# 题目

给定一个字符串，请你找出其中不含有重复字符的 最长子串 的长度。

示例 1:

输入: "abcabcbb"

输出: 3

解释: 因为无重复字符的最长子串是 "abc"，所以其长度为 3。

示例 2:

输入: "bbbbb"

输出: 1

解释: 因为无重复字符的最长子串是 "b"，所以其长度为 1。

示例 3:

输入: "pwwkew"

输出: 3

解释: 因为无重复字符的最长子串是 "wke"，所以其长度为 3。

  请注意，你的答案必须是 子串 的长度，"pwke" 是一个子序列，不是子串。

# 分析

## 方法一：滑动窗口

class Solution {

public:

int lengthOfLongestSubstring(string s) {

// 哈希集合，记录每个字符是否出现过

unordered\_set<char> occ;

int n = s.size();

// 右指针，初始值为 -1，相当于我们在字符串的左边界的左侧，还没有开始移动

int rk = -1, ans = 0;

// 枚举左指针的位置，初始值隐性地表示为 -1

for (int i = 0; i < n; ++i) {

if (i != 0) {

// 左指针向右移动一格，移除一个字符

occ.erase(s[i - 1]);

}

while (rk + 1 < n && !occ.count(s[rk + 1])) {

// 不断地移动右指针

occ.insert(s[rk + 1]);

++rk;

}

// 第i到rk个字符是一个极长的无重复字符子串

ans = max(ans, rk - i + 1);

}

return ans;

}

};

或：

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]);

}

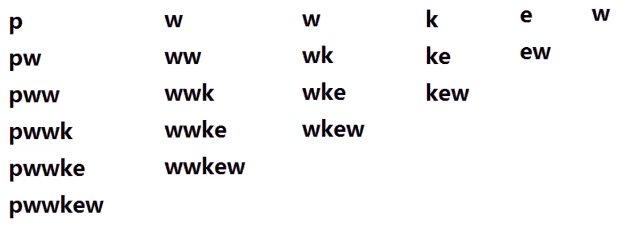
return maxStr;

}

};

## 方法二：双指针

枚举s=“pwwkew”的所有子字符串：

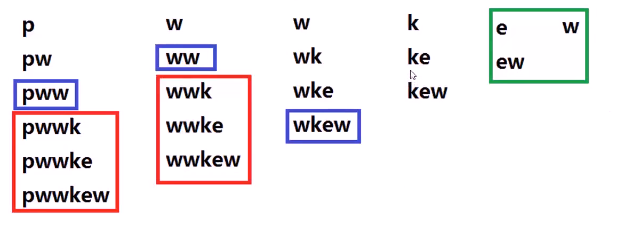


检查以上所有子字符串，是否满足无重复字符的条件，取最长的满足条件的子串作为结果。

该方法的复杂度为：O(n2)

该题目的时间复杂度优化目标是从O(n2)优化至O(n)（中间没有优化至nlogn的条件），所以需要将两重循环的枚举扫描修改为一层扫描。

题目中最关键的条件，无重复字符的子串，观察子串：



红色方框中的枚举均无意义，因为蓝色方框中已出现了重复字符。

绿色方框中的也无意义，因为后续的枚举不会出现满足条件的更优子串。

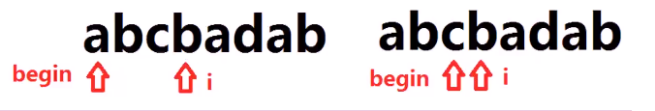
算法思路：

1. 设置一个记录字符数量的字符哈希char\_map；
2. 设置一个记录当前满足条件的最长子串变量word；
3. 设置两个指针（记作指针i与指针begin）指向字符串第一个字符；
4. 设置最长满足条件的子串的长度result；
5. i指针向后逐个扫描字符串中的字符，在这个过程中，使用char\_map记录字符数量；

5.1 如果word中没出现过该字符：对word尾部天剑字符并检查result是否需要更新；

5.2 否则：begin指针向前移动，更新char\_map中的字符数量，直到字符s[i]的数量为1；更新word，将word赋值为begin与i之间的子串。

在整个过程中，使用begin与i维护一个窗口，该窗口中的子串满足题目条件（无重复的字符），窗口线性向前华东，整体时间复杂度为O(n)。



**代码：**



**测试：**



**代码：**

#include <iostream>

#include <vector>

#include <string>

using namespace std;

/\*

题目：最长没有重复字符的子串

思路：假设所有的字符都是26个小写英文字符。以此建立hash表，hash表内记录着字符出现的位置，如果发现当前字符在以前遍历过的字符串中已经存在那么从上次出现这个字符的位置的下一个字符重新遍历那么把hash表清空，继续从当前位置开始遍历

\*/

int Longest\_substring(string& str)

{

vector<int> hash(26,-1);

int i;

int cur =0;

int maxlen =0;

int tmp;

for(i=0;i<str.size();i++)

{

if(hash[str[i]-'a'] ==-1)

{

hash[str[i]-'a'] =i;

++cur;

maxlen = maxlen > cur?maxlen:cur;

}

else

{

tmp = hash[str[i]-'a'];//此时记录的位置

i = tmp;

cur =0;

memset(&hash[0],-1,sizeof(int)\*hash.size());

}

}

return maxlen;

}

/\*

同样的思想 简洁的方法

\*/

int lengthOfLongeststring(string s)

{

vector<int> charIndex(256,-1);//这里扩展到了所有的字符

int longest =0;

int m=0;

for(int i=0;i<s.size();i++)

{

m = max(charIndex[s[i]]+1,m);

charIndex[s[i]] =i;

longest = max(longest,i-m+1);

}

return longest;

}

int main()

{

string str("abcabferafjlkcbb");

cout<<lengthOfLongeststring(str)<<endl;

return 0;

}