LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR

TREE

ABIM MUSTAWA

244107020078

KELAS TI-1B

PRODI D-IV TEKNIK

INFORMATIKA JURUSAN

TEKNOLOGI INFORMASI

POLITEKNIK NEGERI

MALANG 2025

1. Percobaan Praktikum

1.1 Praktikum 1

Langkah-langkah Percobaan

1. Pada Project yang sudah dibuat pada pertemuan sebelumnya, buat package dengan nama Pertemuan14
2. Tambahkan class-class berikut: a. Mahasiswa00.java b. Node00.java c. BinaryTree00.java d. BinaryTreeMain00.java Ganti 00 dengan nomer absen Anda.
3. Implementasikan Class Mahasiswa00, Node00, BinaryTree00 sesuai dengan diagram class berikut ini:
4. Di dalam class Mahasiswa00, deklarasikan atribut sesuai dengan diagram class Mahasiswa di atas. Tambahkan juga konstruktor dan method sesuai diagram di atas.

```java
public class Mahasiswa01 {
    String nim;
    String nama;
    String kelas;
    double ipk;

    public Mahasiswa01() {
    }

    public Mahasiswa01(String nim, String nama, String kelas, double ipk)
        this.nim = nim;
        this.nama = nama;
        this.kelas = kelas;
        this.ipk = ipk;
    }

    public void tampilInformasi() {
        System.out.println("NIM: " + this.nim + " " +
                           "Nama: " + this.nama + " " +
                           "Kelas: " + this.kelas + " " +
                           "IPK: " + this.ipk);
    }
}
```

5. Di dalam class Node00, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```java
public class Node01 {
    Mahasiswa01 mahasiswa;
    Node01 left, right;

    public Node01() {
    }

    public Node01(Mahasiswa01 mahasiswa) {
        this.mahasiswa = mahasiswa;
        left = right = null;
    }
}
```

6. Di dalam class BinaryTree00, tambahkan atribut root.

```
public class BinaryTree01 {
    Node01 root;
```

7. Tambahkan konstruktor dan method isEmpty() di dalam class BinaryTree00.

```
public BinaryTree01() {
    root = null;
}

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

8. Tambahkan method add() di dalam class BinaryTree00. Node ditambahkan di binary search tree pada posisi sesuai dengan besar nilai IPK mahasiswa. Di bawah ini proses penambahan node tidak dilakukan secara rekursif, agar lebih mudah dilihat alur proses penambahan node dalam tree. Sebenarnya dengan proses rekursif penulisan kode akan lebih efisien.

```
public void add(Mahasiswa01 mahasiswa) {
    Node01 newNode = new Node01(mahasiswa);
    if (isEmpty()) {
        root = newNode;
    } else {
        Node01 current = root;
        Node01 parent = null;
        while (true) {
            parent = current;
            if (mahasiswa.ipk < current.mahasiswa.ipk) {
                current = current.left;
                if (current == null) {
                    parent.left = newNode;
                    return;
                }
            } else {
                current = current.right;
                if (current == null) {
                    parent.right = newNode;
                    return;
                }
            }
        }
    }
}
```

9. Tambahkan method find()

```
public void add(Mahasiswa01 mahasiswa) {
    Node01 newNode = new Node01(mahasiswa);
    if (isEmpty()) {
        root = newNode;
    } else {
        Node01 current = root;
        Node01 parent = null;
        while (true) {
            parent = current;
            if (mahasiswa.ipk < current.mahasiswa.ipk) {
                current = current.left;
                if (current == null) {
                    parent.left = newNode;
                    return;
                }
            } else {
                current = current.right;
                if (current == null) {
                    parent.right = newNode;
                    return;
                }
            }
        }
    }
}
```

10. Tambahkan method traversePreOrder(), traverseInOrder() dan traversePostOrder(). Method traverse digunakan untuk mengunjungi dan menampilkan node-node dalam binary tree, baik dalam mode pre-order, in-order maupun post-order.

```
void traversePreOrder(Node01 node) {
    if (node != null) {
        node.mahasiswa.tampilInformasi();
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traverseInOrder(Node01 node) {
    if (node != null) {
        traverseInOrder(node.left);
        node.mahasiswa.tampilInformasi();
        traverseInOrder(node.right);
    }
}

void traversePostOrder(Node01 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        node.mahasiswa.tampilInformasi();
    }
}
```

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

```
Node01 getSuccessor(Node01 del) {
    Node01 successor = del.right;
    Node01 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;
}
```

12. Tambahkan method delete(). Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak, cari posisi node yang akan dihapus.

```
void delete(double ipk) {
    if (isEmpty()) {
        System.out.println(x:"Binary tree kosong");
        return;
    }

    // Cari node yang akan dihapus
    Node01 parent = root;
    Node01 current = root;
    boolean isLeftChild = false;
    while (current != null) {
        if (current.mahasiswa.ipk == ipk) {
            break;
        } else if (ipk < current.mahasiswa.ipk) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else if (ipk > current.mahasiswa.ipk) {
            parent = current;
            current = current.right;
            isLeftChild = false;
        }
    }
```

13. Kemudian tambahkan proses penghapusan di dalam method delete() terhadap node current yang telah ditemukan.

```
// Proses penghapusan
if (current == null) {
    System.out.println(x:"Data tidak ditemukan");
    return;
} else {
    // Jika tidak ada anak (leaf)
    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) { // Jika hanya punya 1 anak (kanan)
        if (current == root) {
            root = current.right;
        } else {
            if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
    } else if (current.right == null) { // Jika hanya punya 1 anak (kiri)
        if (current == root) {
            root = current.left;
        } else {
            if (isLeftChild) {
                parent.left = current.left;
            } else {
                parent.right = current.left;
            }
        }
    } else { // Jika punya 2 anak
        Node01 successor = getSuccessor(current);
        if (current == root) {
            root = successor;
        } else {
            if (isLeftChild) {
                parent.left = successor;
            } else {
                parent.right = successor;
            }
        }
        successor.left = current.left;
    }
}
```

14. Buka class BinaryTreeMain00 dan tambahkan method main() kemudian tambahkan kode berikut ini:

```java
public class BinaryTreeMain01 {
    public static void main(String[] args) {
        // Menambahkan data mahasiswa
        bst.add(new Mahasiswa01(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57));
        bst.add(new Mahasiswa01(nim:"244160227", nama:"Badar", kelas:"B", ipk:3.85));
        bst.add(new Mahasiswa01(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.21));
        bst.add(new Mahasiswa01(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.54));

        System.out.println(x:"\nDaftar semua mahasiswa (in order traversal):");
        bst.traverseInOrder(bst.root);

        System.out.println(x:"\nPencarian data mahasiswa:");
        System.out.print(s:"Cari mahasiswa dengan IPK 3.54 : ");
        String hasilCari = bst.find(ipk:3.54) ? "Ditemukan" : "Tidak ditemukan";
        System.out.println(hasilCari);
        System.out.print(s:"Cari mahasiswa dengan IPK 3.22 : ");
        hasilCari = bst.find(ipk:3.22) ? "Ditemukan" : "Tidak ditemukan";
        System.out.println(hasilCari);

        bst.add(new Mahasiswa01(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.72));
        bst.add(new Mahasiswa01(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.37));
        bst.add(new Mahasiswa01(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.46));

        System.out.println(x:"\nDaftar semua mahasiswa setelah penambahan 3 mahasiswa:");
        bst.traverseInOrder(bst.root);

        System.out.println(x:"\nPreOrder Traversal:");
        bst.traversePreOrder(bst.root);

        System.out.println(x:"\nPostOrder Traversal:");
        bst.traversePostOrder(bst.root);

        // Menghapus mahasiswa dengan IPK 3.57
        bst.delete(ipk:3.57);
        System.out.println(x:"\nDaftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):");
        bst.traverseInOrder(bst.root);
```

15. Compile dan jalankan class BinaryTreeMain00 untuk mendapatkan simulasi jalannya program binary tree yang telah dibuat.

Verifikasi Hasil Percobaan

```
Daftar semua mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160227 Nama: Badar Kelas: B IPK: 3.85

Pencarian data mahasiswa:
Cari mahasiswa dengan IPK 3.54 : Ditemukan
Cari mahasiswa dengan IPK 3.22 : Tidak ditemukan

Daftar semua mahasiswa setelah penambahan 3 mahasiswa:
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160227 Nama: Badar Kelas: B IPK: 3.85

PreOrder Traversal:
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160227 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

PostOrder Traversal:
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160227 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
```

```
Daftar semua mahasiswa setelah penghapusan 1 mahasiswa (in order traversal):
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.21
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.37
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.46
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.54
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160227 Nama: Badar Kelas: B IPK: 3.85
PS C:\Users\lenovo\OneDrive\Documents\Sturktur Data\Jobsheet14>
```

Pertanyaan

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

   Jawab : Karena BST memiliki sifat bahwa semua node di subtree kiri memiliki nilai lebih kecil dari root, dan semua node di subtree kanan memiliki nilai lebih besar dari root. Struktur ini memungkinkan pencarian dengan kompleksitas waktu.

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

   Jawab : Atribut left dan right digunakan untuk menyimpan referensi ke node anak kiri dan anak kanan dari node saat ini. Ini membentuk struktur hierarkis dari tree.

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?

   Jawab :
   a. Atribut root menyimpan referensi ke node paling atas (akar) dari tree. Semua operasi pada tree dimulai dari root.
   b. Ketika tree pertama kali dibuat, nilai root adalah null karena belum ada node yang ditambahkan.

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

   Jawab : Jika tree kosong (root == null), node baru akan langsung menjadi root dari tree tersebut.

5. Perhatikan method add(), di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

   Jawab :
   - parent = current; - Menyimpan node saat ini sebagai parent sebelum berpindah ke child
   - if (mahasiswa.ipk < current.mahasiswa.ipk) - Memeriksa apakah IPK

mahasiswa baru lebih kecil dari IPK node saat ini

- current = current.left; - Jika ya, pindah ke child kiri
- if (current == null) - Jika child kiri kosong
- parent.left = newNode; - Tambahkan node baru sebagai child kiri dari parent
- Proses serupa dilakukan untuk child kanan jika IPK lebih besar

6. Jelaskan langkah-langkah pada method delete() saat menghapus sebuah node yang memiliki dua anak. Bagaimana method getSuccessor() membantu dalam proses ini?

   Jawab :

   - Cari successor (node dengan nilai terkecil di subtree kanan) menggunakan method getSuccessor()
   - Ganti node yang akan dihapus dengan successor
   - Hubungkan subtree kiri dari node yang dihapus ke successor
   - Method getSuccessor() membantu menemukan node pengganti yang tepat untuk mempertahankan sifat Binary Search Tree.

1.2 Praktikum 2

Langkah-langkah 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 BinaryTreeArray00 dan BinaryTreeArrayMain00. Ganti 00 dengan nomer absen Anda.
3. Buat atribut data dan idxLast di dalam class BinaryTreeArray00. Buat juga method populateData() dan traverseInOrder().

```java
public class BinaryTreeArray01 {
    Mahasiswa01[] dataMahasiswa;
    int idxLast;

    public BinaryTreeArray01() {
        this.dataMahasiswa = new Mahasiswa01[10];
    }

    void populateData(Mahasiswa01[] dataMhs, int idxLast) {
        this.dataMahasiswa = dataMhs;
        this.idxLast = idxLast;
    }

    void traverseInOrder(int idxStart) {
        if (idxStart <= idxLast) {
            if (dataMahasiswa[idxStart] != null) {
                traverseInOrder(2 * idxStart + 1);
                dataMahasiswa[idxStart].tampilInformasi();
                traverseInOrder(2 * idxStart + 2);
            }
        }
    }
```

4. Kemudian dalam class BinaryTreeArrayMain00 buat method main() dan tambahkan kode seperti gambar berikut ini di dalam method main().

```java
BinaryTreeArray01 bta = new BinaryTreeArray01();
Mahasiswa01 mhs1 = new Mahasiswa01(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57);
Mahasiswa01 mhs2 = new Mahasiswa01(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.41);
Mahasiswa01 mhs3 = new Mahasiswa01(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.75);
Mahasiswa01 mhs4 = new Mahasiswa01(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.35);
Mahasiswa01 mhs5 = new Mahasiswa01(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.48);
Mahasiswa01 mhs6 = new Mahasiswa01(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.61);
Mahasiswa01 mhs7 = new Mahasiswa01(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.86);

Mahasiswa01[] dataMahasiswa = {mhs1, mhs2, mhs3, mhs4, mhs5, mhs6, mhs7, null, null, null};
int idxLast = 6;
bta.populateData(dataMahasiswa, idxLast);

System.out.println(x:"\nInorder Traversal Mahasiswa: (Praktikum2)");
bta.traverseInOrder(idxStart:0);

BinaryTreeArray01 bta2 = new BinaryTreeArray01();
bta2.add(mhs1);
bta2.add(mhs2);
bta2.add(mhs3);
bta2.add(mhs4);
bta2.add(mhs5);
bta2.add(mhs6);
bta2.add(mhs7);
```

Verifikasi Hasil Percobaan

```
Inorder Traversal Mahasiswa: (Praktikum2)
NIM: 244160220 Nama: Dewi Kelas: B IPK: 3.35
NIM: 244160185 Nama: Candra Kelas: C IPK: 3.41
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.48
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160205 Nama: Ehsan Kelas: D IPK: 3.61
NIM: 244160221 Nama: Badar Kelas: B IPK: 3.75
NIM: 244160170 Nama: Fizi Kelas: B IPK: 3.86
```

Pertanyaan

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

   Jawab :
   - data: Array untuk menyimpan elemen-elemen binary tree
   - idxLast: Indeks terakhir yang berisi data dalam array

2. Apakah kegunaan dari method populateData()?

   Jawab : Untuk mengisi data binary tree sekaligus dari array yang sudah ada dan menetapkan indeks terakhir yang valid

3. Apakah kegunaan dari method traverseInOrder()?

   Jawab : Untuk melakukan traversal in-order (kiri-root-kanan) pada binary tree yang disimpan dalam array

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

   Jawab :
   - Left child: 2 * 2 + 1 = indeks 5
   - Right child: 2 * 2 + 2 = indeks 6

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

   Jawab : Menetapkan bahwa indeks terakhir yang berisi data dalam array adalah 6 (elemen ke-7, karena indeks dimulai dari 0)

6. Mengapa indeks 2*idxStart+1 dan 2*idxStart+2 digunakan dalam pemanggilan rekursif, dan apa kaitannya dengan struktur pohon biner yang disusun dalam array?

   Jawab : Karena dalam representasi array, left child dari node di indeks i berada di 2i+1, dan right child di 2i+2. Ini mempertahankan hubungan hierarkis tree dalam struktur array.

2. Latihan Praktikum

2.1 Tugas Praktikum

a. Buat method di dalam class BinaryTree00 yang akan menambahkan node dengan cara rekursif (addRekursif()).
b. Buat method di dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (cariMinIPK() dan cariMaxIPK()) yang ada di dalam binary search tree.
c. Buat method dalam class BinaryTree00 untuk menampilkan data mahasiswa dengan IPK di atas suatu batas tertentu, misal di atas 3.50 (tampilMahasiswaIPKdiAtas(double ipkBatas)) yang ada di dalam binary search tree.
d. Modifikasi class BinaryTreeArray00 di atas, dan tambahkan :
   • method add(Mahasiswa data) untuk memasukan data ke dalam binary tree
   • method traversePreOrder()