Day 04

Saturday, 2 January 2021 7:24 PM

Homework Assignments :

- Understand the hash table and its members
- Understand the HashSet
- Understand how to use generics in hash table
- Read out the char and string members

- Jewels and Stones
- Intersection of Two Arrays
- contiguous subarray that sums to 0
- https://leetcode.com/problems/continuous-subarray-sum/
- https://leetcode.com/problems/subarray-sum-equals-k
- https://leetcode.com/problems/maximum-size-subarray-sum-equals-k/
- https://leetcode.com/problems/subarray-sums-divisible-by-k/
- https://leetcode.com/problems/valid-anagram
- https://leetcode.com/problems/group-anagrams

Problem Statement :

You're given strings jewels representing the types of stones that are jewels, and stones representir stones you have. Each character in stones is a type of stone you have. You want to know how ma stones you have are also jewels.
Letters are case sensitive, so "a" is considered a different type of stone from "A".

Input: jewels = "aA", stones = "aAAbbbb"
Output :3

Bru

```
Public int  numJewelsInStones(string J , string S)
{
    Int ans = 0 ;
    for(char s :  S.toCharArray() // for each stone ..
        for(char j: J.toCharArray())
        {
            If( j == s )
            {
               ans++;
               break; // stop searching the stone in the jewel
            }
        }
    }
}
```

```
Public int  numJewelsInStones(string J , string S)
{

    Set<Character>   jset = new HashSet();
    For(char j : j.toCharArray())
    {
        Jset.add(j)
    }
```

```
    Int ans = 0 ;
    For(char s : s.toCharArray())
    {
        If(Jset.contains(s))
        {
            ans++;
        }
    }
}
```

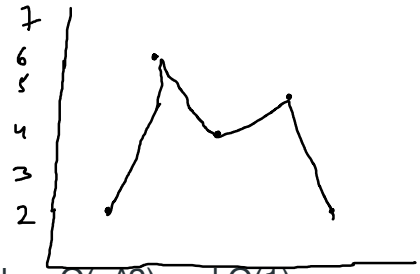
[2, 4, -2, 1, -3, 5, -3]

ps: [2, 6, 4, 5, 2, 7, 4]

Prefix Sums :

Given an array of integers , find the contiguous subarray that sums to 0 . They array can contain b
negative and postive integers .

For Example : [2 , 4 , -2 , 1 , -3 , 5 , -3]
Result = [4 , - 2 , 1 , -3]



1 , - 3, 5 , -3

Approach :

Brute Force Algorithm :
The brute force algorithm (going through each subarray) takes O(n^2) and O(1) space .

We can solve this problem with O(N) and O(N) space .

We use the technique of prefix sums . For all elements
We first calculate the sum s[i] which is equal to sum of all numbers
From o to I .

Interesting property is : If any s[i] is to 0 then a[0] to a[i] sums to 0

So that the subarray [0 .. i] is the answer .

If there is no s[i] equal to zero , we try to find two s[i] that have the same value .
For any j and k , if s[i] and s[k] have the same value then sum of sum of subarray [j+1 , k] is 0 /

In this we case we have two duplicates (0 & 4) and (2 & 6)

```
Public static Pair<Integer> zeroSumSunArray(int[] a)
{

        Int sum = 0 ;
        HashMap<Integer , Integer> map = new HashMap<>();

        For( int  I = 0 ; I < a.length; i++)
        {

        Sum + = a[i];

            If(sum == 0)
            Return new Pair<Integer> (0 , i);
```

```
            If(map.containsKey(sum))
            {
                return new Pair<Integer(map.get(Sum) + 1 , i);
            }

            Map.put(sum , i);

        }

        Return null;

    }



    }
```

Given two strings *s* and *t*, write a function to determine if *t* is an anagram of *s*.
Input: *s* = "anagram", *t* = "nagaram"
Input: *s* = "rat", *t* = "car"

```
public boolean isAnagram(String s, String t) {
    if (s.length() != t.length()) {
        return false;
    }
    int[] table = new int[26];
    for (int i = 0; i < s.length(); i++) {
        table[s.charAt(i) - 'a']++;
    }
    for (int i = 0; i < t.length(); i++) {
        table[t.charAt(i) - 'a']--;
        if (table[t.charAt(i) - 'a'] < 0) {
            return false;
        }
    }
    return true;
}
```

Group Anagram's :

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any orc**
An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, ty
using all the original letters exactly once.
Input: strs = ["eat","tea","tan","ate","nat","bat"]. --> String Array
Output : [["bat"] , ["eat" , "tea" ,"ate"] , ["nat" , tan]] --> List<List<string>>

Approach :

```
For( int I = 0 ; I < strs.length; i++)
{
    For (int j = 0 ; j < strs.length; j++)
```

```
    {

        if(I !=j)
        {
            If (isAnagram(I , j)
        }
    }
}
```

Approach :

Using two looops
Using two pointers
Using sorting

```
  public List<List<String>> groupAnagrams(String[] strs) {
    if (strs.length == 0) return new ArrayList();
    Map<String, List> ans = new HashMap<String, List>();
    for (String s : strs) {

      char[] ca = s.toCharArray();
      Arrays.sort(ca);
      String key = String.valueOf(ca);

      if (!ans.containsKey(key))
       ans.put(key, new ArrayList());

      ans.get(key).add(s);
    }
    return new ArrayList(ans.values());
  }
```

TC: NKLOGK
SC: NK