

# Vasanthan\_T\_DSA\_Practice-3

12.11.2024

## 1. 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.

**Code:**

```
import java.util.*;

class Anagram{

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

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

        for(char i:s1.toCharArray()){
            hp.put(i,hp.getOrDefault(i,0)+1);
        }

        for(char i:s2.toCharArray()){
            if(!hp.containsKey(i)) return false;
        }
    }
}
```

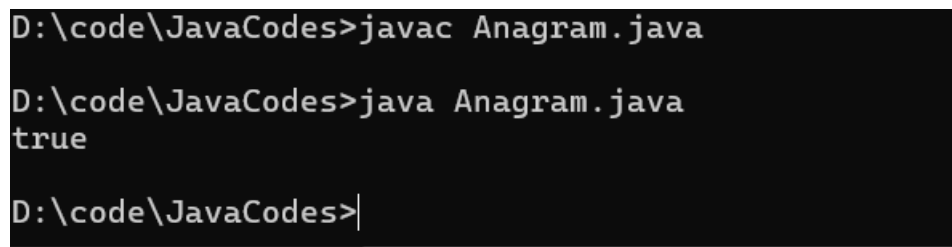
```

        else{
            hp.put(i, hp.get(i)-1);
        }
        if(hp.get(i)==0) hp.remove(i);
    }
    return hp.isEmpty();
}

public static void main(String[] args){
    String s1 = "geeks";
    String s2 = "kseeg";
    System.out.println(areAnagrams(s1,s2));
}
}

```

### Output:



```

D:\code\JavaCodes>javac Anagram.java
D:\code\JavaCodes>java Anagram.java
true
D:\code\JavaCodes>

```

Time Complexity:  $O(n+m)$

Space Complexity:  $O(k)$  (Note:  $k$  denotes no of unique characters)

## 2.Row With Max 1's

You need to find and return the index of the first row that has the most number of 1s. If no such row exists, return -1.

Note: 0-based indexing is followed.

Examples:

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

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

Output: 2

Explanation: Row 2 contains 4 1's.

**Code:**

```
import java.util.*;  
  
class rowMaxones{  
  
    public static int rowWithMax1s(int arr[][]){  
  
        // code here  
  
        int row=arr.length;  
        int col=arr[0].length;  
        int ind=-1;  
        int l=0;  
        int r=col-1;  
        while(l<row && r>=0){  
            if(arr[l][r]==0){  
                l++;  
            }  
            else{  
                ind=l;  
                r--;  
            }  
        }  
        return ind;  
    }  
  
    public static void main(String[] ar){  
        int[][] arr = {{0, 1, 1, 1}, {0, 0, 1, 1},{1, 1, 1, 1},{0, 0, 0, 0}};  
        System.out.println(rowWithMax1s(arr));  
    }  
}
```

```
}  
}
```

**Output:**

```
D:\code\JavaCodes>javac rowMaxones.java  
D:\code\JavaCodes>java rowMaxones.java  
2
```

### 3. Longest consecutive subsequence

Given an array **arr** of non-negative integers. Find the **length** of the longest subsequence 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.

**Code:**

```
import java.util.*;  
  
public class Longestsubsequence{  
  
    public static int findLongestConseqSubseq(int[] arr) {  
        // code here  
  
        int[] sub=new int[100000];  
  
        int max=Integer.MIN_VALUE;
```

```

        int c=0;
        for(int i:arr){
            sub[i+1]++;
        }
        for(int i:sub){
            if(i!=0){
                c++;
                max=Math.max(max,c);
            }
            else c=0;
        }
        return max;
    }

    public static void main(String[] ar){
        int[] arr= {2, 6, 1, 9, 4, 5, 3};
        System.out.println(findLongestConseqSubseq(arr));
    }
}

```

**Output:**

```

D:\code\JavaCodes>javac Longestsubsequence.java
D:\code\JavaCodes>java Longestsubsequence.java
6

```

**Time Complexity:** $O(n)$

**Space Complexity:** $O(1)$