PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

JOBSHEET PERTEMUAN KE-14

RIO TRI PRAYOGO

TI 1A

26

2341720236

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

Tree

Praktikum 1 : Implementasi Binary Search Tree menggunakan Linked List

Percobaan :

Node26

```
package minggu14;
public class Node26 {
    int data;
    Node26 left;
    Node26 right;

    public Node26() {

    }

    public Node26(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

BinaryTree26

```
package minggu14;
public class BinaryTree26 {
    Node26 root;

    public BinaryTree26() {
        root = null;
    }

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

    void add(int data) {
        if (!isEmpty()) {
```

```
            root = new Node26(data);
        } else {
            Node26 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left != null) {
                        current = current.left;
                    } else {
                        current.left = new Node26(data);
                        break;
                    }
                } else if (data > current.data) {
                    if (current.right != null) {
                        current = current.right;
                    } else {
                        current.right = new Node26(data);
                        break;
                    }
                } else {
                    break;
                }
            }
        }
    }

    boolean find(int data) {
        boolean result = false;
        Node26 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;
    }

    void traversePreOrder(Node26 node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traversePostOrder(Node26 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.data);
        }
    }

    void traverseInOrder(Node26 node) {
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(" " + node.data);
            traverseInOrder(node.right);
        }
    }

    Node26 getSuccessor(Node26 del) {
        Node26 successor = del.right;
        Node26 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;
        }
        Node26 parent = root;
        Node26 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 {
        Node26 successor = getSuccessor(current);
        if (current == root) {
            root = successor;
        } else {
            if (isLeftChild) {
                parent.left = successor;
            } else {
                parent.right = successor;
```

```
                    }
                    successor.left = current.left;
                }
            }
        }
    }
}
```

BinaryTreeMain26

```
package minggu14;
public class BinaryTreeMain26 {
    public static void main(String[] args) {
        BinaryTree26 bt = new BinaryTree26();
        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("");
    }
}
```

Output :

```
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?
   - Karena *Binary Search Tree* memiliki sifat dimana semua **left-child** harus lebih kecil daripada **right-child** dan **parent**-nya. Sehingga data menjadi terurut dan pencarian data menjadi lebih efisien.
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   - Atribut left dan right memiliki kegunaan sebagai penunjuk/pointer dari child kanan dan kiri.
     a. Untuk apakah kegunaan dari atribut root di dalam class **BinaryTree**?
        - Atribut **root** memiliki kegunaan sebagai data paling atas atau pertama dimana memiliki sifat tidak memiliki *predesesor*.
     b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
        - Ketika objek tree pertama kali dibuat nilai **root** adalah *null* yang menandakan bahwa tree masih kosong.
3. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   - Ketika tree masih kosong, dan ditambahkan sebuah node baru maka node baru tersebut akan menjadi **root** baru dari tree tersebut.
4. 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;
        }
 }
```

   - Baris program diatas berguan untuk menaruh data baru ke dalam menjadi **left-child** karena data yang baru diinputkan lebih kecil daripada data yang sudah ada. **Current** merupakan **root**/data yang sudah ada dan **data** merupakan data baru Baris pertama mengecek apakah data baru lebih kecil dari data yang sudah ada, jika iya maka masuk ke pengecekkan selanjutnya. Selanjutnya dicek lagi apakah **left-child** dari data yang sudah ada isinya atau tidak jika iya maka `current.left` akan menjadi **current** baru, jika tidak maka `current.left` akan diisikan menjadi data yang baru diinputkan.

Praktikum 2 : Implementasi Binary Tree Dengan Array

Percobaan :

BinaryTreeArray26

```
package minggu14;

public class BinaryTreeArray26 {
    int[] data;
    int idxLast;

    public BinaryTreeArray26() {
        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);
        }
    }
}
```

BinaryTreeArrayMain26

```
package minggu14;

public class BinaryTreeArrayMain26 {
    public static void main(String[] args) {
        BinaryTreeArray26 bta = new BinaryTreeArray26();
        int[] data = { 6, 4, 8, 3, 5, 7, 9, 0, 0, 0 };
        int idxLast = 6;
        bta.populateData(data, idxLast);
```

```
        System.out.print("\nInOrder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("\n");
    }
}
```

Output :

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

Pertanyaan :

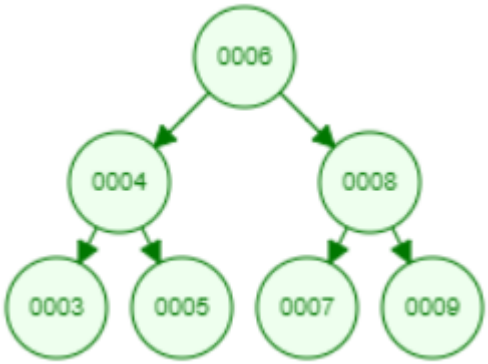
1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
   - Dalam class **BinaryTreeArray**, atribut `data` berfungsi sebagai wadah dari data dalam bentuk array. Sedangkan `idxLast` berfungsi untuk menunjukkan index array terakhir yang akan digunakan.
2. Apakah kegunaan dari method **populateData()**?
   - Method **populateData()** berguna sebagai inisialisasi atau pengisian array dalam objek.
3. Apakah kegunaan dari method **traverseInOrder()**?
   - Method **traverseInOrder()** berfungsi untuk menelusuri seluruh data yang ada dalam tree menggunakan penelusuran *InOrder*.
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   - Jika node disimpan dalam array index ke-2 maka untuk menentukan posisi **left-child** bisa menggunakan rumus '**2*i+1**' dimana i adalah 2 maka posisi **left-child** berada pada index ke-5. Sementara posisi **right-child** menggunakan rumus '**2*i+2**' yang menunjukkan bahwa posisi **right-child** berada pada index ke-6.
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   - Statement `int idxLast = 6` berguna untuk menunjukkan index terakhir dari array data yang akan ditelusuri menggunakan *InOrder Traversal*.

Tugas Praktikum

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

Jawab :

a) **Visualisasi Binary Tree:**



b) **Nomor 1-4**

- **Kode**

  Node26

```
package minggu14.tugas;


public class Node26 {
    int data;
    Node26 left, right;


    public Node26(int data) {
        this.data = data;
        left = right = null;
    }
}
```

BinaryTree26

```
package minggu14.tugas;

public class BinaryTree26 {
    Node26 root;

    public BinaryTree26() {
        root = null;
    }

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

    Node26 addRecursive(Node26 root, int data) {
        if (root == null) {
            root = new Node26(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;
    }

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

    void traversePreOrder(Node26 node) {
        if (node != null) {
            System.out.print(" " + node.data);
            traversePreOrder(node.left);
            traversePreOrder(node.right);
```

```
        }
    }

    void traversePostOrder(Node26 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(" " + node.data);
        }
    }

    void traverseInOrder(Node26 node) {
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(" " + node.data);
            traverseInOrder(node.right);
        }
    }

    int cariMin() {
        if (isEmpty()) {
            System.out.println("Binary Tree kosong!");
            return -1;
        }
        Node26 current = root;
        while (current.left != null) {
            current = current.left;
        }
        return current.data;
    }

    int cariMax() {
        if (isEmpty()) {
            System.out.println("Binary Tree kosong!");
            return -1;
        }
```

```
        Node26 current = root;
        while (current.right != null) {
            current = current.right;
        }
        return current.data;
    }

    void leaf(Node26 node) {
        if (node == null) {
            return;
        }
        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        }
        leaf(node.left);
        leaf(node.right);
    }

    int countLeaf(Node26 node) {
        if (node == null) {
            return 0;
        }
        if (node.left == null && node.right == null) {
            return 1;
        }
        return countLeaf(node.left) + countLeaf(node.right);
    }
}
```

Main26

```
package minggu14.tugas;


public class Main26 {
    public static void main(String[] args) {
        BinaryTree26 bt = new BinaryTree26();
        bt.add(6);
        bt.add(4);
```

```
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        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("Nilai Terkecil dalam Binary Tree: " +
bt.cariMin());
        System.out.println("Nilai Terbesar dalam Binary Tree: " +
bt.cariMax());
        System.out.print("Data yang merupakan Leaf: ");
        bt.leaf(bt.root);
        System.out.println("");
        System.out.println("Jumlah Leaf yang ada dalam Binary Tree: "
+ bt.countLeaf(bt.root));
    }
}
```

- **Output**

```
Preorder Traversal:  6 4 3 5 8 7 9
InOrder Traversal:  3 4 5 6 7 8 9
PostOrder Traversal:  3 5 4 7 9 8 6
Nilai Terkecil dalam Binary Tree: 3
Nilai Terbesar dalam Binary Tree: 9
Data yang merupakan Leaf: 3 5 7 9
Jumlah Leaf yang ada dalam Binary Tree: 4
```

c) **Nomor 5**

- **Kode**

BinaryTreeArray26

```
package minggu14.tugas;


public class BinaryTreeArray26 {
```

```
    int[] array;
    int idxLast = -1;

    public BinaryTreeArray26() {
        array = new int[15];
    }

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

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

    void traversePreOrder(int idxStart) {
        if (idxStart <= idxLast) {
            System.out.print(array[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(array[idxStart] + " ");
        }
    }
```

```
    void add(int data) {
        if (idxLast == array.length - 1) {
            System.out.println("Binary Tree penuh!");
        } else if (idxLast == -1) {
            array[++idxLast] = data;
        } else {
            int root = 0;
            while (true) {
                if (data <= array[root]) {
                    int leftChild = 2 * root + 1;
                    if (array[leftChild] == 0) {
                        array[leftChild] = data;

                        if (leftChild > idxLast) {
                            idxLast = leftChild;
                        }
                        break;
                    } else {
                        root = leftChild;
                    }
                } else {
                    int rightChild = 2 * root + 2;
                    if (array[rightChild] == 0) {
                        array[rightChild] = data;

                        if (rightChild > idxLast) {
                            idxLast = rightChild;
                        }
                        break;
                    } else {
                        root = rightChild;
                    }
                }
            }
        }
    }
```

```
}
```

MainArray26

```
package minggu14.tugas;


public class MainArray26 {

    public static void main(String[] args) {
        BinaryTreeArray26 bta = new BinaryTreeArray26();
        bta.add(6);
        bta.add(4);
        bta.add(8);
        bta.add(3);
        bta.add(5);
        bta.add(7);
        bta.add(9);
        System.out.print("Preorder Traversal : ");
        bta.traversePreOrder(0);
        System.out.println("");
        System.out.print("InOrder Traversal : ");
        bta.traverseInOrder(0);
        System.out.println("");
        System.out.print("PostOrder Traversal : ");
        bta.traversePostOrder(0);
    }
}
```

- **Output**

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