Nama : Muhammad Fayyadh Raihan

Kelas : RPLA 02

NIM : 607062300093

1. Binary Tree
2. class BinaryTree2<E> {
3. private node2 root;
4. public BinaryTree2() {
5. root = null;  // Inisialisasi pohon tanpa simpul akar.
6. }
7. public boolean isEmpty() {
8. return root == null;  // Periksa apakah pohon kosong.
9. }
10. public void insert(E data) {
11. root = insert(root, data);  // Mulai proses sisipkan data.
12. }
13. private node2 insert(node2 node, E data) {
14. if (node == null)
15. node = new node2(data);  // Jika simpul kosong, buat simpul baru.
16. else {
17. if (node.getLeft() == null)
18. node.left = insert(node.left, data);  // Sisipkan di anak kiri jika kosong.
19. else
20. node.right = insert(node.right, data);  // Jika tidak, sisipkan di anak kanan.
21. }
22. return node;  // Kembalikan simpul dengan perubahan.
23. }
24. public int countNodes() {
25. return countNodes(root);  // Hitung jumlah simpul mulai dari akar.
26. }
27. private int countNodes(node2 r) {
28. if (r == null)
29. return 0;  // Jika simpul kosong, tidak ada yang dihitung.
30. else {
31. int l = 1;  // Mulai hitungan dengan simpul ini.
32. l += countNodes(r.getLeft());  // Tambahkan jumlah dari anak kiri.
33. l += countNodes(r.getRight());  // Tambahkan jumlah dari anak kanan.
34. return l;  // Kembalikan total hitungan.
35. }
36. }
37. public boolean search(E val) {
38. return search(root, val);  // Mulai pencarian dari akar.
39. }
40. private boolean search(node2 r, E val) {
41. if (r.getData() == val)
42. return true;  // Jika data sesuai, kembalikan true.
43. if (r.getLeft() != null)
44. if (search(r.getLeft(), val))
45. return true;  // Lanjutkan pencarian di anak kiri.
46. if (r.getRight() != null)
47. if (search(r.getRight(), val))
48. return true;  // Lanjutkan pencarian di anak kanan.
49. return false;  // Jika tidak ditemukan, kembalikan false.
50. }
51. public void inorder() {
52. inorder(root);  // Lakukan traversal inorder dari akar.
53. }
54. private void inorder(node2 r) {
55. if (r != null) {
56. inorder(r.getLeft());  // Kunjungi anak kiri terlebih dahulu.
57. System.out.print(r.getData() + " ");  // Cetak data simpul.
58. inorder(r.getRight());  // Kunjungi anak kanan setelahnya.
59. }
60. }
61. public void preorder() {
62. preorder(root);  // Lakukan traversal preorder dari akar.
63. }
64. private void preorder(node2 r) {
65. if (r != null) {
66. System.out.print(r.getData() + " ");  // Cetak data simpul.
67. preorder(r.getLeft());  // Kunjungi anak kiri.
68. preorder(r.getRight());  // Kunjungi anak kanan.
69. }
70. }
71. public void postorder() {
72. postorder(root);  // Lakukan traversal postorder dari akar.
73. }
74. private void postorder(node2 r) {
75. if (r != null) {
76. postorder(r.getLeft());  // Kunjungi anak kiri.
77. postorder(r.getRight());  // Kunjungi anak kanan.
78. System.out.print(r.getData() + " ");  // Cetak data simpul.
79. }
80. }
81. }

Kelas BinaryTree2 adalah inti dari struktur data Binary Tree. Dalam kelas ini, pohon biner direpresentasikan dengan menggunakan simpul-simpul yang saling terhubung. Variabel instance root digunakan untuk menunjukkan akar dari pohon biner. Metode isEmpty() memeriksa apakah pohon kosong dengan memeriksa apakah root adalah null. Metode insert(E data) digunakan untuk menyisipkan elemen baru ke dalam pohon biner dengan menggunakan rekursi untuk menemukan posisi yang tepat. Metode countNodes() menghitung jumlah simpul dalam pohon biner dengan cara traversal dari akar ke seluruh simpul. Metode search(E val) mencari nilai tertentu dalam pohon biner dengan menggunakan rekursi untuk mencari di anak kiri dan kanan. Metode inorder(), preorder(), dan postorder() digunakan untuk melakukan traversal inorder, preorder, dan postorder dari pohon biner.

2.Jurnal11(Main)

import java.util.Scanner;

public class Jurnal11 {

    public static void main(String[] args) {

        Scanner scan = new Scanner(System.in);

        BinaryTree2 bt = new BinaryTree2();  // Membuat instance dari BinaryTree2.

        System.out.println("Binary Tree Test\n");

        char ch;

        do {

            System.out.println("\nBinary Tree Operations\n");

            System.out.println("1. insert ");  // Pilihan untuk menyisipkan elemen.

            System.out.println("2. search");   // Pilihan untuk mencari elemen.

            System.out.println("3. count nodes");  // Pilihan untuk menghitung jumlah node.

            System.out.println("4. check empty");  // Pilihan untuk memeriksa apakah pohon kosong.

            int choice = scan.nextInt();  // Membaca pilihan pengguna.

            scan.nextLine();  // Bersihkan buffer input.

            switch (choice) {

                case 1:  // Pilihan untuk menyisipkan elemen.

                    System.out.println("Enter integer element to insert");

                    bt.insert(scan.nextLine());

                    break;

                case 2:  // Pilihan untuk mencari elemen.

                    System.out.println("Enter integer element to search");

                    System.out.println("Search result : " + bt.search(scan.nextLine()));

                    break;

                case 3:  // Pilihan untuk menghitung jumlah node.

                    System.out.println("Nodes = " + bt.countNodes());

                    break;

                case 4:  // Pilihan untuk memeriksa kekosongan pohon.

                    System.out.println("Empty status = " + bt.isEmpty());

                    break;

                default:

                    System.out.println("Wrong Entry \n ");

                    break;

            }

            // Menampilkan isi pohon dalam tiga urutan berbeda.

            System.out.print("\nPost order : ");

            bt.postorder();

            System.out.print("\nPre order : ");

            bt.preorder();

            System.out.print("\nIn order : ");

            bt.inorder();

            System.out.println("\n\nDo you want to continue (Type y or n) \n");

            ch = scan.next().charAt(0);  // Membaca karakter untuk memutuskan kelanjutan.

        } while (ch == 'Y' || ch == 'y');  // Lanjutkan jika pengguna memasukkan 'y' atau 'Y'.

    }

}

Kelas Jurnal11 adalah kelas pengguna yang berisi metode main(). Kelas ini berfungsi sebagai titik masuk utama program. Di dalam metode main(), pengguna diberikan opsi untuk melakukan berbagai operasi pada pohon biner, seperti menyisipkan elemen, mencari elemen, menghitung jumlah simpul, dan memeriksa apakah pohon kosong. Setelah setiap operasi, isi pohon ditampilkan dalam tiga urutan traversal yang berbeda: postorder, preorder, dan inorder. Program akan terus berjalan dan meminta input pengguna hingga pengguna memilih untuk berhenti.

3. Node2

class node2<E> {

    node2 left, right;

    E data;

    public node2() {

        left = null;  // Inisialisasi simpul anak kiri dengan null.

        right = null;  // Inisialisasi simpul anak kanan dengan null.

        data = null;  // Inisialisasi data dengan null.

    }

    public node2(E item) {

        left = null;  // Inisialisasi simpul anak kiri dengan null.

        right = null;  // Inisialisasi simpul anak kanan dengan null.

        data = item;  // Tetapkan data simpul dengan item yang diberikan.

    }

    public void setLeft(node2 n) {

        left = n;  // Tetapkan simpul anak kiri dengan simpul baru n.

    }

    public void setRight(node2 n) {

        right = n;  // Tetapkan simpul anak kanan dengan simpul baru n.

    }

    public node2 getLeft() {

        return left;  // Dapatkan simpul anak kiri dari simpul ini.

    }

    public node2 getRight() {

        return right;  // Dapatkan simpul anak kanan dari simpul ini.

    }

    public void setData(E d) {

        data = d;  // Tetapkan data dari simpul ini dengan d.

    }

    public E getData() {

        return data;  // Dapatkan data yang disimpan dalam simpul ini.

    }

}

Kelas node2 merepresentasikan simpul (node) dalam pohon biner. Setiap simpul memiliki dua referensi ke simpul anak kiri dan kanan (left dan right) serta variabel instance data yang menyimpan nilai data dalam simpul tersebut. Kelas ini menyediakan konstruktor untuk membuat simpul baru dengan nilai data yang diberikan, serta metode-metode setter dan getter untuk mengakses dan memodifikasi simpul-simpulnya.

Dengan struktur ini, program dapat membuat, mengelola, dan melakukan operasi pada pohon biner dengan menggunakan antarmuka yang mudah dipahami oleh pengguna. Semua operasi dan manipulasi pada pohon biner dilakukan melalui kelas-kelas yang terorganisir dengan baik, memisahkan logika pengolahan data dari logika penggunaan dan tampilan.