6. Tambahkan method find()

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

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

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

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

```
Node getSuccessor(Node del){
    Node successor = del.right;
    Node 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.print("Tree is Empty!");
        return;
    }
    Node parent = root;
    Node 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;
        }
    }
```

   Kemudian tambahkan proses penghapusan terhadap node current yang telah ditemukan.

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

10. Buka class BinaryTreeMain dan tambahkan method main()

```
package js_13;
public class BinaryTreeMain {
    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();
        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);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find "+bt.find(5));
        bt.delete(8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

11. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
12. Amati hasil running tersebut.

```
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

2.1.2 Pertanyaan Percobaan

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

- Karena pada binary search tree semua anak disebelah kiri (left-child) harus lebih kecil daripada anak disebelah kanan (right-child) dan parentnya, sedangkan binary tree biasa tidak. Binary search tree adalah binary tree yang seluruh children dari tiap node terurut sehingga dalam pencarian data jauh lebih efektif dari awal sampai akhir.

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

- Digunakan untuk menyimpan index. Dimana left untuk menyimpan index yang nilainya lebih kecil dari root, sedangkan right untuk menyimpan index yang nilainya lebih besar dari root.

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   - Untuk menentukan node pertama yang dibuat dalam tree yang tidak memiliki predessesor dan juga sebagai penunjukan

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?
   - Nilai root ketika objek tree pertama kali dibuat adalah bernilai null atau kosong.
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?

- Proses yang akan terjadi adalah proses add dimana node baru tersebut langsung masuk dan menjadi root dalam sebuah 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;
        }
```

- Ketika data kurang dari current.data kemudian current.left tidak bernilai null, maka akan dilakukan current = current.left dan apabila current.left bernilai null, maka proses yang dilakukan current.left = new Node(data) atau penambahan node baru dan break (berhenti).

2.2 Implementasi binary tree dengan array

2.2.1 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 BinaryTreeArray dan BinaryTreeArrayMain
3. Buat atribut data dan idxLast di dalam class BinaryTreeArray. Buat juga method populateData() dan traverseInOrder().

```
package js_13;
public class BinaryTreeArray {
    int[] data;
    int idxLast;
    public BinaryTreeArray() {
        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);
        }
    }
}
```

4. Kemudian dalam class BinaryTreeArrayMain buat method main() seperti gambar berikut ini.

```
package js_13;
public class BinaryTreeArrayMain {
    public static void main(String[] args) {
        BinaryTreeArray bta = new BinaryTreeArray();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder(0);
    }
}
```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ jobsheet_adinda ---
3 4 5 6 7 8 9
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

13.2.1 Pertanyaan Percobaan

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

- Data untuk menyimpan data dan idxLast untuk menyimpan batas data saat dilakukan print out

2. Apakah kegunaan dari method populateData()?

- Fungsi dari method populateData() adalah untuk menyimpan data yang dimasukkan atau untuk atribut data dan idxLast

3. Apakah kegunaan dari method traverseInOrder()?

- Fungsi dari method traverseInOrder() adalah untuk mencetak / menampilkan data dengan InOrder

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

- Left child dari node tersebut akan berada di index 2i + 1. Right child dari node tersebut akan berada di index 2i + 2. Maka left child berada pada index ke-5 dan right child berada index ke 6

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

- Fungsi dari statement int idxLast = 6 adalah untuk menentukan batas melakukan print dengan batas indeks yaitu 6

13.3 Tugas Praktikum

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

```
    void insert(int data){
        root = addRecursive(root, data);
    }
    Node addRecursive (Node root, int data){
        if(root == null){
            root = new Node(data);
            return root;
        }
        if(data < root.data){
            root.left = addRecursive(root.left, data);
        }else if(data > root.data){
            root.right = addRecursive(root.right, data);
        }
        return root;
    }
}
```

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

```
    public int findMin(Node node){
        if(node == null){
            return Integer.MAX_VALUE;
        }
        int minimal = node.data;
        int left = findMin(node.left);
        int right = findMin(node.right);
        if(left < minimal){
            minimal = left;
        }
        if (right < minimal){
            minimal = right;
        }
        return minimal;
    }
```

```
    public int findMax(Node node) {
        if (node == null) {
            return Integer.MIN_VALUE;
        }
        int maximum = node.data;
        int left = findMax(node.left);
        int right = findMax(node.right);
        if (left > maximum) {
            maximum = left;
        }
        if (right > maximum) {
            maximum = right;
        }
        return maximum;
    }
}
```

OUTPUT

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ jobsheet_adinda ---
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Max = 15
Min = 3
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

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

```
    public void dataLeaf(Node node) {
        if (node == null) {
            return;
        }
        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
            return;
        }
        if (node.left != null) {
            dataLeaf(node.left);
        }
        if (node.right != null) {
            dataLeaf(node.right);
        }
    }
```

OUTPUT

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ jobsheet_adinda ---
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Max = 15
Min = 3
3 5 7 15

------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

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

```
    public void jumlahLeaf(Node node) {
        if (node == null) {
            return;
        }
        if (node.left == null && node.right == null) {
            size++;
            return;
        }
        if (node.left != null) {
            jumlahLeaf(node.left);
        }
        if (node.right != null) {
            jumlahLeaf(node.right);
        }
    }
}
```

OUTPUT

```
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6

 find true
 6 4 3 5 9 7 10 15

 Max = 15
 Min = 3
3 5 7 15

 Jumlah leaf pada tree : 4
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

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

```
    public void add(int data, int idx){
        this.data[idx] = data;
    }
```

• method traversePreOrder() dan traversePostOrder()

```
    public void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            if (data[idxStart] == 0) {
                System.out.print(idxLast + " ");
            } else {
                System.out.print(data[idxStart] + " ");
            }
            traverseInOrder((2 * idxStart) + 1);
            traverseInOrder((2 * idxStart) + 2);
        }
    }
    public void traversePostOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traverseInOrder((2 * idxStart) + 1);
            traverseInOrder((2 * idxStart) + 2);
            if (data[idxStart] == 0) {
                System.out.print(idxLast + " ");
            } else {
                System.out.print(data[idxStart] + " ");
            }
        }
    }
}
```

OUTPUT

```
--- exec-maven-plugin:1.5.0:exec (default-cli) @ jobsheet_adinda ---
3 4 5 6 7 8 9
3 4 5 7 8 9 6
6 3 4 5 7 8 9
2 4 5 6 7 8 9
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```