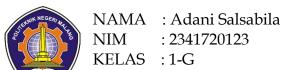


1. Praktikum 1 (Implementasi Binary Search Tree menggunakan Linked List)

```
J Pesanan01.java J Pembeli01.java J Node01.java ...\P14 X J BinaryTree01.j
rc > P14 > J Node01.java > ...
1 package P14;
2
3 public class Node01 {
4 int data;
5 Node01 left;
6 Node01 right;
7
8 public Node01() {
9 }
10
11 public Node01(int data) {
12 this.left = null;
13 this.data = data;
14 this.right = null;
15 }
16 }
17
```

```
| J | Pesnanolijava | J | Pembeliolijava | J | Nodeolijava | J | BinaryTreeOlijava | X | J | BinaryTreeOlijava | J | BinaryTreeOli
```

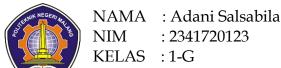
```
src > P14 > 🔳 BinaryTree01.java > 😭 BinaryTree01 > 🗘 add(int)
          boolean find(int data) {
             Node01 current = root;
             while (current != null) {
                 if (current.data == data) {
                 } else if (data < current.data) {
                     current = current.left;
                     current = current.right;
         void traversePreOrder(Node01 node) {
              if (node != null) {
                  System.out.print(" " + node.data);
                  traversePreOrder(node.left);
                  traversePreOrder(node.right);
          void traversePostOrder(Node01 node) {
              if (node != null) {
                  traversePostOrder(node.left);
                  traversePostOrder(node.right);
                  System.out.print(" " + node.data);
          void traverseInOrder(Node01 node) {
              if (node != null) {
                 traverseInOrder(node.left);
                  System.out.print(" " + node.data);
                  traverseInOrder(node.right);
```



MATERI: Tree (Jobsheet 11)

```
Node01 getSuccessor(Node01 del) {
   Node01 successor = del.right;
   Node01 successorParent = del;
      successorParent = successor;
       successor = successor.left;
   if (successor != del.right) {
       successorParent.left = successor.right;
       successor.right = del.right;
void delete(int data) {
   if (isEmpty()) {
      System.out.println(x:"Tree is empty!");
   Node01 parent = root;
   boolean isLeftChild = false;
   while (current != null && current.data != data) {
       parent = current;
       if (data < current.data) {</pre>
           current = current.left;
           current = current.right;
           isLeftChild = false;
   if (current == null) {
       System.out.println(x:"Couldn't find data!");
   if (current.left == null && current.right == null) {
        if (current == root) {
           root = null;
        } else {
           if (isLeftChild) {
               parent.left = null;
               parent.right = null;
```

```
src > P14 > J BinaryTree01.java > ધ BinaryTree01 > 🗘 add(int)
     public class BinaryTree01 {
         void delete(int data) {
              } else if (current.left == null) {
                  if (current == root) {
                      root = current.right;
                  } else {
                     if (isLeftChild) {
                         parent.left = current.right;
                          parent.right = current.right;
              } else if (current.right == null) {
                  if (current == root) {
                     root = current.left;
                  } else {
                     if (isLeftChild) {
                         parent.left = current.left;
                          parent.right = current.left;
                  Node01 successor = getSuccessor(current);
                  if (current == root) {
                     root = successor;
                  } else {
                     if (isLeftChild) {
                          parent.left = successor;
                          parent.right = successor;
                  successor.left = current.left;
```



MATERI: Tree (Jobsheet 11)

```
src > P14 > 🔳 BinaryTreeMain01.java > ધ BinaryTreeMain01 > 🗘 main(String[])
      public class BinaryTreeMain01 {
          public static void main(String[] args) {
             BinaryTree01 bt = new BinaryTree01();
             bt.add(data:6);
             bt.add(data:4);
             bt.add(data:8);
             bt.add(data:3);
             bt.add(data:5);
             bt.add(data:7);
             bt.add(data:9);
             bt.add(data:10);
             bt.add(data:15);
             System.out.print(s:"PreOrder Traversal : ");
             bt.traversePreOrder(bt.root);
              System.out.println(x:"");
             System.out.print(s:"InOrder Traversal : ");
              bt.traverseInOrder(bt.root);
              System.out.println(x:"");
             System.out.print(s:"PostOrder Traversal : ");
              bt.traversePostOrder(bt.root);
              System.out.println(x:"");
              System.out.println("Find Node 5: " + bt.find(data:5));
              System.out.println(x:"Delete Node 8");
             bt.delete(data:8);
              System.out.println(x:"");
              System.out.print(s:"PreOrder Traversal : ");
              bt.traversePreOrder(bt.root);
              System.out.println(x:"");
```

```
PS D:\college\semester 2\Algoritma_dan_Struktur_Data_16_01> d:; cd 'd:\college\semester 2\Algoritma_ava.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\college\semester 2\Algoritma_dan_Struk
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 5: true
Delete Node 8

PreOrder Traversal : 6 4 3 5 9 7 10 15
PS D:\college\semester 2\Algoritma_dan_Struktur_Data_16_01>
```

Pertanyaan!

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

Jawab: Karena dalam binary search tree memiliki aturan semua nilai subtree kiri lebih kecil dari nilai di node dan semua nilai di subtree kanan lebih besar dari nilai node. Jika nilai yang dicari lebih kecil dari nilai node saat ini, maka kita hanya perlu mencari di subtree kiri. Sebaliknya, jika lebih besar, kita hanya perlu mencari di subtree kanan.

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

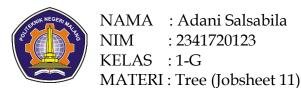
Jawab : Atribut left dan right di dalam kelas Node01 berfungsi sebagai penunjuk ke node-node anak di subtree kiri dan kanan yang memungkinkan navigasi, manipulasi, dan traversal tree.

3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree01?

Jawab : Atribut root di dalam kelas BinaryTree01 menyediakan titik awal untuk semua operasi tree, termasuk pencarian, penambahan, penghapusan, dan traversal, serta memungkinkan kita untuk dengan mudah memeriksa apakah tree kosong atau tidak.

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

Jawab : Ketika objek tree pertama kali dibuat, nilai dari root adalah null, menunjukkan bahwa tree masih kosong dan belum ada node yang ditambahkan.



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

Jawab: Ketika tree masih kosong, proses penambahan node pertama melibatkan pembuatan node baru dan mengatur node tersebut sebagai root. Setelah node pertama ditambahkan, tree tidak lagi kosong, dan node tersebut menjadi titik awal untuk penambahan node-node berikutnya dalam struktur tree.

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 Node01(data);
        break;
    }
}</pre>
```

Jawab : Baris program tersebut untuk menavigasi tree dan menemukan posisi yang tepat untuk menambahkan node baru berdasarkan aturan BST. Jika data lebih kecil dari current.data, kita harus menempatkan node baru di subtree kiri. Baris-baris ini memastikan bahwa kita bergerak ke kiri jika subtree kiri ada, atau menempatkan node baru sebagai anak kiri jika subtree kiri tidak ada.

2. Praktikum 2 (Implementasi binary tree dengan array)

```
PS D:\college\semester 2\Algoritma_dan_Struktur_Data_16_01> d:; cd 'd:\college\semester ava.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'D:\college\semester 2\Algoritm
InOrder Traversal : 3 4 5 6 7 8 9
```

NAMA: Adani Salsabila
NIM: 2341720123
KELAS: 1-G
MATERI: Tree (Jobsheet 11)

Pertanyaan!

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

Jawab : Kegunaan atribut data adalah untuk menyimpan elemen-elemen binary tree dalam bentuk array. Sedangkan atribut idxLast untuk menyimpan indeks dari elemen terakhir yang valid di dalam array data dan membantu dalam mengontrol batasan traversal.

2. Apakah kegunaan dari method populateData()?

Jawab : Method populateData() berfungsi untuk mengisi array data dan mengatur indeks terakhir (idxLast) dalam objek BinaryTreeArray01.

3. Apakah kegunaan dari method traverseInOrder()?

Jawab : Method traverseInOrder() adalah untuk melakukan traversal in-order pada binary tree yang diimplementasikan menggunakan array. Traversal ini memastikan node diakses dalam urutan left child, current node, right child, yang berguna untuk mendapatkan elemen-elemen tree.

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

Jawab: Dalam binary tree yang diimplementasikan menggunakan array, posisi left child dan right child dari suatu node tertentu dapat dihitung menggunakan indeks parent node tersebut. Jika indeks parent node adalah idxParent, maka indeks left child dari node tersebut adalah 2 * idxParent + 1. Jika indeks parent node adalah idxParent, maka indeks right child dari node tersebut adalah 2 * idxParent + 2.

a. Left Child:

o Indeks left child akan menjadi 2 * 2 + 1 = 5.

b. Right Child:

a. Indeks right child akan menjadi 2 * 2 + 2 = 6.

Jadi, jika node binary tree disimpan pada indeks 2, left child-nya akan berada pada indeks 5, dan right child-nya akan berada pada indeks 6 dalam array.

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

Jawab: Kegunaan statement int idxLast = 6; pada BinaryTreeArrayMain01 adalah untuk menentukan indeks terakhir dari elemen yang valid dalam array data yang akan diproses dalam struktur binary tree. Nilai 6 menunjukkan bahwa hanya elemen pada indeks 0 hingga 6 dari array data yang akan dianggap sebagai bagian dari tree, sedangkan elemen pada indeks 7, 8, dan 9 akan diabaikan.