Name : SASTA HARI HARAN R DEPT: CSE

9.11.2024 (Coding Practice Problems)

1. Maximum Subarray Sum – Kadane’s Algorithm:

**CODE:**

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Size");

int n = sc.nextInt();

int ma = 0;

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

int [] ar = new int[n];

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

ar[i] = sc.nextInt();

}

int sum = 0;

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

sum+=ar[i];

if(sum < 0) {

sum =0;

}

ma = Math.*max*(ma, sum);

}

System.***out***.println("The Maximum Subarray Sum is " + ma);

}

}

**OUTPUT :**

Enter the Size

7

Enter the Elements

2

3

-8

7

-1

2

3

The Maximum Subarray Sum is 11

**Time Complexity : O(n)**

**Space Complexity : O(1)**

2. Maximum Product Subarray Given an integer array, the task is to find the maximum product of any subarray.

**CODE:**

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Size");

int n = sc.nextInt();

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

int [] ar = new int[n];

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

ar[i] = sc.nextInt();

}

int pre =1 , suf =1;

int ma = ar[0];

n = ar.length;

int c;

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

if (pre == 0){

pre =1;

}

if(suf == 0){

suf =1 ;

}

pre = pre\*ar[i];

suf = suf \* ar[n-i-1];

c =Math.*max*(pre, suf);

ma = Math.*max*(ma,c);

}

System.***out***.println(" The Maximum product of any subarray is "+ ma);

}

}

**OUTPUT** :

6

Enter the Elements

-2

6

-3

-10

0

2

The Maximum product of any subarray is 180

**Time Complexity : O(n)**

**Space Complexity : O(1)**

3. Search in a sorted and rotated Array Given a sorted and rotated array arr[] of n distinct elements, the task is to find the index of given key in the array. If the key is not present in the array, return -1

**CODE:**

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.*in*);

System.*out*.println("Enter the Size");

int n = sc.nextInt();

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

int [] nums = new int[n];

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

nums[i] = sc.nextInt();

}

System.*out*.println("Enter the Search Element");

int k = sc.nextInt();

int l = 0;

int r = nums.length-1;

boolean f = false;

while(l <= r){

int mid = (l+r)/2;

if (nums[mid] == k){

f = true;

System.*out*.println(mid);

}

if (nums[l] <= nums[mid]){

if(nums[l] <= k && k <= nums[mid]){

r = mid-1;

}

else{

l = mid+1 ;

}

}else{

if (nums[mid] <= k && k <= nums[r]){

l = mid+1;

}

else{

r =mid-1;

}

}

}

if(f==false) {

System.*out*.println(-1);

}

}

}

**OUTPUT** :

Enter the Size

7

Enter the Elements

4

5

6

7

0

1

2

Enter the Search Element

0

4

**Time Complexity : O(log n)**

**Space Complexity : O(1)**

4. Container with Most Water

**CODE**:

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

int ma;

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

int [] h = new int[n];

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

h[i] = sc.nextInt();

}

int l = 0;

int r= h.length-1;

ma = 0;

int area ;

while(l < r){

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

ma = Math.max(area, ma);

if(h[l] < h[r]){

l++;

}

else{

r--;

}

}

System.out.println("The Maximum Subarray Sum is " + ma);

}

}  
  
**OUTPUT**:  
  
Enter the Size

4

Enter the Elements

1

5

4

3  
Maximum amount of water a container can store is  
6  
  
5. Find the Factorial of a large number

**CODE**:

import java.math.BigInteger;

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Element");

int k = sc.nextInt();

BigInteger res = BigInteger.***ONE***;

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

res = res.multiply(BigInteger.*valueOf*(i));

}

System.***out***.println(res);

}

}

**OUTPUT**:

Enter the Element

100

93326215443944152681699238856266700490715968264381621468592963895217599993229915608941463976156518286253697920827223758251185210916864000000000000000000000000

**Time Complexity : O(N)**

**Space Complexity :(1)**

6. Trapping Rainwater Problem states that given an array of n non-negative integers arr[] representing an elevation map where the width of each bar is 1, compute how much water it can trap after rain.

**CODE** :

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Size");

int n = sc.nextInt();

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

int [] height = new int[n];

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

height[i] = sc.nextInt();

}

n = height.length;

int [] pre = new int[n];

int [] suf = new int[n];

int ma = -999999999;

int mi = -999999999;

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

ma = Math.*max*(ma, height[i]);

pre[i] = ma;

mi = Math.*max*(mi, height[n-i-1]);

suf[n-i-1] = mi;

}

int res = 0;

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

int c = Math.*min*(pre[i],suf[i]) - height[i];

if (c > 0){

res+= c;

}

}

System.***out***.println(res);

}

}

**OutPut**:

Enter the Size

7

Enter the Elements

3

0

1

0

4

0

2

10

**Time Complexity : O(N)**

**Space Complexity : O(N)**

7. Chocolate Distribution Problem Given an array arr[] of n integers where arr[i] represents the number of chocolates in ith packet. Each packet can have a variable number of chocolates. There are m students, the task is to distribute chocolate packets such that:

**CODE**:

import java.util.\*;

public class Practice {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.***in***);

System.***out***.println("Enter the Size");

int n = sc.nextInt();

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

int [] arr = new int[n];

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

arr[i] = sc.nextInt();

}

System.***out***.println("Enter the No. of Students");

int m = sc.nextInt();

Arrays.*sort*(arr);

int mi = 999999999;

int l = 0, r = m-1;

while (r < arr.length){

mi = Math.*min*(mi, arr[r]-arr[l]);

l+=1 ;

r+=1 ;

}

System.***out***.println("The minimum difference is " + mi);

}

}

**OUTPUT** :

Enter the Size

7

Enter the Elements

7

3

2

4

9

12

56

Enter the No. of Students

3

The minimum difference is 2

**Time Complexity : O(n log n)**

**Space Complexity : O(1)** - Input array is not considered

8. Merge Overlapping Intervals Given an array of time intervals where arr[i] = [starti, endi], the task is to merge all the overlapping intervals into one and output the result which should have only mutually exclusive intervals.

**CODE**:

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Scanner;

public class MergeIntervals {

public static void main(String[] sasta) {

Scanner sc = new Scanner(System.in);

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

int n = sc.nextInt();

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

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

int a = sc.nextInt();

int b = sc.nextInt();

ar[i][0] = a;

ar[i][1] = b;

}

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

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

res.add(ar[0]);

for (int i = 1; i < ar.length; i++) {

int[] lastInterval = res.get(res.size() - 1);

int[] currentInterval = ar[i];

if (lastInterval[1] >= currentInterval[0]) {

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

} else {

res.add(currentInterval);

}

}

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

for (int[] interval : res) {

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

}

}

}

**Output**:

Enter the Number of Elements

3

1

3

2

3

6

8

Merged intervals:

[1, 3]

[6, 8]

**Time Complexity :** O(n log n)

**Space Complexity** : O(log n)

9. A Boolean Matrix Question Given a boolean matrix mat[M][N] of size M X N, modify it such that if a matrix cell mat[i][j] is 1 (or true) then make all the cells of ith row and jth column as 1.

**CODE** :

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class BooleanMatrix {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int M = scanner.nextInt();

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

int N = scanner.nextInt();

int[][] mat = new int[M][N];

System.out.println("Enter the matrix values (0 or 1):");

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

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

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

}

}

Set<Integer> rows = new HashSet<>();

Set<Integer> cols = new HashSet<>();

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

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

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

rows.add(i);

cols.add(j);

}

}

}

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

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

if (rows.contains(i) || cols.contains(j)) {

mat[i][j] = 1;

}

}

}

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

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

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

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

}

System.out.println();

}

}

}

**Output** :

Enter number of rows: 2

Enter number of columns: 3

Enter the matrix values (0 or 1):

0

0

1

0

0

0

Modified Matrix:

1 1 1

0 0 1

Time Complexity : O(N\*M)

Space Complexity : O(M+N)

10. Print a given matrix in spiral form Given an m x n matrix, the task is to print all elements of the matrix in spiral form.

**CODE** :

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

public class SpiralMatrix {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int row = scanner.nextInt();

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

int col = scanner.nextInt();

int[][] matrix = new int[row][col];

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

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

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

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

}

}

printSpiralOrder(matrix, row, col);

}

public static void printSpiralOrder(int[][] matrix, int row, int col) {

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

int total = row \* col;

int srow = 0;

int scol = 0;

int erow = row - 1;

int ecol = col - 1;

int count = 0;

while (count < total) {

for (int i = scol; i <= ecol && count < total; i++) {

ans.add(matrix[srow][i]);

count++;

}

srow++;

for (int i = srow; i <= erow && count < total; i++) {

ans.add(matrix[i][ecol]);

count++;

}

ecol--;

for (int i = ecol; i >= scol && count < total; i--) {

ans.add(matrix[erow][i]);

count++;

}

erow--;

for (int i = erow; i >= srow && count < total; i--) {

ans.add(matrix[i][scol]);

count++;

}

scol++;

}

System.out.println("Spiral Order:");

for (int num : ans) {

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

}

System.out.println();

}

}

**Output**:

Enter number of rows: 4

Enter number of columns: 4

Enter matrix elements:

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

Spiral Order:

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

**Time Complexity**: O(M×N)

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

11. Check if given Parentheses expression is balanced or not Given a string str of length N, consisting of ‘(‘ and ‘)’ only, the task is to check whether it is balanced or not.

**CODE** :

import java.util.Scanner;

import java.util.Stack;

public class ValidParentheses {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the string

System.out.print("Enter the string: ");

String s = scanner.nextLine();

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

}

public static boolean isValid(String s) {

if (s.length() % 2 == 1) return false;

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

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

if (c == ')') {

if (!stack.isEmpty() && stack.peek() == '(') {

stack.pop();

} else {

stack.push(c);

}

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

if (!stack.isEmpty() && stack.peek() == '[') {

stack.pop();

} else {

stack.push(c);

}

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

if (!stack.isEmpty() && stack.peek() == '{') {

stack.pop();

} else {

stack.push(c);

}

} else {

stack.push(c);

}

}

return stack.isEmpty();

}

}

**OUTPUT**:

Enter the string: ((()))()()

Balanced ?: true

**Time Complexity**: **O(n)**

**Space** **Complexity : O(n)**

12. Check if two Strings are Anagrams of each other Given two strings s1 and s2 consisting of lowercase characters, the task is to check whether the two given strings are anagrams of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

**CODE**:

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class AnagramCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the two strings

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

String s = scanner.nextLine();

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

String t = scanner.nextLine();

System.out.println isAnagram(s, t));

scanner.close();

}

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

if (s.length() != t.length()) return false;

Map<Character, Integer> map1 = new HashMap<>();

Map<Character, Integer> map2 = new HashMap<>();

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

map1.put(c, map1.getOrDefault(c, 0) + 1);

}

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

map2.put(c, map2.getOrDefault(c, 0) + 1);

}

return map1.equals(map2);

}

}

**OUTPUT** :

Enter first string: geeks

Enter second string: kseeg

True

**Time Complexity** : O(N)

**Space Complexity** : O(26) 🡪 O(1) , since 26 characters is fixed

13. Longest Palindromic Substring Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

**CODE** :

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String s = scanner.nextLine();

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

scanner.close();

}

public static String longestPalindrome(String s) {

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

return s;

}

String maxStr = s.substring(0, 1);

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

String odd = expandFromCenter(s, i, i);

String even = expandFromCenter(s, i, i + 1);

if (odd.length() > maxStr.length()) {

maxStr = odd;

}

if (even.length() > maxStr.length()) {

maxStr = even;

}

}

return maxStr;

}

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

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

left--;

right++;

}

return s.substring(left + 1, right);

}

}

**OUTPUT**:

Enter a string: Geeks

Longest Palindromic Substring: ee

Time Complexity : O(n\*2)

Space Complexity : O(1)

14. Longest Common Prefix using Sorting Given an array of strings arr[]. The task is to return the longest common prefix among each and every strings present in the array. If there‟s no prefix common in all the strings, return “-1”.

**CODE** :

import java.util.\*;

public class Solution {

public static void main(String[] sasta) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

scanner.nextLine();

List<String> strs = new ArrayList<>();

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

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

strs.add(scanner.nextLine());

}

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

}

public static String longestCommonPrefix(List<String> strs) {

if (strs.size() == 0) {

return "";

}

Collections.sort(strs);

int i = 0, j = 0;

StringBuilder res = new StringBuilder();

System.out.println("First string: " + strs.get(0));

while (i < strs.get(0).length() && j < strs.get(strs.size() - 1).length()) {

if (strs.get(0).charAt(i) == strs.get(strs.size() - 1).charAt(j)) {

res.append(strs.get(0).charAt(i));

i++;

j++;

} else {

break;

}

}

return res.toString();

}

}

Output:

Enter the number of strings: 4

Enter the strings:

geeksforgeeks

geeks

geek

geezer

First string: geek

Longest Common Prefix: gee

**Time Complexity** : O(n log n)

**Space Complexity** : O(1) apart from input and output

15. Delete middle element of a stack Given a stack with push(), pop(), and empty() operations, The task is to delete the middle element of it without using any additional data structure.

**CODE** :

import java.util.\*;

public class Stack1 {

static void deleteMid(Stack<Character> st, int n, int curr) {

if (st.empty() || curr == n)

return;

char x = st.pop();

deleteMid(st, n, curr + 1);

if (curr != n / 2)

st.push(x);

}

public static void main(String args[]) {

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

st.push('1');

st.push('2');

st.push('3');

st.push('4');

st.push('5');

st.push('6');

st.push('7');

deleteMid(st, st.size(), 0);

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

while (!st.empty()) {

char p = st.pop();

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

}

}

}

**OUTPUT**:

Stack after deleting middle element:

7 6 5 3 2 1

**Time Comple**xity : O(N)

**Space Complexity** : O(N) – For the Recursion call Stack

16. Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element

import java.util.\*;

public class Solution {

public static void main(String[] sasta) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

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

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

nums[i] = scanner.nextInt();

}

Solution solution = new Solution();

int[] result = solution.nextGreaterElements(nums);

System.out.println("Next greater elements:");

for (int num : result) {

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

}

scanner.close();

}

public int[] nextGreaterElements(int[] nums) {

int[] res = new int[nums.length];

int n = nums.length;

Arrays.fill(res, -1);

Deque<Integer> s = new ArrayDeque<>();

for (int i = 0; i < 2 \* n; i++) {

while (!s.isEmpty() && nums[i % n] > nums[s.peek()]) {

res[s.pop()] = nums[i % n];

}

s.push(i % n);

}

return res;

}

}

**OUTPUT** :

Enter the number of elements in the array: 4

Enter the elements of the array:

4

5

2

25

Next greater elements:

5 25 25 -1

**Time Complexity** : O(N)

**Space Complexity** : O(N)

17. Print Right View of a Binary Tree Given a Binary Tree, the task is to print the Right view of it. The right view of a Binary Tree is a set of rightmost nodes for every level.

CODE :

import java.util.\*;

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

public class Solution {

public List<Integer> rightSideView(TreeNode root) {

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

rightView(root, result, 0);

return result;

}

public void rightView(TreeNode curr, List<Integer> result, int currDepth) {

if (curr == null) {

return;

}

if (currDepth == result.size()) {

result.add(curr.val);

}

rightView(curr.right, result, currDepth + 1);

rightView(curr.left, result, currDepth + 1);

}

public static void main(String[] sasta) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

Solution solution = new Solution();

List<Integer> result = solution.rightSideView(root);

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

for (int val : result) {

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

}

System.out.println();

}

}

**OUTPUT**:

Right side view of the tree: 1 3 5

**Time Complexity** : O(N)

**Space Complexity** : O(h) – height of the Binary Tree

18. Maximum Depth or Height of Binary Tree Given a binary tree, the task is to find the maximum depth or height of the tree. The height of the tree is the number of vertices in the tree from the root to the deepest node.

**CODE**:

class TreeNode {

int val;

TreeNode left, right;

TreeNode(int x) {

val = x;

left = right = null;

}

}

public class Solution {

public int maxDepth(TreeNode root) {

if (root == null) {

return 0;

}

int leftDepth = maxDepth(root.left);

int rightDepth = maxDepth(root.right);

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

}

public static void main(String[] sasta) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

root.right.right = new TreeNode(6);

root.right.right.left = new TreeNode(7);

Solution solution = new Solution();

int depth = solution.maxDepth(root);

System.out.println("Maximum depth of the tree: " + depth);

}

}

**OUTPUT**:Sample tree definition

1

// / \

// 2 3

// / / \

// 4 5 6

// /

// 7

Maximum depth of the tree: 4

**Time Complexity** : O(N) – number of nodes visit each once

**Space Complexity** : O(h) – height of the binary Tree