**DSA- PRACTICE DAY 3-DHANUSHSHRUTHI S T AI&DS**

**1.ANAGRAM**

**CODE:**

import java.util.Arrays;

public class Anagram {

public static boolean areAnagrams(String str1, String str2) {

if (str1.length() != str2.length()) {

return false;

}

char[] arr1 = str1.toCharArray();

char[] arr2 = str2.toCharArray();

Arrays.sort(arr1);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

public static void main(String[] args) {

String str1 = "geeks";

String str2 = "skees";

if (areAnagrams(str1, str2)) {

System.out.println(str1 + " and " + str2 + " are anagrams.");

} else {

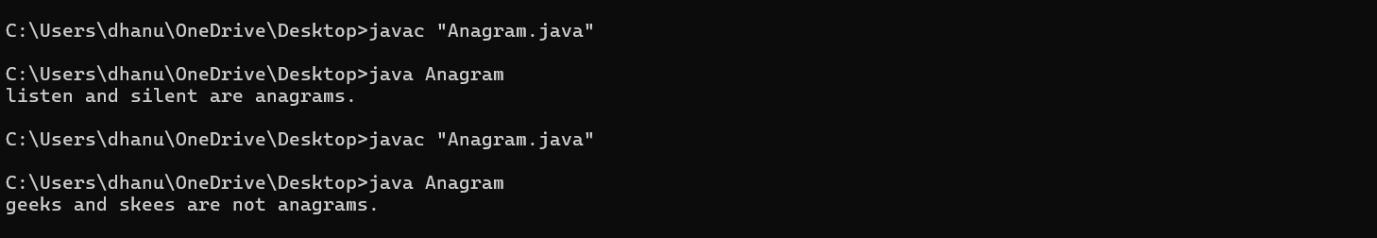
System.out.println(str1 + " and " + str2 + " are not anagrams.");

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(nlogn)**.

**2. Row with Max 1’s:**

**CODE:**

public class Row {

public static int Row(int[][] matrix) {

int maxRowIndex = -1;

int maxCount = -1;

int n = matrix.length;

int m = matrix[0].length;

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

int count = 0;

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

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

count++;

}

}

if (count > maxCount) {

maxCount = count;

maxRowIndex = i;

}

}

return maxRowIndex;

}

public static void main(String[] args) {

int[][] matrix = {

{0, 1, 1, 0},

{1, 0, 1, 1},

{0, 0, 1, 1},

{1, 1, 1, 1}

};

int result = Row(matrix);

if (result != -1) {

System.out.println("The row with the maximum number of 1's is: " + result);

} else {

System.out.println("No row contains 1's.");

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n×m)**

**3.LONGEST CONSECUTIVE SEQUENCE**

**CODE:**

import java.util.HashSet;

public class LongestConsecutiveSequence {

public static int longestConsecutive(int[] nums) {

if (nums.length == 0) {

return 0;

}

HashSet<Integer> numSet = new HashSet<>();

for (int num : nums) {

numSet.add(num);

}

int longestStreak = 0;

for (int num : numSet) {

if (!numSet.contains(num - 1)) {

int currentNum = num;

int currentStreak = 1;

while (numSet.contains(currentNum + 1)) {

currentNum++;

currentStreak++;

}

longestStreak = Math.max(longestStreak, currentStreak);

}

}

return longestStreak;

}

public static void main(String[] args) {

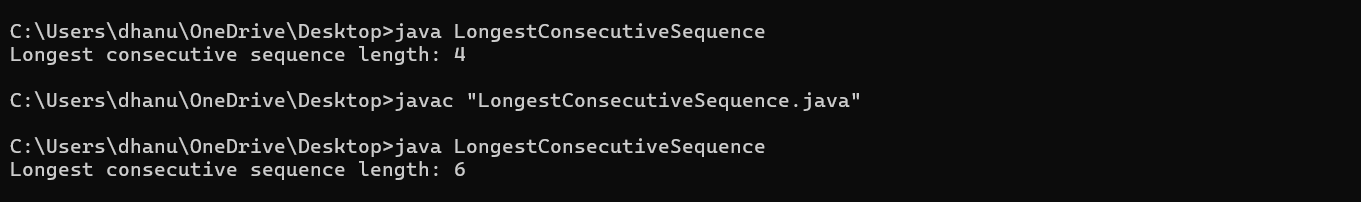
int[] nums = {2, 6, 1, 9, 4, 5, 3};

System.out.println("Longest consecutive sequence length: " + longestConsecutive(nums));

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n)**

**4.LONGEST PALINDROME**

**CODE:**

public class LongestPalindrome {

public static String longestPalindrome(String s) {

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

return "";

}

int start = 0, maxLength = 1;

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

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

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

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

if (len > maxLength) {

maxLength = len;

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

}

}

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

}

private static int expandFromCenter(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) {

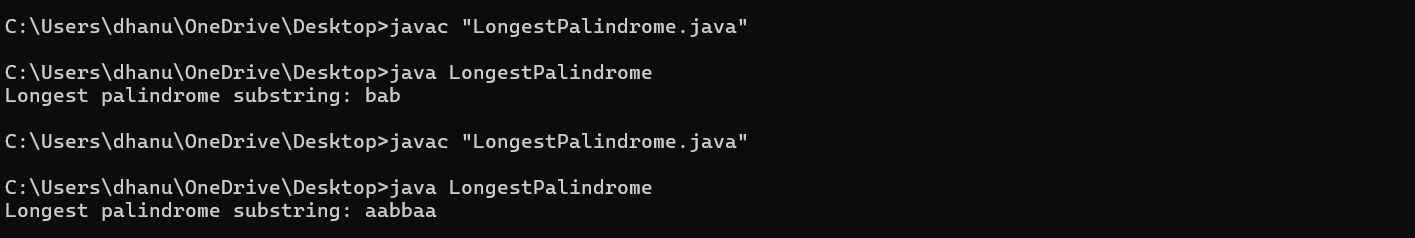
String s = "aaaabbaa";

System.out.println("Longest palindrome substring: " + longestPalindrome(s));

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(n2)**

**5.RAT IN MAZE**

**CODE:**

public class RatInMaze {

public static boolean solveMaze(int[][] maze, int[][] solution, int x, int y) {

int N = maze.length;

if (x == N - 1 && y == N - 1 && maze[x][y] == 1) {

solution[x][y] = 1;

return true;

}

if (isSafe(maze, x, y)) {

solution[x][y] = 1;

if (solveMaze(maze, solution, x + 1, y)) {

return true;

}

if (solveMaze(maze, solution, x, y + 1)) {

return true;

}

solution[x][y] = 0;

return false;

}

return false;

}

public static boolean isSafe(int[][] maze, int x, int y) {

int N = maze.length;

return (x >= 0 && y >= 0 && x < N && y < N && maze[x][y] == 1);

}

public static void printSolution(int[][] solution) {

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

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

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

}

System.out.println();

}

}

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[][] solution = new int[maze.length][maze[0].length];

if (solveMaze(maze, solution, 0, 0)) {

printSolution(solution);

} else {

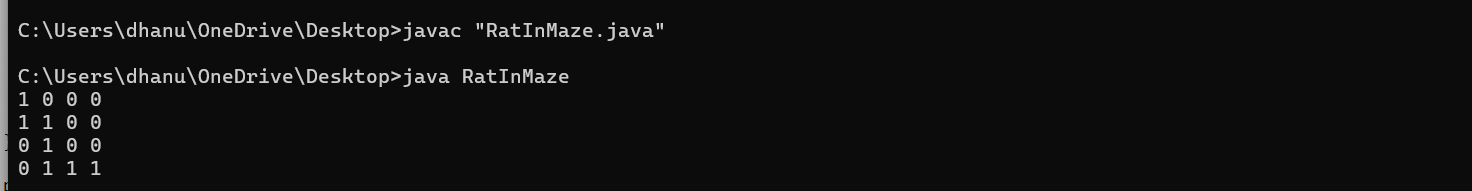
System.out.println("No solution exists");

}

}

}

**OUTPUT:**



**TIME COMPLEXITY:** **O(2^{N^2})**