09/11/24

1)Maximum Subarray Sum – Kadane‟s Algorithm:

import java.util.\*;

public class MaxSubArrSum {

    public static int solver(List<Integer> nums) {

        int cursum = nums.get(0);

        int maxsum = nums.get(0);

        for (int i = 1; i < nums.size(); i++) {

            cursum = Math.max(nums.get(i), cursum + nums.get(i));

            maxsum = Math.max(cursum, maxsum);

        }

        return maxsum;

    }

    public static void main(String[] args) {

        List<Integer> nums = Arrays.asList(2, 3, -8, 7, -1, 2, 3);

        System.out.println(solver(nums));

    }

}

Time complexity: O(N)

Space complexity: O(1)

2) Maximum product sub array

import java.util.\*;

public class MaxSubArrSum {

    public static int solver(List<Integer> nums) {

        int cursum = nums.get(0);

        int maxsum = nums.get(0);

        for (int i = 1; i < nums.size(); i++) {

            cursum = Math.max(nums.get(i), cursum + nums.get(i));

            maxsum = Math.max(cursum, maxsum);

        }

        return maxsum;

    }

    public static void main(String[] args) {

        List<Integer> nums = Arrays.asList(2, 3, -8, 7, -1, 2, 3);

        System.out.println(solver(nums));

    }

}

Time Complexity: O(N)

Space Complexity: O(1)

3)Search In Rotated sorted Array

import java.util.\*;

class SearchInRotArr {

    public static int binary(int left, int right, int[] nums, int tar) {

        while (left <= right) {

            int mid = (left + right) / 2;

            if (nums[mid] == tar) {

                return mid;

            } else if (nums[mid] < tar) {

                left = mid + 1;

            } else {

                right = mid - 1;

            }

        }

        return -1;

    }

    public static int search(int[] nums, int target) {

        int low = 0;

        int high = nums.length - 1;

        if (high == 0) {

            return nums[0] == target ? 0 : -1;

        }

        if (nums[low] < nums[high]) {

            return binary(low, high, nums, target);

        }

        for (; low <= high; low++) {

            if (nums[low] == target) {

                return low;

            }

            if (nums[low] > nums[low + 1]) {

                break;

            }

        }

        low++;

        return binary(low, high, nums, target);

    }

    public static void main(String[] args) {

        int[] nums = {4, 5, 6, 7, 0, 1, 2};

        int target = 0;

        System.out.println("Index of target: " + search(nums, target));

    }

}

Time Complexity: O(logn)

Space complexity: O(1)

4) Container with most water

class ContMostWat {

    public static int maxArea(int[] nums) {

        int i = 0;

        int j = nums.length - 1;

        int ans = 0;

        while (i < j) {

            int area = Math.min(nums[i], nums[j]) \* (j - i);

            ans = Math.max(ans, area);

            if (nums[i] > nums[j]) {

                j--;

            } else {

                i++;

            }

        }

        return ans;

    }

    public static void main(String[] args) {

        int[] nums = {1, 8, 6, 2, 5, 4, 8, 3, 7};

        System.out.println("Max Area: " + maxArea(nums));

    }

}

Time Complexity: O(N)

Space Complexity: O(1)

5)Factorial of long integers

import java.math.BigInteger;

class Factorial {

public static BigInteger factorial(int n) {

BigInteger result = BigInteger.ONE;

for (int i = 2; i <= n; i++) {

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

}

return result;

}

public static void main(String[] args) {

int n = 100;

System.out.println("Factorial of " + n + " is:");

System.out.println(factorial(n));

}

}Time complexity: O(N)

Space complexity: O(1)

6)Trapping Rain water

class TrapRWater {

    public static int trap(int[] height) {

        int left = 0;

        int right = height.length - 1;

        int leftMax = height[left];

        int rightMax = height[right];

        int water = 0;

        while (left < right) {

            if (leftMax < rightMax) {

                left++;

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

                water += leftMax - height[left];

            } else {

                right--;

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

                water += rightMax - height[right];

            }

        }

        return water;

    }

    public static void main(String[] args) {

        int[] height = {0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1};

        System.out.println("Trapped water: " + trap(height));

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

7) Chocolate distribution

import java.util.Arrays;

class ChocolateDistribution {

public static int findMinDiff(int[] arr, int n, int m) {

if (m == 0 || n == 0) {

return 0;

}

Arrays.sort(arr);

if (n < m) {

return -1;

}

int minDiff = Integer.MAX\_VALUE;

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

int diff = arr[i + m - 1] - arr[i];

minDiff = Math.min(minDiff, diff);

}

return minDiff;

}

public static void main(String[] args) {

int[] arr = {7, 3, 2, 4, 9, 12, 56};

int m = 3;

int n = arr.length;

int result = findMinDiff(arr, n, m);

System.out.println("Minimum difference is " + result);

}

}Time Complexity: O(N)

Space Complexity: O(1)

8) OverLapping Intervals

import java.util.\*;

class Solution {

public List<List<Integer>> merge(List<List<Integer>> nums) {

// Sort the intervals based on the first element of each pair

nums.sort((a, b) -> Integer.compare(a.get(0), b.get(0)));

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

int low = nums.get(0).get(0);

int high = nums.get(0).get(1);

for (int i = 1; i < nums.size(); i++) {

if (nums.get(i).get(0) <= high) {

low = Math.min(low, nums.get(i).get(0));

high = Math.max(high, nums.get(i).get(1));

} else {

ans.add(Arrays.asList(low, high));

low = nums.get(i).get(0);

high = nums.get(i).get(1);

}

}

ans.add(Arrays.asList(low, high)); // Add the last interval

return ans;

}

public static void main(String[] args) {

Solution solution = new Solution();

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

nums.add(Arrays.asList(1, 3));

nums.add(Arrays.asList(2, 6));

nums.add(Arrays.asList(8, 10));

nums.add(Arrays.asList(15, 18));

List<List<Integer>> merged = solution.merge(nums);

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

}

}

Time Complexity: O(N)

Space Complexity: O(1)

9)A Boolean matrix

import java.util.Arrays;

class BooleanMatrix {

public static void modifyMatrix(int[][] mat, int M, int N) {

int[] row = new int[M];

int[] col = new int[N];

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

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

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

row[i] = 1;

col[j] = 1;

}

}

}

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

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

if (row[i] == 1 || col[j] == 1) {

mat[i][j] = 1;

}

}

}

}

public static void main(String[] args) {

int[][] mat1 = {{1, 0}, {0, 0}};

int[][] mat2 = {{0, 0, 0}, {0, 0, 1}};

modifyMatrix(mat1, mat1.length, mat1[0].length);

modifyMatrix(mat2, mat2.length, mat2[0].length);

System.out.println("Modified mat1:");

for (int[] row : mat1) {

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

}

System.out.println("Modified mat2:");

for (int[] row : mat2) {

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

}

}

}Time complexity: O(N\*N)

Space Complexity O(N)

10)Spiral matrix

class SpiralMatrix {

public static void printSpiral(int[][] matrix, int m, int n) {

int top = 0, bottom = m - 1, left = 0, right = n - 1;

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

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

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

}

top++;

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

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

}

right--;

if (top <= bottom) {

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

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

}

bottom--;

}

if (left <= right) {

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

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

}

left++;

}

}

}

public static void main(String[] args) {

int[][] matrix = {

{1, 2, 3, 4},

{5, 6, 7, 8},

{9, 10, 11, 12},

{13, 14, 15, 16}

};

printSpiral(matrix, matrix.length, matrix[0].length);

}

}Time Complexity: O(N)

Space Complexity: O(1)

13)Valid paranthesis

import java.util.Stack;

class ValidPara {

    public static boolean isValid(String s) {

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

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

            char c = s.charAt(i);

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

                st.push(c);

            } else {

                if (st.isEmpty()) {

                    return false;

                }

                if (c == ')' && st.peek() != '(') {

                    return false;

                } else if (c == '}' && st.peek() != '{') {

                    return false;

                } else if (c == ']' && st.peek() != '[') {

                    return false;

                }

                st.pop();

            }

        }

        return st.isEmpty();

    }

    public static void main(String[] args) {

        String s = "{[]}";

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

    }

}

Time Complexity: O(N)

Space complexity: O(N)

14) Valid Anagram

import java.util.HashMap;

class ValidAnagram {

    public static boolean isAnagram(String s, String t) {

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

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

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

        }

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

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

        }

        for (int freq : count.values()) {

            if (freq != 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        String s = "anagram";

        String t = "nagaram";

        System.out.println("Is anagram: " + isAnagram(s, t));

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

15) Longest palindromic Subs

class LongPalSubs {

    public static String longestPalindrome(String s) {

        if (s.length() == 1) {

            return s;

        }

        String ans = "";

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

            for (int j = s.length() - 1; j >= i; j--) {

                if (s.charAt(i) == s.charAt(j)) {

                    int a = i;

                    int b = j;

                    boolean isPalindrome = true;

                    while (a <= b) {

                        if (s.charAt(a) != s.charAt(b)) {

                            isPalindrome = false;

                            break;

                        }

                        a++;

                        b--;

                    }

                    if (isPalindrome) {

                        if (ans.length() < j - i + 1) {

                            ans = s.substring(i, j + 1);

                        }

                    }

                }

            }

        }

        return ans;

    }

    public static void main(String[] args) {

        String s = "babad";

        System.out.println("Longest Palindrome: " + longestPalindrome(s));

    }

}

Time Complexity: O(N)

Space Complexity: O(1)

16) Long Palindrome subs using sorting

import java.util.Arrays;

class LongestCommonPrefix {

public static String longestCommonPrefix(String[] arr) {

if (arr == null || arr.length == 0) {

return "-1";

}

Arrays.sort(arr);

String first = arr[0];

String last = arr[arr.length - 1];

int i = 0;

while (i < first.length() && i < last.length() && first.charAt(i) == last.charAt(i)) {

i++;

}

String prefix = first.substring(0, i);

return prefix.isEmpty() ? "-1" : prefix;

}

public static void main(String[] args) {

String[] arr = {"geeksforgeeks", "geeks", "geek", "geezer"};

System.out.println("Longest Common Prefix: " + longestCommonPrefix(arr));

}

}Time Complexity: O(n)

Space Complexity: O(N)

17)Deleting mid element from stack

import java.util.Stack;

class DeleteMiddleElement {

public static void deleteMiddle(Stack<Integer> stack, int current, int size) {

if (stack.isEmpty() || current == size / 2) {

stack.pop();

return;

}

int temp = stack.pop();

deleteMiddle(stack, current + 1, size);

stack.push(temp);

}

public static void main(String[] args) {

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

stack.push(1);

stack.push(2);

stack.push(3);

stack.push(4);

stack.push(5);

int size = stack.size();

deleteMiddle(stack, 0, size);

System.out.println("Stack after deleting middle element: " + stack);

}

}

Time Complexity: O(N)

Space Complexity O(N)

18) Next greater element

import java.util.Stack;

class NextGreaterElement {

public static void printNextGreaterElements(int[] arr) {

int n = arr.length;

int[] nge = new int[n];

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

for (int i = n - 1; i >= 0; i--) {

while (!stack.isEmpty() && stack.peek() <= arr[i]) {

stack.pop();

}

nge[i] = stack.isEmpty() ? -1 : stack.peek();

stack.push(arr[i]);

}

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

System.out.println(arr[i] + " -> " + nge[i]);

}

}

public static void main(String[] args) {

int[] arr = {4, 5, 2, 25};

printNextGreaterElements(arr);

}

}

Time Complexity: O(N)

Space Complexity: O(N)

19)Right view of binary tree

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

left = right = null;

}

}

class BinaryTree {

TreeNode root;

public List<Integer> rightView(TreeNode root) {

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

if (root == null) return result;

Queue<TreeNode> queue = new LinkedList<>();

queue.add(root);

while (!queue.isEmpty()) {

int levelSize = queue.size();

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

TreeNode current = queue.poll();

if (i == levelSize - 1) {

result.add(current.val);

}

if (current.left != null) queue.add(current.left);

if (current.right != null) queue.add(current.right);

}

}

return result;

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

tree.root = new TreeNode(1);

tree.root.left = new TreeNode(2);

tree.root.right = new TreeNode(3);

tree.root.left.right = new TreeNode(5);

tree.root.right.right = new TreeNode(4);

List<Integer> rightView = tree.rightView(tree.root);

System.out.println("Right View of the Binary Tree: " + rightView);

}

}

Time Comopexity: O(N)

Space Complexity: O(N)

20) Depth of binary tree

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int val) {

this.val = val;

left = right = null;

}

}

class BinaryTree {

TreeNode root;

public int maxDepth(TreeNode node) {

if (node == null) {

return 0;

}

int leftDepth = maxDepth(node.left);

int rightDepth = maxDepth(node.right);

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

}

public static void main(String[] args) {

BinaryTree tree = new BinaryTree();

tree.root = new TreeNode(1);

tree.root.left = new TreeNode(2);

tree.root.right = new TreeNode(3);

tree.root.left.left = new TreeNode(4);

tree.root.left.right = new TreeNode(5);

int depth = tree.maxDepth(tree.root);

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

}

}Time complexity: O(N)

Space Complexity: O(N)