

Felix's Blog

Time goes on, life goes on.

[首页](#)[登入](#)[RSS](#)[注册](#)[留言](#)[链接](#)[归档](#)[关于](#)[日志标题 ▾](#)[站内](#)[谷歌](#)[联系我](#)

felix021[#]gmail.com

[最新评论](#)

1.KeepAlive机制很多情况无法..
do_accept是否存在问题? sock..
芝麻糊LOL
主板?
我记得我以前那个台式机在某..
查看上下月这种操作...我都..
我在想额外的两个问题: 1. 能..
我一般是 持续一段时间吃一套..
喝多了不见得对男人好, 女的..
少喝豆浆, 建议完毕.

[分类](#)

杂碎 [12] 
IT [705] 
操作系统 [125] 
Python [9] 
探索设计模式 [5] 
软件 [84] 
硬件 [40] 
手机 [12] 
程序设计 [162] 
网络 [195] 
数据库 [18] 
病毒 [7] 
其他 [48] 

[其他](#)

[登入](#)
[注册](#)
RSS: [日志](#) | [评论](#)
编码: UTF-8
XHTML 1.0

[统计](#)


访问次数 2070983
今日访问 595
日志数量 2097
评论数量 2291
引用数量 1
留言数量 143
注册用户 295
在线人数 15

[链接](#)

默认链接组
[WHU微软俱乐部](#)
朋友们的据点
[czyhd's Blog](#)
[姜南的BLOG](#)

[无聊的scanf](#)[非递归二分查找一个元素在有序数组中应处的位置](#)

★ Manacher's ALGORITHM: O(n)时间求字符串的最长回文子串

类别: [IT](#) » [程序设计](#) | [felix021](#) @ 2011-10-13 12:00 | [评论\(1\)](#) | [阅读\(21546\)](#)[大](#) | [中](#) | [小](#) Translated to [ENGLISH VERSION](#)

源于这两篇文章:

<http://blog.csdn.net/ggqgnypgjj/article/details/6645824><http://zhuhongcheng.wordpress.com/2009/08/02/a-simple-linear-time-algorithm-for-finding-longest-palindrome-sub-string/>

这个算法看了三天, 终于理解了, 在这里记录一下自己的思路, 免得以后忘了又要想很久- -.

首先用一个非常巧妙的方式, 将所有可能的奇数/偶数长度的回文子串都转换成了奇数长度: 在每个字符的两边都插入一个特殊的符号。比如 abba 变成 #a#b#a#, aba变成#a#b#a#。 为了进一步减少编码的复杂度, 可以在字符串的开始加入另一个特殊字符, 这样就不用特殊处理越界问题, 比如\$#a#b#a# (注意, 下面的代码是用C语言写就, 由于C语言规范还要求字符串末尾有一个'\0'所以正好OK, 但其他语言可能会导致越界)。

下面以字符串12212321为例, 经过上一步, 变成了 S[] = "\$#1#2#2#1#2#3#2#1#";

然后用一个数组 P[i] 来记录以字符S[i]为中心的最长回文子串向左/右扩张的长度 (包括S[i]), 比如S和P的对应关系:

```
S # 1 # 2 # 2 # 1 # 2 # 3 # 2 # 1 #  
P 1 2 1 2 5 2 1 4 1 2 1 6 1 2 1 2 1  
(p.s. 可以看出, P[i]-1正好是原字符串中回文串的总长度)
```

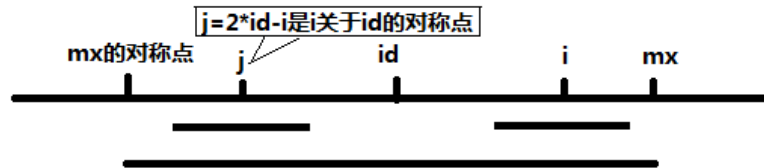
那么怎么计算P[i]呢? 该算法增加两个辅助变量 (其实一个就够了, 两个更清晰) id和mx, 其中id表示最大回文子串中心的位置, mx则为id+P[id], 也就是最大回文子串的边界。

然后可以得到一个非常神奇的结论, 这个算法的关键点就在这里了: 如果mx > i, 那么P[i] >= MIN(P[2 * id - i], mx - i)。就是这个串卡了我非常久。实际上如果把它写得复杂一点, 理解起来会简单很多:

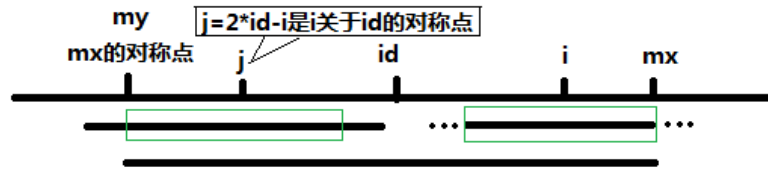
```
//记j = 2 * id - i, 也就是说 j 是 i 关于 id 的对称点。  
if (mx - i > P[j])  
    P[i] = P[j];  
else /* P[j] >= mx - i */  
    P[i] = mx - i; // P[i] >= mx - i, 取最小值, 之后再匹配更新。
```

当然光看代码还是不够清晰, 还是借助图来理解比较容易。

当 mx - i > P[j] 的时候, 以S[j]为中心的回文子串包含在以S[id]为中心的回文子串中, 由于 i 和 j 对称, 以S[i]为中心的回文子串必然包含在以S[id]为中心的回文子串中, 所以必有 P[i] = P[j], 见下图。



当 P[j] >= mx - i 的时候, 以S[j]为中心的回文子串不一定完全包含于以S[id]为中心的回文子串中, 但是基于对称性可知, 下图中两个绿框所包围的部分是相同的, 也就是说以S[i]为中心的回文子串, 其向右至少会扩张到mx的位置, 也就是说 P[i] >= mx - i。至于mx之后的部分是否对称, 就只能老老实实去匹配了。

[Kid的原创Blog](#)[\[GCC\]Feli](#)[彼岸·花开·荼靡](#)[谁见过风?](#)[不敢流泪](#)[Liuw's Thinkpad](#)[Pumpkin's](#)[FOUR](#)[快乐云的小屋](#)[张磊的blog](#)[Felix's Blog](#)[Zhang Jiuan's notes](#)[Frank's Blog](#)[intijk](#)[{Sunny's Blog}](#)[PortWatcher's Blog](#)[链接链接^_^](#)[mlzy](#)[LiuYu's LOGS](#)[dutor](#)

对于 $mx \leq i$ 的情况，无法对 $P[i]$ 做更多的假设，只能 $P[i] = 1$ ，然后再去匹配了。

于是代码如下：

```
//输入，并处理得到字符串s
int p[1000], mx = 0, id = 0;
memset(p, 0, sizeof(p));
for (i = 1; s[i] != '\0'; i++) {
    p[i] = mx > i ? min(p[2*id-i], mx-i) : 1;
    while (s[i + p[i]] == s[i - p[i]]) p[i]++;
    if (i + p[i] > mx) {
        mx = i + p[i];
        id = i;
    }
}
//找出p[i]中最大的
```

OVER.

#UPDATE@2013-08-21 14:27

@zhengyuee 同学指出，由于 $P[id] = mx$ ，所以 $S[id-mx] \neq S[id+mx]$ ，那么当 $P[j] > mx - i$ 的时候，可以肯定 $P[i] = mx - i$ ，不需要再继续匹配了。不过在具体实现的时候即使不考虑这一点，也只是多一次匹配（必然会fail），但是却要多加一个分支，所以上面的代码就不改了。

--

转载请注明出自 <http://www.felix021.com/blog/read.php?2040>，如是转载文则注明原出处，谢谢:)

RSS订阅地址：<http://www.felix021.com/blog/feed.php>。

0

人人分享 | IT » 程序设计 | 引用(0) |

wayne 2011-10-13 21:17

瞄了一眼，有点像用后缀数组的那种方法的感觉

felix021 回复于 2011-10-14 08:22

曾经试图去看后缀数组，最后看晕了-.-

分页: 1/1 ◀ 1 ▶

