

Laporan Praktikum Algoritma dan Struktur Data

Jobsheet 14 - Tree

Dosen Pengampu : Triana Fatmawati, S.T., M.T.



Nama : Annisa
Nim : 2341760032
Kelas : SIB 1E
Prodi : D-IV Sistem Informasi Bisnis

JURUSAN TEKNOLOGI INFORMASI

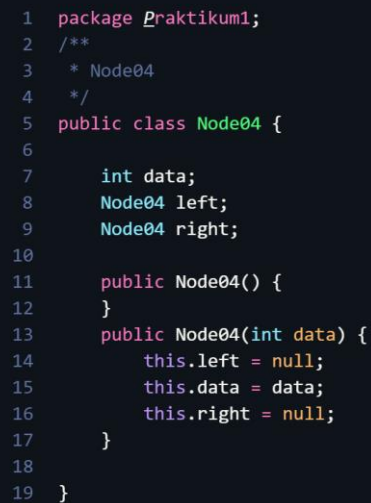
POLITEKNIK NEGERI MALANG

2023/2024

13.2 Kegiatan Praktikum 1

13.2.1 Percobaan 1

1. Buatlah class `NodeNoAbsen`, `BinaryTreeNoAbsen` dan `BinaryTreeMainNoAbsen`. Di dalam class `Node`, tambahkan atribut `data`, `left` dan `right`, serta konstruktor default dan berparameter.

A screenshot of a code editor showing the implementation of the Node04 class. The code is written in Java and includes package declarations, comments, and two constructors: a default constructor and a parameterized constructor that initializes the data, left, and right attributes.

```
1 package Praktikum1;
2 /**
3  * Node04
4  */
5 public class Node04 {
6
7     int data;
8     Node04 left;
9     Node04 right;
10
11     public Node04() {
12     }
13     public Node04(int data) {
14         this.left = null;
15         this.data = data;
16         this.right = null;
17     }
18
19 }
```

2. Di dalam class `BinaryTreeNoAbsen`, tambahkan atribut `root`. Tambahkan konstruktor default dan method `isEmpty()` di dalam class `BinaryTreeNoAbsen`

A screenshot of a code editor showing the implementation of the BinaryTree04 class. The code includes package declarations, comments, a default constructor that initializes the root attribute to null, and an isEmpty() method that checks if the root is null.

```
1 package Praktikum1;
2 /**
3  * BinaryTree04
4  */
5 public class BinaryTree04 {
6     Node04 root;
7
8     public BinaryTree04() {
9         root = null;
10    }
11
12    boolean isEmpty() {
13        return root == null;
14    }
15 }
```

3. 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()) {
3         root = new Node04(data);
4     } else {
5         Node04 current = root;
6         Node04 parent = null;
7         while (true) {
8             parent = current;
9             if (data < current.data) {
10                 if (current.left == null) {
11                     current.left = new Node04(data);
12                     break;
13                 } else {
14                     current = current.left;
15                 }
16             } else if (data > current.data) {
17                 if (current.right == null) {
18                     current.right = new Node04(data);
19                     break;
20                 } else {
21                     current = current.right;
22                 }
23             } else {
24                 break;
25             }
26         }
27     }
28 }
29 }
```

4. Tambahkan method `find()`

```
1 boolean find(int data) {
2     Node04 current = root;
3     while (current != null) {
4         if (current.data == data) {
5             return true;
6         } else if (data < current.data) {
7             current = current.left;
8         } else {
9             current = current.right;
10        }
11    }
12    return false;
13 }
```

5. 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(Node04 node) {
2     if (node != null) {
3         System.out.print(" " + node.data);
4         traversePreOrder(node.left);
5         traversePreOrder(node.right);
6     }
7 }
8
9 void traversePostOrder(Node04 node) {
10    if (node != null) {
11        traversePostOrder(node.left);
12        traversePostOrder(node.right);
13        System.out.print(" " + node.data);
14    }
15 }
16
17 void traverseInOrder(Node04 node) {
18    if (node != null) {
19        traverseInOrder(node.left);
20        System.out.print(" " + node.data);
21        traverseInOrder(node.right);
22    }
23 }
```

6. Tambahkan method `getSuccessor()`. Method ini akan digunakan ketika proses penghapusan node yang memiliki 2 child. Tambahkan method `delete()`. Di dalam method `delete` tambahkan pengecekan apakah tree kosong, dan jika tidak cari posisi node yang akan di hapus.

```
1  Node04 getSuccessor(Node04 del) {
2      Node04 successor = del.right;
3      Node04 successorParent = del;
4      while (successor.left != null) {
5          successorParent = successor;
6          successor = successor.left;
7      }
8      if (successorParent != del) {
9          successorParent.left = successor.right;
10     } else {
11         successorParent.right = successor.right;
12     }
13     return successor;
14 }
15
16 void delete(int data) {
17     if (isEmpty()) {
18         System.out.println("Tree is Empty!");
19         return;
20     }
21
22     Node04 parent = root;
23     Node04 current = root;
24     boolean isLeftChild = false;
25     while (current.data != data) {
26         parent = current;
27         if (data < current.data) {
28             current = current.left;
29             isLeftChild = true;
30         } else {
31             current = current.right;
32             isLeftChild = false;
33         }
34     }
```

7. 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     }
6 }
7
8 if (current.left == null && current.right == null) {
9     if (current == root) {
10         root = null;
11     } else {
12         if (isLeftChild) {
13             parent.left = null;
14         } else {
15             parent.right = null;
16         }
17     }
18 } else if (current.left == null) {
19     if (current == root) {
20         root = current.right;
21     } else {
22         if (isLeftChild) {
23             parent.left = current.right;
24         } else {
25             parent.right = current.right;
26         }
27     }
28 } else if (current.right == null) {
29     if (current == root) {
30         root = current.left;
31     } else {
32         if (isLeftChild) {
33             parent.left = current.left;
34         } else {
35             parent.right = current.left;
36         }
37     }
38 } else {
39     Node04 successor = getSuccessor(current);
40     if (current == root) {
41         root = successor;
42     } else {
43         if (isLeftChild) {
44             parent.left = successor;
45         } else {
46             parent.right = successor;
47         }
48         successor.left = current.left;
49     }
50 }
51 }
52 }
```

8. Buka class BinaryTreeMainNoAbsen dan tambahkan method main() kemudian tambahkan kode berikut ini

```
1 package Praktikum1;
2
3 /**
4  * BinaryTreeMain04
5  */
6 public class BinaryTreeMain04 {
7
8     public static void main(String[] args) {
9         BinaryTree04 bt = new BinaryTree04();
10        bt.add(6);
11        bt.add(4);
12        bt.add(8);
13        bt.add(3);
14        bt.add(5);
15        bt.add(7);
16        bt.add(9);
17        bt.add(10);
18        bt.add(15);
19        System.out.print("PreOrder Traversal : ");
20        bt.traversePreOrder(bt.root);
21        System.out.println("");
22        System.out.print("InOrder Traversal : ");
23        bt.traverseInOrder(bt.root);
24        System.out.println("");
25        System.out.print("PostOrder Traversal : ");
26        bt.traversePostOrder(bt.root);
27        System.out.println("");
28        System.out.println("Find Node : " + bt.find(5));
29        System.out.println("Delete Node 8");
30        bt.delete(8);
31        System.out.println("");
32        System.out.print("PreOrder Traversal : ");
33        bt.traversePreOrder(bt.root);
34        System.out.println("");
35    }
36
37 }
```

9. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.

10. Amati hasil running tersebut.

```
C:\Users\user\Documents\Jobsheet 14> cmd /C "C:\Users\user\.jdk\openjdk-21.0.2\bin\java.exe -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\user\AppData\Roaming\Code\User\workspaceStorage\1f1b7a233161a8a34ebb663cd1dd8493\redhat.java\jdt_ws\Jobsheet 14_cf52ae25\bin" Praktikum1.BinaryTreeMain04"
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:\Users\user\Documents\Jobsheet 14>
```

13.2.2 Pertanyaan Percobaan

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

Karena, pada proses binary search tree telah ditambahkan sebuah aturan baru yaitu, “semua left-child harus lebih kecil dibandingkan right-child dan parentnya” sehingga mempermudah untuk melakukan pencarian data.

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

Pada class node atribut left berfungsi untuk menyimpan "left child" atau nilai yang lebih kecil dari root (node induk) dan atribut right berfungsi untuk menyimpan "right child" atau nilai yang lebih besar dari root (node induk)

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

Didalam BinaryTree root digunakan sebagai kepala atau inti, sama dengan head pada linked list yang digunakan sebagai kepala dari setiap linked list atau inti dari sebuah tree.

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

ketika objek tree pertama kali dibuat nilai dari root bernilai null, karena masih belum ada data yang dimasukan

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

Proses yang akan terjadi adalah penambahan node baru yang akan digunakan sebagai root.

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;
    }
}
```

Pada baris program diatas digunakan untuk mengecek apakah nilai input lebih kecil dari parent atau tidak. Jika iya, dilakukan pengecekan apakah current.left != null atau masih memiliki left child yang dimana memiliki subtree lagi. Jika iya maka dilakukan traversal dengan mengubah nilai current menjadi current.left. lalu, ada pengecekan jika tidak current.left == null atau tidak memiliki subtree atau left child maka posisi current.left tersebut akan menjadi tempat untuk meletakkan data yang diinput.

13.3 Kegiatan Praktikum 2

13.3.1 Tahapan Percobaan

1. Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukkan 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 package Praktikum2;
2
3 /**
4  * BinaryTreeArray04
5  */
6 public class BinaryTreeArray04 {
7
8     int[] data;
9     int idxLast;
10
11     public BinaryTreeArray04() {
12         data = new int[10];
13     }
14     void populateData(int data[], int idxLast) {
15         this.data = data;
16         this.idxLast = idxLast;
17     }
18     void traverseInOrder(int idxStart) {
19         if (idxStart <= idxLast) {
20             traverseInOrder(2*idxStart+1);
21             System.out.print(data[idxStart]+" ");
22             traverseInOrder(2*idxStart+2);
23         }
24     }
25 }
26 }
```

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



```
1 package Praktikum2;
2
3 /**
4  * BinaryTreeArrayMain04
5  */
6 public class BinaryTreeArrayMain04 {
7
8     public static void main(String[] args) {
9         BinaryTreeArray04 bta = new BinaryTreeArray04();
10         int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
11         int idxLast = 6;
12         bta.populateData(data, idxLast);
13         System.out.print("InOrder Traversal: ");
14         bta.traverseInOrder(0);
15         System.out.println();
16     }
17 }
18 }
```

5. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
InOrder Traversal: 3 4 5 6 7 8 9  
C:\Users\user\Documents\Jobsheet 14>
```

13.3.2 Pertanyaan Percobaan

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

Atribut idxLast digunakan untuk mengetahui pada indeks berapa data terakhir kali diletakkan.

2. Apakah kegunaan dari method populateData()?

Fungsi dari method populateData() adalah untuk mengisi data dan nilai idxLast pada objek BinaryTree. Method ini digunakan untuk menginisialisasi atau mengisi data pada objek BinaryTree dengan data yang diberikan.

3. Apakah kegunaan dari method traverseInOrder()?

Method traverseInOrder() digunakan untuk menampilkan data yang ada pada tree dengan cara traversal in order atau sebagai berikut,

- Secara rekursif kunjungi dan cetak seluruh data pada subtree sebelah kiri
- Kunjungi dan cetak data pada root
- Secara rekursif kunjungi dan cetak data pada subtree sebelah kanan

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

Untuk menentukan indeks posisi terdapat 2 cara sebagai Berikut

- Left child = $2*i+1$; jika indeks posisi 2 maka $\Rightarrow 2*2+1 \Rightarrow 5$
- Right child = $2*i+2$; jika indeks posisi 2 maka $\Rightarrow 2*2+2 \Rightarrow 6$

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

Untuk menandakan bahwa indeks yang diletakkan posisi data terakhir adalah 6. Jika dilihat dari data yang diinput setelah indeks 6 data berisi 0 atau kosong sehingga tidak perlu untuk ditampilkan.

13.4 Tugas Praktikum

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

```
1 public void add(int data) {  
2     root = addRecursive(root, data);
```

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

Berikut merupakan method untuk menampilkan nilai paling kecil

```
1 public int findMinValue() {  
2     if (isEmpty()) {  
3         System.out.println("Tree is empty!");  
4         return Integer.MIN_VALUE;  
5     }  
6     Node04 current = root;  
7     while (current.left != null) {  
8         current = current.left;  
9     }  
10    return current.data;  
11 }  
12
```

Berikut merupakan method untuk menampilkan nilai paling besar

```
1 public int findMaxValue() {  
2     if (isEmpty()) {  
3         System.out.println("Tree is empty!");  
4         return Integer.MAX_VALUE;  
5     }  
6     Node04 current = root;  
7     while (current.right != null) {  
8         current = current.right;  
9     }  
10    return current.data;  
11 }
```

```
1 System.out.println("Nilai paling kecil dalam tree: " + bt.findMinValue());  
2 System.out.println("Nilai paling besar dalam tree: " + bt.findMaxValue());
```

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

```
1 public void displayLeafData(Node04 node) {
2     if (node == null) {
3         return;
4     }
5     if (node.left == null && node.right == null) {
6         System.out.print(node.data + " ");
7     }
8     displayLeafData(node.left);
9     displayLeafData(node.right);
10 }
```

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

```
1 public int countLeafNodesRekursif(Node04 node) {
2     if (node == null) {
3         return 0;
4     }
5     if (node.left == null && node.right == null) {
6         return 1;
7     }
8     return countLeafNodesRekursif(node.left) + countLeafNodesRekursif(node.right);
9 }
10 }
```

```
1 System.out.println("Jumlah leaf dalam tree: " + bt.countLeafNodes());
```

Hasil running dari modifikasi 4 method di atas

```
14_cf52ae25\bin" Praktikum1.BinaryTreeMain04 "
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
Nilai paling kecil dalam tree: 3
Nilai paling besar dalam tree: 15

PreOrder Traversal: 6 4 3 5 8 7 9 10 15
Data yang ada di leaf:
3 5 7 15
Jumlah leaf dalam tree: 4

C:\Users\user\Documents\Jobsheet 14>
```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

- method add(int data) untuk memasukan data ke dalam tree
- method traversePreOrder() dan traversePostOrder()

```
1 package Praktikum2;
2
3 /**
4  * BinaryTreeArray04
5  */
6 public class BinaryTreeArray04 {
7
8     int[] data;
9     int idxLast;
10    public BinaryTreeArray04() {
11        data = new int[10];
12        idxLast = -1;
13    }
14
15    public void add(int newData) {
16        if (idxLast + 1 < data.length) {
17            data[++idxLast] = newData;
18        } else {
19            System.out.println("Tree penuh");
20        }
21    }
22
23    void populateData(int data[], int idxLast) {
24        this.data = data;
25        this.idxLast = idxLast;
26    }
27
28    void traverseInOrder(int idxStart) {
29        if (idxStart <= idxLast) {
30            traverseInOrder(2 * idxStart + 1);
31            System.out.print(data[idxStart] + " ");
32            traverseInOrder(2 * idxStart + 2);
33        }
34    }
35
36    public void traversePreOrder() {
37        traversePreOrder(0);
38    }
39
40    private void traversePreOrder(int idxStart) {
41        if (idxStart <= idxLast) {
42            System.out.print(data[idxStart] + " ");
43            traversePreOrder(2 * idxStart + 1);
44            traversePreOrder(2 * idxStart + 2);
45        }
46    }
47
48    public void traversePostOrder() {
49        traversePostOrder(0);
50    }
51
52    private void traversePostOrder(int idxStart) {
53        if (idxStart <= idxLast) {
54            traversePostOrder(2 * idxStart + 1);
55            traversePostOrder(2 * idxStart + 2);
56            System.out.print(data[idxStart] + " ");
57        }
58    }
59 }
```

```

1 package Praktikum2;
2
3 /**
4  * BinaryTreeArrayMain04
5  */
6 public class BinaryTreeArrayMain04 {
7     public static void main(String[] args) {
8         BinaryTreeArray04 bta = new BinaryTreeArray04();
9         int[] data = {6,4,8,3,5,7,9,0,0,0};
10        int idxLast = 6;
11        bta.populateData(data, idxLast);
12        System.out.print("\nInOrder Traversal : ");
13        bta.traverseInOrder(0);
14        System.out.println();
15        System.out.println("PreOrder Traversal: ");
16        bta.traversePreOrder();
17        System.out.println();
18        System.out.println("PostOrder Traversal: ");
19        bta.traversePostOrder();
20    }
21 }

```

```

InOrder Traversal : 3 4 5 6 7 8 9
PreOrder Traversal:
6 4 3 5 8 7 9
PostOrder Traversal:
3 5 4 7 9 8 6
C:\Users\user\Documents\Jobsheet 14>

```