Practice problem : DAY [12/11/2024]

Name:Kowsick K

Dept: CSE

**1.AnagramProgram:**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

class Main {

public static boolean isAnagram(String str1, String str2) {

Map<Character, Integer> count1 = new HashMap<>();

Map<Character, Integer> count2 = new HashMap<>();

for (char ch : str1.toCharArray()) {

count1.put(ch, count1.getOrDefault(ch, 0) + 1);

}

for (char ch : str2.toCharArray()) {

count2.put(ch, count2.getOrDefault(ch, 0) + 1);

}

return count1.equals(count2);

}

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.println("Enter first string: ");

String str1 = input.next();

System.out.println("Enter second string: ");

String str2 = input.next();

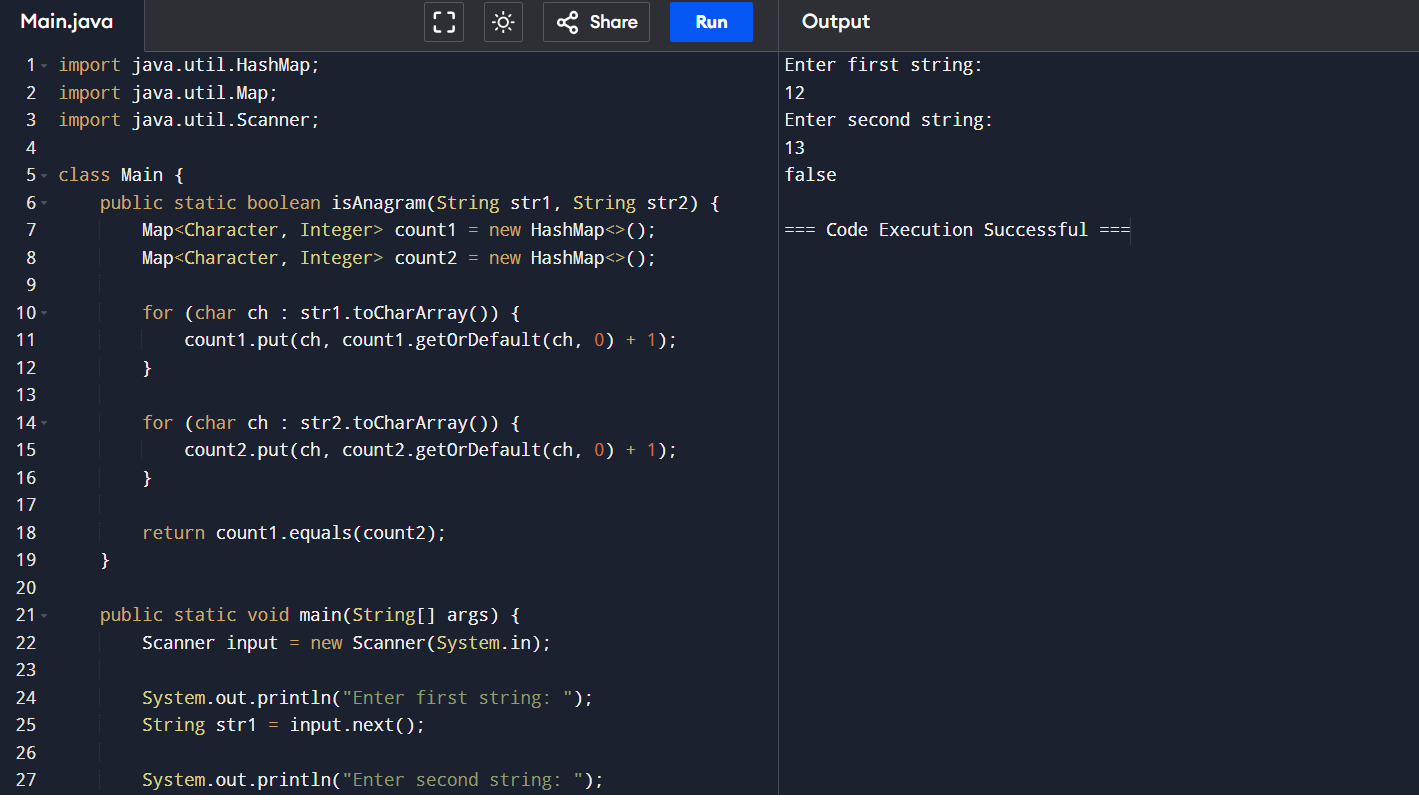
System.out.println(isAnagram(str1, str2));

input.close();

}

}

Output :



Time Complexity: O(n + m)

Space Complexity; O(k)

**2. Binary Search Tree:**

class TreeNode {

int value;

TreeNode left, right;

public TreeNode(int data) {

value = data;

left = right = null;

}

}

class SearchTree {

TreeNode root;

public SearchTree() {

root = null;

}

void add(int value) {

root = addRec(root, value);

}

TreeNode addRec(TreeNode root, int value) {

if (root == null) {

root = new TreeNode(value);

return root;

}

if (value < root.value)

root.left = addRec(root.left, value);

else if (value > root.value)

root.right = addRec(root.right, value);

return root;

}

void remove(int value) {

root = removeRec(root, value);

}

TreeNode removeRec(TreeNode root, int value) {

if (root == null)

return root;

if (value < root.value)

root.left = removeRec(root.left, value);

else if (value > root.value)

root.right = removeRec(root.right, value);

else {

if (root.left == null)

return root.right;

else if (root.right == null)

return root.left;

root.value = minValue(root.right);

root.right = removeRec(root.right, root.value);

}

return root;

}

int minValue(TreeNode root) {

int min = root.value;

while (root.left != null) {

min = root.left.value;

root = root.left;

}

return min;

}

boolean contains(int value) {

return containsRec(root, value);

}

boolean containsRec(TreeNode root, int value) {

if (root == null)

return false;

if (root.value == value)

return true;

if (root.value < value)

return containsRec(root.right, value);

return containsRec(root.left, value);

}

void inOrder() {

inOrderRec(root);

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

}

void inOrderRec(TreeNode root) {

if (root != null) {

inOrderRec(root.left);

System.out.print(root.value + " ");

inOrderRec(root.right);

}

}

void preOrder() {

preOrderRec(root);

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

}

void preOrderRec(TreeNode root) {

if (root != null) {

System.out.print(root.value + " ");

preOrderRec(root.left);

preOrderRec(root.right);

}

}

void postOrder() {

postOrderRec(root);

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

}

void postOrderRec(TreeNode root) {

if (root != null) {

postOrderRec(root.left);

postOrderRec(root.right);

System.out.print(root.value + " ");

}

}

}

public class Main {

public static void main(String[] args) {

SearchTree tree = new SearchTree();

tree.add(50);

tree.add(30);

tree.add(20);

tree.add(40);

tree.add(70);

tree.add(60);

tree.add(80);

System.out.println("Inorder traversal:");

tree.inOrder();

tree.remove(20);

tree.remove(30);

System.out.println("Inorder traversal after deletion:");

tree.inOrder();

int target = 70;

System.out.println("Is " + target + " present in the tree? " + tree.contains(target));

System.out.println("Preorder traversal:");

tree.preOrder();

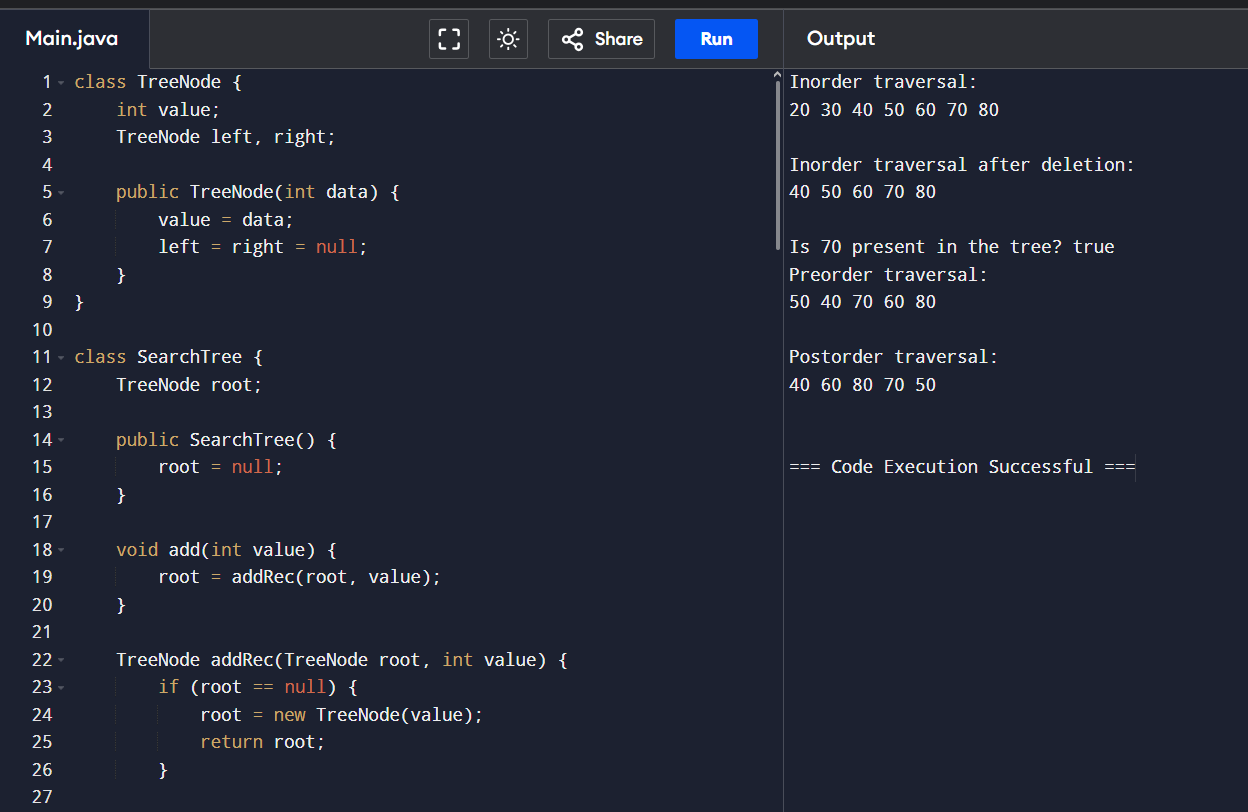
System.out.println("Postorder traversal:");

tree.postOrder();

}

}

Output:



Time Complexity

Average case:

O(logn)

Worst Case:

O(n)

Traversal (inorder, preorder,Postorder)

O(n)

**4. ParenthesesChecker**

import java.util.Stack;

class ParenthesesChecker {

    public static boolean isValid(String s) {

        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {

            if (c == '(' || c == '{' || c == '[') {

                stack.push(c);

            } else if (c == ')' || c == '}' || c == ']') {

                if (stack.isEmpty()) {

                    return false;

                }

                char top = stack.pop();

                if ((c == ')' && top != '(') ||

                    (c == '}' && top != '{') ||

                    (c == ']' && top != '[')) {

                    return false;

                }

            }

        }

        return stack.isEmpty();

    }

    public static void main(String[] args) {

        String[] testCases = {

            "[(())]",

            "{[()]}",

            "[(])",

            "{[()]",

            "[{}((()))]",

            "[{}((())}",

            "[{}((()))",

        };

        for (String s : testCases) {

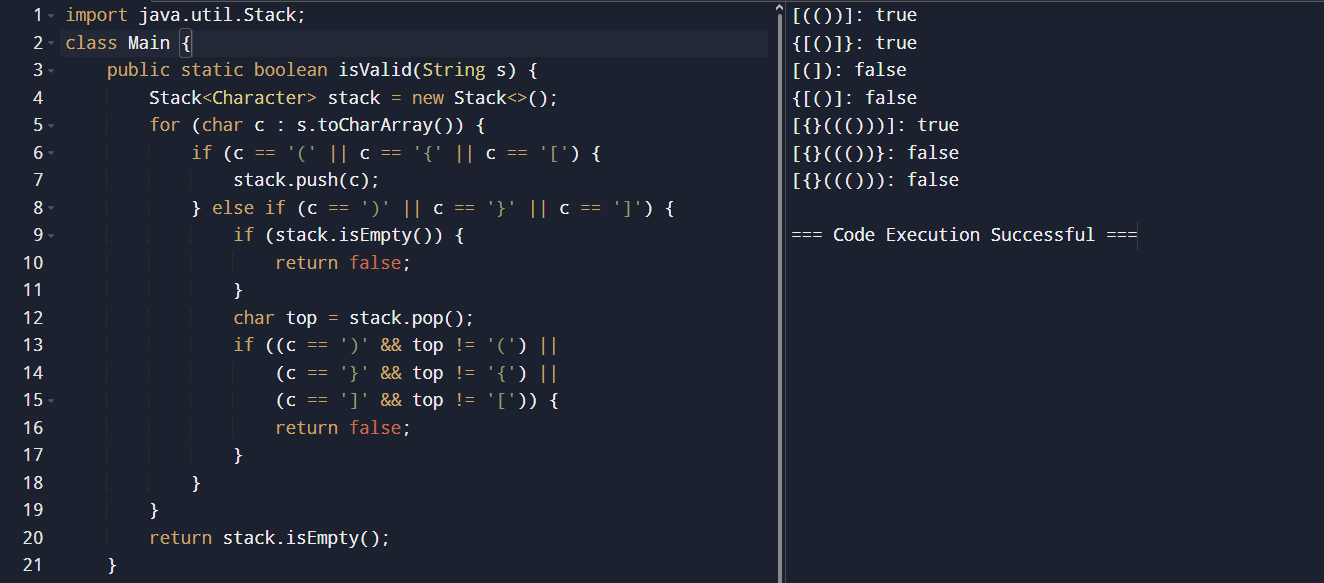
            System.out.println(s + ": " + isValid(s));

        }

    }

}

**Output:**



Time Complexity:O(n)