3. 无重复字符的最长子串

题目

```
给定一个字符串 s ，请你找出其中不含有重复字符的 最长子串 的长度。

"abcabcbb" 最长字串 为 abc -> 3
"bbbbb" -> b -> 1
"" -> 0
```

蛮力解法 BruteForce

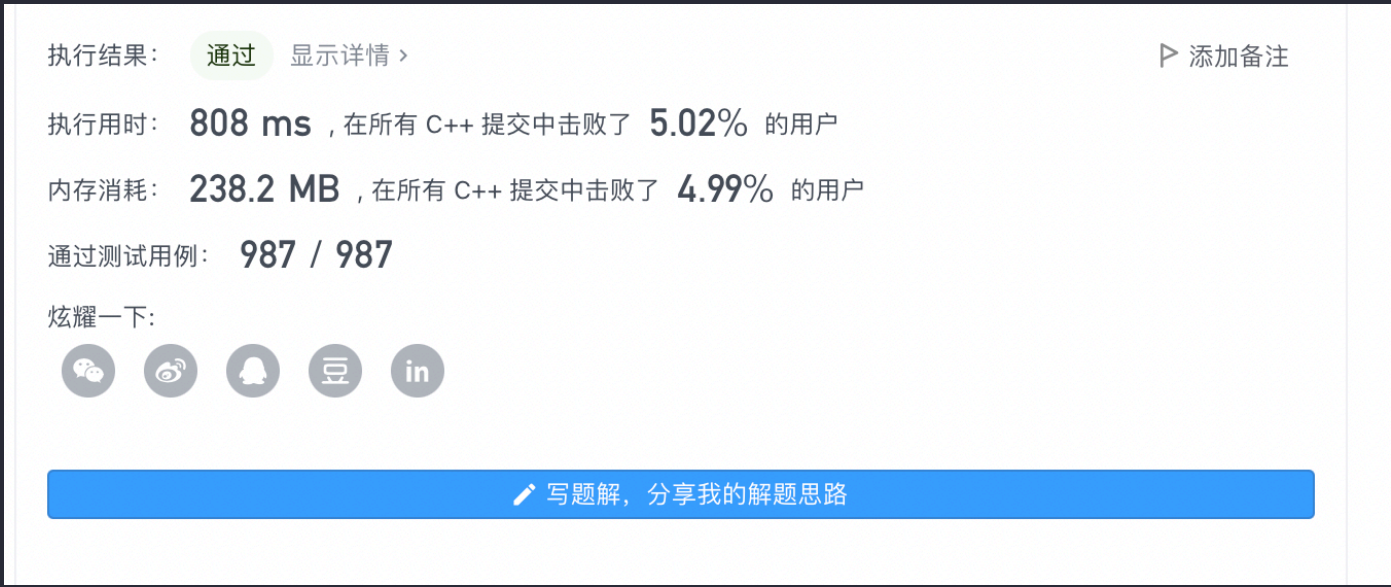
最简单的想法就是枚举每一个子串，然后统计最大长度，配合Set进行优化

```cpp
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        uintmax_t maxLengthOfSubstring = 0;
        // 长度为 0 1 字符串不可能有重复元素 非重复字串就是其本身的长度
        if (s.size() <= 1){
            return s.size();
        }
        // 枚举所有元素作为字串开头
        for (int i = 0; i < s.size(); ++i) {

            unordered_set<char> nps;
            // 从开头往后找到第一个重复元素
            for (int j = i; j < s.size(); ++j) {
                if (nps.find(s[j]) == nps.end()) {
                    nps.insert(s[j]);
                }else {
                    break;
                }
```

```
20              }
21              // 不断取大
22              maxLengthOfSubstring =
    max(nps.size(),maxLengthOfSubstring);
23          }
24          return (int)maxLengthOfSubstring;
25      }
26  };
```

我们不难验证这个是一个可行的解法，我们不妨将所哟关于unordered_set 操作视为O(1) 那么整一个算法的时间复杂度大概约为 O(n^2),同时也确实能在LeetCode上通过但是时间非常长，我们思考一下是不是有更好的解法?



滑动窗口解法

我们不妨从头开始维护一个窗口，每当窗口中出现重复元素，就从头开始删除，直到重复元素消失在往后面找.

我们以

```
abcabcbb
(a)bcabcbb // 窗口为 "a"
(ab)cabcbb // 插入 b 依旧不重复
(abc)abcbb // 插入 c 显然不重复
a(bca)bcbb // 此时再插入 a 发现重复 从left开始删除 第一个删除完（a） 已经不重复结束
ab(cab)cbb // 插入 b 还是发现重复 ，依然从left 开始删除，删除第一个 不重复了
abc(abc)bb // 同上
abcab(cb)b // 此时插入 b 又重复了，删除第一个 a 还是重复 在删除一个 b ok 不重复了
abcabcb(b) // 同上
```

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        // 初始化 保存最长字串长度的变量
        int maxLengthOfSubString = 0;
        // 头部
        int start = 0;
        // 初始化 去重复的set 使用 HashSet
        unordered_set<char> nps;
        // 循环End
        for (int end = 0;end < (int) s.size(); end++){
            // 如果当前 s[end] 重复就不断去除头部元素直到不重复
            while (end < ((int) s.size()) && nps.find(s[end]) != nps.end()) {
                nps.erase(s[start]);
                start ++;
            }
            // 插入自己 nps.size() == end - start + 1 == [start,end]
            nps.insert(s[end]);
            // 两者取大者
            maxLengthOfSubString = max((int)nps.size(),maxLengthOfSubString);
        }
```

```
22          return maxLengthOfSubString;
23      }
24  };
```