Pavadareni R 12-11-2024

DSA PROGRAM

1. Anagram Program

import java.util.Arrays;

public class AnagramCheck {

    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 = "listen";

        String str2 = "silent";

        if (areAnagrams(str1, str2)) {

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

        } else {

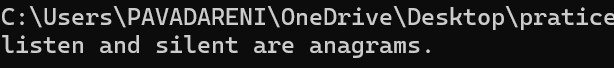
            System.out.println(str1 + " and " + str2 + " are not anagrams.");

        }

    }

}

Output



Time complexity: O(nlogn)

1. Row with Max 1s'

public class RowWithMax1s {

public static int rowWithMax1s(int[][] matrix) {

int m = matrix.length;

int n = matrix[0].length;

int maxRowIndex = -1;

int j = n - 1;

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

while (j >= 0 && matrix[i][j] == 1) {

j--;

maxRowIndex = i;

}

}

return maxRowIndex;

}

public static void main(String[] args) {

int[][] matrix = {

{ 0, 0, 0, 1, 1 },

{ 0, 0, 1, 1, 1 },

{ 0, 1, 1, 1, 1 },

{ 1, 1, 1, 1, 1 }

};

int rowIndex = rowWithMax1s(matrix);

if (rowIndex != -1) {

System.out.println("Row with maximum 1s: " + rowIndex);

} else {

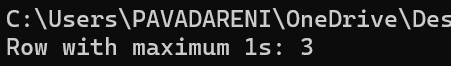
System.out.println("No 1s found in the matrix.");

}

}

}

Output



Time Complexity O(m+n)

1. Longest consequtive subsequence

import java.util.HashSet;

public class LongestConsecutiveSubsequence {

public static int longestConsecutive(int[] nums) {

if (nums == null || nums.length == 0) {

return 0;

}

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

for (int num : nums) {

set.add(num);

}

int longestStreak = 0;

for (int num : nums) {

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

int currentNum = num;

int currentStreak = 1;

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

currentNum++;

currentStreak++;

}

longestStreak = Math.max(longestStreak, currentStreak);

}

}

return longestStreak;

}

public static void main(String[] args) {

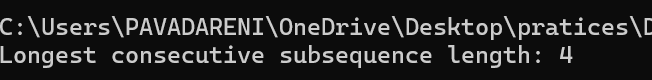
int[] nums = { 100, 4, 200, 1, 3, 2 };

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

}

}

Output



Time Complexity:O(n)

1. Longest Palindrome in a String

public class LongestPalindrome {

public static String expandAroundCenter(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);

}

public static String longestPalindrome(String s) {

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

return "";

}

String longest = "";

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

String palindrome1 = expandAroundCenter(s, i, i);

String palindrome2 = expandAroundCenter(s, i, i + 1);

if (palindrome1.length() > longest.length()) {

longest = palindrome1;

}

if (palindrome2.length() > longest.length()) {

longest = palindrome2;

}

}

return longest;

}

public static void main(String[] args) {

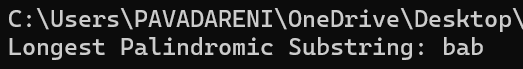
String s = "babad";

System.out.println("Longest Palindromic Substring: " + longestPalindrome(s)); // Output: "bab" or "aba"

}

}

Output



Time Complexity O(n^2)

1. Rat in a Maze Problem

public class RatInMaze {

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

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

    }

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

        if (x == maze.length - 1 && y == maze[0].length - 1) {

            solution[x][y] = 1;

            return true;

        }

        if (isSafe(maze, x, y)) {

            solution[x][y] = 1;

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

                return true;

            }

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

                return true;

            }

            solution[x][y] = 0;

            return false;

        }

        return false;

    }

    public static boolean findPath(int[][] maze) {

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

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

            printSolution(solution);

            return true;

        }

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

        return false;

    }

    public static void main(String[] args) {

        int[][] maze = {

                { 1, 0, 0, 0 },

                { 1, 1, 0, 1 },

                { 0, 1, 0, 0 },

                { 1, 1, 1, 1 }

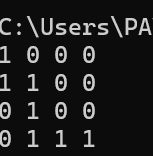
        };

        findPath(maze);

    }

}

Output



Time Complexity O(n^2)