Coding Practice (9/11/2024):

1. Maximum Subarray Sum – Kadane’s Algorithm

Solution: Time Complexity: O(n)

Space Complexity: O(n)  
package problems;

import java.util.\*;

public class MaxSubarraySum {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements in the Array");

int size = scanner.nextInt();

int[] numbers = new int[size];

System.out.println("Enter the Elements");

for (int i = 0; i < size; i++) {

numbers[i] = scanner.nextInt();

}

int currentSum = 0, maxSum = Integer.MIN\_VALUE;

for (int i = 0; i < size; i++) {

currentSum += numbers[i];

maxSum = Math.max(maxSum, currentSum);

if (currentSum <= 0) {

currentSum = 0;

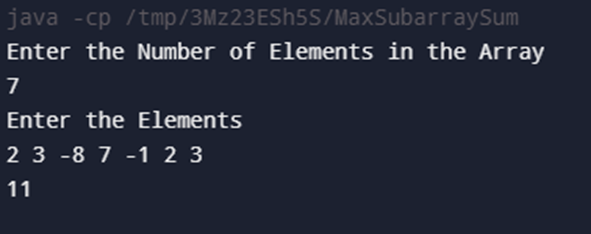
}

}

System.out.println(maxSum);

}

}



1. Maximum Product Subarray

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.\*;

public class MaximumProduct {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements");

int elementCount = inputScanner.nextInt();

System.out.println("Enter the Elements");

int[] numbers = new int[elementCount];

for (int i = 0; i < elementCount; i++) {

numbers[i] = inputScanner.nextInt();

}

int prefixProduct = 1, suffixProduct = 1, maxProduct = Integer.MIN\_VALUE;

for (int i = 0; i < elementCount; i++) {

if (prefixProduct == 0) prefixProduct = 1;

if (suffixProduct == 0) suffixProduct = 1;

prefixProduct \*= numbers[i];

suffixProduct \*= numbers[elementCount - i - 1];

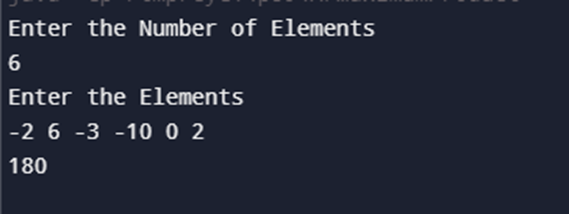
maxProduct = Math.max(maxProduct, Math.max(prefixProduct, suffixProduct));

}

System.out.println(maxProduct);

}

}



1. Search in a sorted and rotated Array

Solution: Time Complexity: O(log n)

Space Complexity: O(1)

package problems;

import java.util.\*;

public class RotatedArraySearch {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements");

int elementCount = inputScanner.nextInt();

System.out.println("Enter the Elements");

int[] numbers = new int[elementCount];

for (int i = 0; i < elementCount; i++) {

numbers[i] = inputScanner.nextInt();

}

System.out.println("Enter Target Value");

int target = inputScanner.nextInt();

int left = 0, right = numbers.length - 1;

boolean isFound = true;

while (left <= right) {

int mid = (left + right) / 2;

if (numbers[mid] == target) {

System.out.println(mid);

isFound = false;

break;

}

if (numbers[left] <= numbers[mid]) {

if (numbers[left] <= target && target <= numbers[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (numbers[mid] <= target && target <= numbers[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

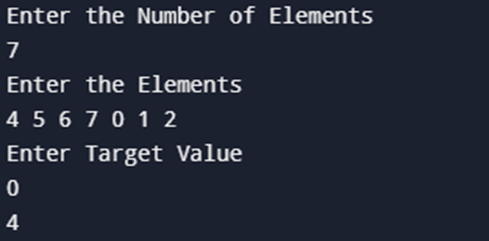
if (isFound) {

System.out.println(-1);

}

}

}



1. Container with Most Water

Solution: Time Complexity: O(n)

Space Complexity: O(1)

package problems;

import java.util.Scanner;

public class MaxAreaContainer {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements");

int elementCount = inputScanner.nextInt();

System.out.println("Enter the Elements");

int[] numbers = new int[elementCount];

for (int i = 0; i < elementCount; i++) {

numbers[i] = inputScanner.nextInt();

}

int result = 0;

int left = 0, right = numbers.length - 1;

while (left < right) {

result = Math.max(result, ((right - left) \* Math.min(numbers[left], numbers[right])));

if (numbers[left] < numbers[right]) {

left++;

} else {

right--;

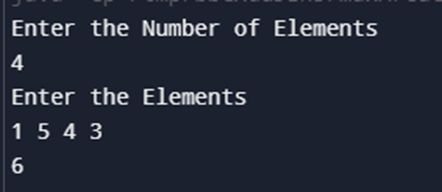
}

}

System.out.println(result);

}

}



1. Find the Factorial of a large number

Solution: Time Complexity: O(n)

Space Complexity: O(1)

package problems;

import java.math.BigInteger;

import java.util.Scanner;

public class FactorialCalculator {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number");

int number = inputScanner.nextInt();

BigInteger result = BigInteger.ONE;

for (int i = 1; i <= number; i++) {

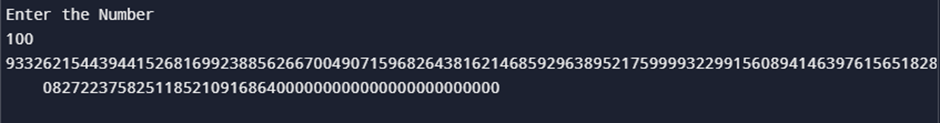
result = result.multiply(BigInteger.valueOf(i));

}

System.out.println(result);

}

}



1. Trapping Rain Water

Solution: Time Complexity: O(n)

Space Complexity: O(1)

package problems;

import java.util.Scanner;

public class RainWaterTrapping {

public static int calculateTrappedWater(int[] heights) {

int left = 0;

int right = heights.length - 1;

int leftMax = heights[left];

int rightMax = heights[right];

int water = 0;

while (left < right) {

if (leftMax < rightMax) {

left++;

leftMax = Math.max(leftMax, heights[left]);

water += leftMax - heights[left];

} else {

right--;

rightMax = Math.max(rightMax, heights[right]);

water += rightMax - heights[right];

}

}

return water;

}

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements");

int numberOfElements = inputScanner.nextInt();

System.out.println("Enter the Heights");

int[] heights = new int[numberOfElements];

for (int i = 0; i < numberOfElements; i++) {

heights[i] = inputScanner.nextInt();

}

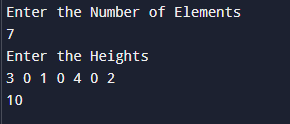
int trappedWater = calculateTrappedWater(heights);

System.out.println(trappedWater);

inputScanner.close();

}

}



1. Chocolate Distribution Problem

Solution: Time Complexity: O(n log n)

Space Complexity: O(1)

package problems;

import java.util.\*;

public class MinDifference {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Elements");

int numberOfElements = inputScanner.nextInt();

System.out.println("Enter the Values");

int[] values = new int[numberOfElements];

for (int i = 0; i < numberOfElements; i++) {

values[i] = inputScanner.nextInt();

}

System.out.println("Enter M Value");

int mValue = inputScanner.nextInt();

Arrays.sort(values);

int minDifference = Integer.MAX\_VALUE;

for (int i = 0; i < numberOfElements - mValue + 1; i++) {

int difference = values[i + mValue - 1] - values[i];

minDifference = Math.min(minDifference, difference);

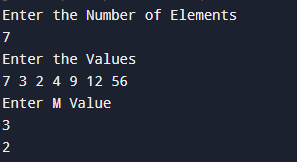
}

System.out.println(minDifference);

inputScanner.close();

}

}



1. Merge Intervals

Solution: Time Complexity: O(n log n)

Space Complexity: O(n)

package problems;

import java.util.\*;

public class IntervalMerger {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Size of the Interval Array");

int numberOfIntervals = inputScanner.nextInt();

int[][] intervals = new int[numberOfIntervals][2];

System.out.println("Enter the Intervals");

for (int i = 0; i < numberOfIntervals; i++) {

intervals[i][0] = inputScanner.nextInt();

intervals[i][1] = inputScanner.nextInt();

}

int[][] mergedIntervals = mergeIntervals(intervals);

System.out.println("Merged Intervals:");

for (int[] interval : mergedIntervals) {

System.out.println(Arrays.toString(interval));

}

}

public static int[][] mergeIntervals(int[][] intervals) {

if (intervals.length <= 1) {

return intervals;

}

Arrays.sort(intervals, (a, b) -> Integer.compare(a[0], b[0]));

ArrayList<int[]> resultIntervals = new ArrayList<>();

int[] currentInterval = intervals[0];

resultIntervals.add(currentInterval);

for (int[] interval : intervals) {

int currentEnd = currentInterval[1];

int nextStart = interval[0];

int nextEnd = interval[1];

if (currentEnd >= nextStart) {

currentInterval[1] = Math.max(currentEnd, nextEnd);

} else {

currentInterval = interval;

resultIntervals.add(currentInterval);

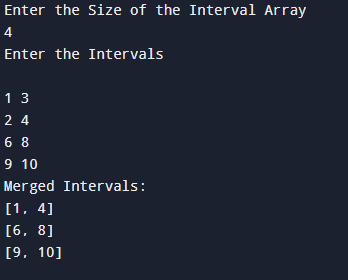
}

}

return resultIntervals.toArray(new int[resultIntervals.size()][]);

}

}



1. A Boolean Matrix Question

Solution: Time Complexity: O(n \* m)

Space Complexity: O(n + m)

package problems;

import java.util.\*;

public class MatrixModifier {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the Number of Rows:");

int numberOfRows = inputScanner.nextInt();

System.out.println("Enter the Number of Columns:");

int numberOfColumns = inputScanner.nextInt();

int[][] matrix = new int[numberOfRows][numberOfColumns];

System.out.println("Enter the Matrix Values (0 or 1 only):");

for (int i = 0; i < numberOfRows; i++) {

for (int j = 0; j < numberOfColumns; j++) {

matrix[i][j] = inputScanner.nextInt();

}

}

modifyMatrix(matrix);

System.out.println("Modified Matrix:");

for (int i = 0; i < numberOfRows; i++) {

for (int j = 0; j < numberOfColumns; j++) {

System.out.print(matrix[i][j] + " ");

}

System.out.println();

}

inputScanner.close();

}

public static void modifyMatrix(int[][] matrix) {

int rows = matrix.length;

int cols = matrix[0].length;

boolean[] rowMark = new boolean[rows];

boolean[] colMark = new boolean[cols];

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

if (matrix[i][j] == 1) {

rowMark[i] = true;

colMark[j] = true;

}

}

}

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

if (rowMark[i] || colMark[j]) {

matrix[i][j] = 1;

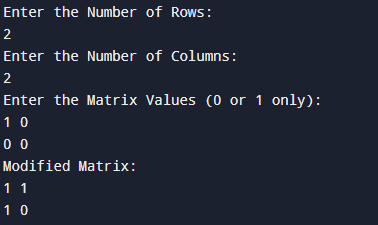
}

}

}

}

}



1. Print a given matrix in spiral form

Solution: Time Complexity: O(n \* m)

Space Complexity: O(n \* m)

package problems;

import java.util.\*;

public class Program10 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of rows:");

int rows = scanner.nextInt();

System.out.println("Enter the number of columns:");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

System.out.println("Enter the matrix values:");

for (int i = 0; i < rows; i++) {

for (int j = 0; j < cols; j++) {

matrix[i][j] = scanner.nextInt();

}

}

List<Integer> result = printSpiral(matrix);

for (int num : result) {

System.out.print(num + " ");

}

System.out.println();

scanner.close();

}

public static List<Integer> printSpiral(int[][] matrix) {

List<Integer> result = new ArrayList<>();

int rows = matrix.length;

int cols = matrix[0].length;

int top = 0, left = 0, bottom = rows - 1, right = cols - 1;

while (top <= bottom && left <= right) {

for (int i = left; i <= right; i++)

result.add(matrix[top][i]);

top++;

for (int i = top; i <= bottom; i++)

result.add(matrix[i][right]);

right--;

if (top <= bottom) {

for (int i = right; i >= left; i--)

result.add(matrix[bottom][i]);

bottom--;

}

if (left <= right) {

for (int i = bottom; i >= top; i--)

result.add(matrix[i][left]);

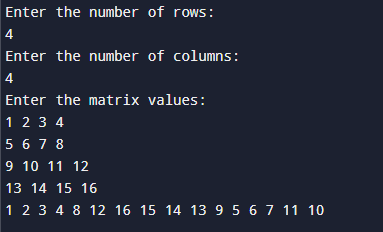
left++;

}

}

return result;

}

}  


1. Check if given Parentheses expression is balanced or not

Solution: Time Complexity: O(n)

Space Complexity: O(1)

package problems;

import java.util.Scanner;

public class Program13 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the expression:");

String expression = scanner.nextLine();

if (checkBalance(expression))

System.out.println("Balanced");

else

System.out.println("Not Balanced");

scanner.close();

}

public static boolean checkBalance(String expression) {

boolean result = true;

int balance = 0;

for (int i = 0; i < expression.length(); i++) {

if (expression.charAt(i) == '(') {

balance++;

} else {

balance--;

}

if (balance < 0) {

result = false;

break;

}

}

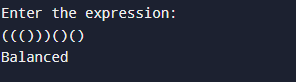
if (balance != 0) {

result = false;

}

return result;

}

}  
  


1. Check if two Strings are Anagrams of each other

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.HashMap;

import java.util.Scanner;

public class Program14 {

static boolean checkAnagrams(String str1, String str2) {

HashMap<Character, Integer> countMap = new HashMap<>();

for (char c : str1.toCharArray())

countMap.put(c, countMap.getOrDefault(c, 0) + 1);

for (char c : str2.toCharArray())

countMap.put(c, countMap.getOrDefault(c, 0) - 1);

for (var entry : countMap.entrySet()) {

if (entry.getValue() != 0) {

return false;

}

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str1 = scanner.nextLine();

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

String str2 = scanner.nextLine();

if(checkAnagrams(str1, str2)) {

System.out.println("True");

}

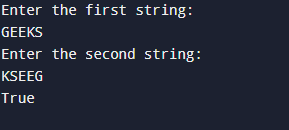
else {

System.out.println("False");

}

scanner.close();

}

}

1. Longest Palindromic Substring

Solution: Time Complexity: O(n²)

Space Complexity: O(n²)

package problems;

import java.util.Scanner;

public class Solution {

static String longestPalindromeSubstring(String str) {

int length = str.length();

boolean[][] dpTable = new boolean[length][length];

int maxLength = 1;

int startIndex = 0;

for (int i = 0; i < length; ++i)

dpTable[i][i] = true;

for (int i = 0; i < length - 1; ++i) {

if (str.charAt(i) == str.charAt(i + 1)) {

dpTable[i][i + 1] = true;

startIndex = i;

maxLength = 2;

}

}

for (int k = 3; k <= length; ++k) {

for (int i = 0; i < length - k + 1; ++i) {

int j = i + k - 1;

if (dpTable[i + 1][j - 1] && str.charAt(i) == str.charAt(j)) {

dpTable[i][j] = true;

if (k > maxLength) {

startIndex = i;

maxLength = k;

}

}

}

}

return str.substring(startIndex, startIndex + maxLength);

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

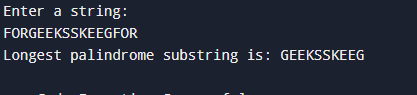
String input = scanner.nextLine();

System.out.println("Longest palindrome substring is: " + longestPalindromeSubstring(input));

scanner.close();

}

}



1. Longest Common Prefix using Sorting

Solution: Time Complexity: O(n log n + m)

Space Complexity: O(1)

package problems;

import java.util.Arrays;

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the number of strings:");

int numStrings = inputScanner.nextInt();

String[] stringArray = new String[numStrings];

inputScanner.nextLine();

System.out.println("Enter the strings:");

for (int i = 0; i < numStrings; i++) {

stringArray[i] = inputScanner.nextLine();

}

System.out.println(findLongestCommonPrefix(stringArray));

inputScanner.close();

}

static String findLongestCommonPrefix(String[] stringArray) {

if (stringArray == null || stringArray.length == 0)

return "-1";

Arrays.sort(stringArray);

String firstString = stringArray[0];

String lastString = stringArray[stringArray.length - 1];

int minLength = Math.min(firstString.length(), lastString.length());

int i = 0;

while (i < minLength && firstString.charAt(i) == lastString.charAt(i)) {

i++;

}

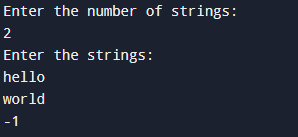
if (i == 0)

return "-1";

return firstString.substring(0, i);

}

}



1. Delete middle element of a stack

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.\*;

public class Solution {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

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

System.out.println("Enter the stack elements:");

String[] inputElements = inputScanner.nextLine().split(" ");

for (String element : inputElements) {

if (!element.isEmpty()) {

charStack.push(element.charAt(0));

}

}

Vector<Character> charVector = new Vector<>();

while (!charStack.empty()) {

charVector.add(charStack.pop());

}

int size = charVector.size();

int middleIndex = size / 2;

for (int i = 0; i < size; i++) {

if (size % 2 == 0 && i == middleIndex || size % 2 != 0 && i == middleIndex) continue;

charStack.push(charVector.get(i));

}

while (!charStack.empty()) {

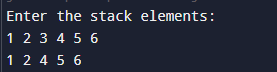
System.out.print(charStack.pop() + " ");

}

inputScanner.close();

}

}



1. Next Greater Element (NGE) for every element in given Array

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

System.out.println("Enter the size of the array:");

int arraySize = inputScanner.nextInt();

int[] inputArray = new int[arraySize];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < arraySize; i++) {

inputArray[i] = inputScanner.nextInt();

}

printNextGreaterElement(inputArray, arraySize);

inputScanner.close();

}

static class CustomStack {

int top;

int[] stackItems = new int[100];

void push(int x) {

if (top == 99) {

System.out.println("Stack full");

} else {

stackItems[++top] = x;

}

}

int pop() {

if (top == -1) {

System.out.println("Underflow error");

return -1;

} else {

return stackItems[top--];

}

}

boolean isEmpty() {

return top == -1;

}

}

static void printNextGreaterElement(int[] arr, int n) {

CustomStack stack = new CustomStack();

stack.top = -1;

stack.push(arr[0]);

for (int i = 1; i < n; i++) {

int currentElement = arr[i];

if (!stack.isEmpty()) {

int topElement = stack.pop();

while (topElement < currentElement) {

System.out.println(currentElement);

if (stack.isEmpty()) break;

topElement = stack.pop();

}

if (topElement > currentElement) stack.push(topElement);

}

stack.push(currentElement);

}

while (!stack.isEmpty()) {

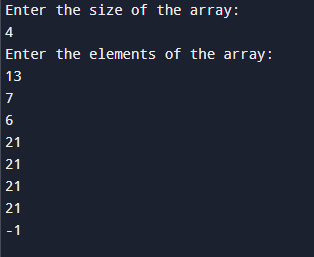
int remainingElement = stack.pop();

System.out.println(-1);

}

}

}



1. Print Right View of a Binary Tree

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.ArrayList;

import java.util.Scanner;

class TreeNode {

int value;

TreeNode leftChild, rightChild;

TreeNode(int x) {

value = x;

leftChild = rightChild = null;

}

}

public class Solution {

public static void main(String[] args) {

TreeNode rootNode = constructTree();

ArrayList<Integer> rightViewResult = getRightView(rootNode);

System.out.print("Right view of the binary tree: ");

displayArray(rightViewResult);

}

static void exploreRightView(TreeNode node, int currentLevel, int[] maxLevel, ArrayList<Integer> result) {

if (node == null) return;

if (currentLevel > maxLevel[0]) {

result.add(node.value);

maxLevel[0] = currentLevel;

}

exploreRightView(node.rightChild, currentLevel + 1, maxLevel, result);

exploreRightView(node.leftChild, currentLevel + 1, maxLevel, result);

}

static ArrayList<Integer> getRightView(TreeNode rootNode) {

ArrayList<Integer> result = new ArrayList<>();

int[] maxLevel = new int[]{-1};

exploreRightView(rootNode, 0, maxLevel, result);

return result;

}

static void displayArray(ArrayList<Integer> arr) {

for (int value : arr) {

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

}

System.out.println();

}

static TreeNode constructTree() {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter the number of nodes:");

int nodeCount = scanner.nextInt();

if (nodeCount <= 0) {

return null;

}

TreeNode[] nodes = new TreeNode[nodeCount];

for (int i = 0; i < nodeCount; i++) {

nodes[i] = new TreeNode(i + 1);

}

System.out.println("Enter node relations as pairs (parent, child, L/R for left/right):");

for (int i = 0; i < nodeCount - 1; i++) {

int parentIndex = scanner.nextInt() - 1;

int childIndex = scanner.nextInt() - 1;

char direction = scanner.next().charAt(0);

if (direction == 'L' || direction == 'l') {

nodes[parentIndex].leftChild = nodes[childIndex];

} else if (direction == 'R' || direction == 'r') {

nodes[parentIndex].rightChild = nodes[childIndex];

}

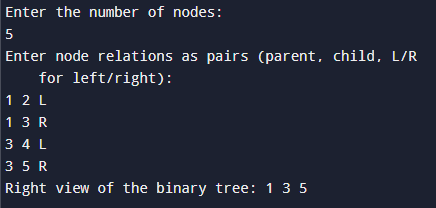
}

scanner.close();

return nodes[0];

}

}



1. Maximum Depth or Height of Binary Tree

Solution: Time Complexity: O(n)

Space Complexity: O(n)

package problems;

import java.util.Scanner;

import java.util.HashMap;

import java.util.Map;

class TreeNode {

int value;

TreeNode leftChild, rightChild;

TreeNode(int val) {

value = val;

leftChild = null;

rightChild = null;

}

}

public class Solution {

static int getMaxDepth(TreeNode node) {

if (node == null)

return 0;

int leftDepth = getMaxDepth(node.leftChild);

int rightDepth = getMaxDepth(node.rightChild);

return Math.max(leftDepth, rightDepth) + 1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Map<Integer, TreeNode> nodes = new HashMap<>();

System.out.println("Enter the number of nodes:");

int n = scanner.nextInt();

System.out.println("Enter each node's parent, child, and direction (L/R) separated by spaces:");

TreeNode rootNode = null;

for (int i = 0; i < n; i++) {

int parentVal = scanner.nextInt();

int childVal = scanner.nextInt();

char direction = scanner.next().charAt(0);

TreeNode parentNode = nodes.computeIfAbsent(parentVal, TreeNode::new);

if (rootNode == null) rootNode = parentNode; // First node becomes root

TreeNode childNode = nodes.computeIfAbsent(childVal, TreeNode::new);

if (direction == 'L') {

parentNode.leftChild = childNode;

} else if (direction == 'R') {

parentNode.rightChild = childNode;

}

}

System.out.println("Maximum Depth of the Binary Tree: " + getMaxDepth(rootNode));

scanner.close();

}

}

