JOBSHEET XIV
Tree

14.1 Tujuan Praktikum

Setelah melakukan praktikum ini, mahasiswa mampu:

1. memahami model *Tree* khususnya *Binary Tree*
2. membuat dan mendeklarasikan struktur algoritma *Binary Tree*.
3. menerapkan dan mengimplementasikan algoritma *Binary Tree* dalam kasus *Binary Search Tree*

14.2 Kegiatan Praktikum 1

Implementasi Binary Search Tree menggunakan Linked List (100 Menit)

14.2.1 Percobaan 1

Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan linked list.

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:

Mahasiswa
nim: String nama: String kelas: String ipk: double
Mahasiswa() Mahasiswa(nm: String, name: String, kls: String, ipk: double) tampilInformasi(): void

Node
data: Mahasiswa left: Node right: Node
Node (left: Node, data:Mahasiswa, right:Node)

BinaryTree
root: Node
BinaryTree() add(data: Mahasiswa): void

find(ipk: double) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) delete(ipk: double): void

1. Di dalam class **Mahasiswa00**, deklarasikan atribut sesuai dengan diagram class Mahasiswa di atas. Tambahkan juga konstruktor dan method sesuai diagram di atas.

```
1   public class Mahasiswa00 {
2       String nim;
3       String nama;
4       String kelas;
5       double ipk;
6
7       public Mahasiswa00() {
8       }
9
10      public Mahasiswa00(String nim, String nama, String kelas, double ipk) {
11          this.nim = nim;
12          this.nama = nama;
13          this.kelas = kelas;
14          this.ipk = ipk;
15      }
16
17      public void tampilInformasi() {
18          System.out.println("NIM: "+this.nim+" "+
19          "Nama: "+this.nama+" "+
20          "Kelas: "+this.kelas+" "+
21          "IPK: "+this.ipk);
22      }
23  }
```

2. Di dalam class **Node**00, tambahkan atribut **data**, **left** dan **right**, serta konstruktor default dan berparameter.

```
1   public class Node00 {
2       Mahasiswa00 mahasiswa;
3       Node00 left, right;
4
5       public Node00() {
6       }
7
8       public Node00(Mahasiswa00 mahasiswa) {
9           this.mahasiswa = mahasiswa;
10          left = right = null;
11      }
12  }
```

3. Di dalam class **BinaryTree**00, tambahkan atribut **root**.

```
1   public class BinaryTree00 {
2       Node00 root;
3
```

4. Tambahkan konstruktor dan method **isEmpty()** di dalam class **BinaryTree**00**.**

```
4       public BinaryTree00() {
5           root = null;
6       }
7
8       public boolean isEmpty(){
9           return root == null;
10      }
```

5. Tambahkan method **add()** di dalam class **BinaryTree**00. 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.

```
12      public void add(Mahasiswa00 mahasiswa) {
13          Node00 newNode = new Node00(mahasiswa);
14          if (isEmpty()){
15              root = newNode;
16          }else{
17              Node00 current = root;
18              Node00 parent = null;
19              while(true){
20                  parent = current;
21                  if (mahasiswa.ipk < current.mahasiswa.ipk) {
22                      current = current.left;
23                      if (current == null) {
24                          parent.left = newNode;
25                          return;
26                      }
27                  } else {
28                      current = current.right;
29                      if (current == null) {
30                          parent.right = newNode;
31                          return;
32                      }
33                  }
34              }
35          }
36      }
```

6. Tambahkan method **find()**

```
51      boolean find(double ipk) {
52          boolean result = false;
53          Node00 current = root;
54          while (current != null){
55              if (current.mahasiswa.ipk == ipk){
56                  result = true;
57                  break;
58              }else if(ipk > current.mahasiswa.ipk){
59                  current = current.right;
60              }else{
61                  current = current.left;
62              }
63          }
64          return result;
65      }
```

7. 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.

```
38      void traversePreOrder(Node00 node){
39          if (node != null){
40              node.mahasiswa.tampilInformasi();
41              traversePreOrder(node.left);
42              traversePreOrder(node.right);
43          }
44      }
45
46      void traverseInOrder(Node00 node){
47          if (node != null){
48              traverseInOrder(node.left);
49              node.mahasiswa.tampilInformasi();
50              traverseInOrder(node.right);
51          }
52      }
53
54      void traversePostOrder(Node00 node){
55          if (node != null){
56              traversePostOrder(node.left);
57              traversePostOrder(node.right);
58              node.mahasiswa.tampilInformasi();
59          }
60      }
```

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

```
83      Node00 getSuccessor(Node00 del){
84          Node00 successor = del.right;
85          Node00 successorParent = del;
86          while (successor.left != null){
87              successorParent = successor;
88              successor = successor.left;
89          }
90          if (successor !=  del.right){
91              successorParent.left = successor.right;
92              successor.right = del.right;
93          }
94          return successor;
95      }
```

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

```
97      void delete(double ipk){
98          if (isEmpty()){
99              System.out.println(x:"Binary tree kosong");
100             return;
101         }
102         //cari node (current) yang akan dihapus
103         Node00 parent = root;
104         Node00 current = root;
105         boolean isLeftChild = false;
106         while (current != null){
107             if(current.mahasiswa.ipk == ipk){
108                 break;
109             }else if (ipk < current.mahasiswa.ipk){
110                 parent = current;
111                 current = current.left;
112                 isLeftChild = true;
113             }else if(ipk > current.mahasiswa.ipk){
114                 parent = current;
115                 current = current.right;
116                 isLeftChild = false;
117             }
118         }
```

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

```
        //penghapusan
        if (current == null){
            System.out.println("Data tidak ditemukan");
            return;
        }else{
            //jika tidak ada anak (leaf), maka node dihapus
            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
                Node00 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;
            }
        }
    }
```

11. Buka class **BinaryTreeMain**00 dan tambahkan method **main()** kemudian tambahkan kode berikut ini:

```
        BinaryTree00 bst = new BinaryTree00();

        bst.add(new Mahasiswa00(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57));
        bst.add(new Mahasiswa00(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.85));
        bst.add(new Mahasiswa00(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.21));
        bst.add(new Mahasiswa00(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.54));

        System.out.println(x:"\nDaftar semua mahasiswa (in oder 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 Mahasiswa00(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.72));
        bst.add(new Mahasiswa00(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.37));
        bst.add(new Mahasiswa00(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.46));
        System.out.println(x:"\nDaftar semua mahasiswa setelah penambahan 3 mahasiswa:");
        System.out.println(x:"InOrder Traversal:");
        bst.traverseInOrder(bst.root);
        System.out.println(x:"\nPreOrder Traversal:");
        bst.traversePreOrder(bst.root);
        System.out.println(x:"\nPostOrder Traversal:");
        bst.traversePostOrder(bst.root);

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

12. Compile dan jalankan class **BinaryTreeMain**00 untuk mendapatkan simulasi jalannya program binary tree yang telah dibuat.
13. Amati hasil running tersebut.

```
Daftar semua mahasiswa (in oder 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

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
```

14.2.2 Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
5. Perhatikan method **add()**, di dalamnya terdapat baris program seperti di bawah ini. Jelaskan secara detil untuk apa baris program tersebut?

```
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;
    }
}
```

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

14.3 Kegiatan Praktikum 2

Implementasi Binary Tree dengan Array (45 Menit)

14.3.1 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**00 dan **BinaryTreeArrayMain**00**.** Ganti **00** dengan nomer absen Anda.
3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArray**00. Buat juga method **populateData()** dan **traverseInOrder().**

```
1   public class BinaryTreeArray00 {
2       Mahasiswa00[] dataMahasiswa;
3       int idxLast;
4
5       public BinaryTreeArray00() {
6           this.dataMahasiswa = new Mahasiswa00[10];
7       }
8
9       void populateData (Mahasiswa00 dataMhs[], int idxLast){
10          this.dataMahasiswa = dataMhs;
11          this.idxLast = idxLast;
12      }
13
14      void traverseInOrder(int idxStart){
15          if(idxStart <= idxLast){
16              if (dataMahasiswa[idxStart] != null){
17                  traverseInOrder(2*idxStart+1);
18                  dataMahasiswa[idxStart].tampilInformasi();
19                  traverseInOrder(2*idxStart+2);
20              }
21          }
22      }
23  }
```

4. Kemudian dalam class **BinaryTreeArrayMain**00 buat method **main()** dan tambahkan kode seperti gambar berikut ini di dalam method main()**.**

```
3           BinaryTreeArray00 bta = new BinaryTreeArray00();
4           Mahasiswa00 mhs1 = new Mahasiswa00(nim:"244160121", nama:"Ali", kelas:"A", ipk:3.57);
5           Mahasiswa00 mhs2 = new Mahasiswa00(nim:"244160185", nama:"Candra", kelas:"C", ipk:3.41);
6           Mahasiswa00 mhs3 = new Mahasiswa00(nim:"244160221", nama:"Badar", kelas:"B", ipk:3.75);
7           Mahasiswa00 mhs4 = new Mahasiswa00(nim:"244160220", nama:"Dewi", kelas:"B", ipk:3.35);
8
9           Mahasiswa00 mhs5 = new Mahasiswa00(nim:"244160131", nama:"Devi", kelas:"A", ipk:3.48);
10          Mahasiswa00 mhs6 = new Mahasiswa00(nim:"244160205", nama:"Ehsan", kelas:"D", ipk:3.61);
11          Mahasiswa00 mhs7 = new Mahasiswa00(nim:"244160170", nama:"Fizi", kelas:"B", ipk:3.86);
12
13          Mahasiswa00[] dataMahasiswas = {mhs1,mhs2,mhs3,mhs4,mhs5,mhs6,mhs7,null, null, null};
14          int idxLast = 6;
15          bta.populateData(dataMahasiswas,idxLast);
16          System.out.println(x:"\nInorder Traversal Mahasiswa: ");
17          bta.traverseInOrder(idxStart:0);
```

5. Jalankan class **BinaryTreeArrayMain00** dan amati hasilnya!

```
Inorder Traversal Mahasiswa:
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
```

14.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
2. Apakah kegunaan dari method **populateData()**?
3. Apakah kegunaan dari method **traverseInOrder()**?
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masing-masing?
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
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?

14.4 Tugas Praktikum

Waktu pengerjaan: 150 menit

1. Buat method di dalam class **BinaryTree**00 yang akan menambahkan node dengan cara rekursif (**addRekursif**()).
2. Buat method di dalam class **BinaryTree**00 untuk menampilkan data mahasiswa dengan IPK paling kecil dan IPK yang paling besar (**cariMinIPK**() dan **cariMaxIPK**()) yang ada di dalam binary search tree.
3. Buat method dalam class **BinaryTree**00 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.
4. Modifikasi class **BinaryTreeArray**00 di atas, dan tambahkan :
   - method **add(**Mahasiswa data**)** untuk memasukan data ke dalam binary tree
   - method **traversePreOrder()**