

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。

【举例】

str="ABA"。

不需要切割，str 本身就是回文串，所以返回 0。

str="ACDCDCDAD"。

最少需要切 2 次变成 3 个回文子串，比如"A"、"CDCDC"和"DAD"，所以返回 2。

【难度】

尉 ★★★☆

【解答】

本题是一个经典的动态规划的题目。定义动态规划数组 dp ， $dp[i]$ 的含义是子串 $str[i..len-1]$ 至少需要切割几次，才能把 $str[i..len-1]$ 全部切成回文子串。那么， $dp[0]$ 就是最后的结果。

从右往左依次计算 $dp[i]$ 的值， i 初始为 $len-1$ ，具体计算过程如下：

1. 假设 j 位置处在 i 与 $len-1$ 位置之间 ($i \leq j < len$)，如果 $str[i..j]$ 是回文串，那么 $dp[i]$ 的值可能是 $dp[j+1]+1$ ，其含义是在 $str[i..len-1]$ 上，既然 $str[i..j]$ 是一个回文串，那么它可以自己作为一个分割的部分，剩下的部分（即 $str[j+1..len-1]$ ）继续做最经济的切割，而 $dp[j+1]$ 值的含义正好是 $str[j+1..len-1]$ 的最少回文分割数。

2. 根据步骤 2 的方式，让 j 在 i 到 $len-1$ 位置上枚举，那么所有可能情况中的最小值就是 $dp[i]$ 的值，即 $dp[i] = \text{Min} \{ dp[j+1]+1 \mid (i \leq j < len, \text{且 } str[i..j] \text{ 必须是回文串}) \}$ 。

3. 如何方便快速地判断 $str[i..j]$ 是否是回文串呢？具体过程如下。

1) 定义一个二维数组 $boolean[][] p$ ，如果 $p[i][j]$ 值为 $true$ ，说明字符串 $str[i..j]$ 是回文串，否则不是。在计算 dp 数组的过程中，希望能够同步、快速地计算出矩阵 p 。

2) $p[i][j]$ 如果为 $true$ ，一定是以下三种情况：

- $str[i..j]$ 由 1 个字符组成。
- $str[i..j]$ 由 2 个字符组成且 2 个字符相等。
- $str[i+1..j-1]$ 是回文串，即 $p[i+1][j-1]$ 为 $true$ ，且 $str[i]==str[j]$ ，即 $str[i..j]$ 上首尾两个字符相等。

3) 在计算 dp 数组的过程中，位置 i 是从右向左依次计算的。而对每一个 i 来说，又依次从 i 位置向右枚举所有的位置 j ($i \leq j < len$)，以此来决策出 $dp[i]$ 的值。所以对 $p[i][j]$ 来说，

$p[i+1][j-1]$ 值一定已经计算过。这就使判断一个子串是否为回文串变得极为方便。

4. 最终返回 $dp[0]$ 的值，过程结束。全部过程请参看如下代码中的 `minCut` 方法。

```
public int minCut(String str) {
    if (str == null || str.equals("")) {
        return 0;
    }
    char[] chas = str.toCharArray();
    int len = chas.length;
    int[] dp = new int[len + 1];
    dp[len] = -1;
    boolean[][] p = new boolean[len][len];
    for (int i = len - 1; i >= 0; i--) {
        dp[i] = Integer.MAX_VALUE;
        for (int j = i; j < len; j++) {
            if (chas[i] == chas[j] && (j - i < 2 || p[i + 1][j - 1])) {
                p[i][j] = true;
                dp[i] = Math.min(dp[i], dp[j + 1] + 1);
            }
        }
    }
    return dp[0];
}
```

字符串匹配问题

【题目】

给定字符串 `str`，其中绝对不含有字符 `'.'` 和 `*`。再给定字符串 `exp`，其中可以含有 `'.'` 或 `*`，`'.'` 字符不能是 `exp` 的首字符，并且任意两个 `*` 字符不相邻。`exp` 中的 `'.'` 代表任何一个字符，`exp` 中的 `*` 表示 `*` 的前一个字符可以有 0 个或者多个。请写一个函数，判断 `str` 是否能被 `exp` 匹配。

【举例】

`str="abc"`，`exp="abc"`，返回 `true`。

`str="abc"`，`exp="a.c"`，`exp` 中单个 `'.'` 可以代表任意字符，所以返回 `true`。

`str="abcd"`，`exp=".*"`。`exp` 中 `*` 的前一个字符是 `'.'`，所以可表示任意数量的 `'.'` 字符，当 `exp` 是 `"...."` 时与 `"abcd"` 匹配，返回 `true`。

`str=""`，`exp=".*"`。`exp` 中 `*` 的前一个字符是 `'.'`，可表示任意数量的 `'.'` 字符，但是 `".*"` 之前还有一个 `'.'` 字符，该字符不受 `*` 的影响，所以 `str` 起码有一个字符才能被 `exp` 匹配。所以返