**12.11.2024**

**1.Anagram problem:**

import java.util.\*;

public class fourteen {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

String s1 = scanner.nextLine();

String s2 = scanner.nextLine();

boolean f = true;

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

HashSet<Character> chars = new HashSet<>();

for (char c : s1.toCharArray()) {

chars.add(c);

}

for (char ch : chars) {

long count1 = s1.chars().filter(c -> c == ch).count();

long count2 = s2.chars().filter(c -> c == ch).count();

if (count1 != count2) {

f = false;

break;

}

}

} else {

f = false;

}

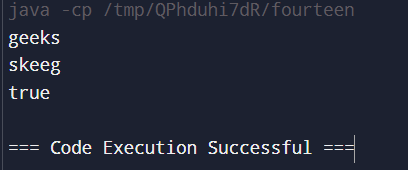
System.out.println(f);

scanner.close();

}

}

Output:



Time complexity: O(n^2)

1. **Row with max 1’s:**

import java.util.\*;

class Main {

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

int m = 0;

int res = 0;

boolean f = false;

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

int x = 0;

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

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

x++;

}

}

if (m < x) {

f = true;

m = x;

res = i;

}

}

return f ? res : -1;

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt(),m=sc.nextInt();

int[][] arr=new int[n][m];

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

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

arr[i][j]=sc.nextInt();

}

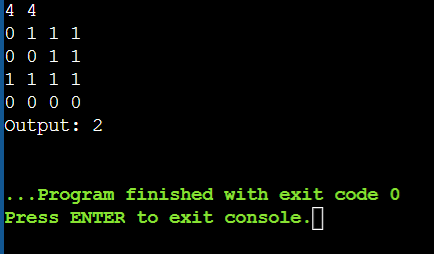
}

System.out.println("Output: "+rowWithMax1s(arr));

}

}

Output:



Time complexity: O(n\*m)

1. **Longest consecutive subsequence:**

import java.util.\*;

class Main {

public static int findLongestConseqSubseq(int[] arr) {

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

for (int i : arr) {

set.add(i);

}

List<Integer> l = new ArrayList<>();

for (int i : set) {

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

int count = 0;

while (set.contains(i)) {

count++;

i++;

}

l.add(count);

}

}

int max = 0;

for (int i : l) {

max = Math.max(max, i);

}

return max;

}

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

int n=sc.nextInt();

int[] arr=new int[n];

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

arr[i]=sc.nextInt();

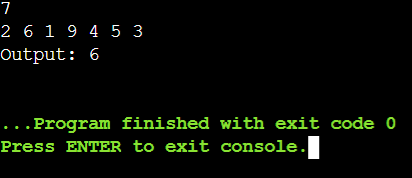
}

System.out.println("Output: "+findLongestConseqSubseq(arr));

}

}

Output:



Time complexity: O(n)

1. **Longest palindrome in a string:**

public class Main {

static String longestPalSubstr(String s) {

int n = s.length();

boolean[][] dp = new boolean[n][n];

int maxLen = 1;

int start = 0;

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

dp[i][i] = true;

for (int i = 0; i < n - 1; ++i) {

if (s.charAt(i) == s.charAt(i + 1)) {

dp[i][i + 1] = true;

start = i;

maxLen = 2;

}

}

for (int k = 3; k <= n; ++k) {

for (int i = 0; i < n - k + 1; ++i) {

int j = i + k - 1;

if (dp[i + 1][j - 1] && s.charAt(i) == s.charAt(j)) {

dp[i][j] = true;

if (k > maxLen) {

start = i;

maxLen = k;

}

}

}

}

return s.substring(start, start + maxLen);

}

public static void main(String[] args) {

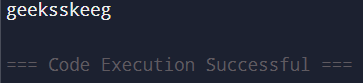
String s = "forgeeksskeegfor";

System.out.println(longestPalSubstr(s));

}

}

Output:



Time complexity: O(n^2)

1. **Rat in a maze problem:**

import java.util.ArrayList;

import java.util.List;

public class Main {

static String direction = "DLRU";

static int[] dr = { 1, 0, 0, -1 };

static int[] dc = { 0, -1, 1, 0 };

static boolean isValid(int row, int col, int n,

int[][] maze)

{

return row >= 0 && col >= 0 && row < n && col < n

&& maze[row][col] == 1;

}

static void findPath(int row, int col, int[][] maze,

int n, ArrayList<String> ans,

StringBuilder currentPath)

{

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

ans.add(currentPath.toString());

return;

}

maze[row][col] = 0;

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

int nextrow = row + dr[i];

int nextcol = col + dc[i];

if (isValid(nextrow, nextcol, n, maze)) {

currentPath.append(direction.charAt(i));

findPath(nextrow, nextcol, maze, n, ans,

currentPath);

currentPath.deleteCharAt(

currentPath.length() - 1);

}

}

maze[row][col] = 1;

}

public static void main(String[] args)

{

int[][] maze = { { 1, 0, 0, 0 },

{ 1, 1, 0, 1 },

{ 1, 1, 0, 0 },

{ 0, 1, 1, 1 } };

int n = maze.length;

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

StringBuilder currentPath = new StringBuilder();

if (maze[0][0] != 0 && maze[n - 1][n - 1] != 0) {

findPath(0, 0, maze, n, result, currentPath);

}

if (result.size() == 0)

System.out.println(-1);

else

for (String path : result)

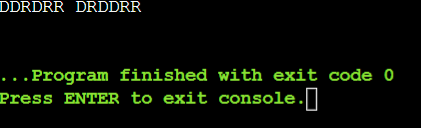
System.out.print(path + " ");

System.out.println();

}

}

Output:



Time complexity: O(3^(m\*n))