**12-11-2024 DSA PRACTICE PROBLEMS SET 3**

**1. Anagram**

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.

**CODE:**

import java.util.Scanner;

public class AnagramChecker {

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

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

return false;

}

int[] charCount = new int[26];

for (int i = 0; i < s1.length(); i++) {

charCount[s1.charAt(i) - 'a']++;

charCount[s2.charAt(i) - 'a']--;

}

for (int count : charCount) {

if (count != 0) {

return false;

}

}

return true;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String s1 = scanner.nextLine();

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

String s2 = scanner.nextLine();

if (areAnagrams(s1, s2)) {

System.out.println("True");

} else {

System.out.println("False");

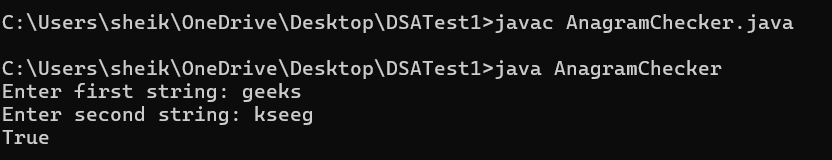
}

scanner.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**2. Rows With Max 1’s**

**CODE:**

class Sol

{

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

{

int r = 0;

int c = M-1;

int MaxVal=0;

while (r<N && c>=0){

if (Mat[r][c]==0){

r++;

}

else{

MaxVal=r;

c--;

}

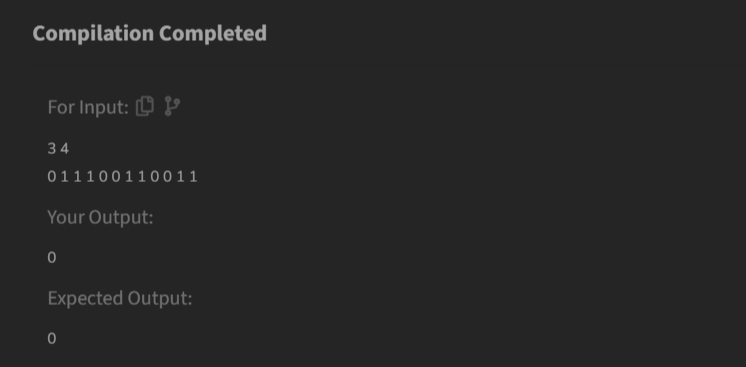
}

return MaxVal;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(M+N)**

**3. Longest Consecutive Subsequence**

**CODE:**

class Solution {

public int findLongestConseqSubseq(int[] arr) {

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

int n = arr.length;

int MaxVal=0;

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

hash.add(arr[i]);

}

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

if(!hash.contains(arr[i]-1)){

int j = arr[i];

while (hash.contains(j)){

j++;

}

if (MaxVal < j - arr[i] ){

MaxVal = j - arr[i];

}

}

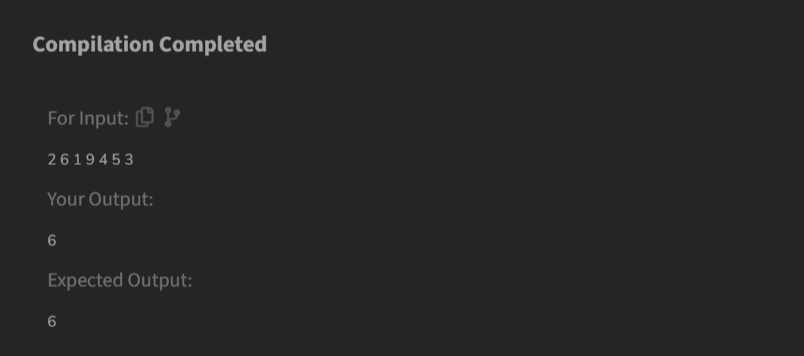
}

return MaxVal;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**4. Longest Palindromic Substring**

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

**CODE:**

import java.util.Scanner;

public class LongPalindrome {

public String longestPalindrome(String s) {

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

return s;

}

int maxLen = 1;

String maxStr = s.substring(0, 1);

s = "#" + s.replaceAll("", "#") + "#";

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

int center = 0;

int right = 0;

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

if (i < right) {

dp[i] = Math.min(right - i, dp[2 \* center - i]);

}

while (i - dp[i] - 1 >= 0 && i + dp[i] + 1 < s.length() && s.charAt(i - dp[i] - 1) == s.charAt(i + dp[i] + 1)) {

dp[i]++;

}

if (i + dp[i] > right) {

center = i;

right = i + dp[i];

}

if (dp[i] > maxLen) {

maxLen = dp[i];

maxStr = s.substring(i - dp[i], i + dp[i] + 1).replaceAll("#", "");

}

}

return maxStr;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String input = scanner.nextLine();

LongPalindrome sol = new LongPalindrome();

String result = sol.longestPalindrome(input);

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

scanner.close();

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(N)**

**5. Rat In A Maze**

**CODE:**

import java.util.ArrayList;

class Solution {

public ArrayList<String> findPath(int[][] mat) {

int n = mat.length;

ArrayList<String> ans = new ArrayList<>();

StringBuilder curr = new StringBuilder();

String st = "DLRU";

int[] r1 = {1, 0, 0, -1};

int[] c1 = {0, -1, 1, 0};

if (mat[0][0] == 1) {

findPathUtil(0, 0, n, mat, ans, curr, st, r1, c1);

}

return ans;

}

private boolean isValid(int r, int c, int n, int[][] mat) {

return (r >= 0 && c >= 0 && r < n && c < n && mat[r][c] == 1);

}

private void findPathUtil(int r, int c, int n, int[][] mat, ArrayList<String> ans, StringBuilder curr, String st, int[] r1, int[] c1) {

if (r == n - 1 && c == n - 1) {

ans.add(curr.toString());

return;

}

mat[r][c] = 0;

for (int i = 0; i < 4; i++) {

int nxtr = r + r1[i];

int nxtc = c + c1[i];

if (isValid(nxtr, nxtc, n, mat)) {

curr.append(st.charAt(i));

findPathUtil(nxtr, nxtc, n, mat, ans, curr, st, r1, c1);

curr.deleteCharAt(curr.length() - 1);

}

}

mat[r][c] = 1;

}

}

**OUTPUT:**

****

**TIME COMPLEXITY: O(4^(N^2)))**