LAPORAN PRAKTIKUM ALGORITMA STRUKTUR DATA

Pertemuan 14: Tree

MARGA RETA NOVIA PUTRI

2341760017

D-1V SISTEM INFORMASI BISNIS

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

2023/2024

13.2 Kegiatan Praktikum 1

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 Node19 {
2
3       int data;
4       Node19 left;
5       Node19 right;
6
7       public Node19() {
8       }
9       public Node19(int data) {
10          this.left = null;
11          this.data = data;
12          this.right = null;
13      }
14  }
```

3. Di dalam class BinaryTreeNoAbsen, tambahkan atribut root.

```
1   public class BinaryTree19 {
2       Node19 root;
```

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

```
4       public BinaryTree19() {
5           root = null;
6       }
7       boolean isEmpty() {
8           return root!=null;
9       }
10
```

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.

```
11      void add(int data) {
12          if (isEmpty()) {
13              root = new Node19(data);
14          } else {
15              Node19 current = root;
16              Node19 parent = null;
17              while (true) {
18                  parent = current;
19                  if (data < current.data) {
20                      if (current.left == null) {
21                        current.left = new Node19(data);
22                          break;
23                      } else {
24                          current = current.left;
25                      }
26                  } else if (data > current.data) {
27                      if (current.right == null) {
28                          current.right = new Node19(data);
29                          break;
30                      } else {
31                          current = current.right;
32                      }
33                  } else {
34                      break;
35                  }
36              }
37          }
38      }
```

6. Tambahkan method find()

```
39      boolean find(int data) {
40          Node19 current = root;
41          while (current!= null) {
42              if (current.data == data) {
43                  return true;
44              } else if (data < current.data) {
45                  current = current.left;
46              } else {
47                  current = current.right;
48              }
49          }
50          return false;
51      }
```

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.

```
39      boolean find(int data) {
40          Node19 current = root;
41          while (current!= null) {
42              if (current.data == data) {
43                  return true;
44              } else if (data < current.data) {
45                  current = current.left;
46              } else {
47                  current = current.right;
48              }
49          }
50          return false;
51      }
52      void traversePreOrder(Node19 node) {
53          if (node!= null) {
54              System.out.print(" " + node.data);
55              traversePreOrder(node.left);
56              traversePreOrder(node.right);
57          }
58      }
59
60      void traversePostOrder(Node19 node) {
61          if (node!= null) {
62              traversePostOrder(node.left);
63              traversePostOrder(node.right);
64              System.out.print(" " + node.data);
65          }
66      }
```

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

```
75      Node19 getSuccessor(Node19 del) {
76          Node19 successor = del.right;
77          Node19 successorParent = del;
78          while (successor.left!= null) {
79              successorParent = successor;
80              successor = successor.left;
81          }
82          if (successorParent!= del) {
83              successorParent.left = successor.right;
84          } else {
85              successorParent.right = successor.right;
86          }
87          return successor;
88      }
```

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

```
    void delete(int data) {
        if (isEmpty()) {
            System.out.println(x:"Tree is Empty!");
            return;
        }

        Node19 parent = root;
        Node19 current = root;
        boolean isLeftChild = false;
        while (current.data!= data) {
            parent = current;
            if (data < current.data) {
                current = current.left;
                isLeftChild = true;
            } else {
                current = current.right;
                isLeftChild = false;
            }
            if (current == null) {
                System.out.println(x:"Couldn't find data!");
                return;
            }
        }
```

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

```
    //DELECTION
    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 (current == root) {
            root = current.right;
        } else {
            if (isLeftChild) {
                parent.left = current.right;
            } else {
                parent.right = current.right;
            }
        }
    }else if (current.right == null) {
        if (current == root) {
            root = current.left;
        } else {
            if (isLeftChild) {
                parent.left = current.left;
            } else {
                parent.right = current.left;
            }
        }
    }else {
        Node19 successor = getSuccessor(current);
        if (current == root) {
            root = successor;
        } else {
            if (isLeftChild) {
                parent.left = successor;
            } else {
                parent.right = successor;
            }
            successor.left = current.left;
        }
```

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

```
public class BinaryTreeMain19 {
    public class BinaryTreeMain14 {
        public static void main(String[] args) {
            BinaryTree19 bt = new BinaryTree19();
            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 : " + 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:"");
        }
    }
}
```

12. Compile dan jalankan class BinaryTreeMain untuk mendapatkan simulasi jalannya program tree yang telah dibuat.
13.

```
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
PS D:\SEMESTER2\ASD\PRAKTIKUM\jobsheet 14>
```

13.2.2 Pertanyaan Percobaan

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

}
```

   Jawab: 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

1. Buatlah class BinaryTreeArrayNoAbsen dan BinaryTreeArrayMainNoAbsen
2. Buat atribut data dan idxLast di dalam class BinaryTreeArrayNoAbsen. Buat juga method populateData() dan traverseInOrder().

```
public class BinaryTreeArray19 {
    int[] data;
    int idxLast;

    public BinaryTreeArray19() {
        data = new int[10];
        idxLast = -1;
    }

    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);
        }
    }
}
```

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

```
    public static void main(String[] args) {
        BinaryTreeArray19 bta = new BinaryTreeArray19();
        int[] data = {6, 4, 8, 3, 5, 7, 9, 0, 0, 0};
        int idxLast = 6;

        bta.populateData(data, idxLast);
        System.out.print(s:"\nInOrder Traversal : ");
        bta.traverseInOrder(idxStart:0);
        System.out.println(x:"\n");
    }
}
```

4. Jalankan class BinaryTreeArrayMain dan amati hasilnya!

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

13.3.2 Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
   Jawab: Atribut idxLast digunakan untuk mengetahui pada indeks berapa data terakhir kali diletakkan.

2. Apakah kegunaan dari method populateData()?
   Jawab: 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()?
   Jawab: 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 rigth child masin-masing?
   Jawab: Untuk menentukan indeks posisi terdapat 2 cara sebagai berikut

   - Left child = 2*i+1 ; jika indeks posisi 2 maka => 2*2+1 => 5

   - Right child = 2*i+2 ; jika indeks posisi 2 maka => 2*2+2 => 6

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

```
12      public void add(int data) {
13          root = addRecursive(root, data);
```

```
39      public Node19 addRecursive(Node19 current, int data) {
40          if (current==null) {
41              return new Node19(data);
42          }
43          if (data<current.data) {
44              current.left = addRecursive(current.left, data);
45          } else if (data>current.data) {
46              current.right = addRecursive(current.right, data);
47          } else {
48              return current;
49          }
50          return current;
51      }
```

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

```
53      public boolean find(int data) {
54          Node19 current = root;
55          while (current != null) {
56              if (current.data == data) {
57                  return true;
58              } else if (data < current.data) {
59                  current = current.left;
60              } else {
61                  current = current.right;
62              }
63          }
64          return false;
65      }
```

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

```
192     public void printLeafNodes() {
193         displayLeafData(root);
194     }
```

```
211     public int countLeafNodesRekursif(Node19 node) {
212         if (node == null) {
213             return 0;
214         }
215         if (node.left == null && node.right == null) {
216             return 1;
217         }
218         return countLeafNodesRekursif(node.left) + countLeafNodesRekursif(node.right);
```

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

```java
public int countLeafNodes() {
    return countLeafNodesRekursif(root);
}

public int countLeafNodesRekursif(Node19 node) {
    if (node == null) {
        return 0;
    }
    if (node.left == null && node.right == null) {
        return 1;
    }
    return countLeafNodesRekursif(node.left) + countLeafNodesRekursif(node.right);

```

5. Modifikasi class BinaryTreeArray, dan tambahkan :

• method add(int data) untuk memasukan data ke dalam tree

```java
public void add(int data) {
    // root = addRecursive(root, data);//
    Node19 newNode = new Node19(data);
    if (isEmpty()) {
        root = newNode;
    } else {
        Node19 current = root;
        Node19 parent;
        while (true) {
            parent = current;
            if (data < current.data) {
                current = current.left;
                if (current == null) {
                    parent.left = newNode;
                    return;
                }
            } else {
                current = current.right;
                if (current == null) {
                    parent.right = newNode;
                    return;
```

• method traversePreOrder() dan traversePostOrder()

```java
public void traversePreOrder(Node19 node) {
    if (node != null) {
        System.out.print(node.data + " ");
        traversePreOrder(node.left);
        traversePreOrder(node.right);
    }
}
```

```java
public void traversePostOrder(Node19 node) {
    if (node != null) {
        traversePostOrder(node.left);
        traversePostOrder(node.right);
        System.out.print(node.data + " ");
    }
```