Longest Substring without repeating character

Given a string `s`, find the length of the **longest** **substring** without repeating characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.
```

kwe kew

B rate

a b c a b c b b

p w w k e w

Check if (prev it exist) or not

cnt = ~~1~~ ~~2~~ 3

cnt = ~~1~~ ~~2~~ ~~3~~ ~~4~~ 3

kew

O(N²)

for(i=0; i<n; i++) → n → string length.

O(N)

if (!prevcheck(s[i])) {

func(prev(char c))

string → for c

cnt++

else

cnt

Naive approach:

Generate all the substring and find non-repeating by using hash set.

T.C. → $O(N^2)$ → $O(N)$ space

Optimized:

a b c a a b c d b d

abc abcd → longest substring.

Approach:

abc a a b c d b d

We Keep two pointer

l → keep track of current element and substring

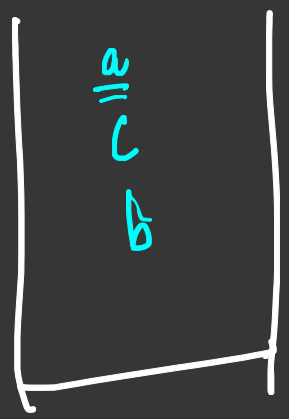
r → to remove ~~check~~ prev duplicate or repeating char.

prefix

I need to know that the current ele has appeared in array previously or not.

hash

we use hashset



Hashset

Store the ele

a b c a a b c d b d

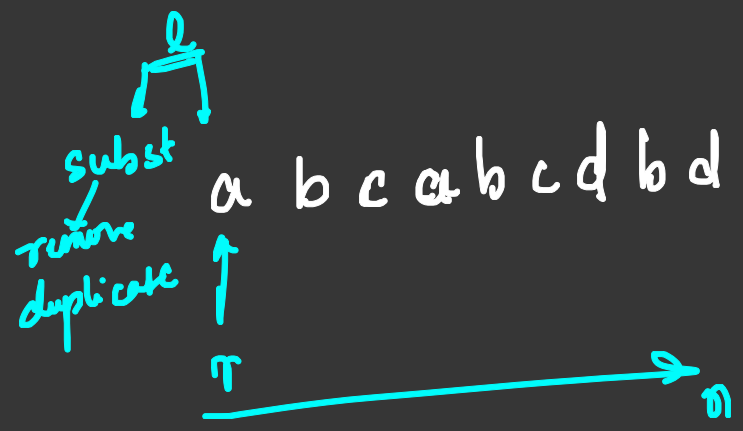
l l l l l l

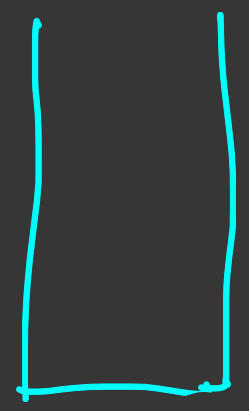
Length = $r - l + 1$

T.C.: $O(2N)$ → Better

can be optimised further.

Pseudocode






```
for (r = 0, r<n; r++)
    if (mp.find(s[r]) != mp.end())
        while( l<r && mp.find(s[r]) != mp.end())
            set.erase(s[l])
            l++;
    mp.insert(s[r])
    maxi = max(maxi, l-r+1);
```

T.C.: O(2N)

Optimized

→ It is use same logic but in case of moving l to the one by one until we find the non-repeating element is taking time. So to avoid it we use hashmap and store both char and its index and directly jump to next of the index to reduce time from O(2N) → O(N)

l l l

abc a a b c d b d

0 1 2 3 3

a, ~~0~~ 3

Pseudocode:

```
unordered map <char, int> mp

for (i=0; i<n; i++)
    if( ~~mp[s[i]]~~ . mp.find( s[i]) != mp.end())
        l = max( l, mp[s[i]]+1);

    mp[s[i]] = r_i
    maxi = max(maxi, r_i - l + 1)
```

T.C. : $O(N)$ S.C = $O(N)$