SARAVANAN BS-AI&DS-DSA-PRACTICE-3

1. anagram program:

class Solution {

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

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

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

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

int n1 = s1.length();

int n2 = s2.length();

if(n1 != n2){

return false;

}

for(int i = 0; i < n1; i++){

char c1 = s1.charAt(i);

char c2 = s2.charAt(i);

map1.put(c1, map1.getOrDefault(c1, 0) + 1);

map2.put(c2, map2.getOrDefault(c2, 0) + 1);

}

for(Map.Entry<Character, Integer> entry: map1.entrySet()){

char key = entry.getKey();

int val = entry.getValue();

if(!map2.containsKey(key) || !map2.get(key).equals(val)){

return false;

}

}

return true;

}

}

Result:

Input: s1 = "geeks", s2 = "kseeg"  
Output: true

Input: s1 = "allergy", s2 = "allergic"  
Output: false

Input: s1 = "g", s2 = "g"  
Output: true

Time complexity: O(N)

2. row with max 1s'

class Solution {

public int countOnes(List<Integer> arr, int col) {

int low = 0, high = col - 1;

// Finding lower bound for 1

while (low <= high) {

int mid = low + ((high - low) >> 1);

if (arr.get(mid) == 1) {

high = mid - 1;

} else {

low = mid + 1;

}

}

return low;

}

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

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

for (int[] row : a) {

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

for (int num : row) {

rowList.add(num); // Convert each int to Integer and add to rowList

}

arr.add(rowList); // Add rowList to main list

}

int n = arr.size();

int m = arr.get(0).size();

int maxCount = 0, row = -1;

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

int count = m - countOnes(arr.get(i), m);

if (count > maxCount) {

maxCount = count;

row = i;

}

}

return row;

}

}  
Result:

Input: arr[][] = [[0, 1, 1, 1],  
 [0, 0, 1, 1],  
 [1, 1, 1, 1],  
 [0, 0, 0, 0]]  
Output: 2

Input: arr[][] = [[0, 0],   
 [1, 1]]  
Output: 1

Time complexity: O(n x m)

3. Longest consequtive subsequence

public int findLongestConseqSubseq(int[] arr) {

int maxi = 0;

int n = arr.length;

int l = 0;

int r = 0;

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

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

hash.add(arr[i]);

}

Integer[] arr1 = hash.toArray(new Integer[0]);

Arrays.sort(arr1);

while (r < arr1.length) {

if (arr1[r] - arr1[l] != r - l) {

l++;

}

if (arr1[r] - arr1[l] == r - l) {

maxi = Math.max(maxi, r - l);

}

r++;

}

return maxi + 1;

}

Result:

Input: arr[] = [2, 6, 1, 9, 4, 5, 3]  
Output: 6

Input: arr[] = [1, 9, 3, 10, 4, 20, 2]  
Output: 4

Input: arr[] = [15, 13, 12, 14, 11, 10, 9]  
Output: 7

Time complexity: O(n log(n))

4. longest palindrome in a string

class Solution {

// Static method to find the longest palindromic substring

static String longestPalindrome(String str) {

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

return "";

}

int start = 0, end = 0;

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

int len1 = expandAroundCenter(str, i, i);

int len2 = expandAroundCenter(str, i, i + 1);

int len = Math.max(len1, len2);

if (len > end - start) {

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

end = i + len / 2;

}

}

return str.substring(start, end + 1);

}

public static int expandAroundCenter(String str, int left, int right) {

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

left--;

right++;

}

return right - left - 1;

}

}

Result:

Input: s = "aaaabbaa"  
Output: "aabbaa"

Input: s = "abc"  
Output: "a"

Input: s = "abacdfgdcaba"   
Output: "aba"

Time Complexity: O(n2)

5. rat in a maze problem:

class Solution {

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

StringBuilder sb = new StringBuilder();

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

find(mat, sb, arr, 0, 0);

return arr;

}

public void find(int[][] mat, StringBuilder str, List<String> arr, int i, int j){

if(i < 0 || j < 0 || j > mat.length-1 || i > mat.length-1|| mat[i][j] == 0){

return;

}

if(i == mat.length-1 && j == mat.length-1){

arr.add(str.toString());

}

mat[i][j] = 0;

str.append("U");

find(mat, str, arr, i - 1, j);

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

str.append("D");

find(mat, str, arr, i + 1, j);

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

str.append("L");

find(mat, str, arr, i, j - 1);

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

str.append("R");

find(mat, str, arr, i , j + 1);

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

mat[i][j] = 1;

}

}

Result:  
Input: mat[][] = [[1, 0, 0, 0],  
 [1, 1, 0, 1],   
 [1, 1, 0, 0],  
 [0, 1, 1, 1]]  
Output: DDRDRR DRDDRR

Input: mat[][] = [[1, 0],  
 [1, 0]]  
Output: -1

Time COmplexity: O(4N\*N)