12/11/2024

1. anagram program:

public class AnagramProgram {

public boolean isAnagram(String S, String T) {

char A[] = S.toCharArray();

char B[] = T.toCharArray();

Arrays.sort(A);

Arrays.sort(B);

boolean F = Arrays.equals(A, B);

return F;

}

Output:

first string: anagram

second string: nagaram

result : true

first string: rat

second string: car

result : false  
  
time complexity: O(Nlog N)

space complexity : O(N)

1. row with max 1s'

import java.util.Arrays;

class Main {

public static void main(String[] args) {

int[][] mat = {{0, 1}, {1, 0}};

System.out.println(Arrays.toString(findMaxOnesRow(mat)));

}

static int[] findMaxOnesRow(int[][] matrix) {

int maxCount = 0;

int[] result = new int[2];

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

int count = 0;

for (int num : matrix[i]) {

if (num == 1) {

count++;

}

}

if (count > maxCount) {

maxCount = count;

result[0] = i;

result[1] = count;

}

}

return result;

}

}

OUTPUT : [1, 1]

Time and space complexity :

O(n\*m) and O(n)

1. Longest consequtive subsequence

import java.util.HashSet;

class Main1 {

public static void main(String[] args) {

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

System.out.println("Length is " +

findLongestConseqSubseq(arr));

}

static int findLongestConseqSubseq(int arr[]) {

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

int maxLength = 0;

for (int num : arr) {

set.add(num);

}

for (int num : arr) {

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

int currentNum = num;

int count = 1;

while (set.contains(currentNum + 1)) {

currentNum++;

count++;

}

maxLength = Math.max(maxLength, count);

}

}

return maxLength;

}

}  
  
output   
length 4

Time complexity : O(N)

Space complexity : O(N)

4.longest palindrome in a string

public class Main1 {

public static void main(String[] args) {

String str1 = "madam";

System.out.println(longestPalSubstr(str1));

String str2 = "level";

System.out.println(longestPalSubstr(str2));

}

static String longestPalSubstr(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);

}

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;

}

}  
  
  
output:

Madan

Level  
  
time complexity : O(N^2)

Space complexity: O(1)