LAPORAN PRAKTIKUM
MATA KULIAH ALGORITMA DAN STRUKTUR DATA

PERTEMUAN 14 : TREE

KAYLA RACHMAUDINA SATITI PUTRI

2341760103

D-IV SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI POLITEKNIK NEGERI MALANG

2024

JOBSHEET 13
Tree

13.1 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*

13.2 Kegiatan Praktikum 1 Implementasi Binary Search Tree menggunakan Linked List (45 Menit)

13.2.1 Percobaan 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.

```
public class Node15 {
    int data;
    Node15 right;
    Node15 left;

    public Node15() {}

    public Node15(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
```

3. Di dalam class **BinaryTreeNoAbsen**, tambahkan atribut **root**.

```
public class BinaryTree15 {
    Node15 root;
```

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

```
public BinaryTree15() {
    root = null;
}

boolean isEmpty() {
    return 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 Node15(data);
    }else {
        Node15 current = root;
        while(true){
            if(data>current.data){
                if(current.left == null){
                current = current.left;
                } else{
                current. left = new Node15(data);
                break;
                }
            }else if(data<current.data) {
                if(current.right != null){
                    current = current.right;
                }else{
                    current.right = new Node15(data);
                break;
            }
            }else { //data is already exist
                break;
            }
```

6. Tambahkan method find()

```
boolean find(int data){
    boolean result = false;
    Node15 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;
}
```

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(Node15 node) {
    if (node != null) {
    System.out.print(" "+ node.data);
    traversePreOrder(node.left);
    traversePreOrder(node.right);
    }
}

void traversePostOrder(Node15 node) {
    if (node != null) {
    traversePostOrder(node.left);
    traversePostOrder(node.right);
    System.out.print(" " + node.data);
    }
}

void traverseInOrder(Node15 node) {
    if (node != null) {
    traverseInOrder(node.left);
    System.out.print(" " + node.data);
    traverseInOrder(node.right);
    }
}
```

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

```
Node15 getSuccessor(Node15 del) {
    Node15 successor = del.right;
    Node15 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;
    }
    //find node (current) that will be deleted
    Node15 parent = root;
    Node15 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;
        }
    }
```

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

```
//deletion
 if(current == null){
     System.out.println("Couldn't find data!");
     return;
 }else{
     //if there is no child, simply delete it
     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 is 2 childs
         Node15 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
public class BinaryTreeMain15 {
    Run | Debug
    public static void main(String[] args) {
        BinaryTree15 bt = new BinaryTree15();
        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.

```
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 : false
Delete Node 8
Tree is empty!

PreOrder Traversal  :  6 4 3 5 8 7 9 10 15
```

13.2.2 Pertanyaan Percobaan

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

   Karena binary search tree melakukan pencarian data dengan cara membagi dua pohon secara berulang. Pada setiap pembagian, bagian pohon yang tidak mungkin berisi data yang dicari dapat diabaikan. Hal ini

membuat proses pencarian data pada BST lebih cepat dibandingkan dengan BT biasa, yang tidak memiliki struktur yang teratur dan dapat menyebabkan pencarian yang tidak efisien.

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

   Digunakan untuk menyimpan referensi ke node anak kiri dan kanan dari node tersebut. Dapat menghubungkan node-node dalam pohon biner dan membangun struktur hierarkis yang memungkinkan operasi seperti penambahan, penghapusan, dan pencarian data menjadi lebih efisien.

3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?

   - Menyimpan referensi ke node teratas (akar) dari pohon biner.
   - Memulai operasi traversal pohon, seperti preorder, inorder, dan postorder.
   - Memudahkan akses ke node-node lain dalam pohon melalui operasi traversal.
   - Mempermudah penambahan dan penghapusan node dalam pohon.

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?

   Nilai dari root adalah null. Hal ini karena tree pada awalnya masih kosong dan tidak memiliki node data. Saat node baru ditambahkan ke tree, nilai root akan diubah untuk menunjuk ke node baru tersebut.

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

   - Membuat node baru: Sebuah objek Node baru dibuat dengan nilai data yang ingin ditambahkan.
   - Menyimpan node baru di root: Karena tree masih kosong, node baru ini akan menjadi node akar (root) dari pohon.
   - Memperbarui atribut root: Atribut root dalam objek BinaryTree diubah untuk menunjuk ke node baru yang telah dibuat.

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;
        }
    }
```

   untuk menavigasi pohon agar menemukan posisi yang tepat untuk memasukkan node baru dengan nilai data data

13.3 Kegiatan Praktikum 2 Implementasi binary tree dengan array (45 Menit)

13.3.1 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 **BinaryTreeArrayNoAbsen** dan **BinaryTreeArrayMainNoAbsen**
3. Buat atribut **data** dan **idxLast** di dalam class **BinaryTreeArrayNoAbsen**. Buat juga method **populateData()** dan **traverseInOrder().**

```
public class BinaryTreeArray15 {
    int[] data;
    int idxLast;

    public BinaryTreeArray15(){
    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);
        }
    }
```

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

```
public static void main(String[] args) {
    BinaryTreeArray15 bta = new BinaryTreeArray15();
    int[] data = {6,4,8,3,5,7,9,0,0,0};
    int idxLast = 6;
    bta.populateData(data, idxLast);
    System.out.print(s:"\nInOrder Traversal : ");
    bta.traverseInOrder(idxStart:0);
    System.out.println(x:"\n");
}
```

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

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

13.3.2 Pertanyaan Percobaan

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

Atribut data digunakan untuk menyimpan data-data dalam binary tree. idxLast menunjukkan indeks terakhir dari array yang terisi dengan data. Indeks ini digunakan untuk melacak batas akhir dari data yang tersimpan dalam array.

2. Apakah kegunaan dari method **populateData()**?

   Digunakan untuk mengisi array data dengan data-data yang ingin disimpan dalam bianrry tree.

3. Apakah kegunaan dari method **traverseInOrder()**?

   method ini melakukan traversal pohon biner secara inorder dengan menerima parameter idxStart yang menunjukkan indeks dari node yang ingin dikunjungi pertama kali. Di dalam method, traversal dilakukan secara rekursif dengan menggunakan rumus berikut:

   Kanan: 2 * idxStart + 1

   Kiri: 2 * idxStart + 2

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

   node binary tree disimpan dalam array indeks 2, maka:

   Left child: Berada di indeks 2 * 2 + 1 = 5.

   Right child: Berada di indeks 2 * 2 + 2 = 6.

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

   digunakan untuk menentukan batas akhir dari data yang tersimpan dalam array data. Nilai 6 menunjukkan bahwa hanya 7 elemen pertama dari array data yang terisi dengan data, yaitu dari indeks 0 hingga 6. Elemen-elemen selanjutnya (indeks 7 hingga 9) diasumsikan kosong atau tidak terisi dengan data.

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

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

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

Node15 addRecursive(Node15 current, int data) {
    if (current == null) {
        return new Node15(data);
    }

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

    return current;
}
```

```
bt.addRec(data:6);
bt.addRec(data:4);
bt.addRec(data:8);
bt.addRec(data:3);
bt.addRec(data:5);
bt.addRec(data:7);
bt.addRec(data:9);
bt.addRec(data:10);
bt.addRec(data:15);
```

```
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 : false
Delete Node 8
Tree is empty!

PreOrder Traversal  :  6 4 3 5 8 7 9 10 15
```

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

```
int minValue() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node15 current = root;
    while (current.left != null) {
        current = current.left;
    }

    return current.data;
}
```

```
int maxValue() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return 0;
    }

    Node15 current = root;
    while (current.right != null) {
        current = current.right;
    }

    return current.data;
}
```

Output :

```
Nilai terkecil : 3
Nilai terbesar : 15
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 : false
Delete Node 8
Tree is empty!

PreOrder Traversal  :  6 4 3 5 8 7 9 10 15
```

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

```
void displayLeafData() {
    if (!isEmpty()) {
        System.out.println(x:"Tree is empty!");
        return;
    }
    displayLeafData(root);
}

void displayLeafData(Node15 node) {
    if (node == null)
        return;

    if (node.left == null && node.right == null) {
        System.out.print(node.data);
        System.out.print(s:" ");
        return;
    }

    displayLeafData(node.left);
    displayLeafData(node.right);
}
```

Output :

```
Nilai terkecil      : 3
Nilai terbesar      : 15
Data Leaf           : 3 5 7 15
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 : false
Delete Node 8
Tree is empty!

PreOrder Traversal  :  6 4 3 5 8 7 9 10 15
```

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

```
int countLeaf() {
    return countLeaf(root);
}

int countLeaf(Node15 node) {
    if (node == null)
        return 0;

    if (node.left == null && node.right == null)
        return 1;

    return countLeaf(node.left) + countLeaf(node.right);
}
```

Output :

```
Nilai terkecil      : 3
Nilai terbesar      : 15
Data Leaf           : 3 5 7 15
Jumlah Data Leaf    : 4
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 : false
Delete Node 8
Tree is empty!

PreOrder Traversal  :  6 4 3 5 8 7 9 10 15
```

5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
   - method **add(int data)** untuk memasukan data ke dalam tree

```
void add(int data) {
    if (idxLast < maxSize - 1) {
        idxLast++;
        this.data[idxLast] = data;
    } else {
        System.out.println(x:"Tree is full!");
    }
}
```

- method **traversePreOrder()** dan **traversePostOrder()**

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

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

void traversePostOrder() {
    traversePostOrder(idxStart:0);
}

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

```
bta.add(data:3);
bta.add(data:4);
bta.add(data:2);
bta.add(data:5);
bta.add(data:1);

System.out.print(s:"\nInOrder Traversal    : ");
bta.traverseInOrder(idxStart:0);
System.out.println();

System.out.print(s:"Pre-order traversal  : ");
bta.traversePreOrder();
System.out.println();

System.out.print(s:"Post-order traversal : ");
bta.traversePostOrder();
System.out.println();
```

Output :

```
InOrder Traversal    : 5 4 1 3 2
Pre-order traversal  : 3 4 5 1 2
Post-order traversal : 5 1 4 2 3
```