# Bab 7 Tree

## Identitas

|  |
| --- |
| **Kajian**  Tree  **Topik**   1. Tree   **Referensi**   1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms Third Edition. Cambridge: The MIT Press. 2. Sedgewick, R. (2002). Algorithm in Java: parts 1-4 Third Edition. Boston: Pearson Education, Inc. 3. P. Deitel and H. Deitel, Java How To Program 9th Edition, Prentice Hall, 2011. 4. Poo, Danny, Derek Kiong, and Swarnalatha Ashok. Object-Oriented Programming and Java, 2nd Edition. Springer, 2010. 5. Wu, C. Thomas. An Introduction to Object-Oriented Programming with Java. McGraw-Hill, 2009.   **Kompetensi Utama**   1. Mahasiswa mampu menggunakan bahasa java untuk membangun sebuah aplikasi sederhana yang digunakan untuk mengelola data dalam struktur Tree dengan menggunakan data nilai dan kelas Object   **Lama Kegiatan Kajian**   1. Pertemuan Terbimbing : 2 x 500 menit 2. Kegiatan Mandiri : 1 x 70 menit   **Parameter Penilaian**   1. Jurnal 60% 2. Tugas Akhir 40%   **Pengumpulan**  Kumpulkan jawaban dari Jurnal tembimbing dan jurnal mandiri pada LMS yang telah disediakan |
|
|
|
|
|

### Jurnal Terbimbing

1. Lengkapi kode program berikut untuk menyusun sebuah program untuk mengelola data dengan menggunakan Tree

### **public class BinaryTree {**

### Node root;

### **public static void main(String[] args) {**

### BinaryTree tree = new BinaryTree();

### /\*

### lakukan proses penambahan data dengan memanggil method insert, ambil studi kasus sebanyak n nilai

### \*/

### tree.dataShow();

### }

### 

### **public BinaryTree() {**

### root = null;

### }

### **public void insert(int data) {**

### root = insertRecord(root, data);

### }

### **private Node insertRecord(Node root, int data) {**

### if (root == null) {

### root = new Node(data);

### return root;

### }

### if (data < root.data)

### root.left = insertRecord(root.left, data);

### else if (data > root.data)

### root.right = insertRecord(root.right, data);

### return root;

### }

### **public void dataShow() {**

### System.out.print("inorder : ");

### inorderRec(root);

### System.out.print("Preorder : ");

### preorderTraversal(root);

### System.out.print("Postorder : ");

### postorderTraversal(root);

### }

### 

### **/\* Inorder traversal**

### **a. Traverse the left subtree**

### **b. Visit the root.**

### **c. Traverse the right subtree**

### **\*/**

### **private void inorderRec(Node root) {**

### if (root != null) {

### inorderRec(root.left);

### System.out.print(root.data + " ");

### inorderRec(root.right);

### }

### }

### **/\* Preorder traversal**

### **a. Visit the root.**

### **b. Traverse the left subtree**

### **c. Traverse the right subtree**

### **\*/**

### **void preorderTraversal(Node node) {**

### if (node == null)

### return;

### System.out.print(node.data + " ");

### preorderTraversal(node.left);

### preorderTraversal(node.right);

### }

### **/\* Postorder traversal**

### **a. Traverse the left subtree**

### **b. Traverse the right subtree**

### **c. Visit the root.**

### **\*/**

### **void postorderTraversal(Node node) {**

### if (node == null)

### return;

### postorderTraversal (node.left);

### postorderTraversal (node.right);

### System.out.print(node.data + " ");

### }

### // Node Class

### **class Node {**

### int data;

### Node left, right;

### public Node(int item) {

### data = item;

### left = right = null;

### }

### }

1. Buatlah method yang dapat digunakan untuk melakukan pencarian sebuah nilai. Mekanisme pencarian yang dilakukan adalah sama, dengan memanfaatkan konsep rekursif untuk mencari berdasarkan komparasi value data dicari dengan data nilai yang ada pada node.
   1. Jika nilai data pada node/root
      1. null, data tidak ditemukan
      2. data cari == data node, data ditemukan
      3. data cari < data node, lakukan pencarian ke child kiri
      4. data cari > data node, lakukan pencarian ke child kanan

public boolean search(int data) {

return searchRec(root, data);

}

**private boolean searchRec(Node root, int data) {**

if (root == null)

return \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_;

if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

return true;

if (\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)

return searchRec(root.left, data);

else

return searchRec(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_);

}

1. Buatlah method yang dapat digunakan untuk melakukan mencari nilai terbesar. Perhatikan bahwa mencari nilai terbesar, berada pada node paling kanan, lakukan rekursif hingga **child kanan** memiliki nilai null

**public int findMax() {**

return findMaxRec(root);

}

**private int findMaxRec(Node root) {**

if (root == null)

throw new IllegalStateException("Empty Tree");

/\*

jika **child kanan** memiliki nilai null, berarti data ditemukan pada node tersebut

\*/

if (root.right == \_\_\_\_\_\_\_\_)

return root.data;

/\*

rekursif jika data child kanan masih memiliki nilai

\*/

return findMaxRec(\_\_\_\_\_\_\_\_);

}

### Jurnal Mandiri

1. Gunakan kode program pada jurnal terbimbing, dan buatlah method yang dapat digunakan untuk melakukan mencari nilai terkecil. Perhatikan bahwa mencari nilai terkecil, berada pada node paling **kiri**, lakukan rekursif hingga **child kiri** memiliki nilai null
2. Modifikasi program pada jurnal terbimbing, sehingga data yang dikelola bukan dalam bentuk data integer saja, namun untuk mengelola kelas berikut ini. Data terurut berdasarkan NIM

// Node Class

**class Node {**

Mahasiswa mahasiswa;

Node left, right;

public Node(Mahasiswa mahasiswa) {

this.mahasiswa = mahasiswa;

left = right = null;

}

}

**class Mahasiswa{**

int nim;

String nama;

public Mahasiswa(int nim, String nama){

this.nim=nim;

this.nama=nama;

}

public int getNim(){

return nim;

}

public String getNama(){

return nama;

}

public String toString(){

return (nim +" "+nama);

}

}