Q.1 Given a string, Find longest substring with distinct characters.
(non-repeated)

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

Str: s i p p e r ans: 3 (sip | per)

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

i) Idea 1

travel on all substrings with the help of s and e. check if substring from s to e is containing distinct chars or not with the help of Hashset.

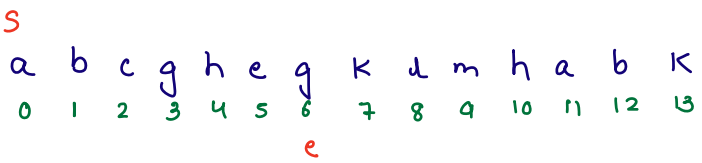
TC: $O(n^3)$

```
for (s → 0 to n-1) {
    for (e → s to n-1) {
        Hashset < C, J > hs = .....;
        for (s to e) {
            // to check whether content from s to is
            // is distinct or not
        }
        ans = Math.max(ans, hs.size());   → based on a check
    }
}
```

0	1	2	3	4	5
a	b	c	e	b	j
	s			e	

b c
e

ii) Idea2 : Expected TC → $O(n^2)$



s = 0

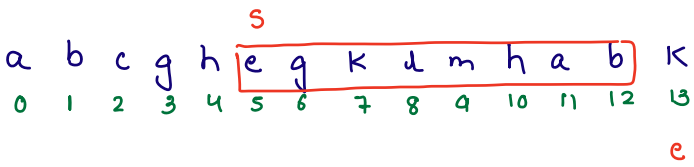
e → 0 to 13

b c g
h e

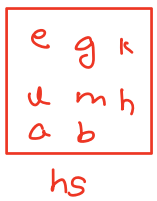
Start from every start point possible and extend till the moment you can (getting non-repeated char).

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

    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;
}
```

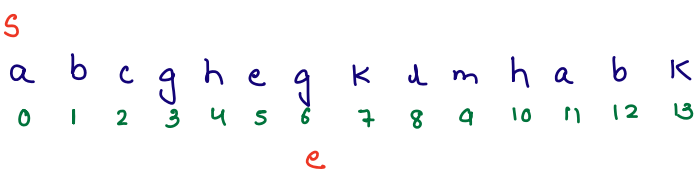



```
for (int s=0; s<n; s++) {
    HashSet<Character> hs = new HashSet<>();
    for (int e=s; e<n; e++) {
        if (hs.contains(str.charAt(e)) == true) {
            break;
        }
        else {
            hs.add(str.charAt(e));
        }
    }
    ans = Math.max(ans, hs.size());
}
```

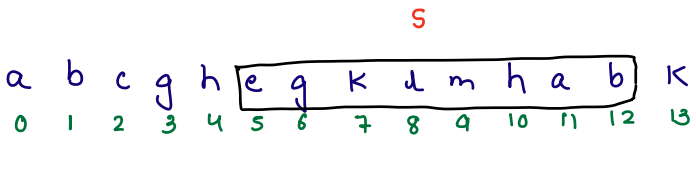


ans = ~~0~~ ~~8~~ 8

Idea 3 : TC → O(n)



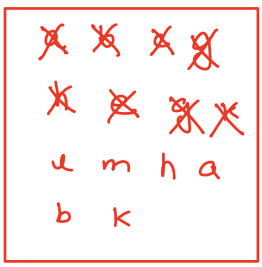
```
int solve (String str) {
    int n = str.length();
    HashSet<Character> hs = new HashSet<>();
    int s=0, e=0;
    int ans=0;
    while ( e < n ) {
        if (hs.contains(str.charAt(e)) == false) {
            hs.add(str.charAt(e));
            e++;
        }
        else {
            //get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



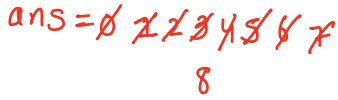

```
while ( e < n )  {
    if ( hs.contains ( str.charAt(e) ) == false ) {
            hs.add (str.charAt(e));
            e++;
    }
    else {
            //get rid of repeated char
            hs.remove (str.charAt(s));
            s++;
    }
    ans = Math.max(ans, hs.size());
}
```

acquire

release



hs



max itr : 2n

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

HashSet < character > hs → SC : O(26) ⇒ O(1)

char → lowercase chars

HashMap < character, Integer > map → SC : O(26) ⇒ O(1)

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

↓ permutation

(diff. rearrangement of same content)

eg1 A = doodle B = ooddle — ans: true

eg2 A = jivei B = vijee — ans: false

Don't use hashset (freq matters)

i) create freq map of str1 → map1

ii) create freq map of str2 → map2

iii) are Map Same (map1, map2)

str1 = a a b c a c — map1 => a → 3, b → 1, c → 2

str2 = b a c c a a — map2 => b → 1, a → 3, c → 2

Check same: travel a to z and check freq of ch in both maps.

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

↓ anagrams

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

Sliding window

Window length : A.length()

A = a a b
0 1 2

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

K = 3

s	e	rem	add
0	2		
1	3	0	3
2	4	1	4
3	5	2	5
4	6	3	6
5	7	4	7
6	8	5	8
→ 7	9	6	9

a → 2
b → 1

Amap
{ never going to change }

a → 2
b → 1

Wmap
{ current window }

if (areMapSame(Wmap, Amap)) {
    ans++;
}

ans = ~~1 2 3~~ 4

TC : O(n)

Doubts

A = [abcdef, abcg , abckt , abtmno]

max ans → length of smallest

↳ slen

ans = ab

idx = 0 , ch = a

idx = 1 , ch = b

idx = 2 , ch = c

```
ans = " ";
for (int idx = 0; idx < slen; idx++) {
    //check if the char at idx is same in all of
    // String or not
    char ch = A[0].charAt(idx);

    for (k = 1; k < A.length; k++) {
        if (A[k].charAt(idx) != ch) {
            return ans;
        }
    }

    ans += ch;
}

return ans;
```

a	b	c	a	d	a	c
0	1	2	3	4	5	6

```
for (s → 0 to n-1) {
    for (e → s to n-1) {
        Hashset < c, g > hs = .....;
        for (s to e) {
            || to check whether content from s to is
            || is distinct or not
        }
        ans = Math.max (ans, hs.size());
    }
}
```

↳ check based

s	e	
0	0	a
0	1	a b
0	2	a b c