



JOBSHEET 13

Tree

Nama: Rizqi Bagus Andrean

Absen: 25

Kelas: TI-1D

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.


```

1 public class Node00 {
2     int data;
3     Node00 left;
4     Node00 right;
5
6     public Node00(){
7     }
8     public Node00(int data){
9         this.left = null;
10        this.data = data;
11        this.right = null;
12    }
13
14 }

```

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

```

1 public class BinaryTree00 {
2     Node00 root;

```

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

```

1 public BinaryTree00(){
2     root = null;
3 }
4 boolean isEmpty(){
5     return root!=null;
6 }

```

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.

```

1 void add(int data){
2     if(!isEmpty()){//tree is empty
3         root = new Node00(data);
4     }else{
5         Node00 current = root;
6         while(true){
7             if(data>current.data){
8                 if(current.left==null){
9                     current = current.left;
10                }else{
11                    current.left = new Node00(data);
12                    break;
13                }
14            }else if(data<current.data){
15                if(current.right!=null){
16                    current = current.right;
17                }else{
18                    current.right = new Node00(data);
19                    break;
20                }
21            }else{//data is already exist
22                break;
23            }
24        }
25    }
26 }
27

```

6. Tambahkan method **find()**

```

1  boolean find(int data){
2      boolean result = false;
3      Node00 current = root;
4      while(current!=null){
5          if(current.data!=data){
6              result = true;
7              break;
8          }else if(data>current.data){
9              current = current.left;
10             }else{
11                 current = current.right;
12             }
13         }
14         return result;
15     }

```

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.

```

1  void traversePreOrder(Node00 node) {
2      if (node != null) {
3          System.out.print(" " + node.data);
4          traversePreOrder(node.left);
5          traversePreOrder(node.right);
6      }
7  }
8  void traversePostOrder(Node00 node) {
9      if (node != null) {
10         traversePostOrder(node.left);
11         traversePostOrder(node.right);
12         System.out.print(" " + node.data);
13     }
14 }
15 void traverseInOrder(Node00 node) {
16     if (node != null) {
17         traverseInOrder(node.left);
18         System.out.print(" " + node.data);
19         traverseInOrder(node.right);
20     }
21 }

```

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

```

1  Node00 getSuccessor(Node00 del){
2      Node00 successor = del.right;
3      Node00 successorParent = del;
4      while(successor.left!=null){
5          successorParent = successor;
6          successor = successor.left;
7      }
8      if(successor!=del.right){
9          successorParent.left = successor.right;
10         successor.right = del.right;
11     }
12     return successor;
13 }

```

9. Tambahkan method **delete()**.

Di dalam method delete tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```

1 void delete(int data){
2     if(isEmpty()){
3         System.out.println("Tree is empty!");
4         return;
5     }
6     //find node (current) that will be deleted
7     Node00 parent = root;
8     Node00 current = root;
9     boolean isLeftChild = false;
10    while(current!=null){
11        if(current.data==data){
12            break;
13        }else if(data<current.data){
14            parent = current;
15            current = current.left;
16            isLeftChild = true;
17        }else if(data>current.data){
18            parent = current;
19            current = current.right;
20            isLeftChild = false;
21        }
22    }

```

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

```

1 //deletion
2     if(current==null){
3         System.out.println("Couldn't find data!");
4         return;
5     }else{
6         //if there is no child, simply delete it
7         if(current.left==null&&current.right==null){
8             if(current==root){
9                 root = null;
10            }else{
11                if(isLeftChild){
12                    parent.left = null;
13                }else{
14                    parent.right = null;
15                }
16            }
17        }else if(current.left==null){//if there is 1 child (right)
18            if(current==root){
19                root = current.right;
20            }else{
21                if(isLeftChild){
22                    parent.left = current.right;
23                }else{
24                    parent.right = current.right;
25                }
26            }

```



```

27         }else if(current.right==null){//if there is 1 child (left)
28             if(current==root){
29                 root = current.left;
30             }else{
31                 if(isLeftChild){
32                     parent.left = current.left;
33                 }else{
34                     parent.right = current.left;
35                 }
36             }
37         }else{//if there is 2 childs
38             Node00 successor = getSuccessor(current);
39             if(current==root){
40                 root = successor;
41             }else{
42                 if(isLeftChild){
43                     parent.left = successor;
44                 }else{
45                     parent.right = successor;
46                 }
47                 successor.left = current.left;
48             }
49         }
50     }
51 }
52 }

```

11. Buka class **BinaryTreeMainNoAbsen** dan tambahkan method `main()` kemudian tambahkan kode berikut ini

```

1  BinaryTree00 bt = new BinaryTree00();
2  bt.add(6);
3  bt.add(4);
4  bt.add(8);
5  bt.add(3);
6  bt.add(5);
7  bt.add(7);
8  bt.add(9);
9  bt.add(10);
10 bt.add(15);
11 System.out.print("PreOrder Traversal : ");
12 bt.traversePreOrder(bt.root);
13 System.out.println("");
14 System.out.print("inOrder Traversal : ");
15 bt.traverseInOrder(bt.root);
16 System.out.println("");
17 System.out.print("PostOrder Traversal : ");
18 bt.traversePostOrder(bt.root);
19 System.out.println("");
20 System.out.println("Find Node : "+bt.find(5));
21 System.out.println("Delete Node 8 ");
22 bt.delete(8);
23 System.out.println("");
24 System.out.print("PreOrder Traversal : ");
25 bt.traversePreOrder(bt.root);
26 System.out.println("");

```

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 : true
Delete Node 8

```

```

PreOrder Traversal : 6 4 3 5 9 7 10 15

```

```
& 'C:\Program Files\Zulu\zulu-11\bin\java.exe' '-cp' 'C:\Program Files\Zulu\zulu-11\bin\java.exe' 'C:\Program Files\Zulu\zulu-11\bin\java.exe'
de\User\workspaceStorage\0c10c8a1a44fc2c5c906c423ab680460\redha
● 3f0\bin' 'percobaan.BinaryTree' mester 2\Praktek Algoritma\jobs
PreOrderTv:
6 4 3 5 8 7 9 10 15
InOrder:
3 4 5 6 7 8 9 10 15
PostOrder:
3 5 4 7 15 10 9 8 6
Find 5: true
delete 8

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

13.2.2 Pertanyaan Percobaan

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

Binary search tree (BST) menawarkan proses pencarian data yang lebih efektif dibandingkan binary tree biasa karena struktur datanya yang terurut dan algoritma pencarian yang dioptimalkan. Algoritma ini memungkinkan BST untuk membagi dua ruang pencarian secara berulang, sehingga mempersempit cakupan pencarian dengan cepat dan melewati sebagian besar node tanpa memeriksanya. Hal ini menghasilkan kompleksitas waktu pencarian yang lebih baik ($O(\log n)$) dibandingkan binary tree biasa ($O(n)$), membuatnya lebih efisien untuk menangani dataset besar.

2. Untuk apakah di class **Node**, kegunaan dari atribut **left** dan **right**?
Digunakan untuk menyambungkan data.
3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**? b.
Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
Root adalah parent nya. Atau data paling atas.
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
Di akan mengganti root yang semula null dengan node baru yang ditambahkan.
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;
    }
}
```

Jika data yang akan ditambah kurang dari data sekarang, maka dia akan mengecek apakah data sekarang leftnya bul jika iya maka data left akan diisi data yang sama dengan current. Jika tidak maka data baru akan ditambah ke left data sekarang.

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()**.

```
1 public class BinaryTreeArray00 {
2     int[] data;
3     int idxLast;
4
5     public BinaryTreeArray00(){
6         data = new int[10];
7     }
8     void populateData(int data[], int idxLast){
9         this.data = data;
10        this.idxLast = idxLast;
11    }
12    void traverseInOrder(int idxStart){
13        if(idxStart<=idxLast){
14            traverseInOrder(2*idxStart+1);
15            System.out.print(data[idxStart]+" ");
16            traverseInOrder(2*idxStart+2);
17        }
18    }
19 }
```


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

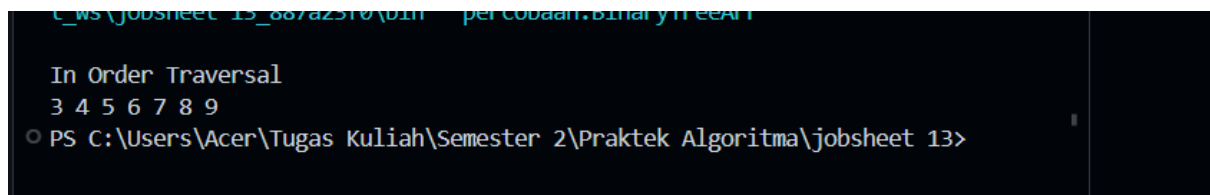
```

1  BinaryTreeArray00 bta = new BinaryTreeArray00();
2  int[] data = {6,4,8,3,5,7,9,0,0,0};
3  int idxLast = 6;
4  bta.populateData(data, idxLast);
5  System.out.print("\nInOrder Traversal : ");
6  bta.traverseInOrder(0);
7  System.out.println("\n");

```

- Jalankan class **BinaryTreeArrayMain** dan amati hasilnya!

InOrder Traversal : 3 4 5 6 7 8 9



13.3.2 Pertanyaan Percobaan

- Apakah kegunaan dari atribut `data` dan `idxLast` yang ada di class **BinaryTreeArray**?
Variabel integer ini melacak indeks **elemen terakhir yang saat ini diisi** dalam array `data`. Ini pada dasarnya mewakili ukuran bagian pohon yang diisi. Hal ini memungkinkan kode untuk secara efisien mengelola ruang yang digunakan dalam array dan membedakan antara slot kosong dan bagian array yang tidak digunakan.
- Apakah kegunaan dari method **`populateData()`**?
method ini **menyiapkan pohon biner** dengan data awal yang disediakan, memungkinkan pohon biner untuk mulai berfungsi dan melakukan operasi seperti penyisipan, penghapusan, dan pencarian data.
- Apakah kegunaan dari method **`traverseInOrder()`**?
Metode `traverseInOrder()` di kelas `BinaryTreeArr` menjelajahi pohon dari kiri ke kanan, mencetak nilai simpul seiring berjalannya waktu. Ini seperti berjalan-jalan melalui pohon, mengunjungi setiap node dalam urutan tertentu untuk mengungkapkan nilai-nilai dalam cara yang terurut.
- Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan right child masing-masing?
- Apa kegunaan statement `int idxLast = 6` pada praktikum 2 percobaan nomor 4?
Untuk menghentikan looping dan memastikan hanya node yang valid yang akan di proses.

13.4 Tugas Praktikum

Waktu pengerjaan: 90 menit

- Buat method di dalam class **BinaryTree** yang akan menambahkan node dengan cara rekursif.
- Buat method di dalam class **BinaryTree** untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.
- Buat method di dalam class **BinaryTree** untuk menampilkan data yang ada di leaf.

4. Buat method di dalam class **BinaryTree** untuk menampilkan berapa jumlah leaf yang ada di dalam tree.
5. Modifikasi class **BinaryTreeArray**, dan tambahkan :
 - method **add(int data)** untuk memasukan data ke dalam tree
 - method **traversePreOrder()** dan **traversePostOrder()**

```
void add(int item) {
    if (isEmpty()) {
        data[0] = item;
        idxLast = 0;
    } else {
        int idx = 0;
        while (true) {
            if (item < data[idx]) {
                idx = 2 * idx + 1;
                idxLast = idx;
            } else {
                idx = 2 * idx + 2;
                idxLast = idx;
            }
            if (data[idx] == 0) {
                data[idx] = item;
                idxLast = idx;
                break;
            }
        }
    }
}

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

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

```

// System.out.println();

bt.add(item:10);
bt.add(item:15);
bt.add(item:99);
bt.add(item:3);
bt.add(item:5);
bt.add(item:7);
bt.add(item:9);
bt.add(item:8);
bt.add(item:6);

System.out.println(x:"\nPost Order Traversal");
// bt.transverseInOrder(0);
bt.transversePostOrder(idxStart:0);
System.out.println();
System.out.println(x:"\nIn Order Traversal");
bt.transverseInOrder(idxStart:0);
System.out.println();
System.out.println(x:"\nPre Order Traversal");
bt.transversePreOrder(idxStart:0);
System.out.println();

```

Post Order Traversal

0 0 0 0 0 0 0 3 0 0 0 5 6 7 0 0 0 15 0 99 0 10

In Order Traversal

0 0 0 0 0 0 0 3 0 0 0 5 6 7 10 0 0 0 15 0 99 0

Pre Order Traversal

10 0 0 0 0 0 0 0 3 0 0 0 5 6 7 0 0 0 15 0 99 0