Todays Content

a) Characters matching
b) Permutations of A in B
c) Class of Permutations

10. Given 2 Strings, check no: of characters are matching.

Ex: S_1 = a n a t   ans = 4

S_2 = t a n a m

Idea1: a. Using 2 Hashmaps & storing frequencies & comparing

b. Using 1 Hashmap & get count

TODO

S_1 = a b a c b e   ans = 4

S_2 = a c a c a b

Idea2: Using 2 Hashmaps

S_1 = a c b a c b a ⟶ HM1 : { a : 3 b : 2 c : 2 }

S_2 = a b c c b c b ⟶ HM2 : { a : 1 b : 3 c : 3 }

cnt = 1 1 1 1 1 0 0 = 5

S_1 = b c b c a b b ⟶ HM1 : { b : 4 c : 2 a : 1 }

S_2 = c a e b a a d c ⟶ HM2 : { c : 2 a : 3 e : 1 b : 1 d : 1 }

1 1 * 1 * * * 1 = 4

```
int match(String s1, String s2){   TC: O(N+M)  SC: O(N+M)
    Hashmap<char, int> hm1;
    Hashmap<char, int> hm2;
    int N=s1.length();
    for(int i=0; i<N; i++){
        char ch= s1.charAt(i);
        if( hm1.containsKey(ch)==true)) {
            int f = hm1.get(ch)
        }   hm1.put(ch, f+1)
        else { hm1.put(ch, 1)}
    }
    int c=0;
    int M = s2.length();
    for(int i=0; i<M; i++){
        char ch= s2.charAt(i);
        if( hm2.containsKey(ch)==true)) {
            int f = hm2.get(ch)
        }   hm2.put(ch, f+1)
        else { hm2.put(ch, 1)}
        // Check if character ch is valid ?
        if( hm1.containsKey(ch) && hm2.get(ch) <= hm1.get(ch)) {
            c = c+1 // ch is valid character.
        }
    }
    return c;
}
```

2Q. Given a S_1 & S_2 of equal lengths

Check if they are permutations of each other.

Permutation: If freq of all characters is same in Both Strings:

Note: If 2 Strings length are diffrent, they can never be permutations of each other.

				match char
Ex:	S_1 = a n a t	S_2 = t a n a	= permutations	4 = 4
	S_1 = a b a c b	S_2 = a c a b c	= not permutation	4 ≠ 5
	S_1 = a b a c b	S_2 = a b b c a	= permutation	5 = 5

Idea: 1. Sorting & compare

S_1 = a b a c b S_2 = a b b c a

S_1 = a a b b c S_2 = a a b b c

TC: O(NlogN + NlogN + N) = O(NlogN)

```
2. boolean permutations(String s1, String s2){
       int c = match(S1, S2); // no: of character matching in S1 & S2
       int N = S1.length // S2.length
       if( c == N) {return true}
       else {return false}
   }
```

3Q: Count no: of substrings of A_N are permutations of String B_k : $N >= k$

	0 1 2 3 4 5 6 7		
Ex: A_N :	a b c b a b b c	ans = 4.	
A[0 3] =	a b c b	permutation	B_k : a c b b
A[1 4] =	b c b a	permutation	B_k : a c b b
A[2 5] =	c b a b	permutation	B_k : a c b b
A[3 6] =	b a b b	not permut	B_k : a c b b
A[4 7] =	a b b c	permutation	B_k : a c b b

<u>Idea 1:</u>

Get all substring of len = k from String A & check if its permutation to B.

$$TC: O(N-k+1) * O(k) \approx O(N^2) \quad SC: O(k)$$

Note: no: of substrings of len = k in size N = N-k+1

$TC: O(N-k+1) * O(k)$

WorstCase:

$k = 1 \quad TC: O(N-1+1)(1) = O(N)$

$k = N \quad TC: O(N-N+1)(N) = O(N)$

$k = N/2 \quad TC: O(N-N/2+1)(N/2) = O(N/2 * N/2) \approx O(N^2)$

Idea2: Fixed Subarray length

B_K : a b a c b $\longrightarrow$ HM1 { a:2 b:2 c:1 }

	0	1	2	3	4	5	6	7	8	9
A_N:	a	c	a	c	a	b	b	c	a	e

cur:

Substring:			HM2:	match:		ans
[0-4]	remove	add	{ a:3 c:2 }	1+1+1	= 3 ≠ 5	not per, 0
$\overset{s}{[1}-\overset{e}{5]}$	ar[0]	ar[5]	{ a:2 c:2 b:1 }	3+1	= 4 ≠ 5	not per, 0
[2-6]	ar[1]	ar[6]	{ a:2 c:1 b:2 }	4+1	= 5=5	permu, 1
[3-7]	ar[2]	ar[7]	{ a:1 c:2 b:2 }	5-1	= 4 ≠ 5	not per, 0
[4-8]	ar[3]	ar[8]	{ a:2 c:1 b:2 }	4+1	= 5=5	permu, 1
[5-9]	ar[4]	ar[9]	{ a:1 c:1 b:2 e:1 }	5-1	= 4 ≠ 5	not per, 0

Note: When we add a valid character:

match count inc by 1

When we remove a valid character:

match count dec by 1

```
int permutation(String A, String B){  TC: O(N)  SC: O(N)
    int N= A.length, k= B.length;
    Hashmap<char, int> hm1
     Step1: Insert all character of B in hm1
     Step2: Insert first k character of A in hm2, get match count
     int match = 0;
     for(int i=0; i<k; i++){
        char ch= A.charAt(i);
        if ( hm2.containskey(ch)){
            int f = hm2.get(ch)
            hm2.put(ch, f+1)
        }
        else {hm2.put(ch, 1)}
        // check if ch is valid character
        if (hm1.containskey(ch) && hm2.get(ch) <= hm1.get(ch)){
            match++;
        }
     }

     int ans = 0;
     if(match==k){ ans = ans+1}
     int s=1, e=k; // 2nd substring.
     while( e<N){
          // remove A[s-1] & add A[e]
          char ch = A.charAt(s-1);
          int f= hm2.get(ch)
          hm2.put(ch, f-1);
          // Check if we removed a valid character
          if (hm1.containskey(ch) && hm2.get(ch) < hm1.get(ch)){
             match = match-1
          }
```

1st substr of len k

$$\begin{tikzcd}[row sep=small] A : \underset{N}{} & {[0..k-1]} \arrow[d] \\ & {[\overset{s}{1}..\overset{e}{k}]} \arrow[d] \\ & {[2..k+1]} \end{tikzcd}$$

```
        char ch = A.charAt(e);
        if (hm2.containsKey(ch)) {
            int f = hm2.get(ch)
            hm2.put(ch, f+1)
        }
        else { hm2.put(ch, 1) }
        // check if ch is valid character
        if (hm1.containsKey(ch) && hm2.get(ch) <= hm1.get(ch)) {
            match++;
        }
        if (match == k) { // permutation.
            ans = ans+1
        }
        s = s+1; e = e+1
    }
    return ans;
}
```

3Q) Given an arr[] of strings return no. of class of permutations are there?

Note: 2 strings are of same class, if they are permutations of each other

Q: How many different class of strings are there

Constraints: N strings, each of length L.

Ex: arr[] =

0	anact
1	babel
2	tacna
3	elbab
4	actna
5	carel
6	lebab
7	lerac
8	valac

class 1 anact : 0 2 4

class 2 babel : 1 3 6

class 3 carel : 5 7

class 4 valac : 8

Ex: arr[] =

		Sorted	insert in hashset
0	anact	aacnt	HS: 4 :
1	babel	abbel	aacnt
2	tacna	aacnt	abbel
3	elbab	abbel	acelr
4	actna	aacnt	aaclv
5	carel	acelr	
6	lebab	abbel	
7	lerac	acelr	
8	valac	aaclv	

$$TC: O(\underbrace{N * L\log L}_{\text{Sorting a String of L size}}) + \underbrace{N * O(L)}_{\text{Insert string in hashset}} = O(N L \log L)$$