DSA PRACTISE – 1 – 09/11/2024

S RAMYA – AIML

1. Khaden’s Algorithm

package dsa;

import java.util.\*;

public class Khaden {

    public static long maximumSubArraySum(int[] nums, int n){

        long maxx = Integer.MIN\_VALUE;

        long summ = 0;

        for (int i = 0; i<n; i++){

            summ+=nums[i];

            maxx = Math.max(maxx, summ);

            if (summ < 0) summ = 0;

        }

        if (maxx < 0) maxx = 0;

        return maxx;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n;

        n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i<n; i++){

            arr[i] = sc.nextInt();

        }

        long res = maximumSubArraySum(arr, n);

        System.out.println(res);

        sc.close();

    }

}

/\*

Test case and O/P:

 15

-7 -8 -16 -4 -8 -5 -7 -11 -10 -12 -4 -6 -4 -16 -10

0

10

18 -6 -6 -5 7 10 16 -6 -2 0

34

 \*/

Time complexity: O(N)

Space complexity: O(1)

1. Maximum Product SubArray

package dsa;

import java.util.\*;

public class MaxProduct {

    public static long maximumSubArrayProduct(int[] arr, int n){

        long prefixSum = 1, suffixSum = 1;

        long ret = Integer.MIN\_VALUE;

        for (int i = 0; i<n; i++){

            if (prefixSum == 0) prefixSum = 1;

            if (suffixSum == 0) suffixSum = 1;

            prefixSum\*=arr[i];

            suffixSum\*=arr[n-i-1];

            ret = Math.max(ret, Math.max(prefixSum, suffixSum));

        }

        return ret;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n;

        n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i<n; i++){

            arr[i] = sc.nextInt();

        }

        long res = maximumSubArrayProduct(arr, n);

        System.out.println(res);

        sc.close();

    }

}

/\*

 Test case and O/P

 5

-2 3 4 -1 -7

84

14

-2 3 4 -1 -7 0 -2 3 1 4 0 4 6 -1

84

 \*/

Time complexity: O(N)

Space complexity: O(1)

1. Search In Rotated Sorted Array

package dsa;

import java.util.Scanner;

public class RotatedSorted {

    public static int rotatedSorted(int[] arr, int n, int target) {

        for (int i = 0; i < n; i++) {

            if (arr[i] == target)

                return i;

        }

        return -1;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n;

        n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        int target = sc.nextInt();

        long res = rotatedSorted(arr, n, target);

        System.out.println(res);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 14

 \* -2 3 4 -1 -7 0 -2 3 1 4 0 4 6 -1

 \* 4

 \*

 \* 2

 \*/

Time Complexity: O(N)

Space Complexity: O(1)

1. Container with Most Water

package dsa;

import java.util.Scanner;

public class MostWater {

    public static long mostWater(int[] arr, int n) {

        int l = 0, r = n - 1;

        int area = Integer.MIN\_VALUE;

        while (l < r) {

            area = Math.max(area, (r - l) \* Math.min(arr[l], arr[r]));

            if (arr[l] < arr[r])

                l++;

            else

                r--;

        }

        return area;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n;

        n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        long res = mostWater(arr, n);

        System.out.println(res);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 9

 \* 1 8 6 2 5 4 8 3 7

 \* 49

 \*

 \*

 \* 2

 \* 1 2

 \* 1

 \*/

Time Complexity: O(N) Space Complexity: O(1)

1. Trapping rain water

package dsa;

import java.util.\*;

public class Trapping {

    public static int waterTrapped(int[] arr, int n) {

        int waterTrapped = 0;

        for (int i = 0; i < n; i++) {

            int j = i;

            int leftMax = 0, rightMax = 0;

            while (j >= 0) {

                leftMax = Math.max(leftMax, arr[j]);

                j--;

            }

            j = i;

            while (j < n) {

                rightMax = Math.max(rightMax, arr[j]);

                j++;

            }

            waterTrapped += Math.min(leftMax, rightMax) - arr[i];

        }

        return waterTrapped;

    }

    public static void main(String[] args) {

        int n;

        Scanner sc = new Scanner(System.in);

        n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        int res = waterTrapped(arr, n);

        System.out.println(res);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 12

 \* 0 1 0 2 1 0 1 3 2 1 2 1

 \* 6

 \*/

Time Complexity: O(N)

Space Complexity: O(1)

1. Factorial of a large number

package dsa;

import java.math.BigInteger;

import java.util.\*;

public class Factorial {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int number = sc.nextInt();

        BigInteger fact = BigInteger.ONE;

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

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

        }

        System.out.println(fact);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 12

 \* 479001600

 \*/

Time complexity: O(N)

Space complexity: O(1)

1. Chocolate Distribution Problem

package dsa;

import java.util.\*;

public class Chocolate {

    public static int distribute(int[] arr, int m) {

        int n = arr.length;

        if (n < m)

            return -1;

        Arrays.sort(arr);

        int mind = Integer.MAX\_VALUE;

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

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

            mind = Math.min(mind, diff);

        }

        return mind;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int m = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        int res = distribute(arr, m);

        System.out.println(res);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 7

 \* 3

 \* 7 3 2 4 9 7 56

 \*

 \* 2

 \*/

Time Complexity: O(nlogn)

Space Complexity: O(1)

1. Merge overlapping intervals

package dsa;

import java.util.\*;

public class Intervals {

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

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

        List<int[]> mergedList = new ArrayList<>();

        int[] currentInterval = intervals[0];

        mergedList.add(currentInterval);

        for (int[] interval : intervals) {

            int currentEnd = currentInterval[1];

            int nextStart = interval[0];

            int nextEnd = interval[1];

            if (nextStart <= currentEnd) {

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

            } else {

                currentInterval = interval;

                mergedList.add(currentInterval);

            }

        }

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

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

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

        for (int i = 0; i < n; i++) {

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

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

        }

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

        for (int[] interval : result) {

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

        }

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 4

 \* 1 3 2 4 6 8 9 10

 \*

 \* [1, 4]

 \* [6, 8]

 \* [9, 10]

 \*

 \*/

Time Complexity: O(nlogn) Space Complexity: O(N)

1. Boolean Matrix Question

package dsa;

import java.util.\*;

public class Matrix {

    public static void modify(int[][] mat, int m, int n) {

        boolean fr = false, fc = false;

        for (int j = 0; j < n; j++)

            if (mat[0][j] == 1)

                fr =

                        true;

        for (int i = 0; i < m; i++)

            if (mat[i][0] == 1)

                fc =

                        true;

        for (int i = 1; i < m; i++) {

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

                if (mat[i][j] == 1) {

                    mat[i][0] = 1;

                    mat[0][j] = 1;

                }

            }

        }

        System.out.println("Matrix after marking rows and columns:");

        print(mat, m, n);

        for (int i = 1; i < m; i++) {

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

                if (mat[i][0] == 1 || mat[0][j] == 1)

                    mat[i][j]

                            = 1;

            }

        }

        if (fr)

            for (int j = 0; j < n; j++)

                mat[0][j] = 1;

        if (fc)

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

                mat[i][0] = 1;

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

        print(mat, m, n);

    }

    public static void print(int[][] mat, int m, int n) {

        for (int i = 0; i < m; i++) {

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

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

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int m = sc.nextInt();

        int n = sc.nextInt();

        int[][] mat = new int[m][n];

        System.out.println("Enter matrix elements:");

        for (int i = 0; i < m; i++)

            for (int j = 0; j < n; j++)

                mat[i][j] = sc.nextInt();

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

        print(mat, m, n);

        modify(mat, m, n);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 3 4

 \* 1 0 0 1

 \* 0 0 1 0

 \* 0 0 0 0

 \*

 \* Enter matrix elements:

 \* Original Matrix:

 \* 1 0 0 1

 \* 0 0 1 0

 \* 0 0 0 0

 \* Matrix after marking rows and columns:

 \* 1 0 1 1

 \* 1 0 1 0

 \* 0 0 0 0

 \* Modified Matrix:

 \* 1 1 1 1

 \* 1 1 1 1

 \* 1 0 1 1 \*/

Time Complexity: O(M\*N) Space Complexity: O(1)

1. 1Print matrix in spiral form

package dsa;

import java.util.Scanner;

public class SpiralMatrix {

    public static void SpiralMatrix(int[][] mat, 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(mat[top][i] + " ");

            }

            top++;

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

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

            }

            right--;

            if (top <= bottom) {

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

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

                }

                bottom--;

            }

            if (left <= right) {

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

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

                }

                left++;

            }

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int m = sc.nextInt();

        int n = sc.nextInt();

        int[][] mat = new int[m][n];

        for (int i = 0; i < m; i++) {

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

                mat[i][j] = sc.nextInt();

            }

        }

        SpiralMatrix(mat, m, n);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 2 4

 \* 3 10 11 12

 \* 13 14 15 16

 \*

 \* 3 10 11 12 16 15 14 13

 \*/

Time complexity: O(m\*n)

Space complexity: O(1)

1. Check if parenthesis expression is balanced or not

package dsa;

import java.util.\*;

public class Parenthesis {

    public static String isBalanced(String str) {

        int count = 0;

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

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

                count++;

            } else if (str.charAt(i) == ')') {

                count--;

            }

            if (count < 0) {

                return "Not Balanced";

            }

        }

        if (count == 0) {

            return "Balanced";

        } else {

            return "Not Balanced";

        }

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String str = sc.nextLine();

        System.out.println(isBalanced(str));

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* (((())))

 \* Balanced

 \*/

Time complexity: O(N)

Space complexity: O(1)

1. Check two strings are anagrams of each other

package dsa;

import java.util.Scanner;

public class Anagrams {

    public static boolean areAnagrams(String s1, String s2) {

        if (s1.length() != s2.length()) {

            return false;

        }

        int[] charCount = new int[26];

        for (int i = 0; i < s1.length(); i++) {

            charCount[s1.charAt(i) - 'a']++;

            charCount[s2.charAt(i) - 'a']--;

        }

        for (int count : charCount) {

            if (count != 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s1 = sc.nextLine();

        String s2 = sc.nextLine();

        if (areAnagrams(s1, s2)) {

            System.out.println("True");

        } else {

            System.out.println("False");

        }

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* abcdef

 \* fedcba

 \*

 \* True

 \*/

Time complexity: O(N) Space complexity: O(1)

1. Longest Palindromic Substring

package dsa;

import java.util.Scanner;

public class PalindromicSubstring {

    public static String lps(String s) {

        if (s == null || s.length() < 1) {

            return "";

        }

        int start = 0, maxLen = 1;

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

            int len1 = expand(s, i, i);

            int len2 = expand(s, i, i + 1);

            int len = Math.max(len1, len2);

            if (len > maxLen) {

                maxLen = len;

                start = i - (len - 1) / 2;

            }

        }

        return s.substring(start, start + maxLen);

    }

    private static int expand(String s, int left, int right) {

        while (left >= 0 && right < s.length() && s.charAt(left)

                == s.charAt(right)) {

            left--;

            right++;

        }

        return right - left - 1;

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        String s = sc.nextLine();

        System.out.println(lps(s));

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* abcdefff

 \*

 \* fff

 \*/

Time complexity: O(N^2)

Space complexity: O(1)

1. Longest common prefix using sorting

package dsa;

import java.util.\*;

public class CommonPrefix {

    public static String lcp(String[] arr) {

        if (arr.length == 0)

            return "-1";

        Arrays.sort(arr);

        String f = arr[0], l = arr[arr.length - 1];

        int i = 0;

        while (i < f.length() && i < l.length() && f.charAt(i)

                == l.charAt(i))

            i++;

        return i == 0 ? "-1" : f.substring(0, i);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        String[] arr = new String[n];

        for (int i = 0; i < n; i++)

            arr[i] = sc.next();

        System.out.println(lcp(arr));

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 3

 \* flower flo fl

 \*

 \* fl

 \*/

Time Complexity: O(nlogn + k)

Space Complexity: O(1)

1. Delete the middle element of stack

package dsa;

import java.util.\*;

public class StackDelete {

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

        if (middle == 0) {

            System.out.println("Removing middle element: " +

                    stack.peek());

            stack.pop();

            return;

        }

        int top = stack.pop();

        System.out.println("Popped element: " + top);

        deleteMiddle(stack, middle - 1);

        stack.push(top);

        System.out.println("Pushed element back: " + top);

    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

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

        int n = sc.nextInt();

        for (int i = 0; i < n; i++) {

            int element = sc.nextInt();

            stack.push(element);

        }

        System.out.println(stack);

        int middle = n / 2;

        deleteMiddle(stack, middle);

        System.out.println(stack);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 4

 \* 1 2 3 4

 \*

 \* Original Stack: [1, 2, 3, 4]

 \* Popped element: 4

 \* Popped element: 3

 \* Removing middle element: 2

 \* Pushed element back: 3

 \* Pushed element back: 4

 \* [1, 3, 4] \*/

Time Complexity: O(N)

Space Complexity: O(N)

1. Next greater element in every element in given array

package dsa;

import java.util.\*;

public class NextGreater {

    public static void printNGE(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) {

        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

        int[] arr = new int[n];

        for (int i = 0; i < n; i++) {

            arr[i] = sc.nextInt();

        }

        printNGE(arr);

        sc.close();

    }

}

/\*

 \* Test cases and O/P

 \* 4

 \* 1 2 3 4

 \*

 \* 1 -> 2

 \* 2 -> 3

 \* 3 -> 4

 \* 4 -> -1

 \*

 \*/

Time Complexity: O(N)

Space Complexity: O(N)

1. Print Right view of Binary Tree

package dsa;

import java.util.\*;

class TreeNode{

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int val){

        this.val = val;

        this.left = left;

        this.right = right;

    }

}

public class RightView {

    public static void recur(TreeNode root, int level, List<Integer> ds){

        if (root == null) return;

        if (ds.size() == level) ds.add(root.val);

        recur(root.right, level+1, ds);

        recur(root.left, level+1, ds);

    }

    public static List<Integer> rightSideView(TreeNode root) {

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

        recur(root, 0, ds);

        return ds;

    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(25);

        root.left = new TreeNode(20);

        root.right = new TreeNode(36);

        root.left.left = new TreeNode(10);

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

        root.left.right = new TreeNode(22);

        root.left.left.right = new TreeNode(12);

        root.right.left = new TreeNode(30);

        root.right.left.left = new TreeNode(28);

        root.right.right = new TreeNode(40);

        System.out.println(rightSideView(root));

    }

}

/\*

 \* Test cases and O/P

 \*

 \* [25, 36, 40, 28]

 \*/

Time Complexity: O(N)

Space Complexity: O(H)

1. Maximum Depth or Height of Binary Tree

package dsa;

import java.util.\*;

class TreeNode{

    int val;

    TreeNode left;

    TreeNode right;

    TreeNode(int val){

        this.val = val;

        this.left = left;

        this.right = right;

    }

}

public class HeightBT {

    public static int maxDepth(TreeNode root) {

        if (root == null) return 0;

        int l = maxDepth(root.left);

        int r = maxDepth(root.right);

        return Math.max(l,r)+1;

    }

    public static void main(String[] args) {

        TreeNode root = new TreeNode(25);

        root.left = new TreeNode(20);

        root.right = new TreeNode(36);

        root.left.left = new TreeNode(10);

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

        root.left.right = new TreeNode(22);

        root.left.left.right = new TreeNode(12);

        root.right.left = new TreeNode(30);

        root.right.left.left = new TreeNode(28);

        root.right.right = new TreeNode(40);

        System.out.println(maxDepth(root));

    }

}

/\*

 \* Test cases and O/P

 \* 4

 \*/

Time Complexity: O(N)

Space Complexity: O(H)