LAPORAN PRAKTIKUM JOBSHEET 13

TREE

MATA KULIAH ALGORITMA DAN STRUKTUR DATA

Disusun Oleh :

Jami'atul Afifah (2341760102)

SIB-1F

PROGRAM STUDI D4 SISTEM INFOEMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2024

PRAKTIKUM 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

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

Node14.java

```java
public class Node14 {
    int data;
    Node14 left;
    Node14 right;

    public Node14() { }

    public Node14(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
public class BinaryTree14 {
    Node14 root;
```

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

```
public class BinaryTree14 {
    Node14 root;

    public BinaryTree14() {
        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()) { // tree is empty
        root = new Node14(data);
    } else {
        Node14 current = root;
        while (true) {
            if (data < current.data) {
                if (current.left == null) {
                    current.left = new Node14(data);
                    break;
                } else {
                    current = current.left;
                }
            } else if (data > current.data) {
                if (current.right == null) {
                    current.right = new Node14(data);
                    break;
                } else {
                    current = current.right;
                }
            } else { // data already exists
                break;
            }
        }
    }
}
```

6. Tambahkan method find()

```
boolean find(int data) {
    Node14 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(Node14 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traverseInOrder(Node14 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
        traverseInOrder(node.right);
    }
}

void traversePostOrder(Node14 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}
```

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

```
Node14 getSuccessor(Node14 del) {
    Node14 successor = del.right;
    Node14 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;
    }

    Node14 parent = root;
    Node14 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 {
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
```

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

```
// no child
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 there is 1 child (right)
    if (current == root) {
        root = current.right;
    } else {
        if (isLeftChild) {
            parent.left = current.right;
        } else {
            parent.right = current.right;
        }
    }
} else if (current.right == null) { // if there is 1 child (left)
    if (current == root) {
        root = current.left;
    } else {
        if (isLeftChild) {
            parent.left = current.left;
        } else {
            parent.right = current.left;
        }
    }
} else { // if there are 2 children
    Node14 successor = getSuccessor(current);
    if (current == root) {
        root = successor;
    } else {
        if (isLeftChild) {
            parent.left = successor;
        } else {
            parent.right = successor;
        }
        successor.left = current.left;
    }
}
```

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

```java
BinaryTreeMain14.java > ...
public class BinaryTreeMain14 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree14 bt = new BinaryTree14();
        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:"");
    }
}
```

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

```
8aca709a1551\redhat.java\jdt_ws\Jobsheet13_64c3bcf7\bin' 'BinaryTreeMa
in14'
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:\Matkul\SEM 2\ASD\Jobsheet13>
```

PERTANYAAN

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

   Dalam Binary Search Tree (BST), data disusun sedemikian rupa sehingga setiap node memiliki nilai yang lebih besar dari semua nilai di subpohon kirinya dan lebih kecil dari semua nilai di subpohon kanannya. Struktur ini memungkinkan kita untuk mengecilkan ruang pencarian pada setiap langkah pencarian dengan memutuskan untuk hanya mencari di subpohon kiri atau kanan berdasarkan perbandingan nilai. Ini mengurangi kompleksitas pencarian dari O(n) dalam binary tree biasa menjadi O(log n) dalam kasus terbaik BST yang seimbang, karena kita hanya perlu memeriksa satu cabang di setiap tingkat pohon
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
   Dalam kelas Node, atribut left dan right digunakan untuk menunjuk ke anak kiri dan anak kanan dari sebuah node, masing-masing. Struktur ini memungkinkan node untuk memiliki maksimum dua anak dan membentuk pohon biner di mana setiap node dapat terhubung ke dua node lain, satu di kiri dan satu di kanan. Dengan demikian, pohon biner ini dapat digunakan untuk mencari, memasukkan, dan menghapus data dengan efisiensi yang lebih baik dibandingkan dengan struktur data lainnya
3. a.Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   Atribut root dalam kelas BinaryTree digunakan untuk menyimpan referensi ke node paling atas dalam pohon, yang merupakan titik awal semua operasi pada pohon biner. Semua operasi seperti penambahan, pencarian, dan penghapusan dimulai dari node root ini.
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
   Ketika objek pohon pertama kali dibuat, nilai dari root adalah null, menandakan bahwa pohon tersebut kosong dan belum memiliki node.
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   Ketika pohon masih kosong (root adalah null), dan sebuah node baru akan ditambahkan, node baru tersebut akan menjadi root dari pohon. Ini berarti node tersebut akan ditempatkan di posisi paling atas dari pohon, dan referensi root akan diubah untuk menunjuk ke node baru ini. Dengan demikian, node baru tersebut akan menjadi titik awal untuk semua operasi pencarian, penghapusan, dan pengaturan data di dalam pohon.
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;

        }
```

   - if(data < current.data): Baris ini memeriksa apakah data yang akan ditambahkan lebih kecil dari data pada node saat ini (current). Jika benar, ini berarti node baru seharusnya

berada di subpohon kiri dari node saat ini karena, dalam BST, semua nilai yang lebih kecil ditempatkan di subpohon kiri.

- if(current.left != null): Baris ini memeriksa apakah anak kiri dari node saat ini (current.left) bukan null, yang berarti ada node lain di subpohon kiri.
- current = current.left: Jika anak kiri tidak null, maka node saat ini (current) diperbarui menjadi anak kiri, dan proses pemeriksaan dilanjutkan dari node anak kiri ini.
- else: Jika anak kiri adalah null, itu berarti kita telah menemukan tempat di mana node baru harus ditempatkan.
- current.left = new Node(data);: Node baru dibuat dengan data yang diberikan dan ditempatkan sebagai anak kiri dari node saat ini.
- break;: Pengulangan dihentikan karena node baru telah berhasil ditambahkan ke pohon.

PRAKTIKUM 2

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

prak2 > BinaryTreeArray14.java > ...

```java
package prak2;

public class BinaryTreeArray14 {
    int[] data;
    int idxLast;

    public BinaryTreeArray14() {
        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);
            if (data[idxStart] != 0) {
                System.out.print(data[idxStart] + " ");
            }
            traverseInOrder(2 * idxStart + 2);
        }
    }
}
```

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

```
prak2 > J BinaryTreeArrayMain14.java > ...
1   package prak2;
2
3   public class BinaryTreeArrayMain14 {
        Run | Debug
4       public static void main(String[] args) {
5           BinaryTreeArray14 bta = new BinaryTreeArray14();
6           int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
7           int idxLast = 6;
8           bta.populateData(data, idxLast);
9           System.out.print(s:"\nInOrder Traversal: ");
10          bta.traverseInOrder(idxStart:0);
11          System.out.println(x:"\n");
12      }
13  }
```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

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

PS D:\Matkul\SEM 2\ASD\Jobsheet13>
```

PERTANYAAN

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
    - data: Atribut ini merupakan array yang digunakan untuk menyimpan elemen-elemen dari binary tree. Setiap elemen dalam array ini merepresentasikan sebuah node dalam binary tree.
    - idxLast: Atribut ini menyimpan indeks terakhir dari elemen yang valid (bukan nol) dalam array data. Ini membantu dalam traversal dan pengelolaan elemen-elemen dalam binary tree untuk memastikan bahwa kita tidak mengakses elemen-elemen di luar batas yang valid.
2. Apakah kegunaan dari method **populateData()**?
   populateData(int[] data, int idxLast): Method ini digunakan untuk mengisi array data dengan elemen-elemen yang diberikan dan menetapkan nilai idxLast. Ini memungkinkan untuk menginisialisasi binary tree dengan data yang sudah ada dan memastikan bahwa informasi tentang batas terakhir dari data yang valid diatur dengan benar.
3. Apakah kegunaan dari method **traverseInOrder()**?
   traverseInOrder(int idxStart): Method ini digunakan untuk melakukan traversal in-order pada binary tree yang disimpan dalam array. Traversal in-order berarti mengunjungi node kiri terlebih dahulu, kemudian node saat ini, dan akhirnya node

kanan. Method ini merekur dan mencetak elemen-elemen dalam urutan in-order.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
    - Untuk node yang disimpan pada indeks i dalam array, posisi anak kiri (left child) adalah di indeks 2 * i + 1 dan posisi anak kanan (right child) adalah di indeks 2 * i + 2.
    - Jika node disimpan di indeks 2:
        - Left child: Indeks = 2 * 2 + 1 = 5
        - Right child: Indeks = 2 * 2 + 2 = 6
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   Statement ini digunakan untuk menetapkan nilai idxLast, yang merupakan indeks terakhir dari elemen valid dalam array data. Dalam kasus ini, elemen-elemen valid dalam array berada dari indeks 0 hingga 6. Hal ini membantu memastikan bahwa traversal tidak melampaui batas elemen-elemen valid dalam array dan hanya memproses elemen-elemen yang merupakan bagian dari binary tree.

TUGAS PRAKTIKUM

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

Node14.java

```java
public class Node14 {
    int data;
    Node14 left;
    Node14 right;

    public Node14() { }

    public Node14(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

BinaryTree14.java

BinaryTree14.java > BinaryTree14

```java
public class BinaryTree14 {
    Node14 root;

    public BinaryTree14() {
        root = null;
    }

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

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

    private Node14 addRecursive(Node14 current, int data) {
        if (current == null) {
            return new Node14(data);
        }

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

        return current;
    }

    void traversePreOrder(Node14 node) {
        if (node != null) {
            System.out.print(node.data + " ");
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traverseInOrder(Node14 node) {
        if (node != null) {
            traverseInOrder(node.left);
            System.out.print(node.data + " ");
            traverseInOrder(node.right);
        }
    }

    void traversePostOrder(Node14 node) {
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            System.out.print(node.data + " ");
        }
    }
}

```

BinaryTreeMain14.java

BinaryTreeMain14.java > BinaryTreeMain14

```java
public class BinaryTreeMain14 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree14 bt = new BinaryTree14();
        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);

        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:"");
    }
}
```

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

BinaryTree14.Java

```java
// terkecil
public int findMin() {
    if (isEmpty()) {
        throw new IllegalStateException(s:"Tree is empty");
    }
    return findMinRecursive(root);
}

private int findMinRecursive(Node14 node) {
    return node.left == null ? node.data : findMinRecursive(node.left);
}

// terbesar
public int findMax() {
    if (isEmpty()) {
        throw new IllegalStateException(s:"Tree is empty");
    }
    return findMaxRecursive(root);
}

private int findMaxRecursive(Node14 node) {
    return node.right == null ? node.data : findMaxRecursive(node.right);
}

void traversePreOrder(Node14 node) {
    if (node != null) {
        System.out.print(node.data + " ");
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}
```

BinaryTreeMain14.java

```java
public class BinaryTreeMain14 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree14 bt = new BinaryTree14();
        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);

        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("Minimum value: " + bt.findMin());
        System.out.println("Maximum value: " + bt.findMax());
    }
}
```

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
Minimum value: 3
Maximum value: 9
PS D:\Matkul\SEM 2\ASD\Jobsheet13>
```

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

BinaryTree14.java

```java
// leaf
public void printLeafNodes() {
    printLeafNodesRecursive(root);
}

private void printLeafNodesRecursive(Node14 node) {
    if (node != null) {
        if (node.left == null && node.right == null) {
            System.out.print(node.data + " ");
        }
        printLeafNodesRecursive(node.left);
        printLeafNodesRecursive(node.right);
    }
}
```

BinaryTreeMain14.java

```java
System.out.print(s:"Leaf Nodes: ");
bt.printLeafNodes();
System.out.println(x:"");
```

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
Leaf Nodes: 3 5 7 9
Minimum value: 3
Maximum value: 9
PS D:\Matkul\SEM 2\ASD\Jobsheet13>
```

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

BinaryTree14.java

```java
// Hitung Leaf
public int countLeafNodes() {
    return countLeafNodesRecursive(root);
}

private int countLeafNodesRecursive(Node14 node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    return countLeafNodesRecursive(node.left) + countLeafNodesRecursive(node.right);
}
```

BinaryTreeMain14.java

```java
int leafCount = bt.countLeafNodes();
System.out.println("Number of Leaf Nodes: " + leafCount);
```

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
Leaf Nodes: 3 5 7 9
Minimum value: 3
Maximum value: 9
Number of Leaf Nodes: 4
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :
    a. method add(int data) untuk memasukan data ke dalam tree
    b. method traversePreOrder() dan traversePostOrder()

prak2 > J BinaryTreeArray14.java > ...

```java
package prak2;

public class BinaryTreeArray14 {
    int[] data;
    int idxLast;

    public BinaryTreeArray14() {
        data = new int[10];
    }

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

    void add(int data) {
        if (idxLast < this.data.length) {
            this.data[++idxLast] = data;
        } else {
            System.out.println("Tree is full, cannot add more data.");
        }
    }
```

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

    private void traversePreOrder(int idx) {
        if (idx <= idxLast && data[idx] != 0) {
            System.out.print(data[idx] + " ");
            traversePreOrder(2 * idx + 1);
            traversePreOrder(2 * idx + 2);
        }
    }

    void traversePostOrder() {
        traversePostOrder(0);
    }

    private void traversePostOrder(int idx) {
        if (idx <= idxLast && data[idx] != 0) {
            traversePostOrder(2 * idx + 1);
            traversePostOrder(2 * idx + 2);
            System.out.print(data[idx] + " ");
        }
    }
}
```