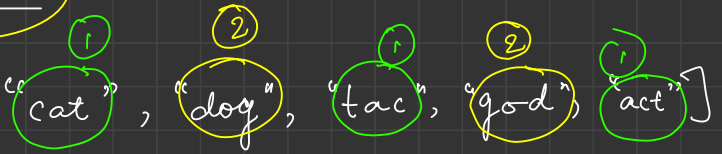
Group Anagrams

eg: ["cat", "dog", "tac", "god", "act"]



S_1 S_2

↳ when no. of occ of each character is same!

① Sort(S_1) Sort(S_2)

↓

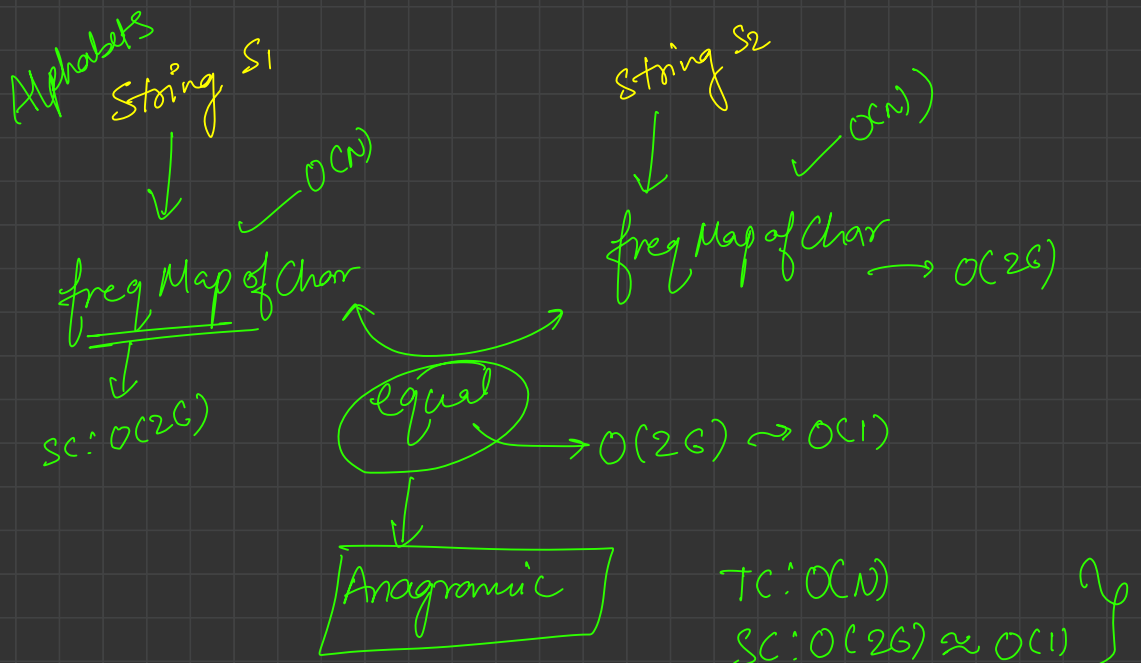
equals → No Not anangramic

↓

yes anagramic

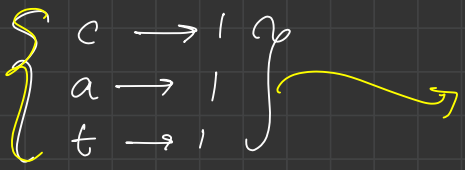
TC: $O(N \log N)$

SC: $O(1)$


Alphabets
String S1
O(N)
freq Map of Char
SC: O(26)
String S2
O(N)
freq Map of Char
O(26)
Equal
O(26) ~ O(1)
Anagramic
TC: O(N)
SC: O(26) ≈ O(1)

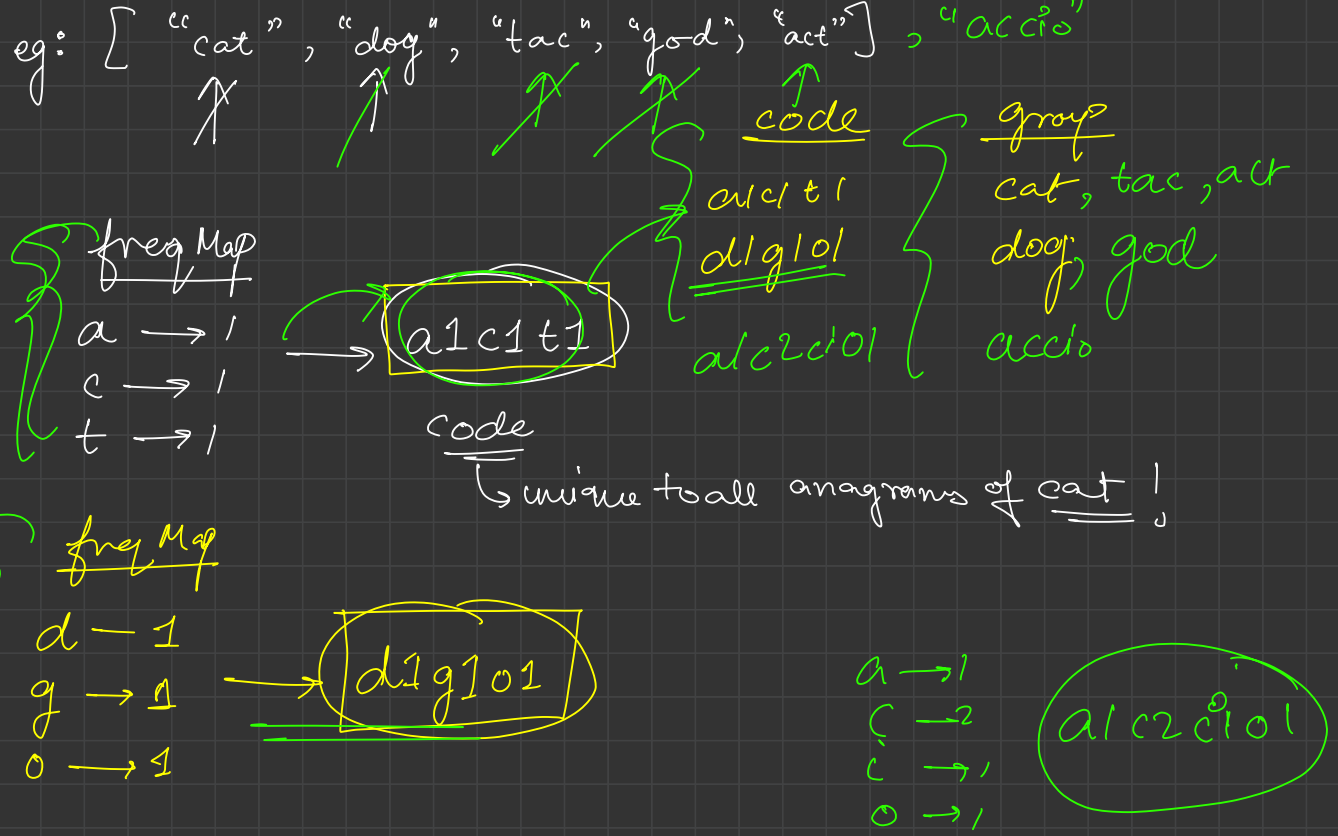
String = "Cat"

freqMap of Char

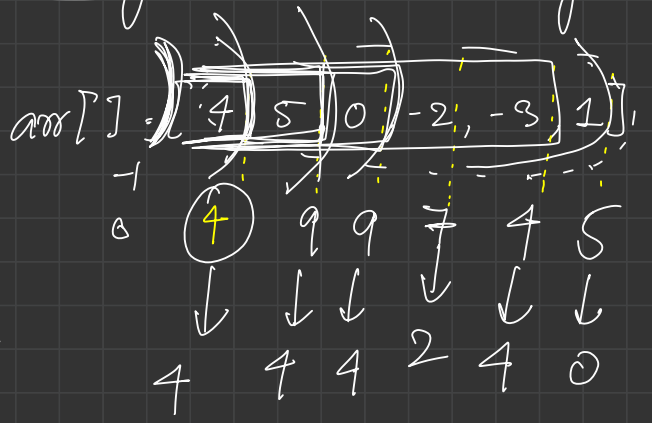


a1c1t1

Same for all anagrams of cat


eg: ["cat", "dog", "tac", "god", "act"] , "accio"
code
group
a1c1t1
dlglo1
a1c2c1o1
cat, tac, act
dog, god
accio
freq Map
a → 1
c → 1
t → 1
a1c1t1
code
↳ unique to all anagrams of cat !
freq Map
d → 1
g → 1
o → 1
dlglo1
a → 1
c → 2
i → 1
o → 1
a1c2c1o1

Subarray Sum divisible by k



K = 5

HashMap

rem	freq.
0	1
4	1 2 3 4
2	1

Sum = 1 + 2 + 3 + 1

= 7

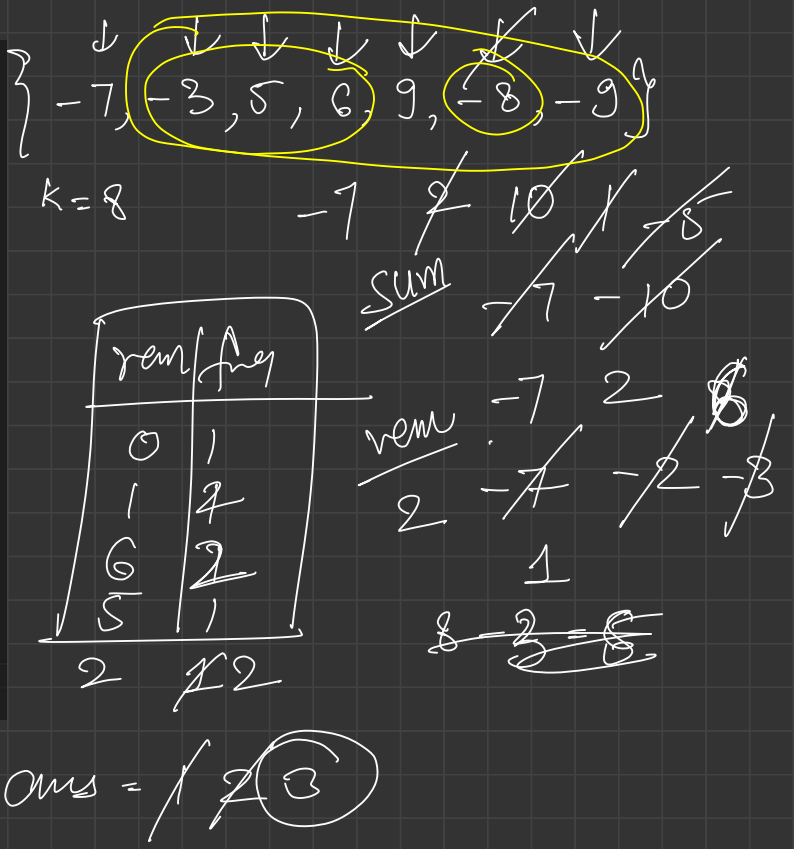
Suppose rem becomes (-)ve

-3

5n - 3 ⇝ 5n - 3 + 5 - 5 = 5n' + 2

rem

```java
public static int subarrayDivisbleByK(int arr[], int n, int k) {
    // Write code here
    HashMap<Integer, Integer> remFreqMap = new HashMap<>();
    remFreqMap.put(key: 0, value: 1);

    int ans = 0;
    int runningSum = 0;

    for (int num : arr) {
        runningSum += num;

        int rem = runningSum % k;

        if (rem < 0) {
            rem += k;
        }

        if (remFreqMap.containsKey(rem)) {
            ans += remFreqMap.get(rem);
        }

        remFreqMap.put(rem, remFreqMap.getOrDefault(rem, defaultValue: 0) + 1);
    }

    return ans;
}
```

Minimum Window Substring

exc

str1. d b a e c b b a b d c a a f b d d c a b g b a

inc

str2. a b b c d c

dmct = str2.length = 6

{ a → 1, b → 2, c → 2, d → 1 }

Freq Map

{ d → 1, b → 1 ~~2~~, a → 1, c → 1 }

mct = ~~1 2 3 4 5~~

O(M)

distinct Windows

S/= "a" "a"

Min Substring, which all unique char present in String

Σ = unique character from here