Q.1 Given a string, Find longest substring with distinct characters.

(distinct = different)

Str: a b c a b c d d ans = 4 (a b c d)

Str: s i p p e r ans = 3 (s i p / p e r)

Str: a b c g h e g k i m h a b k ans = 8

i) Ideal (Brute force)

Go on every substring (s to e), check if it contains all the distinct chars or not.

```
for(s -> 0 to n-1) {
    for(e -> s to n-1) {
        HashSet<character> hs = .....;
        for(s to e) {
            fill hs with char s to e
        }
    }
}
```

TC: $O(n^3)$

ii) idea2 → TC : $O(n^2)$

s = 0

e = 0 to n-1

s

str: a b c g h e g K d m h a b K

e

a b c g h
e

create longest possible ans (no duplicates) starting from s.

If char is getting repeated break at that moment.

for all possible start points

a b c a b c d d

0 1 2 3 4 5 6 7

s	hs	ans
0	a b c	3
1	b c a	3
2	c a b	3
3	a b c d	4

```
int solve (string str) {
    int ans=0;
    int n= str.length();

    for (int s=0; s<n; s++) {
        HashSet <Character> hs= new HashSet<>();

        for(int e=s; e<n; e++) {
            if (hs.contains(str.charAt(e)) == true) {
                break;
            }
            else {
                hs.add(str.charAt(e));
            }
        }
        ans = Math.max(ans, hs.size());
    }

    return ans;
}
```

TC: $O(n^2)$

SC: $O(1)$ { at max 26 size }

dry run

s

a	b	c	g	h	e	g	K	d	m	h	a	b	K
0	1	2	3	4	5	6	7	8	9	10	11	12	13

e

e	g	K
d	m	h
a	b	

```
for (int s=0; s<n; s++) {
    HashSet<Character> hs = new HashSet<>();
    for(int e=s; e<n; e++) {
        if (hs.contains(str.charAt(e)) == true) {
            break;
        }
        else {
            hs.add(str.charAt(e));
        }
    }
    ans = Math.max(ans, hs.size());
}
```

ans = ~~0~~ ~~6~~ 8

iii) idea2 → TC : O(n)

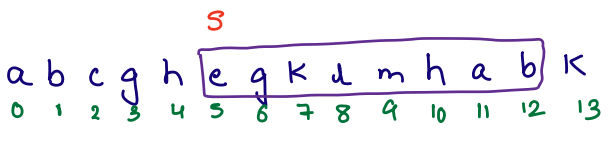
s

a	b	c	g	h	e	g	K	d	m	h	a	b	K
0	1	2	3	4	5	6	7	8	9	10	11	12	13

e

now starting from 1, 2, 3 can't help in creating better ans than 6 length.

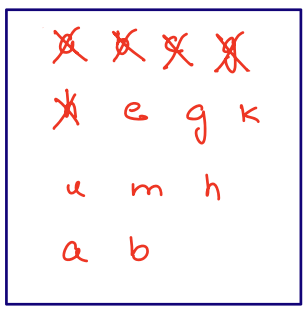
```
int solve (string str) {
    int n= str.length;
    Hashset < character > hs= new Hashset<>();
    int ans= 0;
    int s=0, e=0;
    while ( e < n )  {
        if ( hs. contains ( str.charAt(e) == false ) {
            hs.add( str.charAt (e));
            e++;
        }
        else {
            // let's get rid of repeated char
            hs. remove (str.charAt(s));
            s++;
        }
        ans= Math.max ( ans, hs.size());
    }
    return ans;
}
```




```
while ( e < n ) {
    if ( hs.contains( str.charAt(e) == false ) {
        hs.add( str.charAt(e) );
        e++;
    }
    else {
        // let's get rid of repeated char
        hs.remove( str.charAt(s) );
        s++;
    }
    ans = Math.max( ans, hs.size() );
}
```

acquire

release



ans = ~~1 2 3 4 5 6 7~~

8

itr : 2n

TC : O(n)

SC : O(1)

{ SC : O(26) } ≈ O(1)

Q.2 Given two strings, check if they are anagrams or not.

↓

permutation of each other

eg1 A = doodle, B = ooddle → true

eg2 A = jivei, B = vijee → no

using hashset is wrong {we need count to compare}

HashMap < Character, Integer > map

↳ lowercase

↓

size of map <= 26

i) create freq map of 1st string → map1

ii) create freq map of 2nd string → map2

iii) areMapsSame(map1, map2)

str1 : a a b c a

str2 : b a a a c

map1 =>

a → 3
b → 1
c → 1

map2 =>

b → 1
a → 3
c → 1

Q.3 Count no. of substrings of B which are permutations of A.
(anagram)

eg1
A = a b c
B = a b c b a e a b c ans = 3

eg2
A = a a b
B = a b a a b e b a b a a d ans = 5

Window size : A.length()
is fixed
↳ Sliding window

k = 3

A = a a b

B = a b a a b e b a b a a d
0 1 2 3 4 5 6 7 8 9 10 11

a → 2
b → 1
A map

{ never going change }

	s	e	rem	add
	0	2		
	1	3	0	3
	2	4	1	4
	3	5	2	5
→	4	6	3	6

b → 2
e → 1
B map
(current window)

ans = ~~1~~ ~~2~~ 3

TC : O(n) SC : O(1)