回文树的构建及其应用

Victor Wonder

2015 年 3 月 7 日

1 前言

Palindromic Tree，中文名应当为“回文树”，是一种专门用于处理回文串的数据结构，类似于Manacher算法，但比Manacher算法更加强大。具体可参照BZOJ 3676一题，回文树能以目前最短的代码和最快的速度解决这道题。

根据参考资料[1]和[3]，该数据结构由Mikhail Rubinchik发明，并在Petrozavodsk[1] Summer Training Camp 2014中对该数据结构进行了介绍。所以，相比较后缀树、AC自动机等老牌字符串数据结构和算法，回文树是一个新兴的强有力的字符串处理数据结构（雾），我觉得有一定的必要进行推广。

2 概念

2.1 定义

以下是本文中要用到的定义，部分翻译自参考资料，部分是我为了方便起见在本文中做出的定义。

原串[2]：我们需要进行处理的字符串。

加入串：已经加入到回文树中的字符串，是原串的一条前缀。

特殊串：从第i个位置开始的最长回文串。

最长回文后缀：一条字符串最长的回文后缀，可以为空串。

结点：回文树上的每个结点都代表着原串的一条本质不同的回文子串。子结点代表的字符串是父结点代表的字符串在最左边和最右边各加上一个字符c[3]形成的串。

后缀链[4]：每一个结点都有一条后缀链，除去回文树的两根（关于这一点，我将在后面解释）以外，每个结点后缀链所指向的结点所代表的字符串为当前结点所代表字符串的最长回文后缀。

2.2 符号

我们还需要定义一些符号。

S：表示原串。

p：表示加入串。

t：表示加入串的最长回文后缀。

n：在本文中，如无特殊说明，n均表示需要原串的长度。

m：m为原串的字符集大小。

$Len(p)$：表示字符串p的长度。

[1]彼得罗扎沃茨克市，俄罗斯卡累利阿自治共和国首都和经济、文化中心

[2]普遍叫法为母串，但是我更习惯叫原串，所以本文都将以原串这个说法出现

[3]这里的c并不是指小写字母c，而是一个变量，指代任意字符甚至是数字

[4]此处的后缀链与后缀树中的后缀链不同，类似于AC自动机中的fail指针

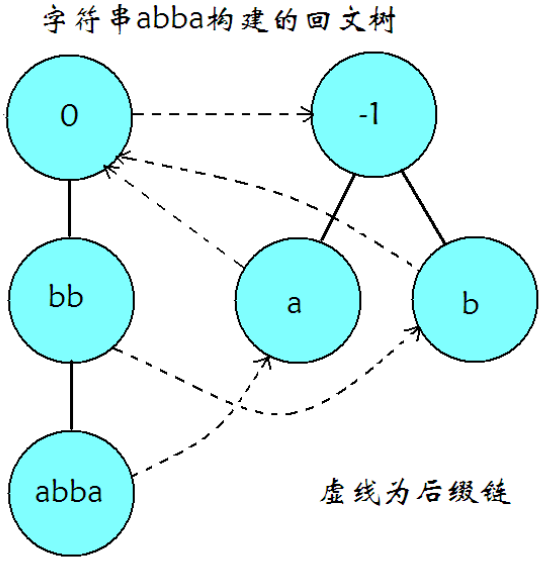
$cnt[i]$：结点i代表的字符串在原串中出现的次数，要注意，建树时其值是不准确的，需要在建树完成之后重新计算一遍，重新计算的复杂度是$O(n)$的。

2.3 形态

回文树其实是由两棵树和许多后缀链构成的，从结构上来看应该算是森林。其中的一棵树储存长度是偶数的回文串，另一棵树储存长度是奇数的回文串，我们定义他们为$tree0$和$tree1$。

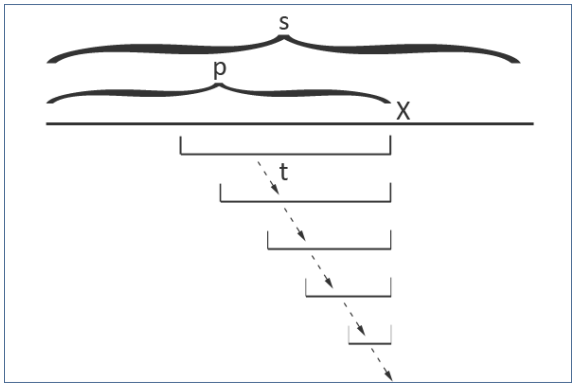
$tree0$的根是空串，长度为0，而$tree1$的根则比较有趣，长度为-1。我们每一次扩展一个结点，都是在原先结点的两边加上一个字符，也就是说，新结点的长度是原结点的长度加2，一个字符也算是回文串，方便起见，$tree1$根的长度应当定义为-1。另外，在一开始的时候，$tree0$的根需要向$tree1$的根连一条后缀链，$tree1$根的后缀链指向的则是它自己。

下面是回文树的一个例子：



3 构建[1]

如图，我们要在加入串p的后面加入一个新的字符x构成一条新的加入串p′，t为加入串p的最长回文后缀。图中的虚线为后缀链，指向的是加入串p的所有回文后缀。

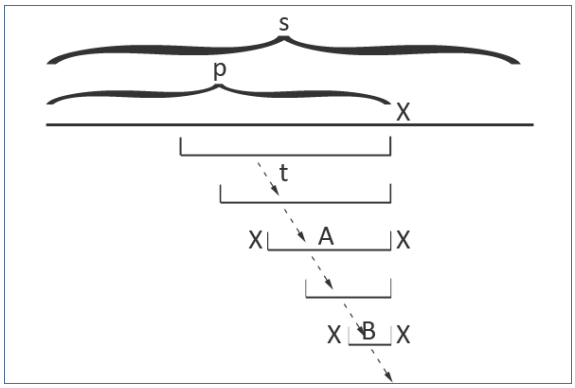


我们需要找到p的一条回文后缀A，使得A的左端是一个字符x，因为A本身就是回文串，那么在其左右各加上一个字符x之后仍然是一条回文串，而且，当A长度最长时，形成的回文串xAx就是t′,即p′的最长回文后缀。A可以是空串甚至是长度为-1的串。

[1]本部分所有图片均来自参考资料[1]

这里就又体现出-1的优越性了，我们判定条件字符串A符合要求的条件是这样：$s[Len(p')-Len(A)-1]=s[Len(p')]$，当新加入的字符x自己构成一条回文串的时候，$Len(A)=-1$，我们不需要再加上额外的判定语句。

如果已经有一个结点i代表xAx了，那么我们就将t改成i，退出该过程就行了，否则我们需要新建一个结点，并且还需要加入一条后缀链，具体方法也是一样，继续沿着后缀链找，直到找到符合要求的字符串B为止。



4 证明

回文树的构建大致就是这么回事，如果还有不懂的可以自己根据我给出的代码YY。下面我们要开始复杂度的证明。

作为一名民间科学家($Level\ 0$)，我很讨厌证明，算法、数据结构只要会用就行了。不过这是一篇严谨的文章，我觉得还是有必要证明一下复杂度什么的，如果不喜欢看证明的话，直接跳过吧。

4.1 一条定理

首先，让我们来看一条定理：

长度为n的字符串中本质不同的回文串的数目最多为n。

我想很多人应该都是知道这个定理的，但是，我上面说过，我只是一个民科，我不知道这个定理该如何证明。去网上搜索了一下，估计是因为我的搜索姿势不太对，没有搜索到相关资料。好在最后还是证明了出来，我觉得有必要分享一下证明方法。

4.1.1 Manacher证明法

Claris向我推荐了这种证明方法，据说业界人士都是用这种方法证明的→ _ →。不过我却难以接受这种证明方法，我的第六感告诉我，这种证明方法似乎有什么地方不太对，但从逻辑上来说这种证明方法应该是没有问题的。

首先，我们都知道Manacher算法的复杂度是$O(n)$的。Manacher算法只会往右扩展而不会往左扩展，因此最多扩展n次，而每一次扩展都是找到了一条本质不同的回文串，所以，本质不同回文串的数目不超过n。[1]

4.1.2 回文串性质证明法

这种方法是我自己在英语课的时候YY出来的。大致的证明过程如下：

[1]大致是这个意思，具体请参照参考资料[4]

我们上面已经定义过了特殊串，即从某一位置开始的最长回文子串，因为特殊串的数目只有n（从任意位置开始都只有一条特殊串），所以我们只要证明，所有回文串都一定和某条特殊串相同，那么定理就得证了。而这一点是很显然的。

我们假设从i开始的特殊串为S，从i开始的其他回文子串为T，必然有|T| < |S|。因为回文串是对称的，所以，S要么是TAT这样的字符串，要么是BCB（B同时为T的前缀和后缀，T=B+C[1]）这样的字符串，这个时候，如果T是从i+|S|-|T|-1位置开始的特殊串，那么定理得证，否则，我们可以一直对称下去，直到某个位置j，使得T是该位置的特殊串。

4.2 复杂度证明

4.2.1 空间复杂度

空间复杂度最主要在于每个结点对于其子结点的储存，一般情况下都采取类似于Trie的储存方式，即每个结点都用nxt[][c]数组记录该结点代表的字符串在左右各增加一个字符c之后指向的编号。这样的话，复杂度是$O(nm)$[2]。如果内存卡的比较紧的话，可以用链表存树，时间复杂度要高一些，如果不是故意卡的话[3]，应该是没有关系的。当然，也可以用map存，一般情况下就卡不掉了。

4.2.2 时间复杂度

参考资料[1]给出的时间复杂度是$O(n)$，但是参考资料[2]给出的时间复杂度是$O(nlogm)$，参考资料[3]给出的时间复杂度是$O(nlog\sum)$，我猜测$\sum$应该也是指字符集大小。

我至今为止也不知道$O(nlogm)$是怎么证明出来的，所以我只给出$O(n)$的证法，不过没有关系，一般情况下$logm$都是很小的，我们就当它是常数忽略了吧。我的证明是这样的：最长回文后缀的对称中心是只会往右移，不会往左移的，比如说dabcbad开始的时候对称中心是1，随着一个个字符的加入，不停地往右移，直到c这个位置之后，再继续加入字符，最长回文后缀的对称中心是不会往左回移的。最长回文后缀的对称中心最多往右移动n次，所以复杂度是$O(n)$的。

结果还是出了点新状况，上面说过，我们建立一个新的结点之后，我们需要重新建立一条后缀链，这个时候，我们需要往后继续走，万一后缀链很长呢？那岂不是要走很久？实际上这点是不用担心的，不管后缀链有多长，我们每个结点只会访问1次，而整棵树最多有n+2个结点，所以复杂度仍旧是$O(n)$的。

另：有人提出回文树的常数比较大，我无法算出回文树的常数是多少，但是根据我的这种证法，回文树的常数应该不是很大才对，如果m特别大的话则另当别论。

5 应用

5.1 基础应用

回文树的基础应用比较少，只有三条。

1、在原串最后再加入一个字符会产生新的多少回文串，我们只要看看新加入一个结点或者原有结点后缀链的长度是多少就行了，可以用一个数组记录一下，一般情况下可以做到$O(1)$查询。

2、查询原串中一共有多少本质不同的字符串，数一下结点数就行了，$O(1)$查询。

3、查询原串中每条回文串的出现次数，这个时候就要用到上面说的cnt数组了，cnt[i]的值为结点i代表的字符串作为最长回文后缀的次数，在建树的过程中就可以统计出来，但这样子记录出来的值是不完整的，我们需要从后到前更新一遍获得答案。

[1]这里的+相当于c++中字符串的加法，即将两个字符串连接起来

[2]上文说过，m代表字符集大小

[3]比如：字符集大小为n，每一个字符都不重复，那么$tree1$就要连出去n条边，形成了菊花图

【例1】回文串[1]

考虑一个只包含小写拉丁字母的字符串s。我们定义s的一个子串t的“出现价值”为t在s中的出现次数乘以t的长度。请你求出s的所有回文子串中的最大“出现价值”。

$1 \leq |s| \leq 300000$

这道题标算是Manacher求回文串+后缀数组求字符串出现次数，也有人用hash过了这道题，但是，学会回文树之后，这道题目就变成了模板题。先建出回文树，然后再求出cnt数组的值，最后再扫一遍求出答案即可。

5.2 其他应用

我们用类似于Manacher的做法，在每两个字符中间都加上一个#，那么最后回文串的长度就都是奇数了，原本的两棵树也可以并成一棵正常的树了。

既然已经是树形结构了，那么就可以开始各种乱搞了，比如说可持久化、树套树什么的。还可以出树形DP，或者根据后缀链建树乱搞。不过好像由于回文的限制，前景好像不是很乐观？毕竟大多数情况下用Manacher似乎就可以解决了。

参考文献

[1] 《Palindromic tree.》 Adilet Zhaxybay

http://adilet.org/blog/25-09-14/

[2] 《Palindromic Tree——回文树【处理一类回文串问题的强力工具】》 YueYueZha

http://blog.csdn.net/u013368721/article/details/42100363

[3] 《Palindromic tree: behind the scenes》 adamant

http://codeforces.com/blog/entry/13959

[4] 《浅谈回文子串问题》 江苏省常州高级中学 徐毅

[1] 题目来源：APIO 2014