**SANJAY\_M – CSE – DSA – PRACTICE – 4**

**Q1**. **anagram program**

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

Explanation: Both the string have same characters with same frequency. So, they are anagrams.

Input: s1 = “allergy” s2 = “allergic”

Output: false

Explanation: Characters in both the strings are not same. s1 has extra character ‘y’ and s2 has extra characters ‘i’ and ‘c’, so they are not anagrams.

CODE:

import java.util.Arrays;

class anagram {

static boolean areAnagrams(String s1, String s2) {

char[] s1Array = s1.toCharArray();

char[] s2Array = s2.toCharArray();

Arrays.sort(s1Array);

Arrays.sort(s2Array);

return Arrays.equals(s1Array, s2Array);

}

public static void main(String[] args) {

String s1 = "geeks";

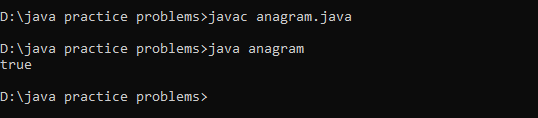
String s2 = "kseeg";

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

}

}

OUTPUT:



Time Complexity: O(m\*log(m) + n\*log(n))

**Q2. Row with max 1’s**

Given a m x n binary matrix mat, find the 0-indexed position of the row that contains the maximum count of ones, and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with the smallest row number should be selected.

Return an array containing the index of the row, and the number of ones in it.

Input: mat = [[0,1],[1,0]]

Output: [0,1]

Explanation: Both rows have the same number of 1's. So we return the index of the smaller row, 0, and the maximum count of ones (1). So, the answer is [0,1].

CODE:

import java.util.Arrays;

class maxones {

public int[] rowAndMaximumOnes(int[][] m) {

int x = 0;

int p = 0;

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

int c = m[i][0];

for (int j = 1; j < m[0].length; j++) {

m[i][j] += c;

c = m[i][j];

}

if (c > x) {

x = c;

p = i;

}

}

return new int[]{p, x};

}

public static void main(String[] args) {

int[][] mat = {

{0, 1},

{1, 0}};

maxones mo = new maxones();

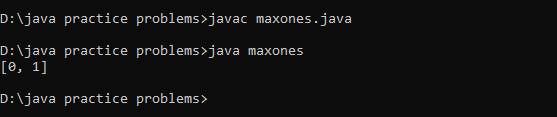
int[] result = mo.rowAndMaximumOnes(mat);

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

}

}

OUTPUT:



Time Complexity: O(m \* n)

**Q3. Longest consecutive subsequence**

Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Input: arr[] = {1, 9, 3, 10, 4, 20, 2}

Output: 4

Explanation: The subsequence 1, 3, 4, 2 is the longest subsequence of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}

Output: 5

Explanation: The subsequence 36, 35, 33, 34, 32 is the longest subsequence of consecutive elements.

CODE:

import java.io.\*;

import java.util.\*;

class longestconsecseq {

static int findLongestConsecSubseq(int arr[], int n) {

Arrays.sort(arr);

ArrayList<Integer> v = new ArrayList<>();

v.add(arr[0]);

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

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

v.add(arr[i]);

}

}

int ans = 1, count = 1;

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

if (v.get(i) == v.get(i - 1) + 1) {

count++;

} else {

count = 1;

}

ans = Math.max(ans, count);

}

return ans;

}

public static void main(String[] args) {

int arr[] = { 1, 9, 3, 10, 4, 20, 2 };

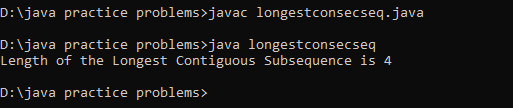
int n = arr.length;

System.out.println("Length of the Longest Contiguous Subsequence is " + findLongestConsecSubseq(arr, n));

}

}

OUTPUT:



Time Complexity: O(n log n)

**Q4. longest palindrome in a string**

Given a string s, your task is to find the longest palindromic substring within s. A substring is a contiguous sequence of characters within a string, defined as s[i...j] where 0 ≤ i ≤ j < len(s).

A palindrome is a string that reads the same forward and backward. More formally, s is a palindrome if reverse(s) == s.

Note: If there are multiple palindromes with the same length, return the first occurrence of the longest palindromic substring from left to right.

Input: s = "aaaabbaa"

Output: "aabbaa"

Explanation: The longest palindromic substring is "aabbaa".

Input: s = "abc"

Output: "a"

Explanation: "a", "b", and "c" are all palindromes of the same length, but "a" appears first.

CODE:

class longpalin {

static String longestPalin(String s) {

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

return s;

}

int maxLen = 1;

int start = 0;

int end = 0;

boolean[][] dp = new boolean[s.length()][s.length()];

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

dp[i][i] = true; // Each character is a palindrome of length 1

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

if (s.charAt(j) == s.charAt(i) && (i - j <= 2 || dp[j + 1][i - 1])) {

dp[j][i] = true;

if (i - j + 1 > maxLen) {

maxLen = i - j + 1;

start = j;

end = i;

}

}

}

}

return s.substring(start, end + 1);

}

public static void main(String[] args) {

String s = "aaaabbaa";

longpalin sol = new longpalin();

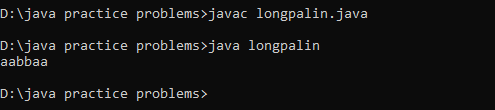
String result = sol.longestPalin(s);

System.out.println(result);

}

}

OUTPUT:



Time Complexity: O(n^2)

**Q5. rat in a maze problem**

Consider a rat placed at (0, 0) in a square matrix of order N \* N. It has to reach the destination at (N – 1, N – 1). Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are ‘U'(up), ‘D'(down), ‘L’ (left), ‘R’ (right). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it. Return the list of paths in lexicographically increasing order.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell.

Input:



Output: DDRDRR DRDDRR

Explanation:



CODE:

import java.util.ArrayList;

import java.util.List;

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)

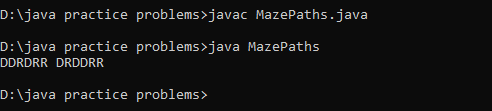
System.out.print(path + " ");

System.out.println();

}

}

OUTPUT:



Time Complexity: O(3^(m\*n))