**Longest Substring Without Repeating Characters**

Given a string, find the length of the longest substring without repeating characters. For example, the longest substring without repeating letters for "abcabcbb" is "abc", which the length is 3. For "bbbbb" the longest substring is "b", with the length of 1.

Q求最长无重复子串,第一想到的是办法，一共有多少子串,求出最长的,模拟最长的是以哪个开始的

For I –> [0,n-1]

For j –> [i+1,n-1]

最长序列肯定是以某个元素为header 所以当equal的时候要break 说明找到以i为头的最长无重复串.

注意的是当最长串是末尾时候：如"abcdeabcedft"考虑下



还可以考虑dp

一般DP都是顺序来的从左到右不断加入字符,那么是否最长不重复子串和结尾的字符串有联系？

最长无重复字符串 肯定是以某个字符结尾的最长字串, 并且最长字串加入一个字符时候, 能通过前一个最长字串判断加入后的最长子串.

**DP方案**

前面刚刚讨论过[最长递增子序列](http://www.ahathinking.com/archives/117.html)的问题，咋一想就觉得二者有点类似，何不向DP方面想一下，为什么说二者类似，在[LIS](http://www.ahathinking.com/archives/117.html)问题中，对于当前的元素，要么是与前面的LIS构成新的最长递增子序列，要么就是与前面稍短的子序列构成新的子序列或单独构成新子序列；

同理，对于最长不重复子串，某个当前的字符，如果它与前面的最长不重复子串中的字符没有重复，那么就可以以它为结尾构成新的最长子串；如果有重复，且重复位置在上一个最长子串起始位置之后，那么就与该起始位置之后的稍短的子串构成新的子串或者单独成一个新子串。

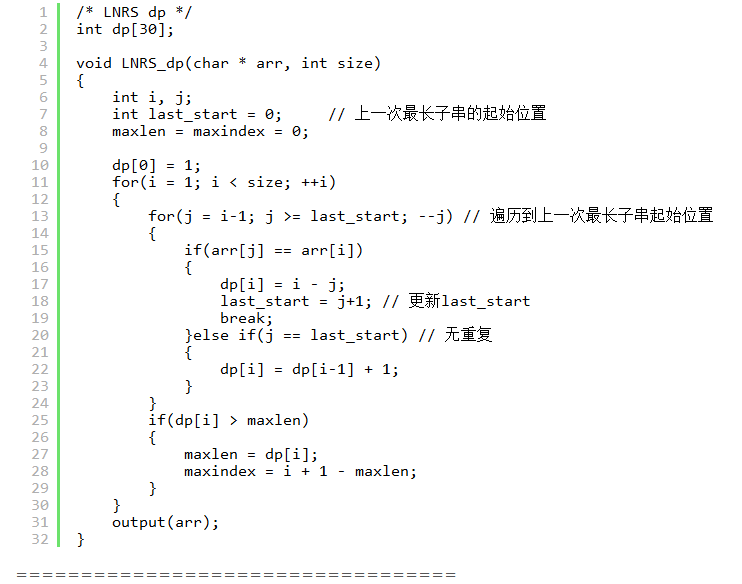
举个例子：例如字符串“abcdeab”，第二个字符a之前的最长不重复子串是“abcde”，a与最长子串中的字符有重复，但是它与稍短的“bcde”串没有重复，于是它可以与其构成一个新的子串，之前的最长重复子串“abcde”结束；

再看一个例子：字符串“abcb”，跟前面类似，最长串“abc”结束，第二个字符b与稍短的串“c”构成新的串；

这两个例子，可以看出些眉目：当一个最长子串结束时（即遇到重复的字符），**新的子串的长度是与第一个重复的字符的下标有关的，如果该下标在上一个最长子串起始位置之前，则dp[i] = dp[i-1] + 1，即上一个最长子串的起始位置也是当前最长子串的起始位置；如果该下标在上一个最长子串起始位置之后，则新的子串是从该下标之后开始的**。

于是类似LIS，对于每个当前的元素，我们“回头”去查询是否有与之重复的，如没有，则最长不重复子串长度+1，如有，则考察上一个子串起始位置与重复字符下标的关系，当然，如果DP使用O(n^2)的方案，则我们只需在内层循环遍历到上一个最长子串的起始位置即可，如下。

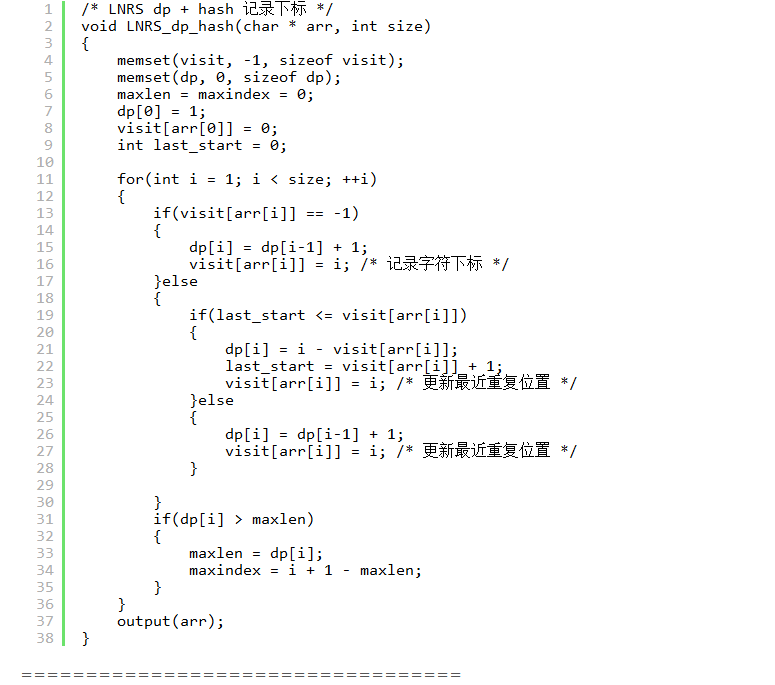
O(N^2)的DP方案，我们可以与[LIS的DP方案](http://www.ahathinking.com/archives/117.html)进行对比，是一个道理的。代码如下：



**DP + Hash 方案**

上面的DP方案是O(n^2)的，之所以是n^2，是因为“回头”去寻找重复元素的位置了，受启发于最初的Hash思路，我们可以用hash记录元素是否出现过，我们当然也可以用hash记录元素出现过的下标，既然这样，在DP方案中，我们何不hash记录重复元素的位置，这样就不必“回头”了，而时间复杂度必然降为O(N)，只不过需要一个辅助的常数空间visit[256]，典型的空间换时间。

代码如下：这样遍历一遍便可以找到最长不重复子串



DP + Hash 优化方案

写到这里，还是有些别扭，因为辅助的空间多了，是不是还能优化，仔细看DP最优子问题解的更新方程：

**dp[i] = dp[i-1] + 1;**

dp[i-1]不就是更新dp[i]当前的最优解么？这与最大子数组和问题的优化几乎同出一辙，我们不需要O(n)的辅助空间去存储子问题的最优解，而只需O(1)的空间就可以了，至此，我们找到了时间复杂度O(N)，辅助空间为O(1)（一个额外变量与256大小的散列表）的算法，代码如下：

注意：当前最长子串的构成是与上一次最长子串相关的，故要记录上一次最长子串的起始位置！