**12-11-2024**

**CODING PRACTICE PROBLEMS**

1. **anagram program**

import java.util.Arrays;

import java.util.Scanner;

public class problem1 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the first string (s1): ");

String s1 = scanner.nextLine();

System.out.print("Enter the second string (s2): ");

String s2 = scanner.nextLine();

if (areAnagrams(s1, s2)) {

System.out.println("true");

} else {

System.out.println("false");

}

scanner.close();

}

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

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

return false;

}

char[] arr1 = s1.toCharArray();

char[] arr2 = s2.toCharArray();

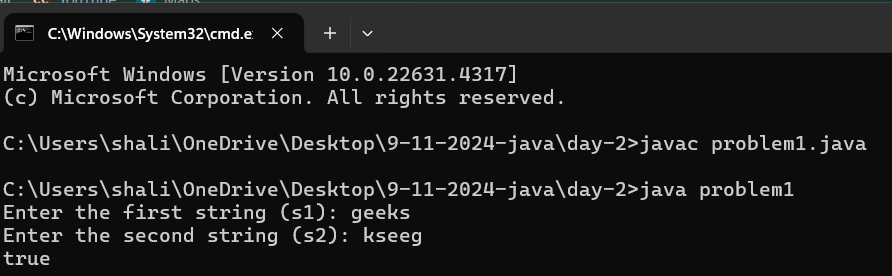
Arrays.sort(arr1);

Arrays.sort(arr2);

return Arrays.equals(arr1, arr2);

}

}



1. **row with max 1s'**

import java.util.Scanner;

public class problem2 {

public static int maxOnes(int[][] Mat, int N, int M) {

int maxRowIndex = -1;

int maxOnesCount = 0;

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

int onesCount = countOnes(Mat[i], M);

if (onesCount > maxOnesCount) {

maxOnesCount = onesCount;

maxRowIndex = i;

}

}

return maxRowIndex;

}

private static int countOnes(int[] row, int M) {

int low = 0, high = M - 1;

while (low <= high) {

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

if (row[mid] == 1) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return M - low;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of rows (N): ");

int N = scanner.nextInt();

System.out.print("Enter the number of columns (M): ");

int M = scanner.nextInt();

int[][] Mat = new int[N][M];

System.out.println("Enter the elements of the matrix row by row:");

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

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

Mat[i][j] = scanner.nextInt();

}

}

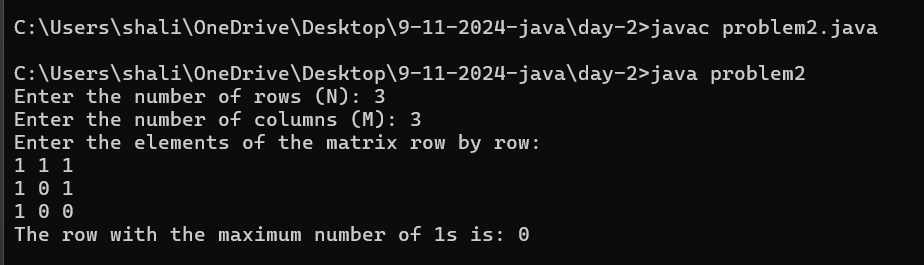
int result = maxOnes(Mat, N, M);

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

scanner.close();

}

}



1. **Longest consecutive subsequence**

import java.util.HashSet;

import java.util.Scanner;

public class problem3 {

public static int findLongestConsecutiveSubsequence(int[] arr) {

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

for (int num : arr) {

set.add(num);

}

int longestStreak = 0;

for (int num : set) {

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

int currentNum = num;

int currentStreak = 1;

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

currentNum += 1;

currentStreak += 1;

}

longestStreak = Math.max(longestStreak, currentStreak);

}

}

return longestStreak;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the array: ");

int n = scanner.nextInt();

int[] arr = new int[n];

System.out.println("Enter the elements of the array:");

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

arr[i] = scanner.nextInt();

}

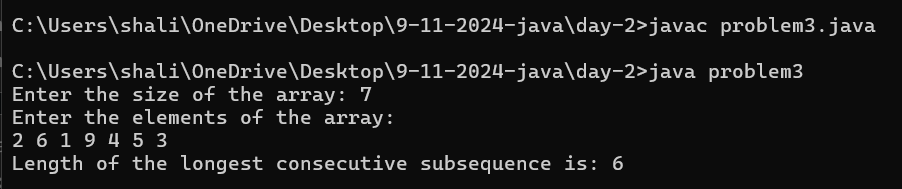
int result = findLongestConsecutiveSubsequence(arr);

System.out.println("Length of the longest consecutive subsequence is: " + result);

scanner.close();

}

}



1. **longest palindrome in a string**

import java.util.Scanner;

public class problem4 {

public static String longestPalindromicSubstring(String s) {

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

return s;

}

int start = 0, end = 0;

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

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

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

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

if (len > end - start) {

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

end = i + len / 2;

}

}

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

}

private static int expandAroundCenter(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) {

Scanner scanner = new Scanner(System.in);

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

String s = scanner.nextLine();

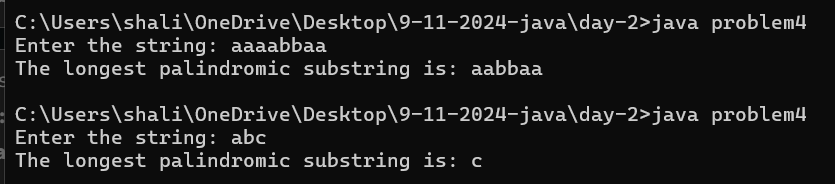
String result = longestPalindromicSubstring(s);

System.out.println("The longest palindromic substring is: " + result);

scanner.close();

}

}



1. **rat in a maze problem**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Scanner;

public class problem5 {

public static List<String> findPaths(int[][] mat, int n) {

List<String> paths = new ArrayList<>();

if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {

return paths;

}

boolean[][] visited = new boolean[n][n];

findPathsUtil(mat, n, 0, 0, "", visited, paths);

Collections.sort(paths);

return paths;

}

private static void findPathsUtil(int[][] mat, int n, int row, int col, String path, boolean[][] visited, List<String> paths) {

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

paths.add(path);

return;

}

visited[row][col] = true;

// Down

if (isSafe(mat, visited, n, row + 1, col)) {

findPathsUtil(mat, n, row + 1, col, path + "D", visited, paths);

}

// Left

if (isSafe(mat, visited, n, row, col - 1)) {

findPathsUtil(mat, n, row, col - 1, path + "L", visited, paths);

}

// Right

if (isSafe(mat, visited, n, row, col + 1)) {

findPathsUtil(mat, n, row, col + 1, path + "R", visited, paths);

}

// Up

if (isSafe(mat, visited, n, row - 1, col)) {

findPathsUtil(mat, n, row - 1, col, path + "U", visited, paths);

}

visited[row][col] = false;

}

private static boolean isSafe(int[][] mat, boolean[][] visited, int n, int row, int col) {

return row >= 0 && row < n && col >= 0 && col < n && mat[row][col] == 1 && !visited[row][col];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the size of the matrix (n): ");

int n = scanner.nextInt();

int[][] mat = new int[n][n];

System.out.println("Enter the matrix elements (0 or 1):");

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

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

mat[i][j] = scanner.nextInt();

}

}

List<String> result = findPaths(mat, n);

if (result.isEmpty()) {

System.out.println("-1");

} else {

for (String path : result) {

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

}

}

scanner.close();

}

}

