Programs for Advanced Tree Data Structures

1. Self-Balancing Binary Search Tree (AVL Tree)

```
public class AVLTree {

    class Node {

        int key, height;

        Node left, right;



        Node(int d) {

            key = d;

            height = 1;

        }

    }



    Node root;



    int height(Node N) {

        if (N == null) return 0;

        return N.height;

    }



    int max(int a, int b) {

        return (a > b) ? a : b;

    }



    Node rightRotate(Node y) {

        Node x = y.left;
    Node T2 = x.right;

    x.right = y;
    y.left = T2;

    y.height = max(height(y.left), height(y.right)) + 1;
    x.height = max(height(x.left), height(x.right)) + 1;

    return x;
}

Node leftRotate(Node x) {
    Node y = x.right;
    Node T2 = y.left;

    y.left = x;
    x.right = T2;

    x.height = max(height(x.left), height(x.right)) + 1;
    y.height = max(height(y.left), height(y.right)) + 1;

    return y;
}

int getBalance(Node N) {
    if (N == null) return 0;
    return height(N.left) - height(N.right);
```

```
}

Node insert(Node node, int key) {
    if (node == null) return (new Node(key));

    if (key < node.key) node.left = insert(node.left, key);
    else if (key > node.key) node.right = insert(node.right, key);
    else return node;

    node.height = 1 + max(height(node.left), height(node.right));

    int balance = getBalance(node);

    if (balance > 1 && key < node.left.key) return rightRotate(node);
    if (balance < -1 && key > node.right.key) return leftRotate(node);
    if (balance > 1 && key > node.left.key) {
        node.left = leftRotate(node.left);
        return rightRotate(node);
    }
    if (balance < -1 && key < node.right.key) {
        node.right = rightRotate(node.right);
        return leftRotate(node);
    }

    return node;
}
```

```
    void preOrder(Node node) {

        if (node != null) {

            System.out.print(node.key + " ");

            preOrder(node.left);

            preOrder(node.right);

        }

    }

}
```

2. Segment Tree for Range Queries

```
class SegmentTree {

    int[] st;



    SegmentTree(int[] arr, int n) {

        int x = (int) (Math.ceil(Math.log(n) / Math.log(2)));

        int max_size = 2 * (int) Math.pow(2, x) - 1;

        st = new int[max_size];

        constructSTUtil(arr, 0, n - 1, 0);

    }



    int constructSTUtil(int[] arr, int ss, int se, int si) {

        if (ss == se) {

            st[si] = arr[ss];

            return arr[ss];

        }
        int mid = ss + (se - ss) / 2;
        st[si] = constructSTUtil(arr, ss, mid, si * 2 + 1) +
                 constructSTUtil(arr, mid + 1, se, si * 2 + 2);
        return st[si];
    }

    int getSum(int n, int qs, int qe) {
        return getSumUtil(0, n - 1, qs, qe, 0);
    }

    int getSumUtil(int ss, int se, int qs, int qe, int si) {
        if (qs <= ss && qe >= se) return st[si];
        if (se < qs || ss > qe) return 0;

        int mid = ss + (se - ss) / 2;
        return getSumUtil(ss, mid, qs, qe, 2 * si + 1) +
               getSumUtil(mid + 1, se, qs, qe, 2 * si + 2);
    }
}
```

3. Trie for String Search

```
class Trie {
    class TrieNode {
        TrieNode[] children = new TrieNode[26];
        boolean isEndOfWord;
    }
    TrieNode root;


    Trie() {
        root = new TrieNode();
    }


    void insert(String key) {
        TrieNode node = root;
        for (char c : key.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) node.children[index] = new TrieNode();
            node = node.children[index];
        }
        node.isEndOfWord = true;
    }


    boolean search(String key) {
        TrieNode node = root;
        for (char c : key.toCharArray()) {
            int index = c - 'a';
            if (node.children[index] == null) return false;
            node = node.children[index];
        }
        return node.isEndOfWord;
    }
}
```