题目：

Given a string, find the length of the **longest substring** without repeating characters.

**Examples:**

Given "abcabcbb", the answer is "abc", which the length is 3.

Given "bbbbb", the answer is "b", with the length of 1.

Given "pwwkew", the answer is "wke", with the length of 3. Note that the answer must be a **substring**, "pwke" is a *subsequence* and not a substring.

体会：

首先，最最最重要的是理解题意，题目说找出最大子字符串的长度，注意是长度。因为没有要求打印出子字符串减了题目难度。

Subsequence是不重复的字符个数不是子字符串，Substring是子字符串。

例如：String : pwwkew

Subsequence : pwke substring : wke

这个问题我一直再考虑怎么连substring以及substring’s length一起打出来，并没有想出来。但是在解题过程中，我学会了：

Char[] = String.toCharArray()是把字符串string变成char[]数组并且返回。

String = String.valueOf(char[])是把char[]数组转换成为String字符串

String strStringType="my string"; //创建一个字符串变量strStringType

char[] chrCharArray; //创建一个字符数组chrCharArray

chrCharArray = strStringType.toCharArray(); //将字符串变量转换为字符数组

strStringType= String.valueOf(chrCharArray ); //将字符数组转换为字符串

解题

首先要通过String. CharAt（i）把字符串分解成一个一个的单个char字符来判断。

然后要解决的问题是没有重复的子字符串，怎么来判断没有重复？set是一个没有重复数值的数据集合，正好满足题目要求，就能判断。然后是子字符串的问题，需要按顺序一个一个比较，重复不能跳过，跳过的话记录的便是不重复的字符的个数了。

通过Set来判断方法一：

（1）判断的函数是否为无重复的字符串：将String中的每个字符按顺序一个一个的加入set中，在加入之前用set.contains（）判断，无，add加入set中，如果一个字符重复返回false，没有的话为无重复字符的字符串返回true。

public boolean allUnique(String s, int start, int end) {

Set<Character> set = new HashSet<>();

for (int i = start; i < end; i++) {

if (set.contains(s.charAt(i)))

return false;

set.add(s.charAt(i));

}

return true;

}

（2）然后判断String中的所有的子字符串。String字符串的其实位置start记为i，结束位置end纪为j，然后通过嵌套循环遍历所有子字符串，对每个子字符串判断是否为无重复的substring。Substring长度为（j-i），然后与每个子字符串长度相比，ans为判断的最长的substring长度。

public int lengthOfLongestSubstring(String s) {

int n = s.length();

int ans = 0;

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

for (int j = i + 1; j <= n; j++)

if (allUnique(s, i, j))

ans = Math.max(ans, j - i);

return ans;

}

当然这个方法通俗易懂，但是这个算法运行起来比较复杂，嵌套了三个循环，所以时间复杂度为0（n\*n\*n）

通过Set来判断方法二：

同方法一，将一个字符串拆成若干char字符挨个用set来判断，没有则加入，用i记录位置，并且实现逐个加入。如果遇到重复的char字符，需要做的是从头开始删除直到第i个字符能加入到set中位置，删除的位置用j记录，并且实现逐个删除。在每个字符串添加的时候记录i-j的位置差，因为只有无重复的char字符才能够加入到set中，删除的时候也是从头开始删除的并不是查找出重复字符才删除，所以set中为无重复字符的子字符串，i-j为set中的字符个数，也就是无重复子字符串的长度。每次都需要做个比较，输出较长的。用Math.max(a,b)来判断。

public static

int lengthOfLongestSubstring\_bySet(String s){  
 int n = s.length();  
 Set<Character> set = new HashSet<>();  
 int i=0,j=0,ans=0;  
 while(i<n&&j<n){  
 if(!set.contains(s.charAt(i))){  
 set.add(s.charAt(i));  
 i++;  
 ans=Math.*max*(ans,i-j);  
 }  
 else {  
 set.remove(s.charAt(j));  
 j++;  
 }  
 }  
 System.*out*.println(set);  
 return ans;  
}

但是set中存储的只是无重复字符串的字符并不是字符串，输出的时候是随机输出的，打印set显示的并不是无重复字符的最大子字符串，只是无重复字符的最大子字符串所包含的char字符。

算法i循环n次，j循环n次。时间复杂度为O（n+n）= O（n）

方法三：用Map解决

将字符串拆分成char字符，将每个char字符按顺序加入到Hashmap中。Map的key为char（字符），value为int（字符位置）。Map特性，每次提交重复的key值的键值对的时候会覆盖之前的同为key值的键值对。用i表示当前的位置，j表示每次重复字符（key重复）的位置（value值），因为每次map都将以前的重复字符覆盖掉相当于删除掉，当前位置i减去上个重复字符的位置j为无重复字符的长度。每次需要比较长度，取最大长度。

函数实现如下：

public static int lengthOfLongestSubstring\_byMap(String s){  
 int n = s.length();  
 int ans = 0;  
 Map<Character,Integer> map = new HashMap<>();  
 for(int i=0,j=0;i<n;i++){  
 if(map.containsKey(s.charAt(i))){  
 j=Math.*max*(map.get(s.charAt(i)),j);  
 }  
 ans = Math.*max*(ans,i-j+1);  
 map.put(s.charAt(i),i+1);  
 }  
  
 System.*out*.println(map);  
 return ans;  
}

算法只有一个循环，所以算法的复杂度为O（n）

看了解题答案后，我学会了要如何用各种集合的各种性质来解决不同的问题，set用来解决数据重复问题。

建立set集合Set<> set = new HashSet<>()

建立map集合Map<key,value> set = new HashMap<>()