Day2-12/11/24

Geeksforgeeks

1.Anagrams

class Solution {

// Function is to check whether two strings are anagram of each other or not.

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

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

return false;

}

int [] arr=new int[26];

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

arr[i -'a']++;

}

for(char i:s2.toCharArray()){

arr[i -'a']--;

}

for(int j:arr){

if(j!=0){

return false;

}

}

return true;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(n)

2. Row with max 1s

class Sol {

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

int maxRowIndex = -1;

int maxOnes = 0;

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

int onesCount = 0;

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

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

onesCount++;

}

}

if (onesCount > maxOnes) {

maxOnes = onesCount;

maxRowIndex = i;

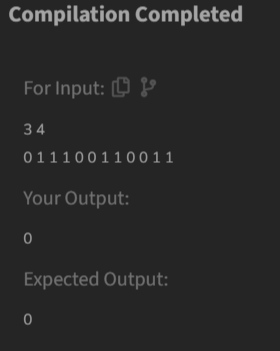
}

}

return maxRowIndex;

}

}



Time complexity: O(MXN)

3. Longest consecutive subsequence

class Solution {

public int findLongestConseqSubseq(int[] arr) {

if (arr.length == 0) {

return 0;

}

Arrays.sort(arr);

int maxLength = 1;

int currentLength = 1;

for (int i = 1; i < arr.length; i++) {

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

continue;

}

if (arr[i] == arr[i - 1] + 1) {

currentLength++;

} else {

maxLength = Math.max(maxLength, currentLength);

currentLength = 1;

}

}

maxLength = Math.max(maxLength, currentLength);

return maxLength;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(NlogN)

4. Longest Palindrome in a String

import java.io.\*;

import java.util.\*;

class GFG {

public static void main(String args[]) throws IOException {

BufferedReader read = new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(read.readLine());

while (t-- > 0) {

String S = read.readLine();

Solution ob = new Solution();

System.out.println(ob.longestPalindrome(S));

}

}

}

// } Driver Code Ends

// User function Template for Java

class Solution {

// Static method to find the longest palindromic substring

static String longestPalindrome(String s) {

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

return "";

}

int start = 0, maxLength = 1;

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 > maxLength) {

maxLength = len;

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

}

}

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

}

static int expandAroundCenter(String s, int left, int right) {

while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {

left--;

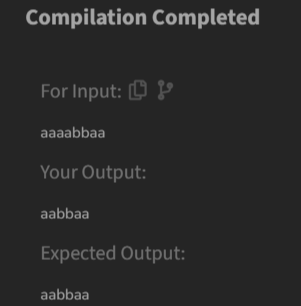
right++;

}

return right - left - 1;

}

}



Time complexity: O(n)

5. Rat in a Maze problem-1

import java.util.\*;

class Rat {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int t = sc.nextInt();

while (t-- > 0) {

int n = sc.nextInt();

int[][] a = new int[n][n];

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

for (int j = 0; j < n; j++) a[i][j] = sc.nextInt();

Solution obj = new Solution();

ArrayList<String> res = obj.findPath(a);

Collections.sort(res);

if (res.size() > 0) {

for (int i = 0; i < res.size(); i++) System.out.print(res.get(i) + " ");

System.out.println();

} else {

System.out.println(-1);

}

System.out.println("~");

}

}

}

// } Driver Code Ends

class Solution {

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

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

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

result.add("-1");

return result;

}

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

solve(mat, 0, 0, "", visited, result);

if (result.isEmpty()) {

result.add("-1");

}

return result;

}

private void solve(int[][] mat, int x, int y, String path, boolean[][] visited, ArrayList<String> result) {

if (x < 0 || x >= mat.length || y < 0 || y >= mat[0].length || mat[x][y] == 0 || visited[x][y]) {

return;

}

if (x == mat.length - 1 && y == mat[0].length - 1) {

result.add(path);

return;

}

visited[x][y] = true;

solve(mat, x + 1, y, path + "D", visited, result);

solve(mat, x, y - 1, path + "L", visited, result);

solve(mat, x, y + 1, path + "R", visited, result);

solve(mat, x - 1, y, path + "U", visited, result);

visited[x][y] = false;

}

}

A screenshot of a phone

Description automatically generated

Time complexity: O(3^n^2)

Backtracking method