LAPORAN PRAKTIKUM STRUKTUR DATA
"*Tree*"

OLEH :

EZZA ADDINI

2311532001

DOSEN PENGAMPU: Dr. WAHYUDI, M. T.

DEPARTEMEN INFORMATIKA

FAKULTAS TEKNOLOGI INFORMASI

UNIVERSITAS ANDALAS

PADANG

2024

A. PENDAHULUAN

Tree adalah salah satu struktur data yang sangat penting dalam ilmu komputer dan pemrograman, termasuk dalam bahasa pemrograman Java. Struktur data tree terdiri dari node-node yang terhubung satu sama lain dengan cara hierarkis, menyerupai struktur pohon dengan akar (root) di bagian atas dan cabang-cabang (branches) yang menyebar ke bawah.

Tree terdiri dari berbagai komponen, yaitu:

a. Node: Elemen dasar dari tree. Setiap node dapat menyimpan data dan memiliki referensi (pointer) ke node lain.
b. Root: Node paling atas dari tree. Tree hanya memiliki satu root.
c. Child: Node yang berada di bawah node lain disebut child.
d. Parent: Node yang memiliki cabang (child) disebut parent.
e. Leaf: Node yang tidak memiliki child disebut leaf (daun).
f. Subtree: Bagian dari tree yang terdiri dari satu node dan semua turunannya.

Terdapat beberapa jenis tree, diantaranya adalah:

a. Binary Tree: Setiap node memiliki paling banyak dua child (left dan right).
b. Binary Search Tree (BST): Binary tree di mana setiap node kiri memiliki nilai lebih kecil dan setiap node kanan memiliki nilai lebih besar dari node induknya.
c. AVL Tree: Jenis binary search tree yang secara otomatis menyeimbangkan dirinya untuk memastikan operasi yang efisien.
d. Red-Black Tree: Binary search tree yang menggunakan aturan pewarnaan untuk menjaga keseimbangan.
e. Heap: Binary tree yang memenuhi properti heap, di mana node parent memiliki nilai lebih besar (max heap) atau lebih kecil (min heap) dari child-nya.
f. Trie: Tree yang digunakan untuk menyimpan himpunan string, biasanya digunakan dalam aplikasi pencarian string.

Operasi dasar yang umum digunakan pada tress adalah sebagai berikut:

a. Traversal: Mengunjungi semua node dalam tree. Ada beberapa metode traversal seperti:
   - In-order: Mengunjungi left child, kemudian root, lalu right child.
   - Pre-order: Mengunjungi root, kemudian left child, lalu right child.
   - Post-order: Mengunjungi left child, kemudian right child, lalu root.
   - Level-order: Mengunjungi node berdasarkan level-nya dari

atas ke bawah (breadth-first search).

b. Insertion: Menambahkan node baru ke tree.
c. Deletion: Menghapus node dari tree.
d. Searching: Mencari node tertentu dalam tree.

B. TUJUAN

Tujuan dari praktikum ini adalah:

1. Membuat program Node untuk sebuah Binary Tree.
2. Membuat program Binary Tree.
3. Membuat program untuk menjalankan Binary Tree.

C. LANGKAH KERJA

a. Program Node

1. Buat package dan class baru dan namakan sesuai dengan yang diperintahkan atau yg diinginkan.

```
Node.java × BTree.java TreeMain.java
1 package Pekan7;
2
3 public class Node {
```

2. Deklarasikan variabelnya.

```
int data;
Node left;
Node right;
```

3. Berikan sintaks instruktur node-nya.

```
7●    public Node(int data) {
8         this.data = data;
9         left = null;
10        right = null;
11    }
```

4. Buat metode setLeft untuk menetapkan node kiri dari node saat ini dalam tree. Metode ini hanya mengatur node kiri jika saat ini belum diatur (null).

```
12●   public void setLeft(Node node) {
13        if (left == null)
14            left = node;
15    }
```

5. Buat metode setRight untuk menetapkan node kanan dari node saat ini dalam tree. Metode ini hanya mengatur node kanan jika saat ini belum diatur (null).

```
16●   public void setRight(Node node) {
17        if (right == null)
18            right = node;
19    }
```

6. Buat metode getLeft untuk mengembalikan node kiri dari node saat ini dalam tree. Ini adalah metode accessor (getter) yang memungkinkan untuk mengakses nilai private dari variabel instance left.

```
20●   public Node getLeft() {
21        return left;
22    }
```

7. Buat metode getRight untuk mengembalikan node kanan dari node saat ini dalam tree. Ini adalah metode accessor (getter) yang memungkinkan untuk mengakses nilai private dari variabel instance right.

```
23●   public Node getRight() {
24        return right;
25    }
```

8. Buat metode getData untuk mengembalikan nilai data yang disimpan dalam node saat ini. Ini adalah metode accessor (getter)

yang memungkinkan untuk mengakses nilai private dari variabel instance data.

```
26●     public int getData() {
27          return data;
28      }
```

9. Buat metode setData untuk menetapkan nilai baru ke variabel data dalam node. Ini adalah metode mutator (setter) yang memungkinkan untuk mengubah nilai private dari variabel instance data.

```
29●     public void setData(int data) {
30          this.data = data;
31      }
```

10. Buat metode printPreorder untuk melakukan traversal pre-order pada tree dan mencetak nilai dari setiap node. Traversal pre-order mengunjungi node dalam urutan berikut: root, left, right. Lakukan pengecekan node null dan buat sintaks untuk menampilkan data node dengan operasi traversal pre-order.

```
*Node.java ×  BTree.java  TreeMain.java
31      }
32●     void printPreorder(Node node) {
33          if (node == null)
34              return;
35          System.out.print(node.data + " ");
36          printPreorder(node.left);
37          printPreorder(node.right);
38      }
```

11. Buat metode printPostorder untuk melakukan traversal post-order pada tree dan mencetak nilai dari setiap node. Traversal post-order mengunjungi node dalam urutan berikut: left, right, root. Lakukan pengecekan node null dan buat sintaks untuk menampilkan data node dengan operasi traversal post-order.

```
39●     void printPostorder(Node node) {
40          if (node == null)
41              return;
42          printPostorder(node.left);
43          printPostorder(node.right);
44          System.out.print(node.data + " ");
45      }
```

12. Buat metode printInorder untuk melakukan traversal in-order pada tree dan mencetak nilai dari setiap node. Traversal in-order mengunjungi node dalam urutan berikut: left, root, right. Lakukan pengecekan node null dan buat sintaks untuk menampilkan data node dengan operasi traversal in-order.

```
46●     void printInorder(Node node) {
47          if (node == null)
48              return;
49          printInorder(node.left);
50          System.out.print(node.data + " ");
51          printInorder(node.right);
52      }
```

13. Buat metode print untuk mengembalikan representasi string dari tree. Metode ini diasumsikan akan mengembalikan representasi berbasis string dari tree dalam bentuk yang lebih terstruktur.

```
53●     public String print() {
54          return this.print("", true, "");
55      }
```

14. Buat metode print menggunakan parameter prefix untuk menentukan indentasi, isTail untuk menentukan apakah node yang sedang diproses adalah node terakhir dalam level saat ini, dan sb untuk menyimpan representasi string dari tree. Buat kode yang terdiri atas rekursi ke child nodes, menampilkan data node, kembali ke parent node, serta mengembalikan string representasi.

```
56●     public String print(String prefix, boolean isTail, String sb) {
57          if (right != null) {
58              right.print(prefix + (isTail ? "|   " : "    "), false, sb);
59          }
60          System.out.println(prefix + (isTail ? "\\-- " : "/-- ") + data);
61          if (left != null) {
62              left.print(prefix + (isTail ? "    " : "|   "), true, sb);
63          }
64      return sb;
65      }
66  }
```

b. Program Binary Tree

1. Buat class baru dan namakan sesuai yang diinginkan.

```
*Node.java   BTree.java ×
 1 package Pekan7;
 2
 3 public class BTree {
```

2. Buat metode untuk dua variabel anggota root dan currentNode yang mewakili node root dan node saat ini dalam pohon.

```
 4      private Node root;
 5      private Node currentNode;
 6●     public BTree() {
 7          root = null;
 8      }
```

3. Buat metode search untuk mencari nilai tertentu dalam pohon dengan mulai dari root node. Ini adalah metode rekursif yang akan mencari nilai dalam pohon secara rekursif, dimulai dari root node.

```
 9●     public boolean search(int data) {
10          return search(root, data);
11      }
```

4. Buat metode search untuk melakukan pencarian rekursif dalam pohon dengan memeriksa setiap node, membandingkan nilai pada node dengan nilai yang ingin dicari. Jika nilai yang dicari ditemukan, maka metode akan mengembalikan true, jika tidak ditemukan, maka metode akan melanjutkan pencarian ke anak kiri dan/atau anak kanan dari node saat ini.

```
12●     private boolean search(Node node, int data) {
13          if (node.getData() == data)
14              return true;
15          if (node.getLeft() != null)
16              if (search(node.getLeft(), data))
17                  return true;
18          if (node.getRight() != null)
19              if (search(node.getRight(), data))
20                  return true;
21          return false;
22      }
```

5. Buat metode printInorder untuk mencetak traversal in-order dari pohon dengan memulai dari root node.

```
23●     public void printInorder() {
24          root.printInorder(root);
25      }
```

6. Buat metode printPreorder untuk mencetak traversal in-order dari pohon dengan memulai dari root node.

```
26●     public void printPreorder() {
27          root.printPreorder(root);
28      }
```

7. Buat metode printPostorder untuk mencetak traversal in-order dari pohon dengan memulai dari root node.

```
29●     public void printPostorder() {
30          root.printPostorder(root);
31      }
```

8. Buat metode getRoot untuk mengembalikan node root dari pohon. Ini adalah metode getter yang mengembalikan referensi ke root node dari pohon.

```
32●     public Node getRoot() {
33          return root;
34      }
```

9. Buat metode isEmpty untuk memeriksa apakah pohon kosong atau tidak. Ini adalah metode yang mengembalikan nilai boolean true jika pohon kosong (tidak memiliki root node), dan false jika tidak.

```
35●     public boolean isEmpty() {
36          return root == null;
37      }
```

10. Buat metode countNodes untuk menghitung jumlah total node dalam pohon dengan memulai dari root node.

```
38●     public int countNodes() {
39          return countNodes(root);
40      }
```

11. Eksekusi metode countNodes untuk menghitung jumlah total node dalam pohon dengan menggunakan pendekatan rekursif. Metode ini akan memeriksa setiap node dalam pohon dan menghitung jumlahnya.

```
41●     private int countNodes(Node node) {
42          int count = 1;
43          if (node == null) {
44              return 0;
45          } else {
46              count += countNodes(node.getLeft());
47              count += countNodes(node.getRight());
48              return count;
49          }
50      }
```

12. Buat metode print untuk mencetak representasi visual dari pohon dengan memulai dari root node.

```
51●     public void print() {
52          root.print();
53      }
```

13. Buat metode getCurrent untuk mengembalikan node saat ini dalam pohon.

```
54●     public Node getCurrent() {
55          return currentNode;
56      }
```

14. Buat metode setCurrent untuk mengatur node saat ini dalam pohon.

```
    public void setCurrent(Node node) {
        this.currentNode = node;
    }
```

15. Buat metode setRoot untuk mengatur node root dari pohon. Ini adalah metode setter yang mengatur nilai dari variabel root ke node yang disediakan sebagai argumen.

```
    public void setRoot(Node root) {
        this.root = root;
    }
}
```

c. Program untuk Menjalankan Binary Tree

1. Buat class baru dan namakan sesuai perintah atau yang diinginkan.

```
*Node.java  BTree.java  *TreeMain.java ×
public class TreeMain {
    public static void main(String[] args) {
```

2. Buat sintaks untuk membuat tree baru.

```
BTree tree = new BTree();
System.out.println("Jumlah simpul pohon: ");
System.out.println(tree.countNodes());
```

3. Buat sintaks untuk menambahkan simpul data 1 dan jadikan simpul 1 sebagai root.

```
Node root = new Node(1); //menambahkan simpul data 1
tree.setRoot(root); //menjadikan simpul 1 sebagai root
System.out.println("Jumlah simpul jika hanya ada root: ");
```

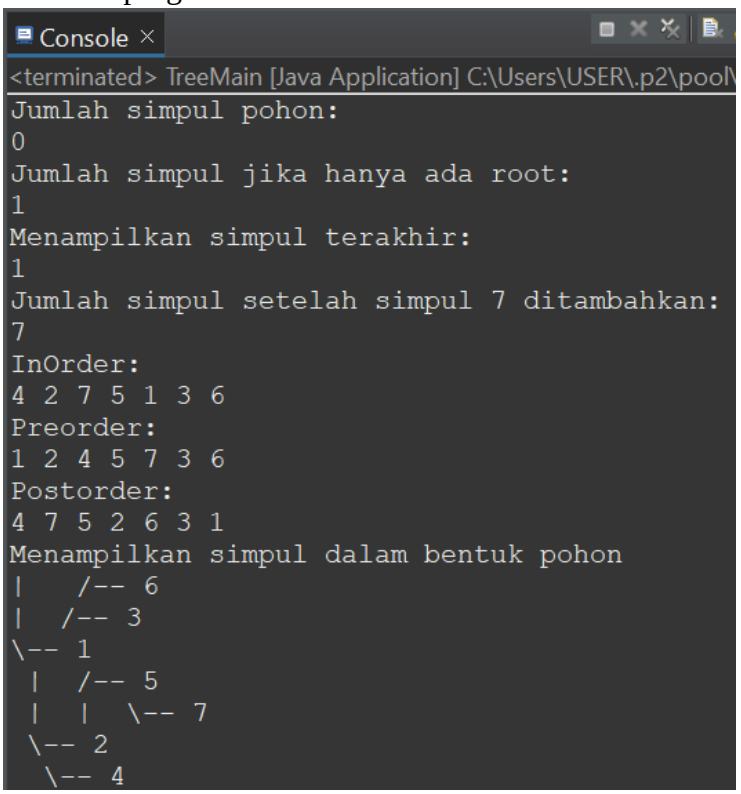
4. Buat metode countNodes untuk menghitung jumlah node dalam pohon dan tampilkan hasilnya. Kemudian, buat sintaks untuk membuat node baru dan mengatur struktur pohon dengan menetapkan anak kiri dan kanan dari node root dan beberapa node lainnya.

```
System.out.println(tree.countNodes());
Node node2 = new Node(2);
Node node3 = new Node(3);
Node node4 = new Node(4);
Node node5 = new Node(5);
Node node6 = new Node(6);
Node node7 = new Node(7);
root.setLeft(node2);
node2.setLeft(node4);
node2.setRight(node5);
node5.setLeft(node7);
root.setRight(node3);
node3.setRight(node6);
```

5. Buat sintaks untuk set root.

```
tree.setCurrent(tree.getRoot());
System.out.println("Menampilkan simpul terakhir: ");
System.out.println(tree.getCurrent().getData());
System.out.println("Jumlah simpul setelah simpul 7 ditambahkan: ");
System.out.println(tree.countNodes());
System.out.println("InOrder: ");
tree.printInorder();
System.out.println("\nPreorder: ");
tree.printPreorder();
System.out.println("\nPostorder: ");
tree.printPostorder();
System.out.println("\nMenampilkan simpul dalam bentuk pohon");
tree.print();
```

6. Jalankan program.



D. KESIMPULAN

Dalam praktikum ini, kita telah mengeksplorasi berbagai aspek dari struktur data tree dalam Java. Berikut adalah beberapa kesimpulan yang dapat diambil dari praktikum ini:

1. Implementasi dan Manipulasi Tree:
    - Kita mempelajari cara mendefinisikan dan mengimplementasikan node dalam tree, serta cara mengatur hubungan antara node, seperti mengatur anak kiri dan kanan dari node.
    - Metode seperti setRoot dan setCurrent memungkinkan kita untuk mengatur node root dan node saat ini dalam tree.
2. Traversal dan Operasi Dasar:
    - Metode traversal seperti printInorder, printPreorder, dan printPostorder menunjukkan cara mengunjungi setiap node dalam tree dengan urutan tertentu.
    - Operasi pencarian (search), penambahan node (insertion), dan penghitungan jumlah node (countNodes) memberikan wawasan tentang bagaimana manipulasi dan interaksi dengan tree dapat dilakukan secara efisien.
3. Pentingnya Struktur Data Tree:
    - Struktur data tree sangat penting dalam berbagai aplikasi, seperti pencarian cepat, pengurutan, dan hierarki data. Dengan pemahaman yang baik tentang operasi dasar dan jenis-jenis tree, kita dapat mengimplementasikan solusi yang lebih efisien dan terstruktur dalam pemrograman.
4. Pemahaman Rekursi dalam Operasi Tree:
    - Banyak operasi pada tree memanfaatkan rekursi, yang memungkinkan kita untuk mengunjungi dan memproses setiap node secara elegan. Ini menunjukkan kekuatan dan keindahan dari pendekatan rekursif dalam memecahkan masalah yang kompleks.

Secara keseluruhan, praktikum ini memberikan dasar yang kuat dalam memahami dan mengimplementasikan struktur data tree dalam Java. Dengan pemahaman ini, kita dapat lebih siap untuk menghadapi tantangan pemrograman yang melibatkan struktur data hierarkis dan operasi yang kompleks.