**Coding Practice day 3**

**1.** **Anagram program :**

**Code**

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String str1 = scanner.nextLine();

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

String str2 = scanner.nextLine();

boolean result = checkIfAnagram(str1, str2);

System.out.println("Are the two strings anagrams? " + result);

scanner.close();

}

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

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

return false;

}

Map<Character, Integer> charCount1 = new HashMap<>();

Map<Character, Integer> charCount2 = new HashMap<>();

for (char c : str1.toCharArray()) {

charCount1.put(c, charCount1.getOrDefault(c, 0) + 1);

}

for (char c : str2.toCharArray()) {

charCount2.put(c, charCount2.getOrDefault(c, 0) + 1);

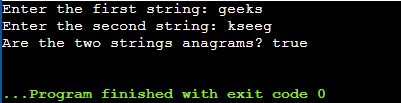
}

return charCount1.equals(charCount2);

}

}

**Output**



Time Complexity : O(N)

Space Complexity : O(n)

**2. Row with Max One’s**

**Code**

import java.util.Scanner;

class Main {

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

int rows = matrix.length;

int cols = matrix[0].length;

int maxOnesCount = 0;

int rowIndex = -1;

int onesCount;

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

onesCount = 0;

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

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

onesCount++;

}

}

if (maxOnesCount < onesCount) {

maxOnesCount = onesCount;

rowIndex = i;

}

}

return rowIndex;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int rows = scanner.nextInt();

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

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

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

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

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

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

}

}

int result = rowWithMaxOnes(matrix);

if (result != -1) {

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

} else {

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

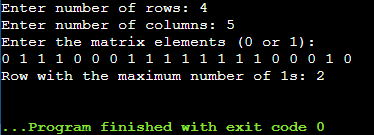
}

scanner.close();

}

}

**Output**

****

Time Complexity : O(n\*m)

Space Complexity : O(1)

**3. Longest consequtive subsequence**

**Code**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner inputScanner = new Scanner(System.in);

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

String inputString = inputScanner.nextLine();

System.out.println("Longest Palindromic Substring: " + findLongestPalindromicSubstring(inputString));

inputScanner.close();

}

public static String findLongestPalindromicSubstring(String inputString) {

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

return inputString;

}

String longestPalindrome = inputString.substring(0, 1);

for (int center = 0; center < inputString.length() - 1; center++) {

String oddPalindrome = expandAroundCenter(inputString, center, center);

String evenPalindrome = expandAroundCenter(inputString, center, center + 1);

if (oddPalindrome.length() > longestPalindrome.length()) {

longestPalindrome = oddPalindrome;

}

if (evenPalindrome.length() > longestPalindrome.length()) {

longestPalindrome = evenPalindrome;

}

}

return longestPalindrome;

}

private static String expandAroundCenter(String inputString, int leftIndex, int rightIndex) {

while (leftIndex >= 0 && rightIndex < inputString.length() && inputString.charAt(leftIndex) == inputString.charAt(rightIndex)) {

leftIndex--;

rightIndex++;

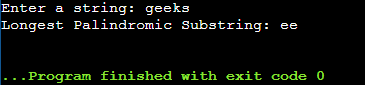
}

return inputString.substring(leftIndex + 1, rightIndex);

}

}

**Output**

****

Time Complexity : O(n\*2)

Space Complexity : O(1)

**6. Rat in a Maze**

**Code**

import java.util.ArrayList;

import java.util.List;

public class Main {

public static List<String> findPathsInMaze(int[][] maze) {

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

if (maze[0][0] == 0) return allPaths;

boolean[][] visitedCells = new boolean[maze.length][maze.length];

explorePaths(0, 0, maze.length, maze, visitedCells, "", allPaths);

return allPaths;

}

private static void explorePaths(int row, int col, int size, int[][] maze, boolean[][] visitedCells, String currentPath, List<String> allPaths) {

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

allPaths.add(currentPath);

return;

}

visitedCells[row][col] = true;

if (row + 1 < size && !visitedCells[row + 1][col] && maze[row + 1][col] == 1) {

explorePaths(row + 1, col, size, maze, visitedCells, currentPath + "D", allPaths);

}

if (col - 1 >= 0 && !visitedCells[row][col - 1] && maze[row][col - 1] == 1) {

explorePaths(row, col - 1, size, maze, visitedCells, currentPath + "L", allPaths);

}

if (col + 1 < size && !visitedCells[row][col + 1] && maze[row][col + 1] == 1) {

explorePaths(row, col + 1, size, maze, visitedCells, currentPath + "R", allPaths);

}

if (row - 1 >= 0 && !visitedCells[row - 1][col] && maze[row - 1][col] == 1) {

explorePaths(row - 1, col, size, maze, visitedCells, currentPath + "U", allPaths);

}

visitedCells[row][col] = 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}

};

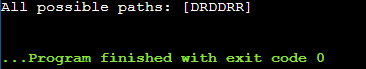
List<String> possiblePaths = findPathsInMaze(maze);

System.out.println("All possible paths: " + possiblePaths);

}

}

**Output**

****

Time Complexity :O(4^(m\*n)

Space Complexity: O(n\*n)