最小包含子串的长度

【题目】

给定字符串 str1 和 str2，求 str1 的子串中含有 str2 所有字符的最小子串长度。

【举例】

str1="abcde"，str2="ac"。因为"abc"包含 str2 的所有字符，并且在满足这一条件的 str1 的所有子串中，"abc"是最短的，返回 3。

str1="12345"，str2="344"。最小包含子串不存在，返回 0。

【难度】

校 ★★★☆

【解答】

如果 str1 的长度为 N，str2 的长度为 M，本书提供的方法时间复杂度为 $O(N)$。

如果 str1 或者 str2 为空，或者 N 小于 M，那么最小包含子串必然不存在，直接返回 0。接下来讨论一般情况，即 str1 和 str2 不为空且 N 不小于 M。为了便于理解，现在以 str1="adabbca"，str2="acb"来举例说明整个过程。

1．在开始遍历 str1 之前，先通过遍历 str2 来生成哈希表 map 的一些记录如下：

Map	key	value
	'a'	1
	'b'	1
	'c'	1

哈希表记为 map，key 为 char 类型，value 为 int 型。每条记录的意义是，对于 key 字符，str1 字符串目前还欠 str2 字符串 value 个。

2．需要定义如下 4 个变量。

1）left：遍历 str1 的过程中，str1[left..right]表示被框住的子串，所以 left 表示这个子串的左边界，初始时，left=0。

2）right：right 表示被框住子串的右边界，初始时，right=0。

3）match：表示对所有的字符来说，str1[left..right]目前一共欠 str2 多少个。对本例来说，初始时，match=3，即开始时欠 1 个'a'、1 个'c'和 1 个'b'。

4）minLen：最终想要的结果为最小包含子串的长度，初始时为 32 位整数最大值。

3．接下来开始通过 right 变量从左到右遍历 str1。

1）right==0，str[0]=='a'。在 map 中把 key 为'a'的 value 减 1，减完后变为('a',0)。减完之后 value 为 0，说明减之前大于 0，那么 str1 归还了 1 个'a'，match 值也要减 1，表示对 str2 的所有字符来说，str1 目前归还了 1 个。目前变量状况如下：

map	key	value
	'a'	0
	'b'	1
	'c'	1

match==2，left==0，right==0，minLen==Integer.MAX_VALUE

2）right==1，str[1]=='d'。在 map 中，把 key 为'd'的 value 减 1，但是发现 map 中没有 key 为'd'的记录，就加一条记录('d',-1)，表示'd'字符 str1 多归还了 1 个。此时 value 为-1，说明当前这个字符是 str2 不需要的，所以 match 不变。目前变量状况如下：

map	key	value
	'a'	0
	'b'	1
	'c'	1
	'd'	-1

match==2，left==0，right==1，minLen==Integer.MAX_VALUE

3）right==2，str[2]=='a'。在 map 中，把 key 为'a'的 value 减 1，变为('a',-1)。减之后 value 为-1，说明减之前 str1 根本就不欠 str2 当前的字符，还是多归还的，故 match 不变。

map	key	value
	'a'	-1
	'b'	1
	'c'	1
	'd'	-1

match==2，left==0，right==2，minLen==Integer.MAX_VALUE

4）right==3，str[3]=='b'。('b',1)变为('b',0)，减之后 value 为 0，说明当前字符'b'归还有效，match 值减 1。

Map	key	value
	'a'	-1
	'b'	0
	'c'	1
	'd'	-1

match==1，left==0，right==3，minLen==Integer.MAX_VALUE

5）right==4，str[4]=='b'。('b',0)变为('b',-1)，减之后 value 为-1，说明当前字符'b'归还无效，match 值不变。

Map	key	value
	'a'	-1
	'b'	-1
	'c'	1
	'd'	-1

match==1，left==0，right==4，minLen==Integer.MAX_VALUE

6）right==5，str[5]=='c'。('c',1)变为('c',0)，减之后 value 为 0，说明当前字符'c'归还有效，match 值减 1。

Map	key	value
	'a'	-1
	'b'	-1
	'c'	0
	'd'	-1

match==0，left==0，right==5，minLen==Integer.MAX_VALUE

此时 match 第一次变成了 0，说明遍历到目前为止，str1 把需要归还的字符都还完了，此时被框住的子串也就是 str1[0..5]，肯定是包含 str2 所有字符的。但是当前被框住的子串是在必须以位置 5 结尾的情况下最短的吗？不一定，因为有些字符归还得很多余，所以步骤 6）还要继续如下过程。

left 开始往右移动，left==0，str1[0]=='a'，key 为'a'的记录为('a',-1)，当前 value==-1，说明 str1 即便拿回这个字符，也不会欠 str2。所以拿回来，令记录变为('a',0)，left++。left==1，str1[1]=='d'，key 为'd'的记录为('d',-1)，当前 value==-1，说明 str1 即便拿回'd'，也不会欠 str2。

所以拿回来，令记录变为('d',0)，left++。left==2，str1[2]=='a'，key 为'a'的记录为('a',0)，当前 value==0，说明 str1 如果拿回这个位置的字符，就要亏欠 str2 了，所以此时 left 停止向右移动。str1[2..5]就是在必须以位置 5 结尾的情况下的最小窗口子串。minLen 更新为 4。

步骤 6）（即 right==5）这一步揭示了整个解法最关键的逻辑，先通过 right 向右扩，让所有的字符被“有效”地还完，都还完时，被框住的子串肯定是符合要求的，但还要经过 left 向右缩的过程来看被框住的子串能不能变得更短。至此，关于位置 5 结尾的情况下的最短窗口子串已经找到。同时从 left 位置开始的最短窗口子串也是 str1[left..right]。所以，之后如果更小的窗口子串也一定不会从 left 的位置开始，而是从 left 之后的位置开始。str1[2]=='a'，令记录('a',0)变为('a',1)，match++，然后 left++。表示现在的 str1[3..5]又开始欠 str2 字符了，right 继续往右扩。目前变量的状况如下：

map	key	value
	'a'	1
	'b'	-1
	'c'	0
	'd'	-1

match==1，left==3，right==5，minLen==4

7）right==6，str[6]=='a'。('a',1)变为('a',0)，减之后 value 为 0，说明当前字符'a'归还有效，match 值减 1。match 又一次等于 0，进入 left 向右缩的过程。left==3，str1[0]=='b'，key 为'b'的记录为('b',-1)，当前 value==-1，说明 str1 即便拿回这个位置的字符，也不会欠 str2，所以拿回，记录变为('b',0)，left++。left==4，str1[1]=='b'，key 为'b'的记录为('b',0)，当前 value==0，说明如果拿回当前字符'b'，就要亏欠 str2。所以此时的 str1[4..6]就是在必须以位置 6 结尾的情况下的最小窗口子串，令 minLen 更新为 3。同步骤 6）的逻辑一样，left==4，str1[4]=='b'，令('b',0)变为('b',1)，match++，left++。表示现在的 str1[5..6]又开始欠 str2 字符，right 继续往右扩。

Map	key	value
	'a'	0
	'b'	1
	'c'	0
	'd'	-1

match==1，left==5，right==6，minLen==3

8）right==7，遍历结束。

4．如果 minLen 此时依然等于 Integer.MAX_VALUE，说明从始至终都没有符合条件的窗口出现过，当然 minLen 也从未被设置过，则返回 0，否则返回 minLen 的值。

left 和 right 始终向右移动，right 移动到右边界过程停止，所以该时间复杂度必然是 $O(N)$。具体请参看如下代码中的 minLength 方法。

```
public int minLength(String str1, String str2) {
        if (str1 == null || str2 == null || str1.length() < str2.length()) {
                return 0;
        }
        char[] chas1 = str1.toCharArray();
        char[] chas2 = str2.toCharArray();
        int[] map = new int[256];
        for (int i = 0; i != chas2.length; i++) {
                map[chas2[i]]++;
        }
        int left = 0;
        int right = 0;
        int match = chas2.length;
        int minLen = Integer.MAX_VALUE;
        while (right != chas1.length) {
                map[chas1[right]]--;
                if (map[chas1[right]] >= 0) {
                        match--;
                }
                if (match == 0) {
                        while (map[chas1[left]] < 0) {
                                map[chas1[left++]]++;
                        }
                        minLen = Math.min(minLen, right - left + 1);
                        match++;
                        map[chas1[left++]]++;
                }
                right++;
        }
        return minLen == Integer.MAX_VALUE ? 0 : minLen;
}
```

回文最少分割数

【题目】

给定一个字符串 str，返回把 str 全部切成回文子串的最小分割数。