3.Longest Substring Without Repeating Characters

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

My Answer

Difficulty	Status	Runtime	Distribution(%)	Memory	Distribution(%)
Medium	AC	7ms	65.52	5.9MB	65.13

```
int lengthOfLongestSubstring(char * s){
    int lenth = strlen(s);
    int i, j;
    int head = 0;
    int tail = 0;
    int result = 0;
    for (i = 0; i < lenth; i++) {
        for (j = head; j < tail; j++) {
            if (s[j] == s[tail]) {
                head = j + 1;
                break;
            }
        }
        result = (result < (tail - head + 1)) ? (tail - head + 1) : result;
        tail++;
    }
    return result;
}
```

Algorithm

Time complexity	Space complexity
$O(n^2)$	$O(n)$

If a substring S_{ij} from index i to $j-1$ is already checked to have no duplicate characters. We

only need to check if $S[j]$ is already in the substring S_{ij}

Another Answer

Difficulty	Status	Runtime	Distribution(%)	Memory	Distribution(%)
Medium	AC	3ms	89.64	5.9MB	42.59

```
int lengthOfLongestSubstring(char * s){
    int head = 0;
    int tail = 0;
    int hash[128] = {0};
    int result = 0;
    while (s[tail]) {
        if (hash[s[tail]] && head < hash[s[tail]]) {
            head = hash[s[tail]];
        }
        hash[s[tail]] = tail + 1;
        result = (result < (tail - head + 1)) ? (tail - head + 1) : result;
        tail++;
    }
    return result;
}
```

Algorithm

Time complexity	Space complexity
$O(n)$	$O(n)$

If a substring S_{ij} from index i to $j-1$ is already checked to have no duplicate characters.We only need to check if $S[j]$ is already in the substring S_{ij}.

Another Answer

Difficulty	Status	Runtime	Distribution(%)	Memory	Distribution(%)
Medium	AC	0ms	100.00	5.8MB	65.04

```
int lengthOfLongestSubstring(char * s){
    int lenth = strlen(s);
    int head = 0;
    int tail = 0;
    int hash[128] = {0};
    int result = 0;
    int i;
    for (i = 0; i < lenth; i++) {
        if (hash[s[tail]] != 0) {
            if (head < hash[s[tail]]) {
                head = hash[s[tail]];
            }
        }
        hash[s[tail]] = tail + 1;
        result = (result < (tail - head + 1)) ? (tail - head + 1) : result;
        tail++;
    }
    return result;
}
```

Algorithm

Time complexity	Space complexity
$O(n)$	$O(n)$

Instead of using a set to tell if a character exists or not,we could define a mapping of the characters to its index.Then we can skip the characters immediately when we found a repeated character.