3. 无重复字符串最长字串

分类：第三类：数学概念，遍历，动归的综合应用

优化思路：a.遍历可以解决，就想办法去优化遍历，排除掉某些情况

b.多种约束关系，求某个值，往数学方程组上香

c.问题可以被递归，并且能够将递归方式转化为顺序方式，想动归

可以使用暴力遍历实现，但是时间复杂度较高，O(n) > n2

S=”sdfrerrgerqeradft”

定义两个窗口值，minwindow和maxwindow

minwindow每次都从1开始增加，maxwindow全局 取minwindow的最大值保存

思考过程中错误的几点：

1. 检测到当前字符与滑动窗口里的字符相同时，没有管重复字符的位置，就直接将滑动窗口后移了一个单位
2. 考虑到要将滑动窗口的起始位置移到重复字符的后一个字符时，没有考虑之后没有被遍历的字符串里是否有与前面重复的字符，将滑动窗口的大小取值为当前滑动窗口的最大值。反例：“ggububgvfk” 在第二个u的位置遍历时，检测到与第一个u相同时，滑动窗口的起始位置变成了b，此时直接加上滑动窗口的最大值为3，下一个遍历的点变成了g，出现错误。

考虑滑动窗口时，

1是如果到了滑动窗口后移的触发条件，滑动窗口应该后移到哪里

2是滑动窗口的大小，是取当前最大值还是取最小值再慢慢开始增长

这是两个需要考虑到的地方。

优化一：使用HashSet作为滑动窗口，使查找重复字符的时间复杂度变成O(1)

需要注意的点：触发滑动窗口开始滑动后，每次只将起始位置向后滑动一位，因为此种情况下无法得知重复字符的具体位置

因为i和j初值都是0，在算滑动窗口大小时，直接是j-i



import java.util.HashSet;

class Solution {

public int lengthOfLongestSubstring(String s){

if(s == null || s.length() == 0){

return 0;

}

HashSet<Character> hashSet = new HashSet<>();

int i = 0;

int j = 0;

int length = s.length();

int maxWindow = 0;

while (i < length && j < length()){

if(!hashSet.contains(s.charAt(j))){

hashSet.add(s.charAt(j++));

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

}else {

hashSet.remove(s.charAt(i++));

}

}

return maxWindow;

}

}

优化二：是优化一的优化版，优化一中使用了set使得查找次变成O(1)，但查找到重复元素时却不能直接跳过该位置从该位置的后面算起，此时便考虑采用HashMap来进行存储，将元素与位置一起存储。

public int lengthOfLongestSubstring2(String s){

if(s == null || s.length() == 0){

return 0;

}

HashMap<Character,Integer> map = new HashMap<>();

int j = 0;

int maxWindow = 0;

for (int i = 0; i<s.length() && j < s.length() ; j++) {

if(map.containsKey(s.charAt(j))){

i = Math.max(map.get(s.charAt(j)),i);

}

maxWindow = Math.max(j-i+1,maxWindow);

map.put(s.charAt(j),j+1);

}

return maxWindow;

}

1. 找出两个有序数组的中位数，要求复杂度为O(log(m+n))

普通解法：找出中位数的位置，再遍历两个数组比较排序。时间复杂度O(m+n)复杂度没有达到要求。

正解：考虑中位数的定义，

定义：中位数在集合中是这样的数，中位数划分的前后两个子集的长度相同，且前一个子集的元素的值全部小于后一个子集的元素的值

1. 条件一的满足i从0遍历到m，而为了保证前后两个子集的大小都相同，

j = (m+n+1)/2-i。

1. 考虑将两个数组分别进行划分，划分完之后，nums1和nums2的前半部分合成前一个子集，后半部分合成后一个子集。

因为nums1[i-1] < nums1[i]只需考虑nums2[j-1] < nums1[i]

nums2[j-1] < nums2[j] 只需考虑 nums1[i-1] < nums2[j]

此时便能保证前一个子集的值都小于后一个子集的值

考虑边界值，

i=0 此时nums1[i-1]不存在，只用考虑nums2[j-1] < nums1[i]是否成立

i=m此时nums1[i]不存在,只考虑nums1[i-1]<nums2[j]是否成立

考虑越界问题

j = (m+n+1)/2-i

此时若m>n j可能会出现负数，因此要保证m<n

解答

<https://leetcode-cn.com/problems/median-of-two-sorted-arrays/submissions/>

1. 找出一个字符串中的最长回文子字符串

思路：写一个判断是否为回文字符串的函数，再遍历字符串。遍历字符串时，分为和后一个字符相同的情况’aa’和不相同的情况‘a‘。如果为回文子字符串，则考虑以i为中心，从i-m到i+m的字串。

几个没有考虑到的地方

1. 传入参数：当判断为’aa’时，传入的right不再是i+m而是i+m+1。分情况调用同一函数时要考虑不同情况传入的参数是否一致。
2. 边界条件：s.length=0时。

时间复杂度>O（n^2）

for循环内部再加了两个while循环。虽然考虑到将本次的窗口赋值为为最大窗口可以减少一些重复遍历的时间，但是时间复杂度还是至少是O(n^2)

优化一：

代码中在遍历里的while循环分不同情况时有重复代码，考虑返回回文字符串时直接返回其最长子字符串，此时已经比较过与老最长字符串的长度，返回的总是较长的那一个。原回文函数中，传入参数有五个：String s,int k ,int left,int right,boolean same

其中可以不用k，只需要传递left和right即可，也不用传递same，将right取不同的值即可。新回文函数中还传递了旧最长子回文串，用于比较长度，避免在遍历的过程中不同情况下再进行重复判断。

<https://leetcode-cn.com/problems/longest-palindromic-substring/submissions/>

1. 正则表达式匹配

我的思路：遍历s和p，判断不同的条件。但是没有AC有些情况下还是没有匹配上

比如s=”ab”,p=”.\*c”此时我的条件是只要遇到.\*就返回true。

正解：