思路：

这道题主要用到思路是：滑动窗口

什么是滑动窗口？

其实就是一个队列,比如例题中的 `abcabcbb` ，进入这个队列（**窗口**）为 `abc` 满足题目要求，当再进入 `a` ，队列变成了 `abca` ，这时候不满足要求。所以，我们要移动这个队列！

如何移动？

我们只要把队列的左边的元素移出就行了，直到满足题目要求！

一直维持这样的队列，找出队列出现最长的长度时候，求出解！

时间复杂度：O(n)O(n)

代码：

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if not s:return 0
        left = 0
        lookup = set()
        n = len(s)
        max_len = 0
        cur_len = 0
        for i in range(n):
            cur_len += 1
            while s[i] in lookup:
                lookup.remove(s[left])
                left += 1
                cur_len -= 1
            if cur_len > max_len:max_len = cur_len
            lookup.add(s[i])
        return max_len
```

下面介绍关于滑动窗口的**万能模板**,可以解决相关问题,相信一定可以对滑动窗口有一定了解!

模板虽好,还是少套为好!多思考!更重要!

还有类似题目有:

3. 无重复字符的最长子串

```
class Solution:
    def lengthOfLongestSubstring(self, s):
        """
        :type s: str
        :rtype: int
        """
        from collections import defaultdict
        lookup = defaultdict(int)
        start = 0
        end = 0
        max_len = 0
        counter = 0
        while end < len(s):
            if lookup[s[end]] > 0:
                counter += 1
            lookup[s[end]] += 1
            end += 1
            while counter > 0:
                if lookup[s[start]] > 1:
                    counter -= 1
                lookup[s[start]] -= 1
                start += 1
            max_len = max(max_len, end - start)
        return max_len
```

76. 最小覆盖子串

```
class Solution:
    def minWindow(self, s: 'str', t: 'str') -> 'str':
        from collections import defaultdict
        lookup = defaultdict(int)
        for c in t:
            lookup[c] += 1
        start = 0
        end = 0
        min_len = float("inf")
        counter = len(t)
        res = ""
        while end < len(s):
            if lookup[s[end]] > 0:
                counter -= 1
            lookup[s[end]] -= 1
            end += 1
            while counter == 0:
                if min_len > end - start:
                    min_len = end - start
                    res = s[start:end]
                if lookup[s[start]] == 0:
                    counter += 1
                lookup[s[start]] += 1
                start += 1
        return res
```

159. 至多包含两个不同字符的最长子串

```
class Solution:
    def lengthOfLongestSubstringTwoDistinct(self, s: str) -> int:
        from collections import defaultdict
        lookup = defaultdict(int)
        start = 0
        end = 0
        max_len = 0
        counter = 0
        while end < len(s):
            if lookup[s[end]] == 0:
                counter += 1
            lookup[s[end]] += 1
            end +=1
            while counter > 2:
                if lookup[s[start]] == 1:
                    counter -= 1
                lookup[s[start]] -= 1
                start += 1
            max_len = max(max_len, end - start)
        return max_len
```

340. 至多包含 K 个不同字符的最长子串

```
class Solution:
    def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:
        from collections import defaultdict
        lookup = defaultdict(int)
        start = 0
        end = 0
        max_len = 0
        counter = 0
        while end < len(s):
            if lookup[s[end]] == 0:
                counter += 1
            lookup[s[end]] += 1
            end += 1
            while counter > k:
                if lookup[s[start]] == 1:
                    counter -= 1
                lookup[s[start]] -= 1
                start += 1
            max_len = max(max_len, end - start)
        return max_len
```

滑动窗口题目:

3. 无重复字符的最长子串

30. 串联所有单词的子串

76. 最小覆盖子串

159. 至多包含两个不同字符的最长子串

209. 长度最小的子数组

239. 滑动窗口最大值

567. 字符串的排列

632. 最小区间

727. 最小窗口子序列

下一篇：画解算法：3. 无重复字符的最长子串

360

条评论 >_
编辑

预览

精选评论(5)

对java代码进行举例解释！

```
public int lengthOfLongestSubstring(String s) {
        HashMap<Character, Integer> map = new HashMap<>();
        int maxLen = 0;//用于记录最大不重复子串的长度
        int left = 0;//滑动窗口左指针
        for (int i = 0; i < s.length() ; i++)
        {
            /**
            1、首先，判断当前字符是否包含在map中，如果不包含，将该字符添加到map（字符，字符在数组下标），
             此时没有出现重复的字符，左指针不需要变化。此时不重复子串的长度为：i-left+1，与原来的maxLen比较，取最大值；

            2、如果当前字符 ch 包含在 map中，此时有2类情况：
             1）当前字符包含在当前有效的子段中，如：abca，当我们遍历到第二个a，当前有效最长子段是abc，我们又遍历到a，
             那么此时更新 left 为 map.get(a)+1=1，当前有效子段更新为 bca；
             2）当前字符不包含在当前最长有效子段中，如：abba，我们先添加a,b进map，此时left=0，我们再添加b，发现map中包含b，
             而且b包含在最长有效子段中，就是1）的情况，我们更新 left=map.get(b)+1=2，此时子段更新为 b，而且map中仍然包含a，map.get(a)=0；
             随后，我们遍历到a，发现a包含在map中，且map.get(a)=0，如果我们像1）一样处理，就会发现 left=map.get(a)+1=1，实际上，left此时
             应该不变，left始终为2，子段变成 ba才对。

             为了处理以上2类情况，我们每次更新left，left=Math.max(left , map.get(ch)+1).
             另外，更新left后，不管原来的 s.charAt(i) 是否在最长子段中，我们都要将 s.charAt(i) 的位置更新为当前的i，
             因此此时新的 s.charAt(i) 已经进入到 当前最长的子段中！
             */
            if(map.containsKey(s.charAt(i)))
            {
                left = Math.max(left , map.get(s.charAt(i))+1);
            }
            //不管是否更新left，都要更新 s.charAt(i) 的位置！
            map.put(s.charAt(i) , i);
            maxLen = Math.max(maxLen , i-left+1);
        }

        return maxLen;
    }
```

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        if(s.size() == 0) return 0;
        unordered_set<char> lookup;
        int maxStr = 0;
        int left = 0;
        for(int i = 0; i < s.size(); i++){
            while (lookup.find(s[i]) != lookup.end()){
                lookup.erase(s[left]);
                left ++;
            }
            maxStr = max(maxStr,i-left+1);
            lookup.insert(s[i]);
    }
        return maxStr;

    }
};
```

小菜鸡，刚开始刷力扣，很多东西不明白，基本都是看答案写的。针对003 无重复字符的最长字串，参考如上答案，C++版本。编者很辛苦，没有给出具体解释，我想说出自己的一些想法。

1.对于大多数人比较纠结的一点

```
while (lookup.find(s[i]) != lookup.end()){
                lookup.erase(s[left]);
                left ++;
            }
```

这段代码结果是不断从左缩小窗口，直到窗口中不存在与下一个字符重复的字符。

lookup.find() 查找对应元素，成功返回对应的迭代器，失败返回最后一个元素迭代器（即 lookup.end() ）

lookup.end() 大多数人认为是最后添加进去的元素对应的迭代器，其实不然，是最后添加进去的元素对应的迭代器的下一个（最后一个元素对应的迭代器），此处有点绕，大家可以去找找相关资料，就很清晰了。

2.小白刚开始刷力扣，其实刷了三题，发现一个规律，最初的算法思想很多都是来自遍历过程。也就是所谓的暴力算法，但是由于哈希和一些高级数据结构的操作，实现了查询时间复杂度为常数的方法，进而衍生出优化方法（这是小白自我感觉，无论对错，希望得到老哥们的指点）

我来加一下java版的注释，方便理解

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s.length()==0) return 0;
        HashMap<Character, Integer> map = new HashMap<Character, Integer>();
        int max = 0;//最长子串长度
        int left = 0;//滑动窗口左下标，i相当于滑动窗口右下标
        for(int i = 0; i < s.length(); i ++){
            if(map.containsKey(s.charAt(i))){//charAt() 方法用于返回指定索引处的字符。索引范围为从 0 到 length() - 1。
                left = Math.max(left,map.get(s.charAt(i)) + 1);       //map.get():返回字符所对应的索引，当发现重复元素时，窗口左指针右移
            }        //map.get('a')=0,因为map中只有第一个a的下标，然后更新left指针到原来left的的下一位
            map.put(s.charAt(i),i);      //再更新map中a映射的下标
            max = Math.max(max,i-left+1);     //比较两个参数的大小
        }
        return max;

    }
}
```

不一定是左边的元素移除吧。。

python带注释，应该比较简洁易懂

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        if not s: return 0
        max_len = 0        # 最大长度
        tp = []                 # 放字符串的一个队列
        for a in s:
            while a in tp:
                del tp[0]           # 删除队列左边第一个，直到没有重复的字符串
            tp.append(a)
            if len(tp) > max_len: max_len = len(tp)
        return max_len
```

评论(360)

我觉得你们都被第一个人给带走偏了,看到第一个说滑动,你们就按照他的换个变量,加点注释,就觉得是自己的了,都在滑动滑动什么math.max,只是简单的说得出不重复字母的个数,都在循环string,不重复就计数不行吗?单独拿出个math函数来计算,性能真的有提升吗?

滑动啥滑动,string转char数组,放到set,返回set.size(),很难吗

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
        if (s.length() == 0) return 0;
            HashMap<Character, Integer> map = new HashMap<Character, Integer>(); //
key ：每一个字符 value ：每个字符的最大下标
            int max = 0;
            int left = 0; // 维护滑动窗口的起始位置

            for (int i = 0; i < s.length(); i++) {
                if (map.containsKey(s.charAt(i))) {
                    // 出现重复的字符时：
                    // 更新窗口的起始位置，当然要根据该命中的字符上一次出现的索引与当前窗口起始位
置做比较，取最大值
                    int hitVal = map.get(s.charAt(i)) + 1; // +1 是为了把重复的那个字符从
窗口左边移出！
                    left = Math.max(hitVal, left);
                }
                map.put(s.charAt(i), i);
                // 更新最大子串长度：用已攒的最大值与当前窗口大小作比较，取最大值
                max = Math.max(max, i - left + 1); // i - left + 1 是当前窗口的大小
            }
            return max;
    }
}
```

采用队列的方式很舒适了 那么用队列就用到底吧 使用数组而非set作为队列 给出一个相对题主较为简洁的py代码

```
class Solution(object):
    def lengthOfLongestSubstring(self, s):
        lst = []
        n = len(s)
        ans = 0
        for i in range(n):
            while s[i] in lst:
                del lst[0]  # 队首元素出队
            lst.append(s[i]) # 排除重复元素后 新元素入队
            ans = max(ans, len(lst))
        return ans
```

对c++关键代码进行解释： 新手入门，还得是看题解才能理解，以此为记录。 题解要点： 可以看得出来题解的关键部分在如下：

```
code block
```

class Solution {

public:

```
int lengthOfLongestSubstring(string s)

{

    unordered_set<char> check;
    int left = 0, max_str = 0,temp=0;
    if (s.length() != 0)
    {
        for (int i = 0; i < s.length(); i++)
        {
            while (check.find(s[i]) != check.end())
            {    //  此处下标是i,对字符串进行遍历，每进一个字符串，就判断之前的字符串里是否有相同的字符
                    //如果存在相同字符，就在已经检查过的字符串里从左到右一直删除，直到检查过的字符串里不存在与当前检查的字符相同的字符了，这个判断条件是对是否存在重复字符的关键，本题解中对于笔者的解释，由于个人理解问题，一开始不太明白这个地方，故以此作为记录

                check.erase(s[left]);
               //可见在移除已检查过的字符的同时，左边指针也在右移，实际上就是一个更新指针的过程
                left++;
            }
         //若不存在相同的字符，则将当前检查过的字符加入已检查过的字符队列中，更新不重复字符串的长度
            check.insert(s[i]);

           max_str=max(max_str,i-left+1);

        }

    }
    return max_str;

}

};

code block
```

感谢本题解提供清晰明了的思路！学习到了很多

把相似题型列举出来真的是太棒了！！！

每次ac后都感觉自己在变强，再看题解发现自己还是原来的那个自己

```
class Solution {
    public int lengthOfLongestSubstring(String s) {
       int N = s.length();
       int[] map = new int[128];  // s 由英文字母、数字、符号和空格组成, 包含着ASCII码范围内
       Arrays.fill(map,-1);       // 初始化 map 所有位置的值为 -1, 作为标记
       int start = 0, maxLength = 0, i = 0;  // start 纪录当前不重复子串的起始位置,
maxLength 纪录最长不重复子串, i 用于扫描字符串 s
       while(i < N) {
           char ch = s.charAt(i);    // 获取 s[i]
           Integer index = map[ch];  // 获取 map[ch] 上的值
           if(index != -1 && index >= start) {  // 如果 index != -1 (表示 ch 已经添加进
map中) && index >= start (表示获取的 ch 是在当前遍历不重复子串的范围内)
               maxLength = Math.max(maxLength, i - start);  // 更新maxLength, 取较大值
               start = index + 1;   // 更新 start 为重复字符 ch 所在索引 + 1
           }
           map[ch] = i;   // 更新map[ch] 的值, 该位置无论是否已经被赋值都进行更新, 因为两种情
况下都需要更新 map[ch] 的值
           i++;  // i 继续向后遍历
       }
       return Math.max(maxLength, i - start);  // maxLength 只会在进入条件语句后才更新, 如
过从 start 开始一直到字符结尾没有发现重复字符, 那么 maxLength将不会再次更新, 所以返回值还需要取
Math.max(maxLength, i - start)
    }
}
```

哭死，刚开始刷真痛苦，写点注释，增加理解。 别说了，我还没吃饭，希望自己坚持下去，我爱力扣。我爱刷题，刷题使我快乐。 每天暗示一遍，以后就养成习惯了； 写完收工，吃饭去啦，hahahaha.

```
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
       int maxstr=0;   //存放最大的长度
       int left=0;     //标记队列最左端位置
       unordered_set<char>window;
       for(int i=0;i<s.size();i++)      //循环遍历每个字符
       {
          while(window.find(s[i])!=window.end())  //开始判断是否有重复字符
          {
             window.erase(s[left]);              //有重复字符就将最左端移动出去，
             left++;                              //while循环将重复字母前的子串去除
          }
          maxstr=max(maxstr,i-left+1);            //每插入一个字符前统计长度，看是否超过临
时的最大长度
          window.insert(s[i]);
       }
       return maxstr;
    }
};
```

感谢题解！C++ 带注释版：

```
// 滑动窗口
// 每一时刻，窗口都表示一个无重复字符的子串

class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int ans = 0;
        int n = s.length();
        int left = 0;
        unordered_set<char> book; // 哈希表，记录窗口中已经出现的字符
        for (int right = 0; right < n; ++right) {
            // 当新加入窗口右侧的字符是重复字符，让窗口左边界（左指针）右移
            while (book.find(s[right]) != book.end()) {
                book.erase(s[left]);
                ++left;
            }
            // 将右指针处的字符加入窗口
            book.insert(s[right]);
            ans = max(ans, right - left + 1);
        }
        return ans;
    }
};
```

我想问模板在哪


2k+


...