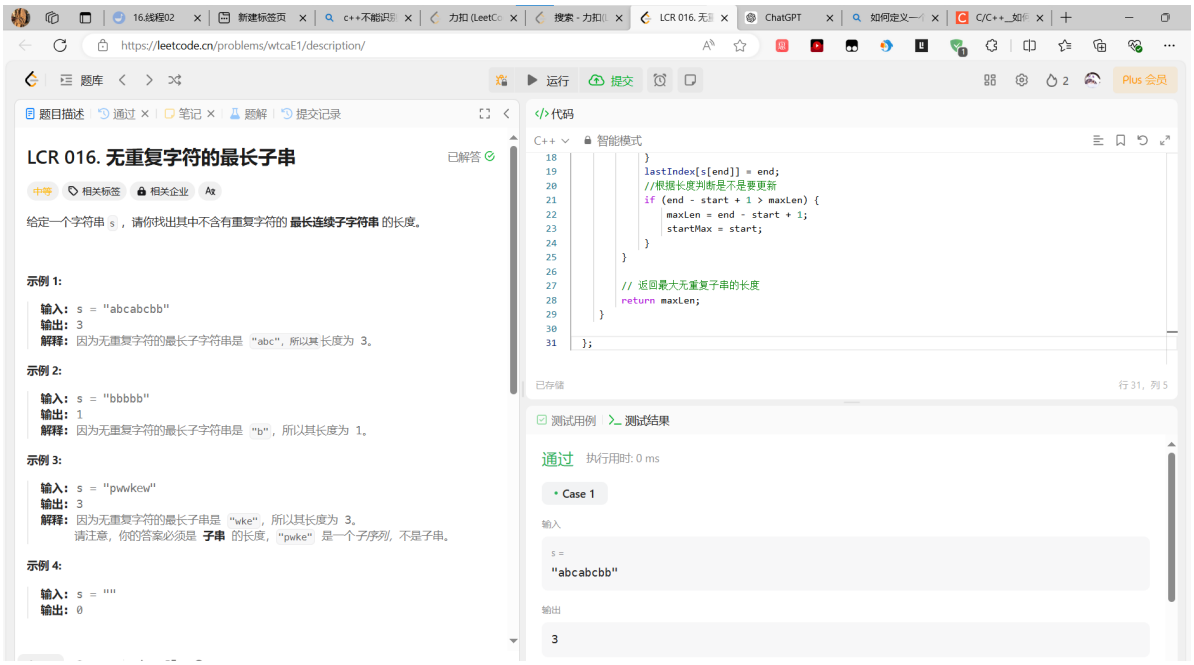
# 题目：无重复子串的最大长度




# 解题思路一：

对于这个问题，我们需要拆开来看，假设成一个个小的字符串，假设当前字符串的长度为3，形如：“aba",假设他是字符串"abac"的子串，很明显，可以得到，下次的最大长度一定是从字符a的下一个元素开始计算的，我们再假设为字符串为”abcb",它是"abcbafjhkiul"的子串，这时候由于a和b重复出现，事实上的start是c而不是b;因此，我们假设start=0,最后一个元素的下标为end,使用一个哈希表来存储index,那么当lastIndex[s.[end]]再次出现时，start就应该更新元素为，max(start,lastIndex[s[end]]+1)

```
class Solution
{
 public:
     int lengthOfLongestSubstring(string s)
     {
         if (s.empty()) return NULL;  // 如果字符串为空，直接返回空字符串

         unordered_map<char, int> lastIndex;  // 哈希表，用于存储每个字符最后出现的位置
         int maxLen = 0;  // 记录最大无重复子串的长度
         int start = 0;  // 当前无重复子串的开始位置
         int startMax = 0;  // 记录最大无重复子串的开始位置

         // 遍历字符串
         for (int end = 0; end < s.size(); ++end) {
             //如果出现重复字符串，更新起始位置为重复出现的字符的下一个的位置与start的最大值
             if (lastIndex.find(s[end]) != lastIndex.end()) {
                 start = max(start, lastIndex[s[end]] + 1);
             }
             lastIndex[s[end]] = end;
             //根据长度判断是不是要更新
             if (end - start + 1 > maxLen) {
                 maxLen = end - start + 1;
                 startMax = start;
             }
```

```
        }

        // 返回最大无重复子串的长度
        return maxLen;
    }

};
```

## 解题思路二：

滑动数组：我们把数组从长度1开始看，不难发现数组一直在向右变长，假设出现了重复元素，其实就相当于数组一下子向右滑动且收缩，即这个窗口是滑动的，对于每个元素，如果重复了，我们只需要删除重复的元素的记录下标，更新起点，然后令重复节点的新下标入队即可

```
class Solution
{
 public:
     int lengthOfLongestSubstring(string s)
     {
      if (s.empty()) return NULL;  // 如果字符串为空，直接返回空字符串

    unordered_set<char> charSet;  // 集合，用于存储当前窗口内的字符
    int maxLen = 0;  // 记录最大无重复子串的长度
    int start = 0;  // 当前无重复子串的开始位置
    int startMax = 0;  // 记录最大无重复子串的开始位置

    // 使用滑动窗口遍历字符串
    for (int end = 0; end < s.size(); ++end) {
        while (charSet.find(s[end]) != charSet.end()) {
            charSet.erase(s[start]);  // 移除重复字符，调整窗口起始位置
            start++;
        }
        charSet.insert(s[end]);  // 插入新字符到集合
        if (end - start + 1 > maxLen) {
            maxLen = end - start + 1;
            startMax = start;
        }
    }

    // 返回最大无重复子串
    return maxLen;
     }
};
```