Date 20/11/2023 Page

Day - 57

Sliding Window

* **Longest substring without repeating char:**

⇒ Exi: a b a b c → 3
aba ✗
abab ✗

⇒ Start from the basic approach, first we find the all substring

⇒ 1 → a b a b c
2 → ab ba ab bc
3 → aba bab abc → 3 $O(n^3)$
4 → abab babc
5 → ababc

⇒ This is our brute force approach.

⇒ In the second approach, we will first start from first letter and start adding the character until any char repeats.

⇒ $O(n^2)$

⇒ In the third approach, we will use sliding window approach.

⇒ First we take a one char window then add another letter until any char repeats. After repeating, we will delete

Date _____ Page _____

that char. So, for removing, we will remove from start to that letter.

⇒ After doing this same, we will get our answer.

Ex: abcdecbeadf

len → 0 1 2 3 4 5

a

ab → abc → abcd → abcde

~~abcdec~~ → decb → ~~decb~~e

cbea → cbead → cbeadf

len = 6

⇒ This is the concept of sliding window

⇒ Here, we use two pointers - first & second.

Code

```
string s;
vector <bool> count (256, 0);
int first = 0, second = 0, len = 0;
while ( second < s.size() ){
    while (count[s[second]]){
        count[s[first]] = 0;
        first++;
    }
    count[s[second]] = 1;
    len = max( len, second - first + 1);
    second++;
}
return len;
```

Date _____ Page _____

* **Smallest Distinct Window:**

=> we have to return the min. window that have all distinct chars.

Ex: A A B B B C B B A C

A B B B C → 5 ↳ B A C → 3 min

=> we will use the same previous approach. So,

A A B B B C → 6

=> But we want min —

A B B B C → 5

=> B B B C B B A

C B B A → 4

B B A C

B A C → (3) → min

=> We will take len as our answer as the length of string.

=> We will also create an array of 256 size to check the occurance.

=> Here, we take a diff variable to check all the characters are come into the string or not.

=> If the diff → 0 , first increases

=> If the diff → 1 , second increases.

=> From diff to 0 , we will increase second