12.11.2024

1. Anagram program :

CODE:

import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class AnagramCheck {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input the two strings

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

String s = scanner.nextLine();

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

String t = scanner.nextLine();

System.out.println isAnagram(s, t));

scanner.close();

}

public static boolean isAnagram(String s, String t) {

if (s.length() != t.length()) return false;

Map<Character, Integer> map1 = new HashMap<>();

Map<Character, Integer> map2 = new HashMap<>();

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

map1.put(c, map1.getOrDefault(c, 0) + 1);

}

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

map2.put(c, map2.getOrDefault(c, 0) + 1);

}

return map1.equals(map2);

}

}

**OUTPUT** :

Enter first string: geeks

Enter second string: kseeg

True

**Time Complexity** : O(N)

**Space Complexity** : O(n)

2. Row with Max One’s

CODE:

import java.util.Scanner;

class Solution {

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

int n = arr.length;

int m = arr[0].length;

int maxOnes = 0;

int index = -1;

int count;

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

count = 0;

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

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

count++;

}

}

if (maxOnes < count) {

maxOnes = count;

index = i;

}

}

return index;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

int n = scanner.nextInt();

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

int m = scanner.nextInt();

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

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

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

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

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

}

}

Solution solution = new Solution();

int result = solution.rowWithMax1s(arr);

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:

Enter number of rows: 4 Enter number of columns: 5

Enter the matrix elements (0 or 1): 0 1 1 1 0 0 0 1 1 1 1 1 1 1 1 0 0 0 1 0

Row with the maximum number of 1s: 2

Time Complexity : O(n\*m)

Space Complexity : O(1)

3. Longest consequtive subsequence

CODE:

class Solution {

// Function to return length of longest subsequence of consecutive integers.

public int findLongestConseqSubseq(int[] arr) {

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

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

set.add(arr[i]);

}

int ma = 0;

int cnt;

int n ;

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

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

n = arr[i];

cnt = 0;

while(set.contains(n)){

n+=1 ;

cnt+=1;

}

ma = Math.max(ma, cnt);

}

}

return ma;

}

}

Time Complexity : O(n)

Space Complexity: O(n)

5 . Longest Palindromic Substring

**CODE** :

import java.util.Scanner;

public class Solution {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

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

String s = scanner.nextLine();

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

scanner.close();

}

public static String longestPalindrome(String s) {

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

return s;

}

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

for (int i = 0; i < s.length() - 1; i++) {

String odd = expandFromCenter(s, i, i);

String even = expandFromCenter(s, i, i + 1);

if (odd.length() > maxStr.length()) {

maxStr = odd;

}

if (even.length() > maxStr.length()) {

maxStr = even;

}

}

return maxStr;

}

private static String expandFromCenter(String s, int left, int right) {

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

left--;

right++;

}

return s.substring(left + 1, right);

}

}

**OUTPUT**:

Enter a string: Geeks

Longest Palindromic Substring: ee

Time Complexity : O(n\*2)

Space Complexity : O(1)

6. Rat in a Maze

CODE :

import java.util.ArrayList;

import java.util.List;

class Solution {

public List<String> findPath(int[][] m) {

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

if (m[0][0] == 0) return paths; // If starting cell is blocked, return empty list

boolean[][] visited = new boolean[m.length][m.length];

findPaths(0, 0, m.length, m, visited, "", paths);

return paths;

}

private void findPaths(int i, int j, int n, int[][] m, boolean[][] visited, String path, List<String> paths) {

// If reached the bottom-right corner, add the path to result list

if (i == n - 1 && j == n - 1) {

paths.add(path);

return;

}

// Mark the cell as visited

visited[i][j] = true;

// Move Down

if (i + 1 < n && !visited[i + 1][j] && m[i + 1][j] == 1) {

findPaths(i + 1, j, n, m, visited, path + "D", paths);

}

// Move Left

if (j - 1 >= 0 && !visited[i][j - 1] && m[i][j - 1] == 1) {

findPaths(i, j - 1, n, m, visited, path + "L", paths);

}

// Move Right

if (j + 1 < n && !visited[i][j + 1] && m[i][j + 1] == 1) {

findPaths(i, j + 1, n, m, visited, path + "R", paths);

}

// Move Up

if (i - 1 >= 0 && !visited[i - 1][j] && m[i - 1][j] == 1) {

findPaths(i - 1, j, n, m, visited, path + "U", paths);

}

// Backtrack and unmark the cell as visited

visited[i][j] = false;

}

}

public class Main {

public static void main(String[] args) {

Solution solution = new Solution();

int[][] maze = {

{1, 0, 0, 0},

{1, 1, 0, 1},

{0, 1, 0, 0},

{1, 1, 1, 1}

};

List<String> paths = solution.findPath(maze);

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

}

}

Output:

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

Space Complexity: O(n\*n)