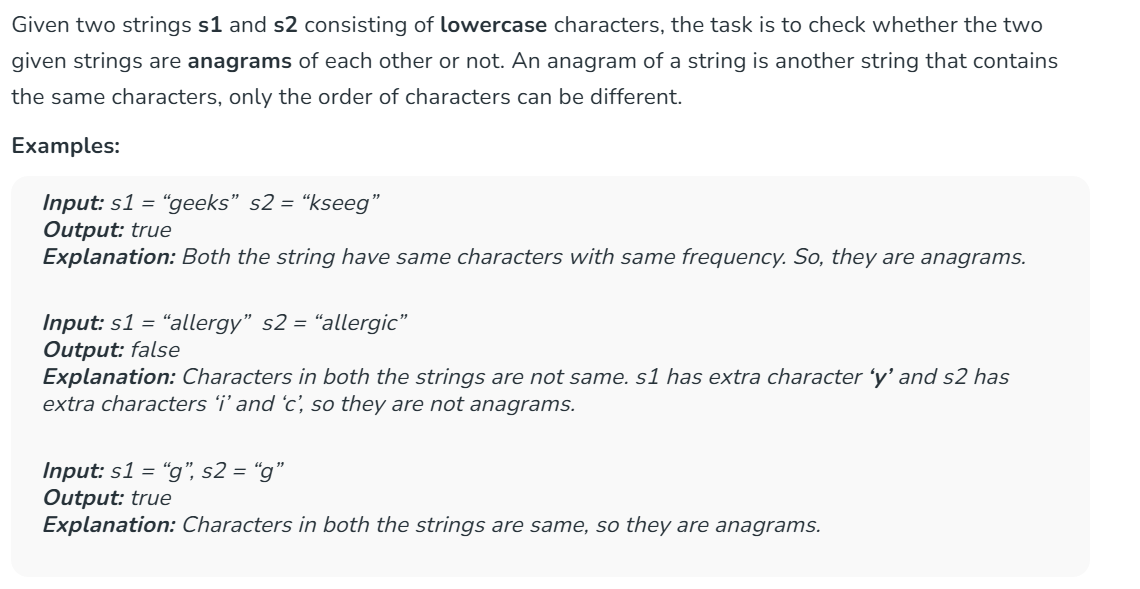
**DSA Practice Test – 3** 12th Nov 2024

**1. Anagram Program**

****

**Code:**

import java.util.HashMap;

import java.util.Scanner;

public class Anagram {

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

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

return false;

}

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

for (char c : s1.toCharArray()) {

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

}

for (char c : s2.toCharArray()) {

if (!charCountMap.containsKey(c) || charCountMap.get(c) == 0) {

return false;

}

charCountMap.put(c, charCountMap.get(c) - 1);

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String s1 = scanner.nextLine();

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

String s2 = scanner.nextLine();

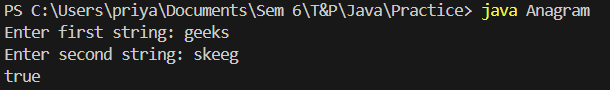
System.out.println(isAnagram(s1, s2));

scanner.close();

}

}

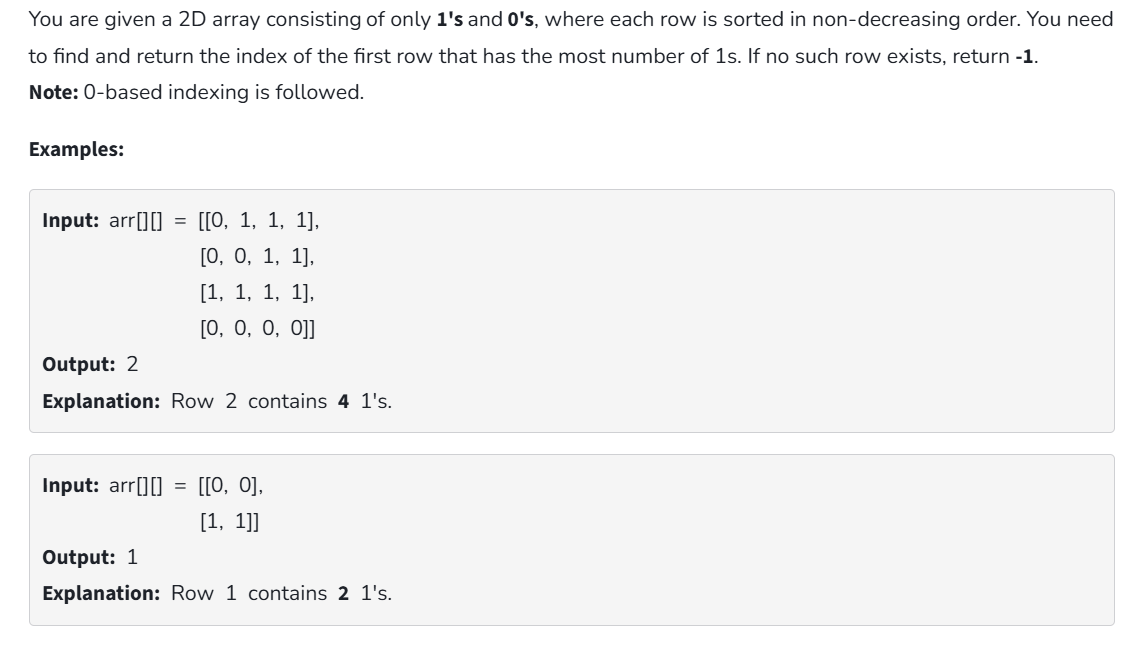
**Output:**



**Time Complexity:** O (n)

**Space Complexity:** O (1)

**2. Row with max 1’s**



**Code:**

class Solution {

static int rowWithMax1s(int arr[][]) {

int m=0;

int mn=-1;

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

int n=0;

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

if(arr[i][j]==1)

n++;

}

if(n>m){

mn=i;

m=n;

}

}

return mn;

}

public static void main (String ar[]){

int[][] arr=new int[][]{

{1,0,0,1},

{1,1,1,0},

{0,0,1,1},

{1,1,0,0}

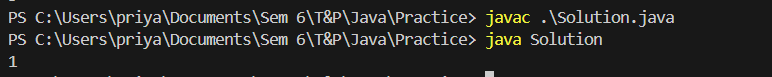
};

System.out.print(rowWithMax1s(arr));

}

}

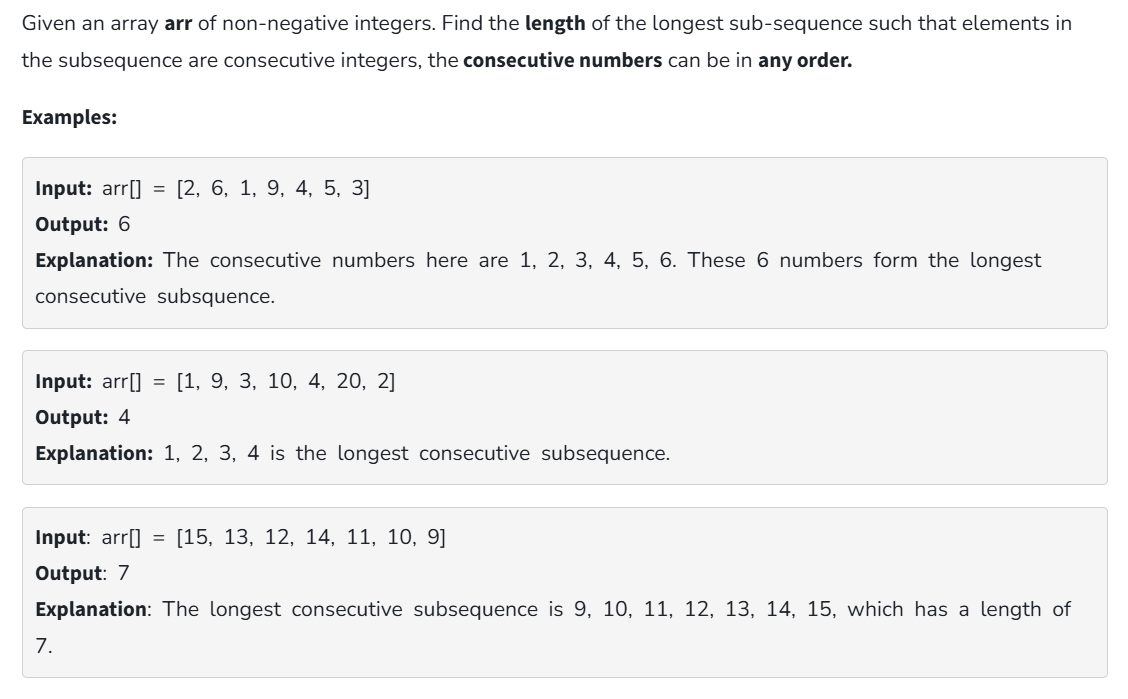
**Output:**



**Time Complexity:** O (n2)

**Space Complexity:** O (1)

**3. Longest Consecutive Subsequence:**



**Code:**

import java.util.Arrays;

class Solution {

static int findLongestConseqSubseq(int[] arr) {

if (arr.length == 0) return 0;

Arrays.sort(arr);

int longestStreak = 1;

int currentStreak = 1;

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

if (arr[i] != arr[i - 1]) {

if (arr[i] == arr[i - 1] + 1) {

currentStreak++;

} else {

longestStreak = Math.max(longestStreak, currentStreak);

currentStreak = 1;

}

}

}

return Math.max(longestStreak, currentStreak);

}

public static void main (String ar[]){

int[] arr=new int[]{

2,6,1,9,4,5,3

};

System.out.print(findLongestConseqSubseq(arr));

}

}

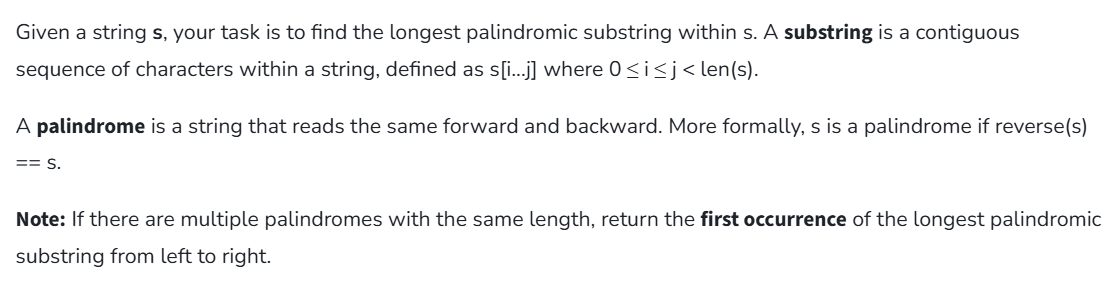
**Output:**

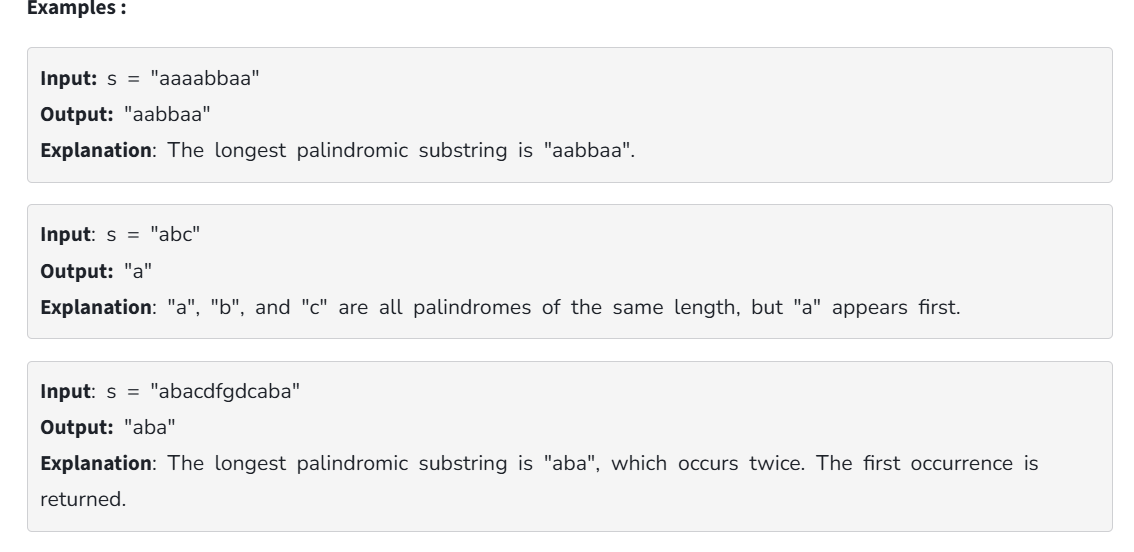


**Time Complexity:** O (n log n)

**Space Complexity:** O (1)

**4. Longest Palindrome in a String**

****

****

**Code:**

public class LongestPalindrome {

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;

}

static String longestPalindrome(String s) {

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

return s;

}

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);

}

}

}

return maxStr;

}

public static void main(String ar[]){

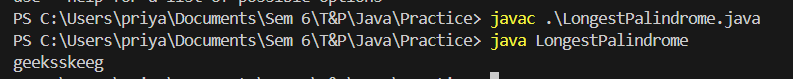
String str="forgeeksskeegfor";

System.out.print(longestPalindrome(str));

}

}

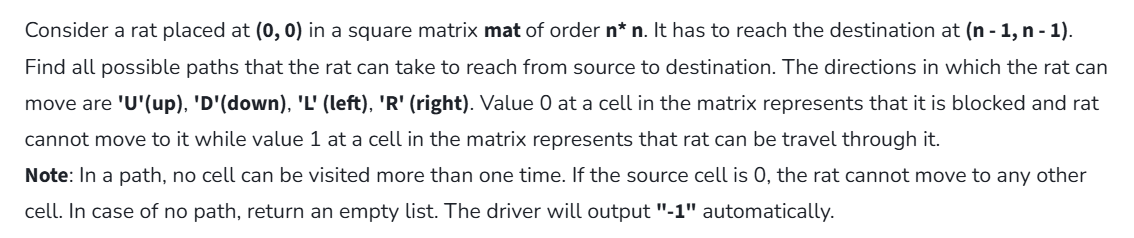
**Output:**

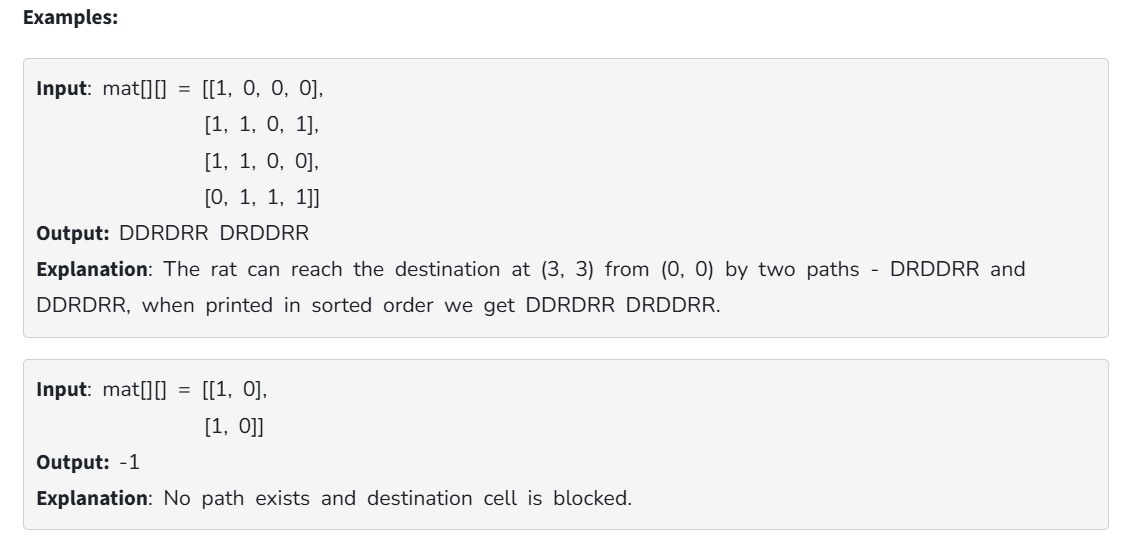


**Time Complexity:** O (n2)

**Space Complexity:** O (1)

**5. Rat in a Maze**





**Code:**

import java.util.ArrayList;

public class MazePaths {

static String direction = "DLRU";

static int[] dr = { 1, 0, 0, -1 };

static int[] dc = { 0, -1, 1, 0 };

static boolean isValid(int row, int col, int n,

int[][] maze)

{

return row >= 0 && col >= 0 && row < n && col < n

&& maze[row][col] == 1;

}

static void findPath(int row, int col, int[][] maze,

int n, ArrayList<String> ans,

StringBuilder currentPath)

{

if (row == n - 1 && col == n - 1) {

ans.add(currentPath.toString());

return;

}

maze[row][col] = 0;

for (int i = 0; i < 4; i++) {

int nextrow = row + dr[i];

int nextcol = col + dc[i];

if (isValid(nextrow, nextcol, n, maze)) {

currentPath.append(direction.charAt(i));

findPath(nextrow, nextcol, maze, n, ans,

currentPath);

currentPath.deleteCharAt(

currentPath.length() - 1);

}

}

maze[row][col] = 1;

}

public static void main(String[] args)

{

int[][] maze = { { 1, 0, 0, 0 },

{ 1, 1, 0, 1 },

{ 1, 1, 0, 0 },

{ 0, 1, 1, 1 } };

int n = maze.length;

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

StringBuilder currentPath = new StringBuilder();

if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {

findPath(0, 0, maze, n, result, currentPath);

}

if (result.size() == 0)

System.out.println(-1);

else

for (String path : result)

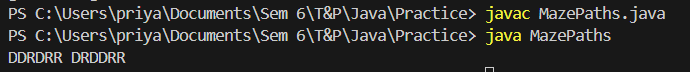
System.out.print(path + " ");

System.out.println();

}

}

**Output:**



**Time Complexity:** O (2\*\* n2 )