Tugas 5

Nama : M irfan S

NRP 17111051

Source code

1.

```
1.  public class BinaryTreeNode {
2.
3.      BinaryTreeNode parent;
4.      BinaryTreeNode left;
5.      BinaryTreeNode right;
6.      int data;
7.
8.      BinaryTreeNode(int new_data) {
9.          this.data = new_data;
10.         this.parent = null;
11.         this.left = null;
12.         this.right = null;
13.     }
14.
15.
16.     void set_parent(BinaryTreeNode other) {
17.         this.parent = other;
18.         if (other != null) {
19.             if (other.data > this.data) {
20.                 other.left = this;
21.             } else {
22.                 other.right = this;
23.             }
24.         }
25.     }
26.
27.     void set_left(BinaryTreeNode other) {
28.         this.left = other;
29.         if (other != null) {
30.             other.parent = this;
31.         }
32.     }
33.
34.     void set_right(BinaryTreeNode other) {
35.         this.right = other;
36.         if (other != null) {
37.             other.parent = this;
38.         }
39.     }
40.     boolean is_left() {
41.         return this.parent != null && parent.left == this;
42.     }
43.
44.     boolean is_right() {
45.         return this.parent != null && parent.right == this;
46.     }
47.
48.     boolean has_right_and_left() {
49.         return this.left != null && this.right != null;
```

```
50.     }
51.
52.     boolean only_has_left() {
53.         return this.left != null && this.right == null;
54.     }
55.
56.     boolean only_has_right() {
57.         if (this.right != null || this.left == null) {
58.
59.         }
60.         return this.right != null && this.left == null;
61.     }
62.
63.     boolean has_no_child() {
64.         return this.left == null && this.right == null;
65.     }
66.     void unset_parent() {
67.         if (this.is_left()) {
68.             parent.left = null;
69.             this.parent = null;
70.
71.         } else if (this.is_right()) {
72.             parent.right = null;
73.             this.parent = null;
74.
75.         }
76.     }
77.
78.
79.     BinaryTreeNode most_left_child() {
80.         BinaryTreeNode child = this.left;
81.         while (child.left != null) {
82.             child = child.left;
83.
84.         }
85.         return child;
86.     }
87.
88.     BinaryTreeNode most_right_child() {
89.         BinaryTreeNode child = this.right;
90.         while (child.right != null) {
91.             child = child.right;
92.         }
93.         return child;
94.     }
95.
96.     void print(String spaces, String label) {
97.         System.out.println(spaces + label + this.data);
98.         if (this.left != null) {
99.             this.left.print(spaces +" ", " LEFT ");
100.            }
101.            if (this.right != null) {
102.                this.right.print(spaces+ " ", " RIGHT ");
103.            }
104.        }
105.
106.        void print() {
107.            this.print(" ", "NODE ");
108.        }
109.
110.        void infix() {
111.            System.out.print("( ");
112.            if (this.left != null) {
113.                left.infix();
114.            } else {
115.                System.out.print("null");
```

```
                }
                System.out.print(" " + this.data + " ");
                if (this.right != null) {
                    right.infix();
                } else {
                    System.out.print("null");
                }
                System.out.print(")");
            }

            void prefix() {
                System.out.print(this.data + "(");
                if (this.left != null) {
                    left.prefix();
                } else {
                    System.out.print("null");
                }
                System.out.print(" ");
                if (this.right != null) {
                    right.prefix();
                } else {
                    System.out.print("null");
                }
                System.out.print(") ");
            }
            void postfix() {
                System.out.print("( ");
                if (this.left != null) {
                    left.postfix();
                } else {
                    System.out.print("null");
                }
                System.out.print(" ");
                if (this.right != null) {
                    right.postfix();
                } else {
                    System.out.print("null");
                }
                System.out.print(")" + this.data);
            }
        }
```

2.

```
import java.util.Scanner;

public class search {

    public static void main(String[] args) {
        BinaryTree bt = new BinaryTree();
        Scanner sc = new Scanner(System.in);
        int angka = 0, jumangka, cari;
        char ulang = 'y';

        System.out.println("    ** Binary Search dalam Binary Tree **");
        System.out.println("---------------------------------------");
        System.out.print("Masukan jumlah angka\t: ");
        jumangka = sc.nextInt();
        for (int i = 0; i < jumangka; i++) {
            System.out.print("Angka ke " + (i + 1) + "\t: ");
            angka = sc.nextInt();
            bt.push(new BinaryTreeNode(angka));
        }
```

```
        System.out.println("--------------------------------");
        bt.print();
        do {
            System.out.println("--------------------------------");
            System.out.print("Masukan angka yang anda cari : ");
            cari = sc.nextInt();
            bt.caricari(cari);
            do {
                System.out.print("Cari Angka lagi? (Y / T)\t");
                ulang = sc.next().charAt(0);
            } while (ulang != 't' && ulang != 'y');

        } while (ulang == 'y');

    }

}
```

3.

```
public class BinaryTree {

    BinaryTreeNode root;

    public BinaryTree() {
        this.root = null;
    }

    void print() {
        if (this.root != null) {
            this.root.print();
        }
    }

    void prefix() {
        if (this.root != null) {
            this.root.prefix();
        }
        System.out.println("");
    }

    void infix() {
        if (this.root != null) {
            this.root.infix();
        }
        System.out.println("");
    }
    void postfix() {
        if (this.root != null) {
            this.root.postfix();
        }
        System.out.println("");
    }
    void push(BinaryTreeNode new_node) {
        if (this.root == null) {
            this.root = new_node;
        } else {
            BinaryTreeNode current = this.root;
            while (current != null) {
                if (new_node.data > current.data) {
                    if (current.right == null) {
                        current.set_right(new_node);
                        break;
```

```
                    } else {
                        current = current.right;
                    }
                } else {
                    if (current.left == null) {
                        current.set_left(new_node);
                        break;
                    } else {
                        current = current.left;
                    }
                }
            }
        }
    }

    void delete(BinaryTreeNode deleted) {
        if (this.root != null) {
            if (deleted.has_no_child()) {
                if (deleted == this.root) {
                    this.root = null;
                } else {
                    deleted.unset_parent();
                }
            } else if (deleted.only_has_left() || deleted.only_has_right()) {
                BinaryTreeNode replacement = null;
                if (deleted.only_has_left()) {
                    replacement = deleted.left;
                } else {
                    replacement = deleted.right;
                }
                if (deleted == this.root) {
                    this.root = replacement;
                    this.root.unset_parent();

                } else if (deleted.is_left()) {
                    deleted.parent.set_left(replacement);
                    deleted.unset_parent();

                } else if (deleted.is_right()) {
                    deleted.parent.set_right(replacement);
                    deleted.unset_parent();

                }
            } else {
                BinaryTreeNode replacement = deleted.left;
                if (replacement.right != null) {
                    replacement = replacement.most_right_child();
                }
                BinaryTreeNode parent_of_replacement = replacement.parent;
                if (replacement.only_has_right()) {
                    parent_of_replacement.set_left(replacement.right);
                }
                replacement.unset_parent();
                replacement.set_left(deleted.left);
                replacement.set_right(deleted.right);
              if (deleted == this.root) {
                            this.root = replacement;
                        } else if (deleted.is_left()) {
                            deleted.parent.set_left(replacement);
                        } else if (deleted.is_right()) {
                            deleted.parent.set_right(replacement);
                        }
                    }
                }
            }

```

```
110.          void caricari(int key) {
111.              if (this.root == null) {
112.                  System.out.println("kosong );
113.              } else {
114.                  BinaryTreeNode current = this.root;
115.                  while (current != null) {
116.                      if (key == current.data) {
117.                          System.out.println("Angka tersebut ada");
118.                          break;
119.                      }
120.                      if (key > current.data) {
121.                         current = current.right;
122.                      } else {
123.                          current = current.left;
124.                      }
125.                      }
126.              }
127.
128.          }
129.
130.      }
```

Output: