Eswar Arjun(AI&DS )

Sunday(9/11/2024)

DSA **Training Practice – 1**

**1 . Maximum Subarray Sum – Kadane‟s Algorithm: Given an array arr[], the task is to find the subarray that has the maximum sum and return its sum.**

**Input: arr[] = {2, 3, -8, 7, -1, 2, 3}**

**Output: 11**

**Time Complexity : O(n )**

**Code:**

**import java.util.\*;**

**public class kadanes{**

**public static void main(String[] args){**

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

**int[] arr={2, 3, -8, 7, -1, 2, 3};**

**int sum=0,max=Integer.*MIN\_VALUE*;**

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

**sum=sum+arr[i];**

**if(sum>max){**

**max=sum;**

**}**

**if(sum<0){**

**sum=0;**

**}**

**}**

**System.*out*.println("Maximum SubArray Length: "+max);**

**}**

**}**

**Output:**

****

**------------------------------------------------------------------------------------------------------------------------------------------**

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

**Input: arr[] = {-2, 6, -3, -10, 0, 2}**

**Output: 180**

**Time Complexity:O(n)**

**Code:**

**package ArrayList;**

**import java.util.\*;**

**public class stack {**

**public static void main(String[] args) {**

**int[] arr= {-2, 6, -3, -10, 0, 2};**

**int prefix=1;**

**int suffix=1;**

**int max=Integer.*MIN\_VALUE*;**

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

**prefix=prefix\*arr[i];**

**suffix=suffix\*arr[arr.length-i-1];**

**if(prefix==0) {**

**prefix=1;**

**}**

**if(suffix==0) {**

**suffix=1;**

**}**

**max=Math.*max*(max,Math.*max*(prefix,suffix));**

**}**

**System.*out*.println("Maximum Product:"+max);**

**}**

**}**

**Output:**

****

**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.**

**Input : arr[] = {4, 5, 6, 7, 0, 1, 2}, key = 0**

**Output : 4**

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

**public class sorted {**

**public static int searchInRotatedArray(int[] arr, int key) {**

**int low = 0, high = arr.length - 1;**

**while (low <= high) {**

**int mid = (low + high) / 2;**

**if (arr[mid] == key) {**

**return mid;**

**}**

**if (arr[low] <= arr[mid]) {**

**if (arr[low] <= key && key < arr[mid]) {**

**high = mid - 1;**

**} else {**

**low = mid + 1;**

**}**

**}**

**else {**

**if (arr[mid] < key && key <= arr[high]) {**

**low = mid + 1;**

**} else {**

**high = mid - 1;**

**}**

**}**

**}**

**return -1;**

**}**

**public static void main(String[] args) {**

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

**int key = 0;**

**int result = *searchInRotatedArray*(arr, key);**

**System.*out*.println("Index of " + key + ": " + result);**

**}**

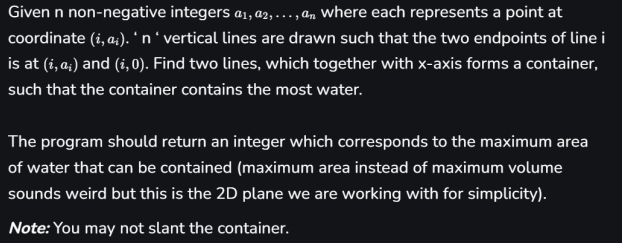
**}**

**Output:**

****

**------------------------------------------------------------------------------------------------------------------------------------------**

**4.**

****

**Input: arr = [1, 5, 4, 3]**

**Output: 6**

**Time Complexity:O(n)**

**Code:**

**package ArrayList;**

**public class Mostwater {**

**public static int container(int[] arr){**

**int left=0,right=arr.length-1,maxx=0;**

**while(left<right){**

**int width=right-left;**

**int height=Math.*min*(arr[left],arr[right]);**

**maxx=Math.*max*(maxx,width\*height);**

**if(arr[left]<arr[right]) left++;**

**else right--;**

**}**

**return maxx;**

**}**

**public static void main(String[] args) {**

**int[] arr1={1, 5, 4, 3};**

**int[] arr2 = {3, 1, 2, 4, 5};**

**System.*out*.println(*container*(arr1));**

**System.*out*.println(*container*(arr2));**

**}**

**}**

**Output:**

****

**------------------------------------------------------------------------------------------------------------------------------------------**

**5.** **Find the Factorial of a large number**

**Input: 100**

**Output: 933262154439441526816992388562667004907159682643816214685929638952175999932299 156089414639761565182862536979208272237582511852109168640000000000000000000000 00**

**Time Complexity:** **O(n2)**

**Code:**

**package ArrayList;**

**import java.math.\*;**

**public class Big {**

**public static void main(String[] args) {**

**BigInteger b=BigInteger.*ONE*;**

**int n=100;**

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

**b=b.multiply(b.*valueOf*(i));**

**}**

**System.*out*.println("Maximum Product: "+ b);**

**}**

**}**

**Output:**

****

**------------------------------------------------------------------------------------------------------------------------------------------**

**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. Input: arr[] = {3, 0, 1, 0, 4, 0, 2}**

**Output: 10**

**Time Complexity: O(n)**

**package util;**

**public class trapping {**

**public static int trapRainwater(int[] arr) {**

**int n = arr.length;**

**if (n < 3) return 0;**

**int left = 0, right = n - 1;**

**int leftMax = 0, rightMax = 0;**

**int waterTrapped = 0;**

**while (left < right) {**

**if (arr[left] < arr[right]) {**

**if (arr[left] >= leftMax) {**

**leftMax = arr[left];**

**} else {**

**waterTrapped += leftMax - arr[left];**

**}**

**left++;**

**} else {**

**if (arr[right] >= rightMax) {**

**rightMax = arr[right];**

**} else {**

**waterTrapped += rightMax - arr[right];**

**}**

**right--;**

**}**

**}**

**return waterTrapped;**

**}**

**public static void main(String[] args) {**

**int[] arr = {3, 0, 1, 0, 4, 0, 2};**

**System.*out*.println("Total water trapped: " + *trapRainwater*(arr));**

**}**

**}**

**Output:**

****

**------------------------------------------------------------------------------------------------------------------------------------------**

**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: Each student gets exactly one packet. The difference between the maximum and minimum number of chocolates in the packets given to the students is minimized.**

**Input: arr[] = {7, 3, 2, 4, 9, 12, 56}, m = 3**

**Output: 2**

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

**Code:**

package ArrayList;

import java.util.\*;

public class choclate {

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

if (arr.length < m) {

return -1;

}

Arrays.*sort*(arr);

int minDiff = Integer.***MAX\_VALUE***;

for (int i = 0; i <= arr.length - m; 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 result = *findMinDiff*(arr, m);

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

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

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.

Input: arr[] = [[1, 3], [2, 4], [6, 8], [9, 10]]

Output: [[1, 4], [6, 8], [9, 10]]

TimeComplexity:O(n)

Code:

package ArrayList;

import java.util.\*;

public class intervels {

public static void main(String[] args) {

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

HashMap<Integer,Integer> h1=new HashMap<>();

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

h1.put(arr[i][0],arr[i][1]);

}

List<Map.Entry<Integer,Integer>> li=new ArrayList<>(h1.entrySet());

li.sort(Map.Entry.*comparingByKey*());

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

int start = li.get(0).getKey();

int end = li.get(0).getValue();

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

int next = li.get(i).getKey();

int last = li.get(i).getValue();

if (end >= next) {

end = Math.*max*(end, last);

} else {

merged.add(new int[]{start, end});

start = next;

end = last;

}

}

merged.add(new int[]{start, end});

int[][] nums=new int[merged.size()][2];

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

nums[i]=merged.get(i);

}

for (int[] interval : merged) {

System.***out***.println("[" + interval[0] + ", " + interval[1] + "]");

}

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

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.

Input: {{1, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 0, 0}}

Output: {{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 0, 1, 1}}

Time Complexity: O(n\*m)

Code:

package ArrayList;

public class booleanMatrix {

public static void main(String[] args) {

int[][] mat = {{0, 0, 0},

{0, 0, 1}};

int[][] updat=new int[mat.length][mat[0].length];

int n=mat.length;int m=mat[0].length;

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

for(int j=0;j<mat[i].length;j++) {

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

for(int x=0;x<m;x++) {

updat[i][x]=1;

}

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

updat[x][j]=1;

}

}

}

}

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

for(int j=0;j<mat[i].length;j++) {

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

}

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

}

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

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.

int[][] matrix= {{1,2,3},{4,5,6},{7,8,9}};

Time complexity: O(n∗m)

Code:

package ArrayList;

import java.util.\*;

public class spiral {

public static void main(String[] args) {

int[][] matrix= {{1,2,3},{4,5,6},{7,8,9}};

int rows = matrix.length;

int cols = matrix[0].length;

int x = 0;

int y = 0;

int dx = 1;

int dy = 0;

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

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

res.add(matrix[y][x]);

matrix[y][x] = -101;

if (!(0 <= x + dx && x + dx < cols && 0 <= y + dy && y + dy < rows) || matrix[y+dy][x+dx] == -101) {

int temp = dx;

dx = -dy;

dy = temp;

}

x += dx;

y += dy;

}

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

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

13. 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.

Input: str = “((()))()()”

Output: Balanced

Time Complexity:O(n)

Code:

package ArrayList;

import java.util.\*;

public class parantheses {

public static void main(String[] args) {

String str1 = "((()))()()";

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

for(int i=0;i<str1.length();i++) {

char ch=str1.charAt(i);

if(ch=='(') {

stack.push(ch);

}else if(stack.isEmpty()) {

System.***out***.println("Unbalanced");

return;

}else {

stack.pop();

}

}

if(stack.isEmpty()) {

System.***out***.println("Balanced");

}else {

System.***out***.println("Unbalanced");

}

}

}

Output:



14. 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.

Input: s1 = “geeks”

s2 = “kseeg”

Output: true

TimeComplexity: O(nlogn)

Code:

package ArrayList;

import java.util.\*;

public class string {

public static void main(String[] args) {

String str1="allergy";

String str2="allergic";

char[] ch1=str1.toCharArray();

Arrays.*sort*(ch1);

str1=new String(ch1);

char[] ch2=str2.toCharArray();

Arrays.*sort*(ch2);

str2=new String(ch2);

System.***out***.println(str1+" "+str2);

if(str1==str2) {

System.***out***.println("Ture");

return;

}

System.***out***.println("False");

}

}

Output:



15. 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.

Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

TimeComplexity:o(n2)

Code:

package ArrayList;

public class longestCommon {

private static boolean isPalindrome(String str) {

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

public static void main(String[] args) {

String s="forgeeksskeegfor";

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

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

return;

}

int maxLen = 1;

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

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

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

if (j - i > maxLen && *isPalindrome*(s.substring(i, j))) {

maxLen = j - i;

maxStr = s.substring(i, j);

}

}

}

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

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

16. 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”.

Input: arr[] = [“geeksforgeeks”, “geeks”, “geek”, “geezer”]

Output: gee

TimeComplexity:o(nlogn)

Code:

package ArrayList;

import java.util.\*;

public class commonPrefix {

public static void main(String[] args) {

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

Arrays.*sort*(str);

System.***out***.println(Arrays.*toString*(str));

String s1 = str[0];

String s2 = str[str.length-1];

int i = 0;

while(i< s1.length() && i < s2.length()){

if(s1.charAt(i) == s2.charAt(i)){

i++;

} else {

break;

}

}

System.***out***.println(s1.substring(0,i));

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

17. 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.

Input : Stack[] = [1, 2, 3, 4, 5]

Output : Stack[] = [1, 2, 4, 5]

Time Complexity:O(n)

Code:

package ArrayList;

import java.util.\*;

public class stack1 {

public static void main(String[] args) {

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

int [] arr= {1, 2, 3, 4, 5};

if(arr.length<=1) {

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

return;

}

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

st.push(arr[i]);

}

Stack<Integer> s2=new Stack<>();

for(int i=0;i<=st.size()/2;i++) {

s2.push(st.pop());

}

st.pop();

int len=s2.size();

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

st.push(s2.pop());

}

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

}

}

Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

18.Next Greater Element (NGE) for every element in given Array Given an array, print the Next Greater Element (NGE) for every element. Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [ 4 , 5 , 2 , 25 ]

Output: 4 –> 5 5 –> 25 2 –> 25 25 –> -1

Time Complexity: O(N)

package util;

import java.util.Stack;

import java.util.HashMap;

public class greaterele {

public static void printNextGreaterElements(int[] arr) {

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

HashMap<Integer, Integer> ngeMap = new HashMap<>();

for (int i = arr.length - 1; i >= 0; i--) {

int currentElement = arr[i];

while (!stack.isEmpty() && stack.peek() <= currentElement) {

stack.pop();

}

int nextGreater = stack.isEmpty() ? -1 : stack.peek();

ngeMap.put(currentElement, nextGreater);

stack.push(currentElement);

}

for (int element : arr) {

System.***out***.println(element + " -> " + ngeMap.get(element));

}

}

public static void main(String[] args) {

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

*printNextGreaterElements*(arr);

}

}

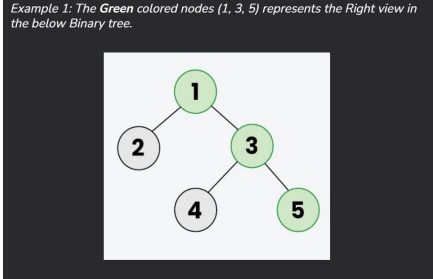
Output:



**------------------------------------------------------------------------------------------------------------------------------------------**

19. 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.

**Time Complexity**: O(n)



Code:

import java.util.ArrayList;

import java.util.List;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int val) { this.val = val; }

}

class Solution {

int maxlevel = 0;

public List<Integer> rightSideView(TreeNode root) {

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

right(root, 1, list);

return list;

}

void right(TreeNode root, int level, List<Integer> list) {

if (root == null) {

return;

}

if (maxlevel < level) {

list.add(root.val);

maxlevel = level;

}

right(root.right, level + 1, list);

right(root.left, level + 1, list);

}

public static void main(String[] args) {

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

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

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

Solution solution = new Solution();

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

System.out.println(result);

}

}

Output:

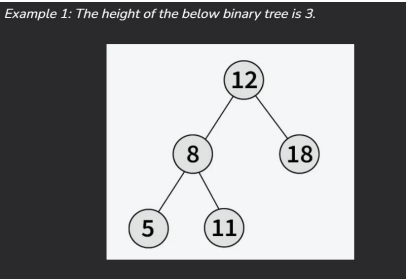


**------------------------------------------------------------------------------------------------------------------------------------------**

20. 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

Time Complexity:O(n)

TestCase:



Code:

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int val) { this.val = val; }

}

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int left = maxDepth(root.left);

int right = maxDepth(root.right);

return Math.max(left, right) + 1;

}

public static void main(String[] args) {

TreeNode root = new TreeNode(3);

root.left = new TreeNode(9);

root.right = new TreeNode(20);

root.right.left = new TreeNode(15);

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

Solution solution = new Solution();

int result = solution.maxDepth(root);

System.out.println(result);

}

}

Output:

