Sliding Window

Key is to find out the condi

Fixed

i) Window size is fixed.

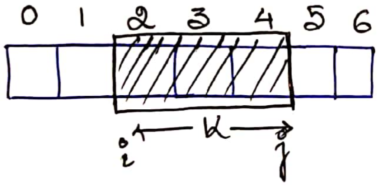
ii) Once you reach the window size k, the same window will slide over the entire array/string.

Keep 2 pointers;

i ← start of window index;

j ← end of window index;

k = length of window.



$k = 3; \; j = 4; \; i = 2$

$\therefore \boxed{j - i + 1 == k}$

Code

```
while (j < N){
    Calculation; Add j
    if ~~while~~ (j-i+1) < k)
        j++;
    else if (j-i+1 == k)
        get answer
        i++; j++;
}.
```

Variable.

i) Window size is variable

In Variable size window the size of the window will be based on some condition.

Keep 2 pts

i ← start of window

j ← end of window.

General Code

```
while (j < N){
    calculations
    if (condition < k)
        j++
    else if (condition == k){
        answer ← Calculations
        j++
    }.
    else if (condition > k){
        while (cond > k){
            remove calculations for i;
            i++
        }
        j++
}.
```

Variable Size Sliding Window

Given a string you need to print the size of the longest possible substring ~~of~~ with exactly k unique characters. If such a substring is not possible return -1.

— x —

Algorithm :- • This is variable sliding window problem as the length of the window is not fixed.

• Condition

Creating a map. to store the frequency of chars in the substring / window.

So; $\boxed{\text{size of the map}}$ will give the count of unique characters within the window.

∴ $\boxed{\text{map.size() == k}}$ ← Condition

Similar Problems - Pick Toys.
Fruits into Baskets (LC) $\{ k=2 \}$.

Code

```
int longest_k (string str, int k) {
    int i=0, j=0;
    map <char, int> hashmap;
    int len = 0;          // stores the length of the ~~map~~ window

    while (j < str.length()) {
        if (hashmap.find (str[j]) == hashmap.end())
            hashmap [str[j]] = 1;
        else
            hashmap [str[j]] ++;

        if (hashmap.size() < k)
            j++;
        else if (hashmap.size() == k) {
            len = max (len, j-i+1);
            j++;
        }
        else if (hashmap.size() > k) {
            while (hashmap.size() > k) {
                if (hashmap.find (str[i] != hashmap.end())
                    hashmap [str[i]]--
                if (hashmap [str[i]] == 0)     // Remove to
                                                  decrease
                hashmap.erase (str[i])            size
                i++;
            } j++;
        } if (len <= 0) return -1;
    return length;
}
```

Given a string, find the length of the longest substring without repeating characters

—— x ——

Without repeating characters = <u>All unique Characters</u>

So map.size() must be equal to window size (j-i+1)

<u>Condit^n</u>: $\boxed{\text{map.size()} == (j-i+1)}$

map.size() can never be greater than j-i+1

<u>Code</u>

```
int lengthofLongSubstr(string str){
    int i=0, j=0, maxLen=0;
    map<char,int> hashmap;
    while(j < str.length()){
        if (hashmap.find(str[j]) == hashmap.end())
            hashmap[str[j]]=1;
        else
            hashmap[str[j]]++;

        if (hashmap.size() == j-i+1){
            maxLen = max(maxLen, j-i+1);
            j++;
        }
        else if (hashmap.size() < j-i+1){
            while (hashmap.size() < j-i+1){
                if (hashmap.find(str[i]) != hashmap.end())
                    hashmap[str[i]]--;
                if (hashmap[str[i]] == 0)
                    hashmap.erase(str[i]);
                i++
            }j++;
        }
    } return maxLen;
}
```