**Bhuvaneswari P AIDS DSA Practice-3 12/11/2024**

**1.Anagram String**

public class Solutions {

static int areAnagram(String S1, String S2) {

if (S1.length() != S2.length()) {

return 0;

} else {

int count = 0;

for (int i = 0; i < S1.length(); i++) {

int a = countOccurrences(S1, S1.charAt(i));

int b = countOccurrences(S2, S1.charAt(i));

if (a == b) {

count++;

}

}

return count == S1.length() ? 1 : 0;

}

}

static int countOccurrences(String str, char ch) {

int count = 0;

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

if (str.charAt(i) == ch) {

count++;

}

}

return count;

}

public static void main(String[] args) {

String S1 = "listen";

String S2 = "silent";

System.out.println(areAnagram(S1, S2));

String S1\_3 = "hello";

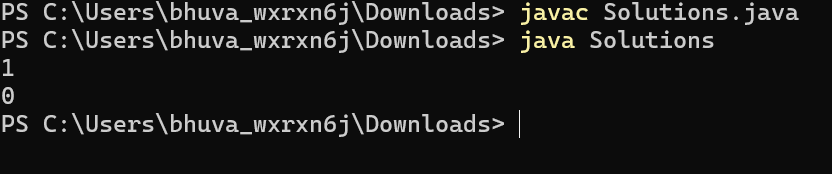
String S2\_3 = "world";

System.out.println(areAnagram(S1\_3, S2\_3));

}

}

**Output:**



Time Complexity : O(n^2)

Space Complexity : O(1)

**2. Row with max 1s'**

public class RowMatrix {

public int rowWithMax1s(int arr[][]) {

int maxi\_count = 0;

int index\_value = -1;

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

int count = countOnes(arr[i]);

if (maxi\_count < count) {

maxi\_count = count;

index\_value = i;

}

}

return index\_value;

}

private int countOnes(int[] row) {

int count = 0;

for (int value : row) {

if (value == 1) {

count++;

}

}

return count;

}

public static void main(String[] args) {

RowMatrix rm = new RowMatrix();

int arr1[][] = {

{0, 1, 1, 0},

{1, 1, 1, 0},

{0, 1, 0, 1},

{1, 1, 0, 1}

};

System.out.println(rm.rowWithMax1s(arr1));

int arr2[][] = {

{0, 0, 0},

{0, 0, 0},

{0, 0, 0}

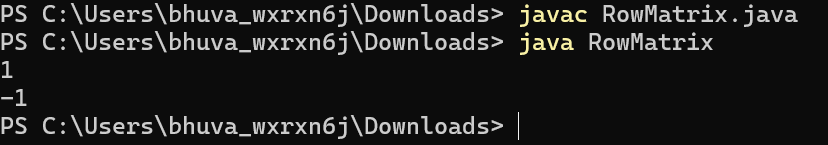
};

System.out.println(rm.rowWithMax1s(arr2));

}

}

**Output :**



Time Complexity: O(m\*n)

Space Complexity: O(1)

**3. Longest consecutive subsequence**

import java.util.\*;

public class Subsequence {

public int findLongestConseqSubseq(int[] arr) {

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

for (int num : arr) {

set.add(num);

}

int max\_val = 1;

for (int num : set) {

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

int current\_num = num;

int count = 1;

while (set.contains(current\_num + 1)) {

current\_num++;

count++;

}

max\_val = Math.max(max\_val, count);

}

}

return max\_val;

}

public static void main(String[] args) {

Subsequence seq = new Subsequence();

int[] arr1 = {2, 6, 1, 9, 4, 5, 3};

System.out.println(seq.findLongestConseqSubseq(arr1));

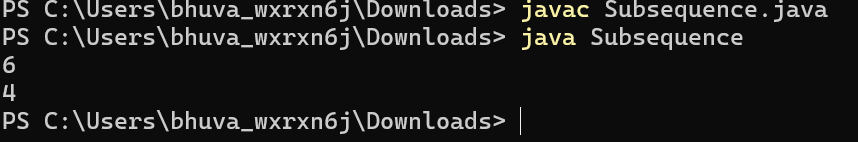
int[] arr2 = {1, 9, 3, 10, 4, 20, 2};

System.out.println(seq.findLongestConseqSubseq(arr2));

}

}

**Output:**



Time Complexity: O(n)

Space Complexity: O(n)

**4. Longest palindrome in a string**

class LongestSubstring {

static String longestPalindrome(String s) {

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

return "";

}

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 class Main {

public static void main(String[] args) {

LongestSubstring sol = new LongestSubstring();

System.out.println(sol.longestPalindrome("aaaabbaa"));

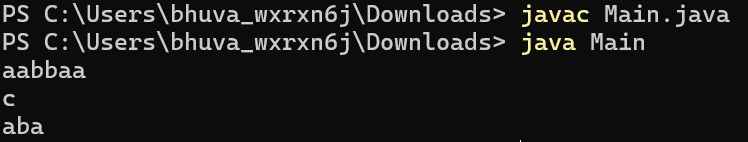
System.out.println(sol.longestPalindrome("abc"));

System.out.println(sol.longestPalindrome("abacdfgdcaba"));

}

}

**Output:**



Time Complexity: O(n^2)

Space Complexity: O(1)

**5.Rat in the Maze Problem**

import java.util.ArrayList;

class MazeSolver {

public ArrayList<String> findPaths(int[][] grid) {

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

if (grid == null || grid[0][0] == 0) {

return paths;

}

boolean[][] visited = new boolean[grid.length][grid[0].length];

StringBuilder currentPath = new StringBuilder();

explorePaths(grid, 0, 0, visited, currentPath, paths);

paths.sort(String::compareTo);

return paths;

}

private void explorePaths(int[][] grid, int row, int col, boolean[][] visited, StringBuilder currentPath, ArrayList<String> paths) {

if (row == grid.length - 1 && col == grid[0].length - 1) {

paths.add(currentPath.toString());

return;

}

visited[row][col] = true;

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

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

char[] directions = {'U', 'D', 'L', 'R'};

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

int newRow = row + rowMoves[i];

int newCol = col + colMoves[i];

if (isValidMove(grid, newRow, newCol, visited)) {

currentPath.append(directions[i]);

explorePaths(grid, newRow, newCol, visited, currentPath, paths);

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

}

}

visited[row][col] = false;

}

private boolean isValidMove(int[][] grid, int row, int col, boolean[][] visited) {

return (row >= 0 && row < grid.length && col >= 0 && col < grid[0].length && grid[row][col] == 1 && !visited[row][col]);

}

}

public class Maze {

public static void main(String[] args) {

MazeSolver solver = new MazeSolver();

int[][] grid = {

{1, 0, 0, 0},

{1, 1, 0, 1},

{1, 1, 0, 0},

{0, 1, 1, 1}

};

ArrayList<String> paths = solver.findPaths(grid);

if (paths.isEmpty()) {

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

} else {

for (String path : paths) {

System.out.println(path);

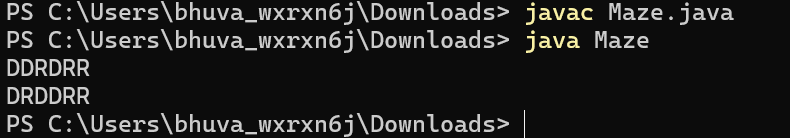
}

}

}

}

**Output:**



Time Complexity: O(4^(n^2) \* x log x)

Space Complexity: O(n^2 + x \* l)