

DSA PRACTICE PROGRAM -DAY3

Name: Sundara Vinayagam V

Dept: B.Tech-CSBS 3rd year

Anagram Program:

Given two strings **s1** and **s2** consisting of lowercase characters. The task is to check whether two given strings are an anagram 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. For example, act and tac are an anagram of each other. Strings **s1** and **s2** can only contain lowercase alphabets.

Note: You can assume both the strings s1 & s2 are **non-empty**.

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, so they are not anagrams.

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

Output: true

Explanation: Character in both the strings are same, so they are anagrams.

Constraints:

$1 \leq s1.size(), s2.size() \leq 10^5$

Code:

```
class Solution {  
    public static boolean areAnagrams(String s1, String s2) {  
        char[] s1a=s1.toCharArray();  
        char[] s2a=s2.toCharArray();  
        Arrays.sort(s1a);  
        Arrays.sort(s2a);  
        return Arrays.equals(s1a,s2a);  
    }  
}
```

Output:

```

PS F:\cse\java> javac .\AnagramCheck.java
PS F:\cse\java> java .\AnagramCheck.java
Enter the first string (s1): "geeks"
Enter the second string (s2): "forgeeks"
false
PS F:\cse\java> |

```

Row with minimum number of 1's

Given a 2D **binary matrix**(1-based indexed) **mat** of dimensions **nxm** , determine the **row** that contains the **minimum number of 1's**.

Note: The matrix contains only **1's** and **0's**. Also, if two or more rows contain the **minimum number of 1's**, the answer is the **lowest** of those **indices**.

Examples :

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

Output: 2

Explanation: Rows 2 and 3 contain the minimum number of 1's (2 each). Since, row 2 is less than row 3. Thus, the answer is 2.

Input: mat = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

Output: 1

Explanation: All the rows contain the same number of 1's (0 each). Among them, index 1 is the smallest, so the answer is 1.

Constraints:

$1 \leq n, m \leq 1000$

$0 \leq \text{mat}[i][j] \leq 1$

Solution:

```

class Solution {
    public int rowWithMax1s(int arr[][]){
        int[] a=new int[arr.length];
        // code here
        for(int i=0;i<arr.length;i++){
            int c=0;
            for(int j=0;j<arr[i].length;j++){
                if(arr[i][j]==1){
                    c+=1;
                }
            }
            a[i]=c;
        }
        int ans=0;
        int index=0;
        for(int i=0;i<a.length;i++){
            if (a[i]>ans){
                ans=a[i];
                index=i;
            }
        }
        return index;
    }
}

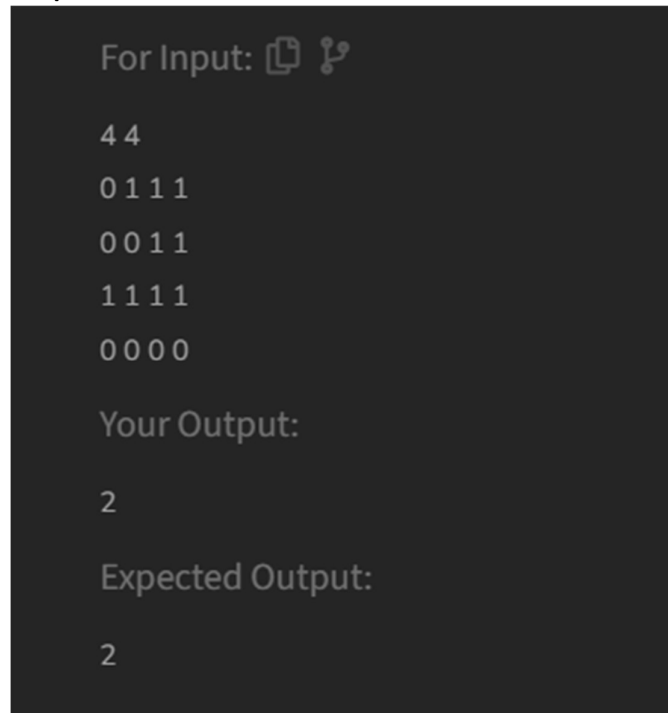
```

```

    }
}
if(ans<1) return -1;
return index;
}
}

```

Output:



Longest consecutive subsequence

Given an array **arr** of non-negative 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[] = [2, 6, 1, 9, 4, 5, 3]

Output: 6

Explanation: The consecutive numbers here are 1, 2, 3, 4, 5, 6. These 6 numbers form the longest consecutive subsequence.

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

Output: 4

Explanation: 1, 2, 3, 4 is the longest consecutive subsequence.

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

Output: 7

Explanation: The longest consecutive subsequence is 9, 10, 11, 12, 13, 14, 15, which has a length of 7.

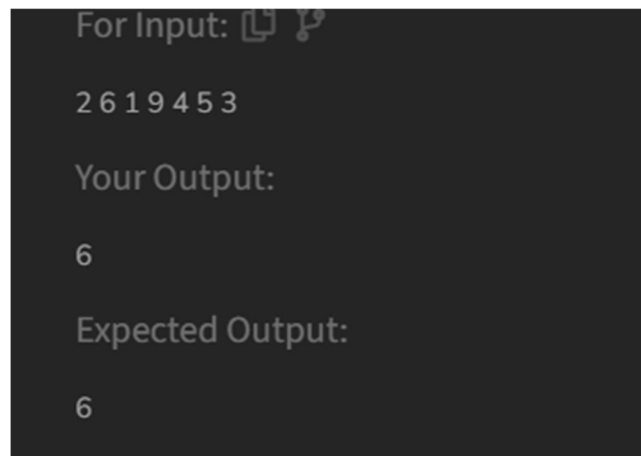
Constraints:

$1 \leq \text{arr.size()} \leq 10^5$

$0 \leq \text{arr}[i] \leq 10^5$

Solution:

```
class Solution {  
    public int findLongestConseqSubseq(int[] arr) {  
        Arrays.sort(arr);  
        int c=1;  
        for(int i=0;i<arr.length-1;i++){  
            if(arr[i+1]-arr[i]==1){  
                c+=1;  
            }  
        }  
        return c;  
    }  
}
```

Output:**Longest Palindromic Substring****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.

Input: str = "Geeks"

Output: "ee"

Input: str = "abc"

Output: "a"

Input: str = ""

Output: ""

Solution:

```
import java.util.Scanner;
```

```
public class LongestPalindromicSubstring {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter the string: ");
        String str = scanner.nextLine();

        int n = str.length();
        if (n == 0) {
            System.out.println("");
            scanner.close();
            return;
        }

        int start = 0;
        int maxLength = 1;

        for (int i = 0; i < n; i++){
            int left = i;
            int right = i;
            while (left >= 0 && right < n && str.charAt(left) == str.charAt(right)) {
                if (right - left + 1 > maxLength) {
                    start = left;
                    maxLength = right - left + 1;
                }
                left--;
                right++;
            }
            left = i;
            right = i + 1;
            while (left >= 0 && right < n && str.charAt(left) == str.charAt(right)) {
                if (right - left + 1 > maxLength) {
                    start = left;
                    maxLength = right - left + 1;
                }
                left--;
                right++;
            }
        }

        System.out.println("Longest Palindromic Substring: " + str.substring(start, start + maxLength));
        scanner.close();
    }
}
```

Output:

```
PS F:\cse\java> javac .\LongestPalindromicSubstring.java
PS F:\cse\java> java .\LongestPalindromicSubstring.java
Enter the string: "leetcode"
Longest Palindromic Substring: ee
PS F:\cse\java> java .\LongestPalindromicSubstring.java
Enter the string: "codingninjas"
Longest Palindromic Substring: ingni
PS F:\cse\java> |
```

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

Next Greater Element:

Given an array, print the Next Greater Element (NGE) for every element.

Note: The Next greater Element for an element x is the first greater element on the right side of x in the array. Elements for which no greater element exist, consider the next greater element as -1.

Input: arr[] = [4 , 5 , 2 , 25]

Output: 4

5

2 ->

5 -> 25 -> 25

25 -> -1

Explanation: Except 25 every element has an element greater than them present on the right side

Input: arr[] = [13 , 7 , 6 , 12]

Output: 13 ->

7 -1 -> 12

6

12 -> 12 -> -1

Explanation: 13 and 12 don't have any element greater than them present on the right side

Solution:

```
import java.util.*;
public class NGE {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter Stack Length");
        int n = sc.nextInt();
        System.out.println("Enter array elements");
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }

        Stack<Integer> stack = new Stack<>();
        int[] result = new int[n];

        for (int i = n - 1; i >= 0; i--) {
```

```

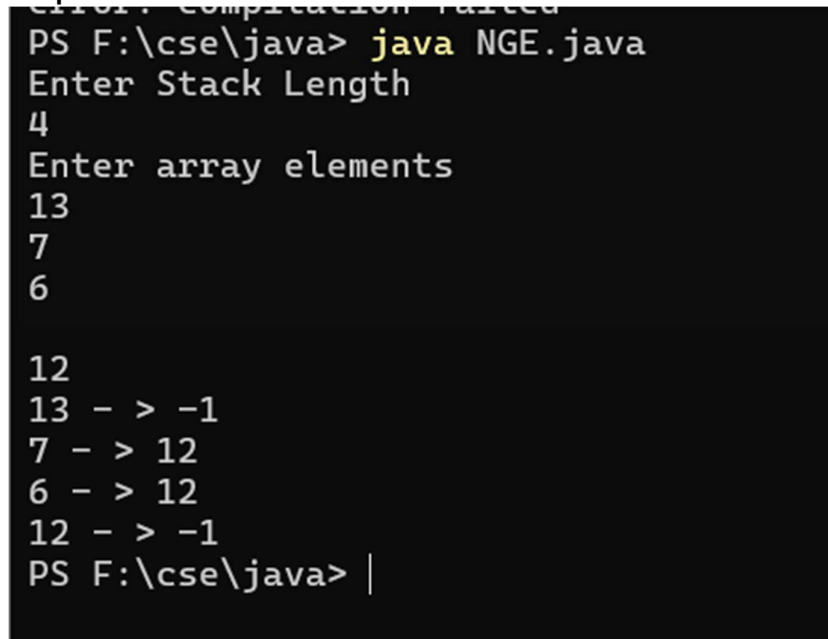
        while (!stack.isEmpty() && stack.peek() <= arr[i]) {
            stack.pop();
        }

        if (stack.isEmpty()) {
            result[i] = -1;
        } else {
            result[i] = stack.peek();
        }
        stack.push(arr[i]);
    }

    for (int i=0;i<n;i++) {
        System.out.println(arr[i] + " - > " + result[i]);
    }
}
}

```

Output:



```

PS F:\cse\java> java NGE.java
Enter Stack Length
4
Enter array elements
13
7
6

12
13 - > -1
7 - > 12
6 - > 12
12 - > -1
PS F:\cse\java> |

```

Time Complexity: $O(n)$

Space Complexity: $O(n)$

Rat In a Maze problem:

Consider a rat placed at **(0, 0)** in a square matrix **mat** of order **$n \times n$** . It has to reach the destination at **($n - 1, n - 1$)**. Find all possible paths that the rat can take to reach from source to destination. The directions in which the rat can move are '**U**'(**up**), '**D**'(**down**), '**L**' (**left**), '**R**' (**right**). Value 0 at a cell in the matrix represents that it is blocked and rat cannot move to it while value 1 at a cell in the matrix represents that rat can be travel through it.

Note: In a path, no cell can be visited more than one time. If the source cell is 0, the rat cannot move to any other cell. In case of no path, return an empty list. The driver will output "**-1**" automatically.

Examples:

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

[1, 1, 0, 1],

[1, 1, 0, 0],

[0, 1, 1, 1]]

Output: DDRDRR DRDDRR

Explanation: The rat can reach the destination at (3, 3) from (0, 0) by two paths - DRDDRR and DDRDRR, when printed in sorted order we get DDRDRR DRDDRR.

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

[1, 0]]

Output: -1

Explanation: No path exists and destination cell is blocked.

Expected Time Complexity: $O(3^{n^2})$

Expected Auxiliary Space: $O(l * x)$

Here l = length of the path, x = number of paths.

Constraints:

$2 \leq n \leq 5$

$0 \leq \text{mat}[i][j] \leq 1$

Code :

```
import java.util.*;
```

```
public class RatMaze {
    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();

            ArrayList<String> res = findPath(a);
            Collections.sort(res);
            if (res.size() > 0) {
                for (String path : res) System.out.print(path + " ");
                System.out.println();
            } else {
                System.out.println(-1);
            }
        }
        sc.close();
    }
}
```

```
public static ArrayList<String> findPath(int[][] mat) {
    ArrayList<String> res = new ArrayList<>();
```



```

int n = mat.length;
if (mat[0][0] == 0 || mat[n - 1][n - 1] == 0) {
    return res;
}
findPaths(mat, 0, 0, "", res, n);
return res;
}

private static void findPaths(int[][] mat, int x, int y, String path, ArrayList<String> res, int n) {
    if (x == n - 1 && y == n - 1) {
        res.add(path);
        return;
    }
    mat[x][y] = 0;
    if (isSafe(mat, x + 1, y, n)) {
        findPaths(mat, x + 1, y, path + "D", res, n);
    }
    if (isSafe(mat, x, y - 1, n)) {
        findPaths(mat, x, y - 1, path + "L", res, n);
    }
    if (isSafe(mat, x, y + 1, n)) {
        findPaths(mat, x, y + 1, path + "R", res, n);
    }
    if (isSafe(mat, x - 1, y, n)) {
        findPaths(mat, x - 1, y, path + "U", res, n);
    }
    mat[x][y] = 1;
}

private static boolean isSafe(int[][] mat, int x, int y, int n) {
    return x >= 0 && y >= 0 && x < n && y < n && mat[x][y] == 1;
}
}

```

Output

Input: 4

```

[[1, 0, 0, 0],
 [1, 1, 0, 1],
 [1, 1, 0, 0],
 [0, 1, 1, 1]]

```

Output: DDRDRR DRDDRR

Time Complexity: $O(3^{n^2})$

Space: $O(l * x)$