Laporan Jobsheet 13 Algoritma dan Struktur Data

Tree

244107020151

Ghazwan Ababil

Teknik Informatika / TI – E (11)

Politeknik Negeri Malang

Jurusan Teknologi Informasi

2025

1. Praktikum

1.1. Percobaan 1

1. Pada folder Praktikum-ASD membuat folder baru dengan nama Jobsheet13
2. Menambahkan class-class:
   a. Mahasiswa11.java
   b. Node11.java
   c. BinaryTree11.java
   d. BinaryTreeMain11.java
3. Mengimplementasikan class Mahasiswa11 sesuai dengan class diagram

```
package Jobsheet13;
public class Mahasiswa11 {
    String nim,nama,kelas;
    double ipk;
    Mahasiswa11(){}
    Mahasiswa11(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.printf("NIM: %s Nama: %s Kelas: %s IPK:
%s\n",this.nim,this.nama,this.kelas,this.ipk);
    }
}
```

4. Mengimplementasikan class Node11 sesuai dengan gambar pada langkah percobaan dan class diagram pada Node11

```
package Jobsheet13;
public class Node11 {
    Mahasiswa11 mahasiswa;
    Node11 left, right;

    public Node11(){}

    public Node11(Mahasiswa11 mahasiswa){
        this.mahasiswa = mahasiswa;
        this.right = this.left = null;
    }
}
```

5. Pada class BinaryTree11, menambahkan attribute root serta mengimplementasikan method-method yang terdapat pada konsep tree seperti:
   - isEmpty(), method untuk mengecek apakah tree kosong.
   - add(), method untuk menambahkan node pada tree dengan posisi sesuai dengan besar nilai IPK mahasiswa
   - find(), untuk mencari data yang diinginkan pada tree

- traversePreOrder(), traverseInOrder(), dan traversePostOrder(), untuk menampilkan node yang ada pada binary tree dengan berbagai cara
- getSuccessor(), method untuk mencari successor pada node. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child.
- delete(), method untuk menghapus sebuah node pada binary tree

6. Class BinaryTree11

```
package Jobsheet13;
public class BinaryTree11 {
    Node11 root;

    BinaryTree11(){};

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

    public void add(Mahasiswa11 mahasiswa){
        Node11 newNode = new Node11(mahasiswa);
        if (isEmpty()) root = newNode;
        else{
            Node11 current = root;
            Node11 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;
                    }
                }
            }
        }
    }

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

```
    void traversePreOrder(Node11 node){
        if (node != null) {
            node.mahasiswa.tampilInformasi();
            traversePreOrder(node.left);
            traversePreOrder(node.right);
        }
    }

    void traverseInOrder(Node11 node){
        if (node != null) {
            traverseInOrder(node.left);
            node.mahasiswa.tampilInformasi();
            traverseInOrder(node.right);
        }
    }

    void traversePostOrder(Node11 node){
        if (node != null) {
            traversePostOrder(node.left);
            traversePostOrder(node.right);
            node.mahasiswa.tampilInformasi();
        }
    }

    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;
    }
```

```
    void delete(double ipk){
        if (isEmpty()) {
            System.out.println("Binary tree kosong");
            return;
        }
        Node11 parent = root;
        Node11 current = root;
        boolean isLeftChild = false;
        // Traverse to find correct node
        while (current != null) {
            if (current.mahasiswa.ipk == ipk) break;
            if (current.mahasiswa.ipk < ipk) {
                parent = current;
                current = current.left;
                isLeftChild = true;
            }
            if (current.mahasiswa.ipk > ipk) {
                parent = current;
                current = current.right;
                isLeftChild = false;
            }
        }

        if (current == null) {
            System.out.println("Data tidak ditemukan");
            return;
        }

        // node is leaf condition
        else if (current.left == null && current.right == null) {
            if (current == root) root = null;
            else if (isLeftChild) parent.left = null;
            else parent.right = null;
        }

        // node has child (right only)
        else if (current.left == null) {
            if (current == root) root = current.right;
            else if (isLeftChild) parent.left = current.right;
            else parent.right = current.right;

        }

        // node has child (left only)
        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);
            System.out.println("Jika 2 anak, current = ");
            successor.mahasiswa.tampilInformasi();
            if (current == root) root = successor;
            else if(isLeftChild) parent.left = successor;
            else parent.right = successor;
            successor.left = current.left;
        }
    }
}
```

7. Pada class BinaryTreeMain11, ditambahkan fungsi main pada class BinaryTreeMain11, kemudian membuat object dari class BinaryTree11. class BinaryTreeMain11 digunakan untuk mengeksekusi semua method yang ada pada class BinaryTree11.

```
package Jobsheet13;
public class BinaryTreeMain11 {
    public static void main(String[] args) {
        BinaryTree11 bst = new BinaryTree11();
        bst.add(new Mahasiswa11("244160121","Ali","A",3.57));
        bst.add(new Mahasiswa11("244160221","Badar","B",3.85));
        bst.add(new Mahasiswa11("244160185","Candra","C",3.21));
        bst.add(new Mahasiswa11("244160220","Dewi","B",3.54));

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

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

        System.out.println("Cari mahasiswa dengan ipk: 3.22 : ");
        hasilCari = bst.find(3.22)?"Ditemukan":"Tidak Ditemukan";
        System.out.println(hasilCari);

        bst.add(new Mahasiswa11("244160131","Devi","A",3.72));
        bst.add(new Mahasiswa11("244160205","Ehsan","D",3.37));
        bst.add(new Mahasiswa11("244160170","Fizi","B",3.46));
        System.out.println("\nDaftar semua mahasiswa setelah penambahan 3
mahasiswa:");
        System.out.println("InOrder Traversal: ");
        bst.traverseInOrder(bst.root);
        System.out.println("PreOrder Traversal: ");
        bst.traversePreOrder(bst.root);
        System.out.println("PostOrder Traversal: ");
        bst.traversePostOrder(bst.root);

        System.out.println("Penghapusan data mahasiswa");
        bst.delete(3.57);
        System.out.println("\nDaftar semua mahasiswa setelah penghapusan
1 mahasiswa (in order traversal):");
        bst.traverseInOrder(bst.root);
    }
}
```

8. 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: 244160221 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:
InOrder 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: 244160121 Nama: Ali Kelas: A IPK: 3.57
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72
NIM: 244160221 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: 244160221 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: 244160221 Nama: Badar Kelas: B IPK: 3.85
NIM: 244160121 Nama: Ali Kelas: A IPK: 3.57
```

```
Penghapusan data mahasiswa
Jika 2 anak, current =
NIM: 244160131 Nama: Devi Kelas: A IPK: 3.72

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: 244160221 Nama: Badar Kelas: B IPK: 3.85
```

9. Pertanyaan

    1. Pada binary search tree proses pencarian data bisa lebih efektif dilakukan dibandingkan binary tree biasa karena, pada binary search tree (BST) data di dalamnya telah terurutkan sehingga dapat dilakukan pencarian dengan metode binary search yang mengimplementasikan divide and conquer. Binary tree biasa hal tersebut

tidak dapat dilakukan karena data pada binary tree masih belum terurut sehingga metode pencarian tidak dapat dengan binary search melainkan dengan sequential.

2. Pada class Node11, atribut left dan right digunakan untuk menunjuk child milik parent maupun root. Atribut left berfungsi untuk menunjuk ke child sebuah parent yang memiliki nilai lebih kecil dari parent. Sedangkan atribut right berfungsi untuk menunjuk ke child sebuah parent yang memiliki nilai lebih besar dari parent.
3. 3a. Pada struktur data tree, root berfungsi sebagai pusat (data teratas) yang berfungsi menghubungkan data (node) lain dalam tree.

   3b. Ketika objek tree pertama kali dibuat, root bernilai null atau kosong karena saat objek tree pertama kali dibuat, tree tersebut masih dalam keadaan kosong sehingga root bernilai null atau kosong.
4. Ketika menambahkan node baru pada tree ketika kondisi tree masih kosong, maka node tersebut akan dijadikan sebagai root dalam struktur tree tersebut.
5. Pada bagian kode tersebut digunakan untuk penambahan node pada tree ketika kondisi tree tidak kosong. Parent digunakan sebagai penanda untuk node yang digunakan sebagai parent pada proses traverse, kemudian dilakukan pengecekan apakah node data baru yang ingin ditambahkan pada tree lebih kecil dari parent (dalam hal ini current) saat ini, jika benar maka traverse current akan berjalan ke arah kiri (left pointer) kemudian dilakukan pengecekan apakah current yang traverse ke arah left tersebut null atau kosong, jika benar atau kosong maka node baru akan ditambahkan disana. Sedangkan jika tidak kosong, maka akan melanjutkan ke perulangan selanjutnya. Lalu sama halnya jika node data baru yang ingin ditambahan pada tree lebih besar dari parent saat ini, maka proses traverse current akan berjalan ke kanan (right pointer) karena pada BST pointer right hanya untuk data yang lebih besar dari parent. Kemudian dilakukan proses yang sama seperti kondisi left pointer dengan perbedaan current yang traverse ke arah right.
6. Ketika menghapus node dengan dua child dalam binary search tree, maka perlu untuk melakukan penggantian node tersebut dengan mencari successor (penerus) nya, kemudian menghapus successor tersebut dari posisi aslinya. Method getSuccessor() berfungsi untuk menemukan successor yang cocok dengan node yang akan dihapus pada proses tersebut dengan cara bergerak ke pointer right (kanan) sekali lalu mencari node terkecil pada pointer left (kiri) milik subtree pada pointer right (kanan) milik node yang akan dihapus.

1.2. Percobaan 2 Implementasi Binary Tree dengan Array

1. Pada percobaan ini mengimplementasikan tree pada array dengan membuat class berikut
   - BinaryTreeArray11
   - BinaryTreeArrayMain11

2. Menambahkan atribut data dan idxLast serta mengimplementasikan method populateData() dan traverseInOrder() pada class BinaryTreeArray11.

```
package Jobsheet13;

public class BinaryTreeArray11 {
    Mahasiswa11[] dataMahasiswa;
    int idxLast;

    public BinaryTreeArray11(){
        this.dataMahasiswa = new Mahasiswa11[10];
    }

    void populateData(Mahasiswa11 dataMhs[], int idxLast){
        this.dataMahasiswa = dataMhs;
        this.idxLast = idxLast;
    }

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

3. Menambahkan method main pada BinaryTreeArrayMain11 untuk mengeksekusi method pada objek dari class BinaryTreeArray11.

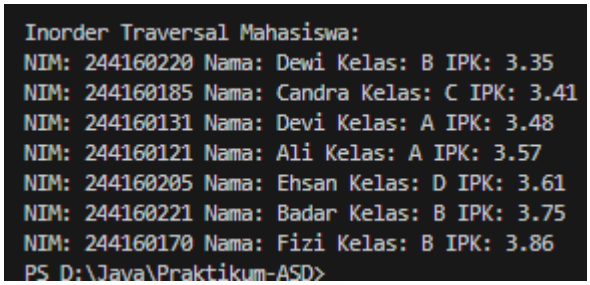
```
package Jobsheet13;

public class BinaryTreeArrayMain11 {
    public static void main(String[] args) {
        BinaryTreeArray11 bta = new BinaryTreeArray11();
        Mahasiswa11 mhs1 = new Mahasiswa11("244160121","Ali","A",3.57);
        Mahasiswa11 mhs2 = new
Mahasiswa11("244160185","Candra","C",3.41);
        Mahasiswa11 mhs3 = new Mahasiswa11("244160221","Badar","B",3.75);
        Mahasiswa11 mhs4 = new Mahasiswa11("244160220","Dewi","B",3.35);

        Mahasiswa11 mhs5 = new Mahasiswa11("244160131","Devi","A",3.48);
        Mahasiswa11 mhs6 = new Mahasiswa11("244160205","Ehsan","D",3.61);
        Mahasiswa11 mhs7 = new Mahasiswa11("244160170","Fizi","B",3.86);

        Mahasiswa11[] dataMahasiswas =
{mhs1,mhs2,mhs3,mhs4,mhs5,mhs6,mhs7,null,null,null};
        int idxLast = 6;
        bta.populateData(dataMahasiswas, idxLast);
        System.out.println("\nInorder Traversal Mahasiswa: ");
        bta.traveseInOrder(0);
    }
}
```

4. Hasil Percobaan



5. Pertanyaan
    1. Atribut idxLast berfungsi untuk pointer child terakhir pada tree dalam visualisasi array, karena pada tree dengan array jumlah child (data) tidak selalu memenuhi kapasitas array (array.length - 1).
    2. Fungsi method populateData() pada BinaryTreeArray11 yaitu untuk memasukkan data array ke dalam binary tree array.
    3. Method traverseInOrder() pada BinaryTreeArray11 berfungsi untuk menampilkan data pada binary tree array dengan urutan in order yaitu dimulai dari yang terkecil hingga terbesar.
    4. Pada representasi binary tree dengan array child pada pointer left dan right ditentukan dengan cara berikut dengan kondisi root dimulai dari indeks-0:
        a. Left child = 2*i + 1
        b. Right child = 2*i + 2

        Apabila sebuah node tersimpan pada indeks ke-2, maka posisi masing-masing left child dan right child nya adalah:

        a. Left Child = 2*2 + 1 = 5 -> indeks ke-5
        b. Right Child = 2*2 + 2 = 6 -> indeks ke-6
    5. Statement idxLast = 6 berfungsi untuk menandai bahwa data terakhir pada binary tree array terletak pada indeks ke-6, hal tersebut karena jumlah data yang dimasukkan ke dalam objek class BinaryTreeArray11 berjumlah 7 sehingga array.length -1 = 6.
    6. Karena pada binary tree dengan menggunakan array left child sebuah binary tree dirumuskan dengan 2*i+1 dan right child sebuah binary tree dirumuskan dengan 2*i+2, sehingga proses traversal untuk binary tree pada array dapat dilakukan.

2. Tugas

1. Menambahkan method di dalam class BinaryTree11 yang akan menambahkan node dengan cara rekursif (addRekursif()).

   Method addRekursif() pada kelas BinaryTree11.

```
public void addRekursif(Node11 current, Node11 newNode){
    if (isEmpty()) {
        root = newNode;
        return;
    }
    if (current != null && current.mahasiswa.ipk == newNode.mahasiswa.ipk) {
        System.out.println("Data mahasiswa dengan IPK " +
newNode.mahasiswa.ipk + " sudah ada, tidak bisa ditambahkan.");
        return;
    }
    Node11 parent = current;
    if (newNode.mahasiswa.ipk < current.mahasiswa.ipk) {
        if (current.left == null) {
            parent.left = newNode;
            return;
        }
        addRekursif(current.left, newNode);
    } else {
        if (current.right == null) {
            parent.right = newNode;
            return;
        }
        addRekursif(current.right, newNode);
    }
}
```

2. Menambahkan method di dalam class BinaryTree11 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (cariMinIPK() dan cariMaxIPK()) yang ada di dalam binary search tree.

   Method cariMinIPK() pada kelas BinaryTree11.

```
public void cariMinIPK(Node11 current){
    if (isEmpty()) {
        System.out.println("Binary tree kosong");
        return;
    }
    if (current.left == null) {
        System.out.println("Mahasiswa dengan IPK Terendah: ");
        current.mahasiswa.tampilInformasi();
    }
    else cariMinIPK(current.left);
}
```

Method cariMaxIPK() pada kelas BinaryTree11.

```
public void cariMaxIPK(Node11 current){
    if (isEmpty()) {
        System.out.println("Binary tree kosong");
        return;
    }
    if (current.right == null) {
        System.out.println("Mahasiswa dengan IPK Tertinggi:
");
        current.mahasiswa.tampilInformasi();
    }
    else cariMaxIPK(current.right);
}
```

3. Menambahkan method dalam class BinaryTree11 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

Method tampilMahasiswaIPKdiAtas(double ipkBatas) pada class BinaryTree11

```
public void tampilMahasiswaIPKdiAtas(Node11 current,double ipk){
    if (isEmpty()) {
        System.out.println("Binary tree kosong");
        return;
    }
    if (current != null) {
        if (current.mahasiswa.ipk > ipk) {
            current.mahasiswa.tampilInformasi();
        }
        tampilMahasiswaIPKdiAtas(current.left, ipk);
        tampilMahasiswaIPKdiAtas(current.right, ipk);
    }
}
```

4. Menambahkan method method add(Mahasiswa data) untuk memasukan data ke dalam binary tree dan method traversePreOrder() pada BinaryTreeArray11.

   Method add pada class BinaryTreeArray11

```
void add(Mahasiswa11 mahasiswa) {
    int idx = 0;
    while (idx <= dataMahasiswa.length-1) {
        if (dataMahasiswa[idx]!=null && dataMahasiswa[idx].ipk ==
mahasiswa.ipk) {
            System.out.println("Data mahasiswa dengan IPK " + mahasiswa.ipk +
" sudah ada, tidak bisa ditambahkan.");
            return;
        }
        if (dataMahasiswa[idx] == null) {
            dataMahasiswa[idx] = mahasiswa;
            idxLast = idx;
            return;
        }
        else if (mahasiswa.ipk < dataMahasiswa[idx].ipk) {
            idx = 2 * idx + 1;
        } else {
            idx = 2 * idx + 2;
        }
    }
}
```

   Method traversePreOrder pada class BinaryTreeArray11

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

Commit dan Push kode program ke github

```
PS D:\Java\Praktikum-ASD> git add .\Jobsheet13\*
PS D:\Java\Praktikum-ASD> git commit -m "Praktikum Jobsheet 13 Tree Percobaan dan Tugas"
[main e7bed45] Praktikum Jobsheet 13 Tree Percobaan dan Tugas
 6 files changed, 382 insertions(+)
 create mode 100644 Jobsheet13/BinaryTree11.java
 create mode 100644 Jobsheet13/BinaryTreeArray11.java
 create mode 100644 Jobsheet13/BinaryTreeArrayMain11.java
 create mode 100644 Jobsheet13/BinaryTreeMain11.java
 create mode 100644 Jobsheet13/Mahasiswa11.java
 create mode 100644 Jobsheet13/Node11.java
PS D:\Java\Praktikum-ASD> git push -u origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 4 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 3.54 KiB | 604.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ghazwanz/Praktikum-ASD.git
   bb85cbd..e7bed45  main -> main
branch 'main' set up to track 'origin/main'.
```