知乎 首发于 大话 CS

一文让你彻底明白马拉车算法

windliang

热爱编程，认真答题，感谢关注

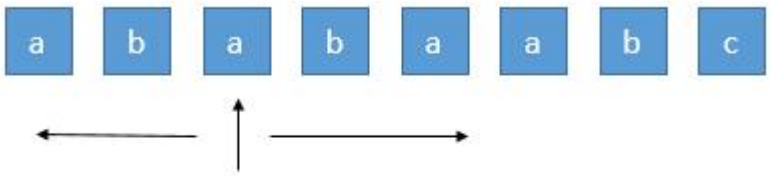
缘起

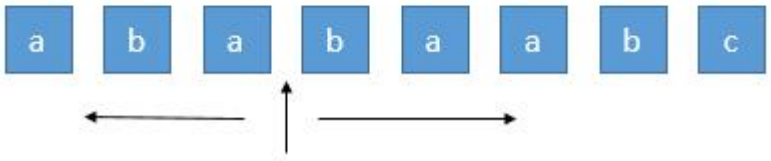
对应于 leetcode 的第 5 题，给定一个字符串，然后输出这个字符串包含的最长回文子串。例如，"cbabfd" 的最长回文子串就是 "bab"。大概是去年刷到的这个题，当时有一种马拉车的算法来解决这个问题。记得当时理解了好几天才明白，当时也总结了一下。这几天看到知乎又有人问这个算法，索性就把这个算法单独拿出来总结一下，在之前的总结上再讲的详细一点。

中心扩展算法

我们先来看一个简单的算法，来解决这个问题。

我们知道回文串一定是对称的，所以我们可以每次循环选择一个中心，进行左右扩展，判断左右字符是否相等即可。





由于存在奇数的字符串和偶数的字符串，所以我们需要从一个字符开始扩展，或者从两个字符之间开始扩展，所以总共有

```java
public String longestPalindrome(String s) {
    if (s == null || s.length() < 1) return "";
    int start = 0, end = 0;
    for (int i = 0; i < s.length(); i++) {
        int len1 = expandAroundCenter(s, i, i); //从一个字符扩展
        int len2 = expandAroundCenter(s, i, i + 1); //从两个字符之间扩展
        int len = Math.max(len1, len2);
        //根据 i 和 len 求得字符串的相应下标
        if (len > end - start) {
            start = i - (len - 1) / 2;
            end = i + len / 2;
        }
    }
    return s.substring(start, end + 1);
}

private int expandAroundCenter(String s, int left, int right) {
    int L = left, R = right;
    while (L >= 0 && R < s.length() && s.charAt(L) == s.charAt(R)) {
        L--;
        R++;
    }
    return R - L - 1;
}
```

时间复杂度：O（n^2）。两层循环，每层循环都是遍历每个字符。

空间复杂度：O（1）。

Manacher's Algorithm 马拉车算法。

> 马拉车算法 Manacher‘s Algorithm 是用来查找一个字符串的最长回文子串的线性方法，由一个叫Manacher的人在1975年发明的，这个方法的最大贡献是在于将时间复杂度提升到了线性。

主要参考了下边链接进行讲解。

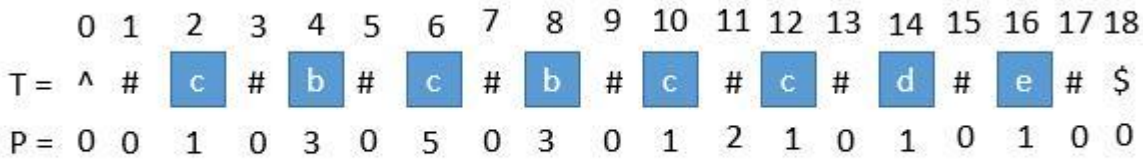
segmentfault.com/a

ju.outofmemory.cn/entry...

articles.leetcode.com/l...

首先我们解决下奇数和偶数的问题，在每个字符间插入"#"，并且为了使得扩展的过程中，到边界后自动结束，在两端分别插入 "^" 和 "$"，两个不可能在字符串中出现的字符，这样中心扩展的时候，判断两端字符是否相等的时候，如果到了边界就一定会不相等，从而出了循环。经过处理，字符串的长度永远都是奇数了。

首先我们用一个数组 P 保存从中心扩展的最大个数，而它刚好也是去掉 "#" 的原字符串的总长度。例如下图中下标是 6 的地方。可以看到 P[6] 等于 5，所以它是从左边扩展 5 个字符，相应的右边也是扩展 5 个字符，也就是 "#c#b#c#b#c#"。而去掉 # 恢复到原来的字符串，变成 "cbcbc"，它的长度刚好也就是 5。



T：处理后的数组 P：从中心扩展的长度

求原字符串下标

用 P 的下标 i 减去 P [i]，再除以 2，就是原字符串的开头下标了。

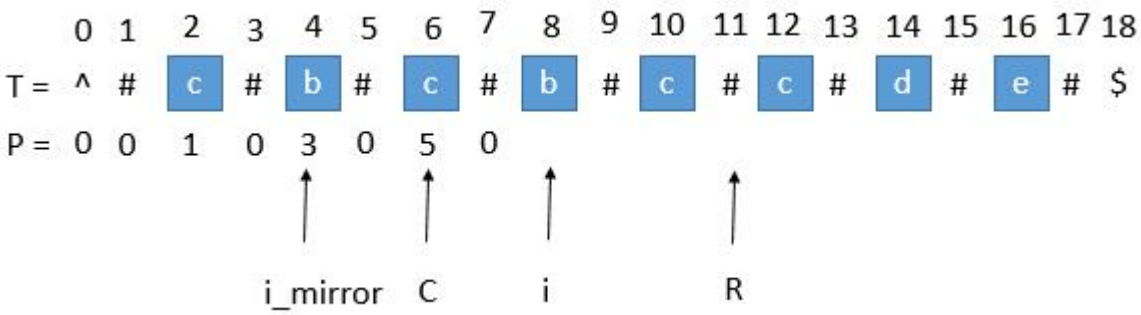
例如我们找到 P[i] 的最大值为 5，也就是回文串的最大长度是 5，对应的下标是 6，所以原字符串的开头下标是 （6 - 5 ）/ 2 = 0 。所以我们只需要返回原字符串的第 0 到 第 （5 - 1）位就可以了。

接下来是算法的关键了，它充分利用了回文串的对称性。

我们用 C 表示回文串的中心，用 R 表示回文串的右边半径。所以 R = C + P[i] 。C 和 R 所对应的回文串是当前循环中 R 最靠右的回文串。

让我们考虑求 P [i] 的时候，如下图。

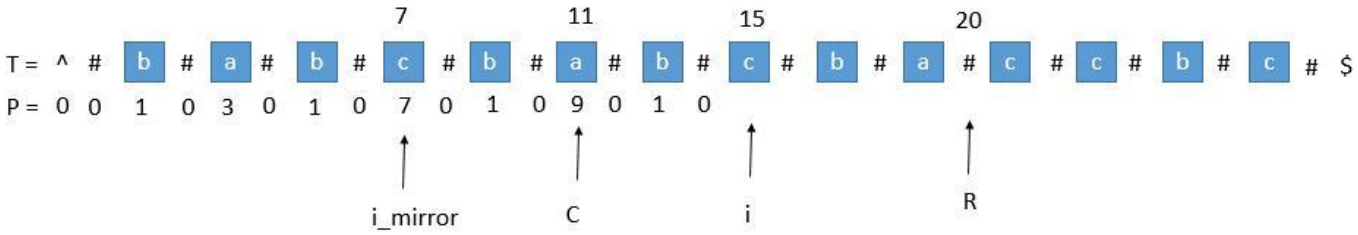
用 i_mirror 表示当前需要求的第 i 个字符关于 C 对应的下标。



我们现在要求 P [i]， 如果是用中心扩展法，那就向两边扩展比对就行了。但是我们其实可以利用回文串 C 的对称性。i 关于 C 的对称点是 i_mirror ， P [i_mirror] = 3，所以 P [i] 也等于 3 。

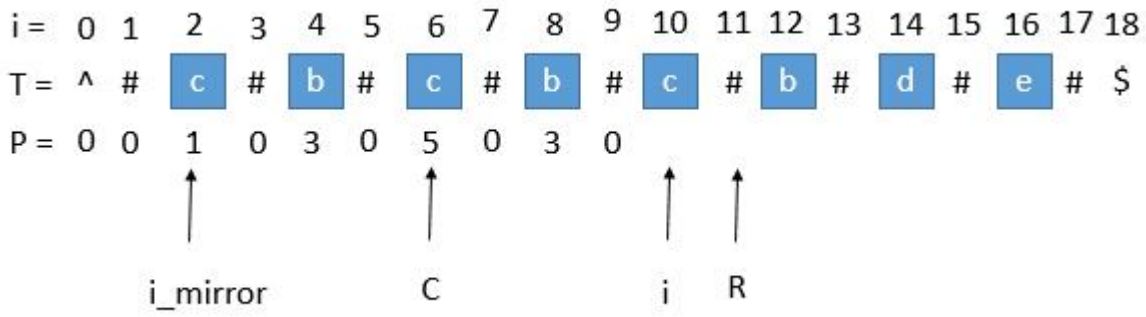
但是有三种情况将会造成直接赋值为 P [i_mirror] 是不正确的，下边一一讨论。

1. 超出了 R



当我们要求 P [i] 的时候，P [mirror] = 7，而此时 P [i] 并不等于 7 ，为什么呢，因为我们从 i 开始往后数 7 个，等于

2. P [i_mirror] 遇到了原字符串的左边界



此时P [i_mirror] = 1，但是 P [i] 赋值成 1 是不正确的，出现这种情况的原因是 P [i_mirror] 在扩展的时候首先是 "#" == "#"，之后遇到了 "^"和另一个字符比较，也就是到了边界，才终止循环的。而 P [i] 并没有遇到边界，所以我们可以继续通过中心扩展法一步一步向两边扩展就行了。

3. i 等于了 R

此时我们先把 P [i] 赋值为 0，然后通过中心扩展法一步一步扩展就行了。

考虑 C 和 R 的更新

就这样一步一步的求出每个 P [i]，当求出的 P [i] 的右边界大于当前的 R 时，我们就需要更新 C 和 R 为当前的回文串了。因为我们必须保证 i 在 R 里面，所以一旦有更右边的 R 就要更新 R。

此时的 P [i] 求出来将会是 3 ， P [i] 对应的右边界将是 10 + 3 = 13，所以大于当前的 R ， 我们需要把 C 更新成 i 的值，也就是 10 ， R 更新成 13。继续下边的循环。

```
public String preProcess(String s) {
    int n = s.length();
    if (n == 0) {
        return "^$";
    }
    String ret = "^";
    for (int i = 0; i < n; i++)
        ret += "#" + s.charAt(i);
    ret += "#$";
    return ret;
}

// 马拉车算法
public String longestPalindrome2(String s) {
    String T = preProcess(s);
    int n = T.length();
    int[] P = new int[n];
    int C = 0, R = 0;
    for (int i = 1; i < n - 1; i++) {
        int i_mirror = 2 * C - i;
        if (R > i) {
            P[i] = Math.min(R - i, P[i_mirror]);// 防止超出 R
        } else {
            P[i] = 0;// 等于 R 的情况
        }

        // 碰到之前
        while (T.c
```

```
            // 判断是否需要更新 R
            if (i + P[i] > R) {
                C = i;
                R = i + P[i];
            }

        }

        // 找出 P 的最大值
        int maxLen = 0;
        int centerIndex = 0;
        for (int i = 1; i < n - 1; i++) {
            if (P[i] > maxLen) {
                maxLen = P[i];
                centerIndex = i;
            }
        }
        int start = (centerIndex - maxLen) / 2; //最开始讲的求原字符串下标
        return s.substring(start, start + maxLen);
    }
```

时间复杂度：for 循环里边套了一层 while 循环，难道不是 O (n^2)？不！其实是 O (n)。不严谨的想一下，因为 while 循环访问 R 右边的数字用来扩展，也就是那些还未求出的节点，然后不断扩展，而期间访问的节点下次就不会再进入 while 了，可以利用对称得到自己的解，所以每个节点访问都是常数次，所以是O (n)。

空间复杂度：O（n）。

总

最后感叹一下，提出马拉车算法的人太天才了，这个算法太美妙了，哈哈。

更多详细通俗的 leetcode 题解可以关注下边的知乎专栏，持续更新。

知乎 首发于 大话 CS

编辑于 02-04

算法 马拉车算法 ACM

文章被以下专栏收录

大话 CS
计算机相关的基础、应用，公众号 windliang

进入专栏

推荐阅读

KMP 算法详解

KMP 算法（Knuth-Morris-Pratt 算法）是一个著名的字符串匹配算法，效率很高，但是确实有点复杂。 很多读者抱怨 KMP 算法无法理解，这很正常，想到大学教材上关于 KMP 算法的讲解，也不知道...

labuladong

[LeetCode] 5. 最长回文子串

九四干

字符串回文问题最优解 - 算法（Manacher's...

概述在计算机科学中，马拉（Manacher's Algor主要解决最长回文子串问题以在 线性时间内找出给定字从任意位置开始的所有回文最长回文子串给定一个字符

胖头鱼 发表于

19 条评论

切换为时间排序

写下你的评论...

瑟箜

其实可以不插入

▲ 赞同 76 ▼ 19 条评论 分享 喜欢 收藏 ...

知乎 首发于 **大话 CS**

王赟 Maigo 2019-07-12

马拉车算法……这翻译……

1

windliang (作者) 回复 王赟 Maigo 2019-07-12

哈哈，开始我还以为和马拉车有什么关系

赞

郑晓白 2019-10-29

最开始介绍中心C的时候，那个恒等式应该是 R=C+P[C]吧？(而不是P[i])

赞

JerryChang 2019-12-10

还没完全弄懂，但是是个人认为看到最好的讲解，感谢！

赞

水货 回复 JerryChang 05-09

推荐这个：cxyxiaowu.com/2665.html

个人认为最详细，最从读者出发的，最能理解透彻的讲解

1

JerryChang 回复 水货 05-20

收到！感谢！

赞

知乎用户 2019-12-22

楼主666，是我见过的讲的最通俗的

赞

BraveY

知乎 首发于 **大话 CS**

赞

windliang (作者) 回复 BraveY 02-19

受样例的数目以及类型的影响

1

知乎用户 04-15

看懂了！谢谢

赞

Shark 06-10

你好，有个疑问。P[8]不应该等于5么

赞

windliang (作者) 回复 Shark 06-10

算的是一个方向的

赞

Shark 回复 windliang (作者) 06-10

明白了。多谢。这篇确实是网上说的最明白的文章了。

赞

长空1452 07-03

其实本质就是充分利用了对称性, 镜像节点就不用重复计算了, 不属于对称范围的话,归根到底还是需要一步一步去匹配.

1

微雨落花 07-20

最开始的那个算法应该是return R-L+1不嘞?

赞

kifish 回复 微雨落花 08-03

(R-1) - (L+1) + 1

赞

释下么

赞

windliang (作者) 回复 gavin 08-03

看上一条评论 kifish 的回复，跳出循环的时候， L和 R 多向外扩展了一次

赞