**Valid Anagram:**

1)Given two strings **s1** and **s2** consisting of **lowercase**characters, the task is to check whether the two given strings are **anagrams**of each other or not. An anagram of a string is another string that contains the same characters, only the order of characters can be different.

**Examples:**

***Input:*** *s1 = “geeks”  s2 = “kseeg”****Output:*** *true****Explanation:*** *Both the string have same characters with same frequency. So, they are anagrams.*

***Input:*** *s1 = “allergy”  s2 = “allergic”****Output:*** *false****Explanation:*** *Characters in both the strings are not same. s1 has extra character* ***‘y’*** *and s2 has extra characters ‘i’ and ‘c’, so they are not anagrams.*

Program:

import java.util.HashMap;

public class ValidAnagram {

public static void main(String[] args){

ValidAnagram obj =new ValidAnagram();

String s1 = "geeks";

String s2 = "kseeg";

boolean res= obj.anagram(s1,s2);

System.out.println(res);

}

public boolean anagram(String s1,String s2){

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

return false;

}

HashMap<Character,Integer> map=new HashMap<>();

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

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

}

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

if(!map.containsKey(c)){

return false;

}

map.put(c,map.get(c)-1);

}

for(int val:map.values()){

if(val!=0){

return false;

}

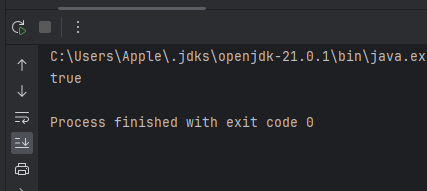
}

return true;

}

}

Output:



Time Complexity:O(N);

Space Complexity:O(1);

**2)MaxSum1:**

Given a **binary**2D array, where each row is **sorted**. Find the row with the maximum number of 1s.

**Examples:**

***Input matrix :*** *0 1 1 1  
                        0 0 1 1  
                        1 1 1 1   
                        0 0 0 0****Output:*** *2****Explanation:*** *Row = 2 has maximum number of 1s, that is 4.*

Program:

public class RowWithmaxOne {

public static void main(String[] args){

RowWithmaxOne obj = new RowWithmaxOne();

// Example matrix

int[][] matrix = {

{1, 0, 1},

{0, 1, 1},

{1, 1, 1}

};

System.out.println(obj.maxone(matrix));

}

public int maxone(int[][] matrix){

int n=matrix.length;

int m=matrix[0].length;

if(n==0){

return 0;

}

int maxsum=0;

int val=-1;

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

int sum=0;

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

sum = sum + matrix[i][j];

}

if(sum>maxsum){

maxsum=sum;

val=i;

}

}

return val;

}

}

Or Another Approach

public int maxone(int[][] matrix){

int val=-1;

int r=matrix.length;

int c=matrix[0].length;

int row=0;

int col=c-1;

while(row<r && col>=0){

if(matrix[row][col]==0){

row++;

}

col--;

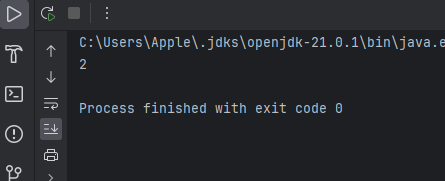
val=row;

}

return val;

}

output:



Time Complexity:O(N);

Space Complexity:O(1);

**3)Longest Consecutive Subsequence** :

Given an array of integers, find the length of the **longest sub-sequence** such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

**Examples**:

***Input****: arr[] = {1, 9, 3, 10, 4, 20, 2}****Output****: 4****Explanation:*** *The subsequence 1, 3, 4, 2 is the longest subsequence of consecutive elements*

Program:

public class LongestSubsequence {

public static void main(String[] args){

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

LongestSubsequence obj= new LongestSubsequence();

System.out.println(obj.longest(arr));

}

public int longest(int[] arr){

Arrays.sort(arr);

int max=Integer.MIN\_VALUE;

int count=1;

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

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

count++;

}

else{

count=1;

}

max=Math.max(max,count);

}

return max;

}

}

TimeComplexity:O(n log\*n)

SpaceComplexity:O(1);

Another Approach using Hashset:

public int longest(int[] arr){

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

int max=0;

for(int num:arr){

set.add(num);

}

for(int num:arr){

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

int curr=num;

int len=1;

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

curr++;

len++;

}

max=Math.max(max,len);

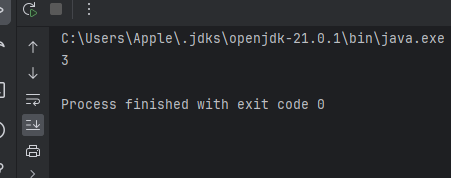
}

}

return max;

}

Output:



TimeComplexity:O(n )

SpaceComplexity:O(n);

**4)Longest Palindromic Substring**

Given a string str, the task is to find the longest substring which is a palindrome. If there are multiple answers, then return the first appearing substring.

Input: str = “forgeeksskeegfor”

Output: “geeksskeeg”

Explanation: There are several possible palindromic substrings like “kssk”, “ss”, “eeksskee” etc. But the substring “geeksskeeg” is the longest among all.

import java.util.Scanner;

public class longestPalindromeSubString {

public static void main(String[] args){

Scanner sc =new Scanner(System.in);

String s=sc.nextLine();

int start=0;

int end=0;

longestPalindromeSubString obj=new longestPalindromeSubString();

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

int l1=obj.subString(s,i,i);

int l2=obj.subString(s,i,i+1);

int max=Math.max(l1,l2);

if(max>(end-start)){

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

end=i+max/2;

}

}

System.out.println(s.substring(start,end+1));

}

public int subString(String str,int start,int end){

while(start>=0 && end <str.length() && str.charAt(start)==str.charAt(end)){

start--;

end++;

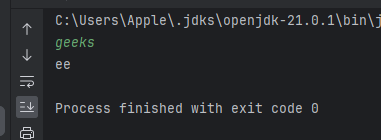
}

return end-start-1;

}

}

Output:



**TimeComplexity** :O(4^(m\*n)) **Space Complexity**:O(m\*n)

**5)Rat in a Maze:**

Program:

import java.util.ArrayList;

public class RatinaMaze {

private static void solve(int i, int j, int a[][], int n, ArrayList < String > ans, String move,

int vis[][]) {

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

ans.add(move);

return;

}

if (i + 1 < n && vis[i + 1][j] == 0 && a[i + 1][j] == 1) {

vis[i][j] = 1;

solve(i + 1, j, a, n, ans, move + 'D', vis);

vis[i][j] = 0;

}

if (j - 1 >= 0 && vis[i][j - 1] == 0 && a[i][j - 1] == 1) {

vis[i][j] = 1;

solve(i, j - 1, a, n, ans, move + 'L', vis);

vis[i][j] = 0;

}

if (j + 1 < n && vis[i][j + 1] == 0 && a[i][j + 1] == 1) {

vis[i][j] = 1;

solve(i, j + 1, a, n, ans, move + 'R', vis);

vis[i][j] = 0;

}

if (i - 1 >= 0 && vis[i - 1][j] == 0 && a[i - 1][j] == 1) {

vis[i][j] = 1;

solve(i - 1, j, a, n, ans, move + 'U', vis);

vis[i][j] = 0;

}

}

public static ArrayList < String > findPath(int[][] m, int n) {

int vis[][] = new int[n][n];

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

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

vis[i][j] = 0;

}

}

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

if (m[0][0] == 1) solve(0, 0, m, n, ans, "", vis);

return ans;

}

public static void main(String[] args) {

int n = 4;

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

RatinaMaze obj = new RatinaMaze();

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

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);

}

}

}

Output:

