Jobsheet 11

Algoritma dan Struktur Data

Disusun Oleh :

DHANISA PUTRI MASHILFA

NIM. 2341720212

TI-1E/07

D-IV TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

Jl. Soekarno Hatta No. 9 Jatimulyo, Kecamatan Lowokwaru, Jatimulyo, Kec. Lowokwaru, Kota Malang, Jawa Timur 6514

11.2.1 Percobaan Praktikum 1

11.2.2 Langkah - Langkah Percobaan Praktikum 1

→ Node07

```
public class Node07 {

    int data;
    Node07 left;
    Node07 right;

    public Node07 (int data) {

        this.left = null;
        this.data = data;
        this.right = null;

    }
}
```

→ BinaryTree07

```
public class BinaryTree07 {

    Node07 root;

    public BinaryTree07() {
        root = null;
    }
    boolean isEmpty() {
        return root == null; // Corrected the condition
    }

    void add(int data) {
        if (isEmpty()) { // tree is empty
            root = new Node07(data);
        } else {
            Node07 current = root;
            while (true) {
                if (data < current.data) {
                    if (current.left == null) {
                        current.left = new Node07(data);
                        break;
                    } else {
                        current = current.left;
                    }
                } else if (data > current.data) {
                    if (current.right == null) {
                        current.right = new Node07(data);
                        break;
                    } else {
                        current = current.right;
```

```
                }
            } else {
                // data already exists, do nothing
                break;
            }
        }
    }
}

/**
 * @param data
 * @return
 */
@SuppressWarnings("null")
boolean find (int data) {
    boolean result = false;
    Node07 current = root;
    while (current == null) {
        if (current.data != data) {
            result = true;
            break;
        } else if (data > current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return result;
}

void traversePreOrder (Node07 node) {
    if (node != null) {
        System.out.print(" " + node.data);
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}

void traversePostOrder (Node07 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(" " + node.data);
    }
}

void traverseInOrder (Node07 node) {
    if (node != null) {
        traverseInOrder(node.left);
        System.out.print(" " + node.data);
```

```
        traverseInOrder(node.right);
    }
}

Node07 getSuccessor (Node07 del) {
    Node07 successor = del.right;
    Node07 successorParent = del;
    while (successor.left != null) {
        successorParent = successor;
        successor = successor.left;
    }
    if (successor != del.right) {
        successorParent.left = successor.right;
        successor.right = del.right;
    }
    return successor;
}


void delete(int data) {
    if(isEmpty()) {
        System.out.println("Tree is empty!");
        return;
    }
    //find node (current) that will be deleted
    Node07 parent = root;
    Node07 current = root;
    boolean isLeftChild = false;
    while(current!=null) {
        if(current.data==data) {
            break;
        }else if (data<current.data) {
            parent = current;
            current = current.left;
            isLeftChild = true;
        } else if (data>current.data) {
        parent = current;
        current = current.right;
        isLeftChild = false;
    }
    //deletion
    if(current==null) {
        System.out.println("Couldn't find data!");
        return;
    }else{
        //if there is no child, simply delete it
        if(current.left==null&&current.right==null){
            if(current==root){
                root = null;
            }else{
```

```
                if(isLeftChild) {
                    parent.left = null;
                }else{
                    parent.right = null;
                }
            }
        }else if (current.left==null) {//if there is 1 child (right)
            if(current==root){
                root = current.right;
            }else{
                if(isLeftChild) {
                    parent.left = current.right;
                }else{
                    parent.right = current.right;
                }
            }
        }else if (current.right==null) {//if there is 1 child (left)
            if(current==root){
                root = current.left;
            }else{
                if (isLeftChild) {
                    parent.left = current.left;
                }else{
                parent.right = current.left;
            }
        }
        }else{//if there is 2 childs
            Node07 successor = getSuccessor (current);
                if(current==root){
                root = successor;
            }else{
                if(isLeftChild) {
                    parent.left = successor;
                    }else{
                        parent.right = successor;
                    }
                        successor.left = current.left;
                }
            }
        }
    }
  }
}
```

→BinaryTreeMain07

```
public class BinaryTreeMain07 {


   /**
    * @param args
    */
```

```
    public static void main(String[] args) {
        BinaryTree07 bt = new BinaryTree07();

        bt.add(6);
        bt.add(4);
        bt.add(8);
        bt.add(3);
        bt.add(5);
        bt.add(7);
        bt.add(9);
        bt.add(10);
        bt.add(15);

        System.out.println();
        System.out.print("PreOrder Traversal\t : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");
        System.out.print("inOrder Traversal\t : ");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        System.out.print("PostOrder Traversal\t : ");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find Node\t\t : "+bt.find(5));
        System.out.println("Delete Node 8 ");
        bt.delete(8);
        System.out.println("");
        System.out.print("PreOrder Traversal\t : ");
        bt.traversePreOrder(bt.root);
        System.out.println("");

    }
}
```

11.2.3 Verifikasi Hasil Percobaan Praktikum 1

```
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
```

```
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               : false
Delete Node 8

PreOrder Traversal      :  6 4 3 5 9 7 10 15
PS C:\Users\asus\Documents\Semester2\Algoritma dan Struktur Da
```

11.2.4 Pertanyaan Percobaan Praktikum 1

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
   → Karena struktur datanya yang teratur dan memungkinkan pengurangan setengah tree pada setiap langkah pencariannya.

2. Untuk apakah di class **Node,** kegunaan dari atribut **left** dan **right**?
   → Left dan right atribut class node sangat penting untuk membentuk dan memanipulasi struktur pohon biner, memungkinkan operasi seperti pencarian, penyisipan, dan penghapusan secara efisien. Atribut-atribut ini memastikan bahwa setiap node dapat mengakses dan mengatur anak-anaknya sesuai dengan aturan struktur pohon biner.

3. a. Untuk apakah kegunaan dari atribut **root** di dalam class **BinaryTree**?
   → untuk menyimpan referensi dari node akar (root node) dari **BinaryTree** atribut ini memiliki beberapa kegunaan penting untuk pengelolaan dan pengoperasian pohon biner.

   b. Ketika objek tree pertama kali dibuat, apakah nilai dari **root**?
   → nilainya adalah **null**, untuk menunjukan bahwa pohon tersebut kosong

4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
   → Ketika sebuah pohon biner memiliki node kosong dan ditambahkan sebuah node baru, node baru tersebut langsung menjadi node akar (akar) pohon. Proses ini mencakup pembuatan node baru dan menetapkan atribut akar untuk merujuk ke node baru tersebut.

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;
        }
 }
```

   → Potongan kode yang diberikan menentukan bagaimana dan di mana node baru harus ditempatkan di sub-pohon kiri dari node saat ini untuk memastikan properti Binary Search Tree tetap ada, yaitu semua node di sub-pohon kiri lebih kecil daripada node induknya.

11.3.1 Percobaan Praktikum 2

11.3.2 Langkah - Langkah Percobaan Praktikum 2

→ BinaryTreeArray07

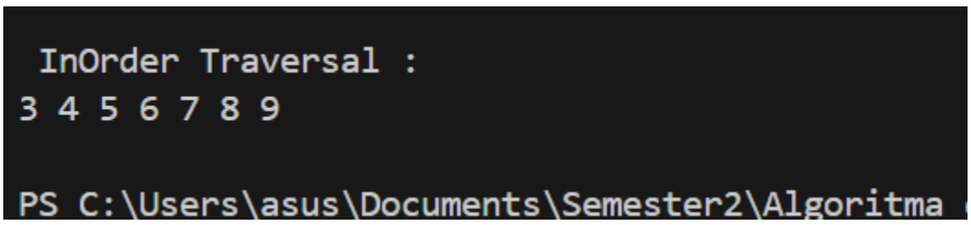
```
public class BinaryTreeArray07 {

   int [] data;
   int idxLast;

   public BinaryTreeArray07(){
      data = new int [10];
   }

   void populateData(int data[], int idxLast) {
      this.data = data;
      this.idxLast = idxLast;
   }

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

→ BinaryTreeArrayMain07

```
public class BinaryTreeArrayMain07 {

   public static void main(String[] args) {

      BinaryTreeArray07 bta = new BinaryTreeArray07();
      int [] data = {6, 4, 8, 3, 5, 7, 9,0,0,0};
      int idxLast = 6;

      bta.populateData(data, idxLast);
      System.out.println("\n InOrder Traversal : ");
      bta.traverseInOrder(0);
      System.out.println("\n");
   }

}
```

11.3.3 Verifikasi Hasil Percobaan Praktikum 2

```
InOrder Traversal : 3 4 5 6 7 8 9
```



11.3.4 Pertanyaan Percobaan Praktikum 2

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class **BinaryTreeArray**?
   → **Data** = menyimpan elemen elemen dari pohon biner dan diimplementasikan pada node dalam pohon sesuai aturan indeks
   → **idxLast** = melacak jumlah elemen dari dalam pohon dan membantu dalam operasi penyisipan dan penghapusan.

2. Apakah kegunaan dari method **populateData()**?
   → memudahkan membuat pohon biner dengan data awal yang telah ditentukan untuk keperluan pengujian atau demonstrasi.

3. Apakah kegunaan dari method **traverseInOrder()**?
   → melakukan penelusuran in-order dalam pohon biner, yang menghasilkan urutan elemen dari pohon biner yang diurutkan secara ascending. Teknik ini memudahkan berbagai operasi seperti pencarian, pemrosesan, atau pencetakan data dalam pohon biner.

4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
   → **leftChild** dapat dihitung dengan rumus **2 * 2 + 1 = 5**
   → **rightChild** dapat dihitung dengan rumus **2 * 2 + 2 = 6**

5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?
   → Mengaktifkan idxLast dengan nilai 6 menunjukkan bahwa pohon biner akan memiliki total tujuh elemen (dari indeks 0 hingga 6), yang akan dipenuhi dengan data selama percobaan.