LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

2022

Praktikan

2141720183

RIDWAN CAESAR RIZQI KARISMA BIWARNI

Daftar Isi

Percobaan implementasi Binary Search Tree menggunakan Linked List

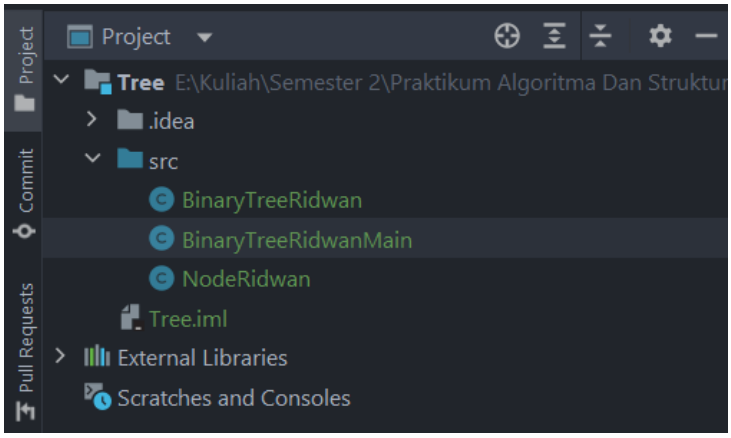
Pada percobaan ini akan diimplementasikan Binary Search Tree dengan operasi dasar, dengan menggunakan array (praktikum 2) dan linked list (praktikum 1). Sebelumnya, akan dibuat class Node, dan Class BinaryTree

Node
data: int left: Node right: Node
Node(left: Node, data:int, right:Node)

BinaryTree
root: Node size : int
DoubleLinkedLists() add(data: int): void find(data: int) : boolean traversePreOrder (node : Node) : void traversePostOrder (node : Node) void traverseInOrder (node : Node): void getSuccessor (del: Node) add(item: int, index:int): void delete(data: int): void

Langkah 1

Buatlah class Node, BinaryTree dan BinaryTreeMain



Langkah 2

Di dalam class Node, tambahkan atribut data, left dan right, serta konstruktor default dan berparameter.

```java
public class NodeRidwan {
    int data;
    NodeRidwan left;
    NodeRidwan right;

    public NodeRidwan() {

    }
    public NodeRidwan(int data) {
        this.left = null;
        this.data = data;
        this.right = null;
    }
}
```

Langkah 3

Di dalam class BinaryTree, tambahkan atribut root.

```java
public class BinaryTreeRidwan {
    NodeRidwan root;
}
```

Langkah 4

Tambahkan konstruktor default dan method isEmpty() di dalam class BinaryTree

```java
public class BinaryTreeRidwan {
    NodeRidwan root;

    public BinaryTreeRidwan() {
        root = null;
    }
    boolean isEmpty() {
        return root == null;
    }
}
```

Langkah 5

Tambahkan method add() di dalam class BinaryTree. 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 NodeRidwan(data);
14          } else {
15              NodeRidwan current = root;
16              while (true) {
17                  if (data < current.data) {
18                      if (current.left != null) {
19                          current = current.left;
20                      } else {
21                          current.left = new NodeRidwan(data);
22                          break;
23                      }
24                  } else if (data > current.data) {
25                      if (current.right != null) {
26                          current = current.right;
27                      } else {
28                          current.right = new NodeRidwan(data);
29                          break;
30                      }
31                  } else { // data sudah ada
32                      break;
33                  }
34              }
35          }
36      }
```

Langkah 6

Tambahkan method find()

```
boolean find(int data) {
    boolean hasil = false;
    NodeRidwan current = root;
    while (current != null) {
        if (current.data == data) {
            hasil = true;
            break;
        } else if (data < current.data) {
            current = current.left;
        } else {
            current = current.right;
        }
    }
    return hasil;
}
```

Langkah 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.

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

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

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

Langkah 8

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

```
80  @       NodeRidwan getSuccessor(NodeRidwan del) {
81              NodeRidwan successor = del.right;
82              NodeRidwan successorParent = del;
83              while (successor.left != null) {
84                  successorParent = successor;
85                  successor = successor.left;
86              }
87              if (successor != del.right) {
88                  successorParent.left = successor.right;
89                  successor.right = del.right;
90              }
91              return successor;
92          }
```

Langkah 9

Tambahkan method delete().

```
94          void delete(int data) {
95
96          }
```

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

```
95          if (isEmpty()) {
96              System.out.println("Tree is empty!");
97              return;
98          }
99          NodeRidwan parent = root;
100         NodeRidwan current = root;
101         boolean isLeftChild = false;
102         while (current != null) {
103             if (current.data == data) {
104                 break;
105             } else if (data < current.data) {
106                 parent = current;
107                 current = current.left;
108                 isLeftChild = true;
109             } else if (data > current.data) {
110                 parent = current;
111                 current = current.right;
112                 isLeftChild = false;
113             }
114         }
```

Kemudian tambahkan proses penghapusan terhadap node current yang telah ditemukan.

```
        // proses hapus
        if (current == null) {
            System.out.println("Couldn't find data!");
            return;
        } else {
            // jika tidak memiliki child
          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) { // 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) { // 1 child left
              if (current == root) {
                  root = current.left;
              } else {
                  if (isLeftChild) {
                      parent.left = current.left;
                  } else {
                      parent.right = current.left;
                  }
              }
          } else { // memiliki dua child
              NodeRidwan successor = getSuccessor(current);
              if (current == root) {
                  root = successor;
              } else {
                  if (isLeftChild) {
                      parent.left = successor;
                  } else {
                      parent.right = successor;
                  }
                  successor.left = current.left;
              }
          }
        }
    }
```

Langkah 10

Buka class BinaryTreeMain dan tambahkan method main().

```java
public class BinaryTreeRidwanMain {
    public static void main(String[] args) {
        BinaryTreeRidwan bt = new BinaryTreeRidwan();

        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);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find " + bt.find( data: 5));
        bt.delete( data: 8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
    }
}
```

Langkah 11

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

```
"C:\Users\Asus TUF DT\.jdks\openjdk-
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15

Process finished with exit code 0
```

Langkah 12

Amati hasil running tersebut.

Pertanyaan Percobaan

1. Mengapa dalam binary search tree proses pencarian data bisa lebih efektif dilakukan dibanding binary tree biasa?
2. Untuk apakah di class Node, kegunaan dari atribut left dan right?
3. a. Untuk apakah kegunaan dari atribut root di dalam class BinaryTree?
   b. Ketika objek tree pertama kali dibuat, apakah nilai dari root?
4. Ketika tree masih kosong, dan akan ditambahkan sebuah node baru, proses apa yang akan terjadi?
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 NodeRidwan(data);
                break;
        }
}
```

Jawaban Percobaan

1. Karena dalam binary search tree seluruh child dari tiap node sudah dalam keadaan terurut sehingga pencarian data lebih efektif.
2. Atribut left digunakan sebagai pointer yang akan menunjuk ke child disebelah kiri sedangkan atribut right digunakan sebagai pointer yang akan menunjuk ke child disebelah kanan.
3. a. Root digunakan sebagai data yang pertama kali dimasukkan ke dalam tree dan akan dimanfaatkan dalam operasi tree.
   b. Nilai dari root ketika tree pertama kali dibuat adalah null.
4. Proses yang akan terjadi adalah root akan di isi dengan node baru.
5. Pada baris kode tersebut akan dilakukan pengecekan apakah data yang akan ditambahkan bernilai kurang dari nilai data current. Kemudian akan dilakukan pengecekan lagi jika nilai child kiri dari current bukan null maka current akan bernilai child sebelah kiri tadi, sedangkan jika tidak maka child di sebelah kiri akan di isi node baru dan akan keluar dari perulangan.

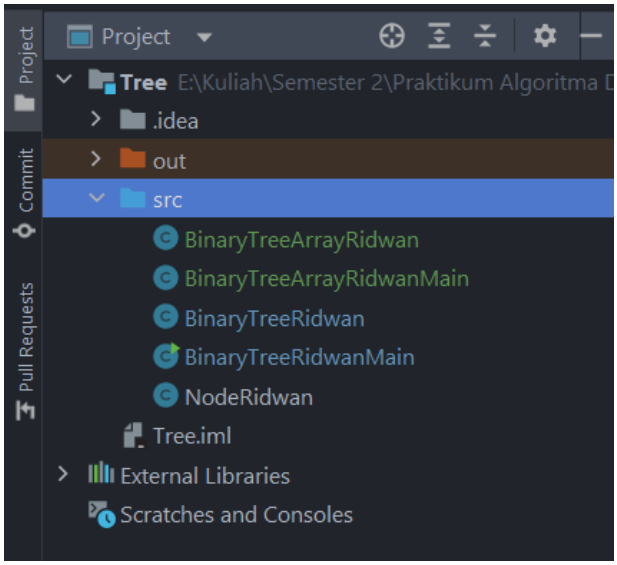
Percobaan implementasi binary tree dengan array

Langkah 1

Di dalam percobaan implementasi binary tree dengan array ini, data tree disimpan dalam array dan langsung dimasukan dari method main(), dan selanjutnya akan disimulasikan proses traversal secara inOrder.

Langkah 2

Buatlah class BinaryTreeArray dan BinaryTreeArrayMain



Langkah 3

Buat atribut data dan idxLast di dalam class BinaryTreeArray. Buat juga method populateData() dan traverseInOrder().

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

    public BinaryTreeArrayRidwan() {
        data = new int[10];
    }
    void populateData(int data[], int idxLast) {
        this.data = data;
        this.idxLast = idxLast;
    }
    void traverseInOrder(int idxStart) {
        if (idxStart <= idxLast) {
            traverseInOrder( idxStart: 2*idxStart+1);
            System.out.print(data[idxStart] + " ");
            traverseInOrder( idxStart: 2*idxStart+2);
        }
    }
}
```

Langkah 4

Kemudian dalam class BinaryTreeArrayMain buat method main() seperti gambar berikut ini.

```
public class BinaryTreeArrayRidwanMain {
    public static void main(String[] args) {
        BinaryTreeArrayRidwan bta = new BinaryTreeArrayRidwan();
        int[] data = {6,4,8,3,5,7,9,0,0,0};
        int idxLast = 6;
        bta.populateData(data, idxLast);
        bta.traverseInOrder( idxStart: 0);
    }
}
```

Langkah 5

Jalankan class BinaryTreeArrayMain dan amati hasilnya!

```
"C:\Users\Asus TUF DT\.jdks\openjdk-1
3 4 5 6 7 8 9
Process finished with exit code 0
```

Pertanyaan Percobaan

1. Apakah kegunaan dari atribut data dan idxLast yang ada di class BinaryTreeArray?
2. Apakah kegunaan dari method populateData()?
3. Apakah kegunaan dari method traverseInOrder()?
4. Jika suatu node binary tree disimpan dalam array indeks 2, maka di indeks berapakah posisi left child dan rigth child masin-masing?
5. Apa kegunaan statement int idxLast = 6 pada praktikum 2 percobaan nomor 4?

Jawaban Percobaan

1. Atribut data akan digunakan untuk menyimpan data dalam array sedangkan idxlast adalah index terakhir dalam array yang sudah di isi pada kasus ini data yang bukan 0.
2. Method tersebut digunakan untuk mengisi variabel data dan idxLast pada class BinaryTreeArray sehingga dapat dilakukan operasi seperti traverse.
3. Method tersebut digunakan untuk mencetak isi dari Tree dengan aturan InOrder.
4. Jika sebuah data berada pada indeks ke 2 maka left child akan berada pada indeks ke 5 dan right child berada pada indeks ke 6.
5. idxLast berfungsi untuk menentukan indeks terakhir pada array yang sudah berisi data yang sesuai dalam kasus ini data yang bukan 0.

Tugas Praktikum

1. Buat method di dalam class BinaryTree yang akan menambahkan node dengan cara rekursif
2. Buat method di dalam class BinaryTree untuk menampilkan nilai paling kecil dan yang paling besar yang ada di dalam tree.
3. Buat method di dalam class BinaryTree untuk menampilkan data yang ada di leaf.
4. Buat method di dalam class BinaryTree untuk menampilkan berapa jumlah leaf yang ada di dalam tree.
5. Modifikasi class BinaryTreeArray, dan tambahkan :
    - method add(int data) untuk memasukan data ke dalam tree
    - method traversePreOrder() dan traversePostOrder()

Jawaban Nomor 1

Menambahkan method add node dengan cara rekursif

```
11      // Modifikasi soal nomor 1 add rekursif
12      void add(int data, NodeRidwan current) {
13          if (isEmpty()) {
14              root = new NodeRidwan(data);
15          } else {
16              if (data < current.data) {
17                  if (current.left != null) {
18                      add(data, current.left);
19                  } else {
20                      current.left = new NodeRidwan(data);
21                  }
22              } else if (data > current.data) {
23                  if (current.right != null) {
24                      add(data, current.right);
25                  } else {
26                      current.right = new NodeRidwan(data);
27                  }
28              } else {
29                  System.out.println("Data sudah ada");
30              }
31          }
32      }
```

Modifikasi pada class BinaryTreeMain

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

        bt.add( data: 6, bt.root);
        bt.add( data: 4, bt.root);
        bt.add( data: 8, bt.root);
        bt.add( data: 3, bt.root);
        bt.add( data: 5, bt.root);
        bt.add( data: 7, bt.root);
        bt.add( data: 9, bt.root);
        bt.add( data: 10, bt.root);
        bt.add( data: 15, bt.root);

        bt.traversePreOrder(bt.root);
        System.out.println("");
        bt.traverseInOrder(bt.root);
        System.out.println("");
        bt.traversePostOrder(bt.root);
        System.out.println("");
        System.out.println("Find " + bt.find( data: 5));
        bt.delete( data: 8);
        bt.traversePreOrder(bt.root);
        System.out.println("");
```

Output

```
"C:\Users\Asus TUF DT\.jdks\openjd
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
```

Jawaban Nomor 2

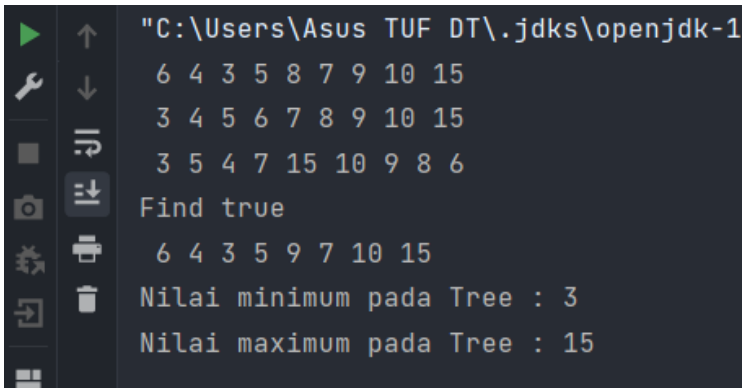
Modifikasi pada class BinaryTree

```
189     int findMax(NodeRidwan node) {
190         if (node == null) {
191             return Integer.MIN_VALUE;
192         }
193         int max = node.data;
194         int left = findMax(node.left);
195         int right = findMax(node.right);
196 
197         if (left > max)
198             max = left;
199         if (right > max)
200             max = right;
201         return max;
202     }
203 
204     int[] findMinMax() {
205         return new int[] {findMin(root), findMax(root)};
206     }
207 
208     int findMin(NodeRidwan node) {
209         if (node == null) {
210             return Integer.MAX_VALUE;
211         }
212         int min = node.data;
213         int left = findMin(node.left);
214         int right = findMin(node.right);
215 
216         if (left < min)
217             min = left;
218         if (right < min)
219             min = right;
220         return min;
221     }
```

Modifikasi pada class BinaryTreeMain

```
25          int[] minMax = bt.findMinMax();
26          if (minMax[0] == Integer.MIN_VALUE || minMax[1] == Integer.MAX_VALUE) {
27              System.out.println("Tree kosong");
28          } else {
29              System.out.println("Nilai minimum pada Tree : " + minMax[0]);
30              System.out.println("Nilai maximum pada Tree : " + minMax[1]);
31          }
```

Output



Jawaban Nomor 3

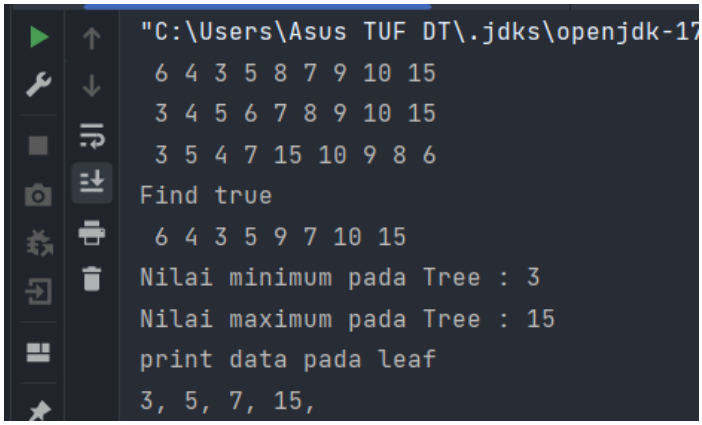
Menambahkan method pada class BinaryTree

```
223     void printLeaf(NodeRidwan node) {
224         if (node == null) {
225             return;
226         }
227         if (node.left == null && node.right == null) {
228             System.out.print(node.data + ", ");
229         }
230         printLeaf(node.left);
231         printLeaf(node.right);
232     }
```

Modifikasi pada class BinaryTreeMain

```
32          System.out.println("print data pada leaf");
33          bt.printLeaf(bt.root);
```

Output



Jawaban Nomor 4

Menambahkan method untuk menghitung leaf pada class BinaryTree

```
234     int countLeaf(NodeRidwan node) {
235         if (node == null) {
236             return 0;
237         }
238         if (node.left == null && node.right == null) {
239             return 1;
240         } else {
241             return countLeaf(node.left) + countLeaf(node.right);
242         }
243     }
```

Modifikasi pada class BinaryTreeMain

```
34          System.out.println();
35          System.out.println("Jumlah leaf : " + bt.countLeaf(bt.root));
```

Output

```
"C:\Users\Asus TUF DT\.jdks\openj
 6 4 3 5 8 7 9 10 15
 3 4 5 6 7 8 9 10 15
 3 5 4 7 15 10 9 8 6
Find true
 6 4 3 5 9 7 10 15
Nilai minimum pada Tree : 3
Nilai maximum pada Tree : 15
print data pada leaf
3, 5, 7, 15,
Jumlah leaf : 4
```

Jawaban Nomor 5

Menambahkan method add

```
20      void add(int input) {
21          if (data[0] == 0 && data[data.length-1] == 0) {
22              data[0] = input;
23          } else {
24              int current = 0;
25              while (true) {
26                  if (input < data[current]) {
27                      if (data[2*current+1] != 0) {
28                          current = 2*current+1;
29                      } else {
30                          data[2*current+1] = input;
31                          idxLast++;
32                          break;
33                      }
34                  } else if (input > data[current]) {
35                      if (data[2*current+2] != 0) {
36                          current = 2*current+2;
37                      } else {
38                          data[2*current+2] = input;
39                          idxLast++;
40                          break;
41                      }
42                  } else {
43                      break;
44                  }
45              }
46          }
47      }
```

Menambahkan method traversePreOrder() dan traversePostOrder()

```
20      void traversePreOrder(int idxStart) {
21          if (idxStart <= idxLast) {
22              System.out.print(data[idxStart] + " ");
23              traversePreOrder( idxStart: 2*idxStart+1);
24              traversePreOrder( idxStart: 2*idxStart+2);
25          }
26      }
27      void traversePostOrder(int idxStart) {
28          if (idxStart <= idxLast) {
29              traversePostOrder( idxStart: 2*idxStart+1);
30              traversePostOrder( idxStart: 2*idxStart+2);
31              System.out.print(data[idxStart] + " ");
32          }
33      }
```

Modifikasi pada class BinaryTreeArrayMain

```java
public class BinaryTreeArrayRidwanMain {
    public static void main(String[] args) {
        BinaryTreeArrayRidwan bta = new BinaryTreeArrayRidwan();
//        int[] data = {6,4,8,3,5,7,9,0,0,0};
//        int idxLast = 6;
//        bta.populateData(data, idxLast);
//        bta.traverseInOrder(0);
        bta.add(6);
        bta.add(4);
        bta.add(8);
        bta.add(3);
        bta.add(5);
        bta.add(7);
        bta.add(9);
        System.out.print("InOrder\n");
        bta.traverseInOrder( idxStart: 0);
        System.out.println();
        System.out.print("PreOrder\n");
        bta.traversePreOrder( idxStart: 0);
        System.out.println();
        System.out.print("PostOrder\n");
        bta.traversePostOrder( idxStart: 0);
    }
}
```

Output

```
"C:\Users\Asus TUF DT\.jdks\openjdk-17
InOrder
3 4 5 6 7 8 9
PreOrder
6 4 3 5 8 7 9
PostOrder
3 5 4 7 9 8 6
Process finished with exit code 0
```