Praktikum DAA: Binary Search Tree

1. Binary Tree

Binary tree adalah tree yang mempunyai maksimal dua buah child (biasanya disebut dengan left child dan right child). Binary tree dapat direpresentasikan secara objek oriented, dimana tree adalah objek yang dibangun dari sekumpulan objek node. Objek node ini memiliki atribut info. Selain itu, terdapat atribut left, right, dan parent, yang secara berturut-turut adalah node yang menjadi left child, right child, dan parent dari sebuah node.

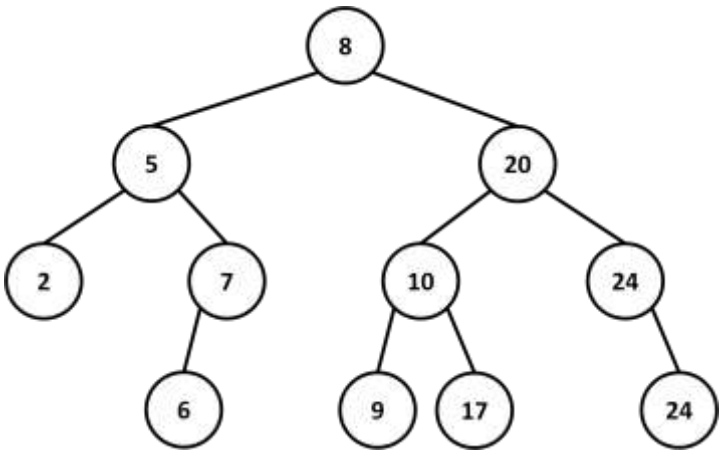
2. Binary Search Tree

Binary search tree adalah binary tree yang memiliki property:

Untuk setiap X yang merupakan sebuah node di dalam binary search tree,

- *Setiap node y yang terdapat pada left subtree dari x memiliki sifat: y.info < x.info.*
- *Setiap node y yang terdapat pada right subtree dari x memiliki sifat: y.info ≥ x.info*.

Contoh BST:



Ada beberapa operasi yang dapat dilakukan pada BST, antara lain:

- Querying (searching, finding maximum/minimum info, finding successor & predecessor)
- Insertion & deletion
- Sorting

Pada modul ini akan diberikan contoh implementasi dari operasi insert, search, dan finding minimum.

Latihan: Membuat Representasi Binary Search Tree dan Menambahkan Method untuk Insert, Search, dan Find Min.

1. Buatlah sebuah kelas dengan nama Node. Kelas ini digunakan untuk merepresentasikan node pada tree. Beri atribut parent, left, right dan info. Tambahkan pula sebuah constructor dengan parameter info bertipe Integer.

```java
/*
*Kelas untuk merepresentasikan Node
*pada tree.
*/
public class Node{
    //atribut info dari sebuah Node
    private int info;
    //atribut parent, anak kiri, dan anak kanan
    private Node parent, left, right;

    Node(int info){
       this.info=info;
       this.parent=null;
       this.left=null;
       this.right=null;
    }
```

2. Karena semua atribut dari kels Node aksesnya adalah *private* ,tambahkan accessor dan mutator untuk kelas Node untuk mengakses dan mengubah nilai dari atribut-atribut tersebut.

```java
    public int getInfo(){
       return this.info;
    }

    public Node getParent(){
       return this.parent;
    }

    public Node getLeft(){
       return this.left;
    }

    public Node getRight(){
       return this.right;
    }

    public void setParent(Node parent){
       this.parent=parent;
    }

    public void setLeft(Node left){
       this.left=left;
    }

    public void setRight(Node right){
       this.right=right;
    }
}
```

3. Untuk mengimplementasikan operasi insert ke dalam sebuah BST, buatlah sebuah sebuah kelas bernama BinarySearchTree. Lengkapi kelas tersebut dengan atribut root.

```java
public class BinarySearchTree{
   private Node root;
```

4. Untuk melakukan insert ke dalam sebuah BST, buatlah method bernama insert dengan parameter info baru yang akan ditambahkan ke dalam BST. Pada slide diberikan pseudocode untuk method insert secara rekursif. Berikut ini alternatif iteratif untuk memasukan node pada tree.

```
//Method untuk memasukan sebuah info ke dalam tree
public void insert(int info){
    Node parent = null;
    Node curr = this.root;
    Node newNode = new Node(info);

    if(this.root == null)
        this.root = newNode;
    else{
        while(curr != null){
            parent = curr;
            if(info < curr.getInfo())
                curr = curr.getLeft();
            else
                curr = curr.getRight();
        }

        newNode.setParent(parent);
        if(info < parent.getInfo())
            parent.setLeft(newNode);
        else
            parent.setRight(newNode);
    }
}
```

5. Untuk melakukan search sebuah info di dalam BST, tambahkan method bernama recSearch. Method ini membutuhkan parameter(Node x, int info). Pada saat dijalankan pertama kali, x adalah root dari Tree. Set hak akses dari method ini dengan private, kemudian bungkuslah dengan sebuah method public dengan nama search. Parameter dari method ini hanya info yang akan dicari di dalam BST, dan akan memanggil method recSearch.

```
//Mencari node yang mengandung info
//Dimulai dari sebuah Node x
//Kemudian akan menelusuri left subtree
//Jika tidak ditemukan akan menelusuri right subtree
private Node recSearch(Node x, int info){
    Node s;
    if(x==null) return null;
    else if(x.getInfo()==info) return x;
    else{
        s=recSearch(x.getLeft(), info);
        if(s!=null) return s;
        else return recSearch(x.getRight(),info);
    }
}

//Melakukan pencarian terhadap sebuah info
//di dalam tree
//pencarian dimulai dari root.
public Node search(int info){
    return this.recSearch(this.getRoot(), info);
}
```

6. Operasi berikutnya adalah mencari node yang mengandung info yang nilainya paling kecil atau minimum. Untuk operasi ini, buatlah method dengan nama FindMin dengan parameter Node x. Sama seperti pada operasi search, method FindMin(Node x) ini harus dibungkus dengan method lain yaitu FindMin().

```
private Node recFindMin(Node x){
    if(x==null) return x;
    else{
        while(x.getLeft()!=null) x=x.getLeft();
        return x;
    }
}
public Node findMin(){
    return (this.recFindMin(this.getRoot()));
}
```

7. Tambahkan method untuk melakukan traversal pada tree dengan cara inorder. Method ini mengembalikan LinkedList yang berisi info node-node pada tree berdasarkan hasil traversal. Untuk keperluan traversal, tambahkan sebuah atribut pada kelas BinarySearchTree yang bernama *result* dan bertipe *LinkedList<Integer>.*

```
public LinkedList<Integer> inorder(){
    this.result = new LinkedList<Integer>();
    recInorder(this.root);
    return result;
}

private void recInorder(Node x){
    if(x!=null){
        recInorder(x.getLeft());
        result.add(x.getInfo());
        recInorder(x.getRight());
    }
}
```

8. Tambahkan kelas Tester untuk menguji implementasi pada kelas Node dan BinarySearchTree

Tugas Praktikum: Lengkapi kelas BinarySearchTree

Lengkapilah kelas BinarySearchTree sehingga dapat dilakukan finding maximum, finding successor, finding predecessor, dan delete pada BST.