Modul

5

BINARY TREE

Tujuan

Mahasiswa diharapkan dapat menerapkan penggunaan binary tree pada suatu kasus. Secara khusus program yang dijalankan pada praktikum inni menerapkan bagian dari binary tree. Mahasiswa juga dapat menggunakan binary tree pada suatu kasus yang akan diselesaikan. Di akhir praktikum ini mahasiswa dapat :

- Menjelaskan konsep dasar binary tree
- Memahami bagaimana menentukan leaf node dan right node pada binary tree
- Memahami bagaimana membentuk suatu binary tree

Pendahuluan

Praktikum ini mengasumsikan bahwa mahasiswa telah dapat mengoperasikan Netbeans. Mahasiswa juga diharapkan sudah memahami dengan baik materi-materi yang telah diberikan sebelumnya (pemrograman dasar I). Agar mahasiswa dapat mencapai tujuan dalam pertemuan praktikum yang pertama ini, mahasiswa harus memiliki pengetahuan (minimal telah membaca) penggunaan konsep dasar binary tree. Bagian dari binary tree yaitu: root, node, left node dan right node. Pemahaman tentang penggunaan tipe data dan variabel juga dibutuhkan dalam praktikum ini.

Proses

Praktikan membaca buku/diktat yang dilakukan selama 20 menit termasuk membuat ringkasan penting, kemudian berdiskusi sesuai dengan panduan aktifitas yang dilakukan selama 20 menit, sedangkan waktu untuk mengerjakan *project* (membuat program/*coding*) secara individual harus diselesaikan di dalam laboratorium dalam waktu 30 menit. Berikutnya untuk sesi latihan, ada soal tentang pengembangan program yang juga harus diselesaikan di laboratorium dalam waktu 50 menit. Terakhir adalah bagian tugas, yaitu *project* yang dikerjakan dirumah dan wajib dikumpulkan pada pertemuan berikutnya.

Aktifitas

1. Mahasiswa membaca buku ajar (jika ada)/diktat kuliah/materi dari sumber lain tentang konsep dasar binary tree sebagai materi bab pertama. Temukan bagian penting dalam topik binary tree ini, kemudian tulis sebagai ringkasan hasil belajar.

2. Mahasiswa berdiskusi tentang definisi binary tree, tujuan adanya root, leaf node,right node child, sebagai bahan diskusi

a. Lengkapilah bagian yang kosong pada class BinaryTreeNode di bawah ini sesuai dengan perintah / command yang tertera.

```
public class BinaryTreeNode{
    BinaryTreeNode parent;
    BinaryTreeNode left;
    BinaryTreeNode right;
    int data;

    /* Set data to new_data
     * set parent, left, and right into null
     */
    BinaryTreeNode(int new_data){
          ……
          ……
          ……
          ……
    }

    /* set other as parent of this node
     * if other is not null, and other's data is bigger than this
data then set this as other's left
     * if other is not null, and other's data is lesser than this
data then set this as other's right
     */
    void set_parent(BinaryTreeNode other){
        ……
        if(……){
            if(……. >  ……){
                ……..
            }else{
                ……..
            }
        }
    }

    /* set other as left of this node
     * if other is not null set other's parent to this node
     */
    void set_left(BinaryTreeNode other){
        ……..
        if(……){
            ……..
        }
    }
/*tobe continued to the next page

    /* set other as right of this node
     * if other is not null set other's parent to this node
     */
    void set_right(BinaryTreeNode other){
```

```
            …….
            if(other != null){
                …….
            }
        }

        /* if this node's parent is not null and parent's left is
this, return true
         * otherwise return false
         */
        boolean is_left(){
            if(……&&…….){
                return true;
            }else{
                return false;
            }
        }
/* if this node's parent is not null and parent's right is this,
return true
         * otherwise return false
         */
        boolean is_right(){
            if(this.parent != null && parent.right == this){
                return true;
            }else{
                return false;
            }
        }

        /* if this left is not null and this right is not null, return
true
         */
        boolean has_right_and_left(){
            if(…….. && ……){
                return true;
            }else{
                return false;
            }
        }

        /* if this left is not null but this right is null, return
true
         */
        boolean only_has_left(){
            if(……&&…….){
                return true;
            }else{
                return false;
            }
        }

        /* if this right is not null but this left is null, return
true
         */
        boolean only_has_right(){
            if(…….. && …….){
                return true;
            }else{
                return false;
            }
        }
```

```
    boolean has_no_child(){
        if(…….... &&……..){
            return true;
        }else{
            return false;
        }
    }

    /* if this node is it's parent left child, set parent's left =
null and this parent = null
     * if this node is it's parent right child, set parent's right
= null and this parent = null
     */
    void unset_parent(){
        if(this.is_left()){
            …….
            …….
        }else if(this.is_right()){
            ……..…..
            ……
        }
    }

    /* set child as left of this node
     * while child's left is not null, set child into child's left
     * return child
     */
    BinaryTreeNode most_left_child(){
        BinaryTreeNode child = this.left;
        while(…….……){
            ……
        }
        return child;
    }

    /* set child as right of this node
     * while child's right is not null, set child into child's
right
     * return child
     */
    BinaryTreeNode most_right_child(){
        BinaryTreeNode child = this.right;
        while(……){
            …….
        }
        return child;
    }

    /* recursively print node
     * print spaces + label + node's data
     * if node's left is not null, call left's print method with
additional space and "LEFT" label
     * if node's right is not null, call right's print method with
additional space and "RIGHT" label
     */
    void print(String spaces, String label){
        System.out.println(…….);
        if(……..){
            this.left.print(……..);
        }
        if(this.right != null){
            this.right.print(……);
```

```
        }
    }

    // Alias for print("", "NODE");
    void print(){
        this.print("", "NODE");
    }

    /* recursively print node with infix mode
     * print "("
     * if node's left is not null, run left's infix method,
otherwise print "null"
     * print space + this node's data + space
     * if node's right is not null, run right's infix method,
otherwise print "null"
     * print ")"
     */
    void infix(){
        System.out.print("(");
        if(……..){
            left.infix();
        }else{
            System.out.print("null");
        }
        System.out.print(" " + this.data + " ");
        if(……){
            right.infix();
        }else{
            System.out.print("null");
        }
        System.out.print(")");
    }

    /* recursively print node with prefix mode
     * print this node's data + "("
     * if node's left is not null, run left's prefix method,
otherwise print "null"
     * print space
     * if node's right is not null, run right's prefix method,
otherwise print "null"
     * print ")"
     */
    void prefix(){
        System.out.print(this.data + "(");
        if(……..){
            left.prefix();
        }else{
            System.out.print("null");
        }
        System.out.print(" ");
        if(……..){
            right.prefix();
        }else{
            System.out.print("null");
        }
        System.out.print(")");
    }

    /* recursively print node with postfix mode
     * print "("
     * if node's left is not null, run left's postfix method,
otherwise print "null"
```

```
     * print space
     * if node's right is not null, run right's postfix method,
otherwise print "null"
     * print ")" + this node's data
     */
    void postfix(){
        System.out.print("(");
        if(……){
            left.postfix();
        }else{
            System.out.print("null");
        }
        System.out.print(" ");
        if(…….){
            right.postfix();
        }else{
            System.out.print("null");
        }
        System.out.print(")" + this.data);
    }
}
```

b. Lengkapilah bagian yang kosong pada class binary tree di bawah ini sesuai dengan perintah / command yang tertera.

```
public class BinaryTree{

    BinaryTreeNode root;

    public BinaryTree(){
        this.root = null;
    }

    /* call method print of root if root is not null */
    void print(){
        if(……){
            ……;
        }
    }
    /* call method prefix of root if root is not null, print new
line */
    void prefix(){
        if(……..){
            ……;
        }
        System.out.println();
    }
    /* call method infix of root if root is not null, print new
line */
    void infix(){
        if(…….){
            ……;
        }
        System.out.println();
    }
    /* call method postfix of root if root is not null, print new
line */
    void postfix(){
        if(……..){
```

```
            this.root.postfix();
        }
        System.out.println();
    }

/* if root is null, set new_node as root
     * otherwise, set current to root
     * while current is not null
     *        if new_node's data bigger than current's data and
current has right, then set current = current's right
     *        if new node's data bigger than current's data and
current doesn't have right, then put new_node as current's right,
break from loop
     *        if new_node's data smaller than current's data and
current has left, then set current = current's left
     *        if new_node's data smaller than current's data and
current doesn't have left, then put new_node as current's left,
break from loop
     */
    void push(BinaryTreeNode new_node){
        if(……..){
            ……;
        }else{
            BinaryTreeNode current = this.root;
            while(……..){
                if(……>…….){
                    if(…….== ……){
                        ……..);
                        break;
                    }else{
                        current = current.right;
                    }
                }else{
                    if(…….){
                        current.set_left(new_node);
                        break;
                    }else{
                        current = current.left;
                    }
                }
            }
        }
    }
/* only able to delete if root is not null
     * if deleted doesn't have any child, then unset_parent
     * if deleted only has one child, then let the child replace
deleted
     * if deleted has two child, then let deleted's left's most
right to replace deleted
     */
    void delete(BinaryTreeNode deleted){
        if(……..){
            if(……..()){
                if(……){
                    this.root = null;
                }else{
                    deleted.unset_parent();
                }
            }else
```

```
 if(deleted.only_has_left()|| deleted.only_has_right()){
                BinaryTreeNode replacement = null;
                if(……..()){
                    replacement = deleted.left;
                }else{
                    replacement = deleted.right;
                }
if(deleted == this.root){
                    ……..;
                    ……..();
                }else if(deleted.is_left()){
                    ……..;
                    ……..();
                }else if(deleted.is_right()){
                    ……..;
                    ……..();

                }
            }else{
                BinaryTreeNode replacement = deleted.left;
                if(replacement.right != null){
                    ……..();
                }
                BinaryTreeNode       parent_of_replacement       =
replacement.parent;
                if(replacement.only_has_right()){

parent_of_replacement.set_left(replacement.right);
                }
                ……..();
                 ……..(deleted.left);
                ……..(deleted.right);
                if(……..){
                    this.root = replacement;
                }else if(deleted.is_left()){
                    deleted.parent.set_left(replacement);
                }else if(deleted.is_right()){
                    deleted.parent.set_right(replacement);
                }
            }
        }
    }

}
```

c. Simpan class BinaryTreeNode.java dan BinaryTree.java pada satu package yang sama. Selanjutnyabuatlah class baru pada package yang sama ketikkan class Test.java di bawah ini dan jalankan.

```
    public class Test{
    public static void main(String Args[]){

BinaryTree bt = new BinaryTree();
        bt.print(); // shoud show nothing
        bt.push(new BinaryTreeNode(20));
        bt.push(new BinaryTreeNode(15));
        bt.push(new BinaryTreeNode(25));
```

```
        bt.push(new BinaryTreeNode(12));
        bt.push(new BinaryTreeNode(17));
        bt.push(new BinaryTreeNode(22));
        bt.push(new BinaryTreeNode(27));
        bt.push(new BinaryTreeNode(28));
        bt.print(); // should show structured tree
        bt.infix(); // should show tree in infix mode
        bt.prefix(); // should show tree in prefix mode
        bt.postfix(); // should show tree in postfix mode
        // delete a node with no child
        BinaryTreeNode deleted = bt.root.most_left_child(); //
Node 12
        System.out.println(deleted.data); // should show "12"
        bt.delete(deleted);
        bt.print(); // should show structured tree without "12"
        // delete a node with a single child
        deleted = bt.root.most_right_child().parent; // Node "27"
        System.out.println(deleted.data); // should show "27"
        System.out.println(deleted.only_has_right());
        System.out.println(deleted.only_has_left());
        bt.delete(deleted);
        bt.print(); // should show structured tree without "27"
        // delete root (has two children)
        deleted = bt.root; // Node "20"
        System.out.println(deleted.data); // should show "20"
        bt.delete(deleted);
        bt.print(); // should show structured tree without "20"

}
}
```

d. Setelah dijalankan akan muncul output seperti di bawah ini :

```
    NODE 20
 LEFT 15
  LEFT 12
  RIGHT 17
 RIGHT 25
  LEFT 22
  RIGHT 27
   RIGHT 28
(((null 12 null) 15 (null 17 null)) 20 ((null 22 null) 25 (null 27
(null 28 null))))
20(15(12(null null) 17(null null)) 25(22(null null) 27(null 28(null
null))))
(((null null)12 (null null)17)15 ((null null)22 (null (null
null)28)27)25)20
12
NODE 20
 LEFT 15
  RIGHT 17
 RIGHT 25
  LEFT 22
  RIGHT 27
   RIGHT 28
27
true
false
NODE 20
```

```
 LEFT 15
  RIGHT 17
 RIGHT 25
  LEFT 22
  RIGHT 28
20
NODE 17
 LEFT 15
 RIGHT 25
  LEFT 22
  RIGHT 28
```

Kesimpulan

Buatlah interface dari program di atas!

Penutup

Tugas

Buatlah sebuah program yang menerapkan konsep binary search menggunakan binary tree!