jmyyc sak

PALINDROMIC TREE

By dy

PREFACE

- 回文树是一种新兴的数据结构，由Mikhail Rubinchik在2015年发表。
- 由于其简单的实现以及高效性，许多回文问题都有了优美的解决方法。
- 本文将着重介绍回文树，并展示回文树强大的可塑性与解决问题的能力。

CONTENTS

- 基础构造
- 拓展
- 应用

基础构造

- 约定
- 回文树的结构
- 基础插入算法
- 复杂度证明
- 基础应用

约定

- Σ 表示字符集大小
- $|s|$ 表示串 s 的长度
- 设有串t与字符c，记tc表示在t后接上字符c对应的字符串。ct表示在t前接上字符对应的字符串。
- 一个串t为s的回文后缀，当且仅当$t \neq s$，且t为s的一个后缀且t是回文的。s的最长回文后缀定义为长度最长的，且不为s的回文后缀的串。注意一个串可能不存在回文后缀，定义他的回文后缀只包含一个空串。

回文树的结构

- 一个串S的回文树为一个森林，由两棵树组成。设两棵树的根分别为*even*, *odd*。树上每个节点对应着一个字符串，除根外与s的回文子串一一对应。树上每条边都有对应的一个字符，且满足一个点的所有出边对应的字符各不相同。
- 对于树上一个点u，令 $fa[u]$ 为其父亲，则u对应的字符串为$fa[u]$对应的字符串在两端加上连接u与$fa[u]$的边对应的字符c。
- 特别的，对于根*even*，令其对应的字符串为空串，根*odd* 对应的字符串为一个长度为−1 的实际并不存在的字符串，*odd*的儿子对应的字符串为连出边对应的字符。

一些定义

- $len[u]$：节点u代表的回文串的长度。

  特别地 $len[even] = 0, len[odd] = -1$。

- $ch[u][c]$：节点u的儿子，其连边上的字符为c。

- $fail[u]$：u的最长回文后缀在回文树上对应的节点。

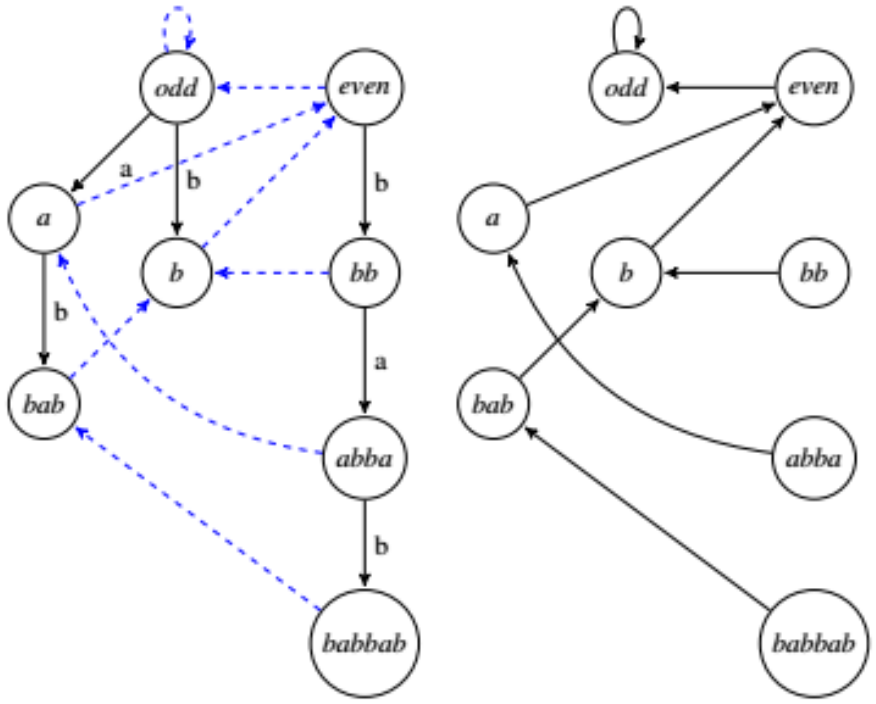
  特别地 $fail[even] = fail[odd] = odd,$

  最长回文后缀为空串时，$fail[u] = even$。

  可以发现$fail$指针几乎构成了一棵树，定义一个节点的$fail$链为这$fail$树上它到根的路径。

一个回文树的例子

在图中用蓝色虚线代表一个节点的*fail*转移，实线代表*go*转移，实线上的字母代表转移的字符，一个点上的字符串代表其对应的*s*中的回文子串。



(a) babbab的回文树　　(b) babbab的fail树

状态数证明

- 定理1.1 对于一个字符串s，不同的回文子串个数最多只有$|s|$个。
- 证明可用数学归纳法
- 即回文树状态和转移数均是$O(|s|)$的。

基础插入算法

- 我们采用增量法来构造出字符串s的回文树。假设已经构造出s的回文树，接下来需要在s的末尾加入一个新的字符c，并维护出sc的回文树。
- 定理1.2 以新加入的字符c为结尾的，未在s中出现过的回文子串最多只有1个，且必为*sc的*最长回文后缀。
- 证明跟之前那个是一样的

基础插入算法

- 每次只需求出sc的最长回文后缀即可。
- 不妨设sc的最长回文后缀为 $s[i..|s|]c$，那么由回文串的定义，要么$i > |s|$，即最长回文后缀只有c这个字符，要么$s[i+1..|s|]$也是一个回文串。
- 也就是说，$s[i+1..|s|]$必须是s的回文后缀，或是长度为0或-1的串（方便来讲就是回文树上的*even*和*odd*）。
- 现在要在s的最长回文后缀对应的$fail$链中找到长度最大的一个节点t，满足 $s\left[|s| - len[t]\right] = c$，则sc的最长回文后缀就是ctc。

基础插入算法

- 插入时，假如回文树上不存在一个节点代表*ctc*，则需要新建一个新的节点来代表ctc，否则直接使用这个节点。
- 假如新建了一个节点，还需要求出这个新节点的$fail$指针对应的节点，相当于在$fail[t]$的$fail$链中再找一个长度最长的节点*v*满足$s[|s| - len[v]] = c$。
- 注意假如$len[t] = -1$，那么ctc的$fail$指针应当指向长度为0 的节点。

复杂度证明

- 定理 1.3 这种实现的总时间复杂度为$O(|s|)$，空间复杂度为$O(|s|\Sigma)$。
- 将两次跳$fail$的复杂度分开考虑。考虑每次跳$fail$时，其代表的回文串的长度至少-1，而每次插入时，其长度相较于其父亲+2。
- 故总时间复杂度为$O(|s|)$。当然，字符集较大时，可每个节点开一个map记录$ch[]$，其时间复杂度为$O(|s|log\Sigma)$，空间复杂度降为$O(|s|)$。
- 注意这种插入算法的复杂度是基于势能分析的，单次插入可能达到$O(|s|)$。

经典问题

- 给定一个只包含小写字母的字符串，对于s的每个前缀$s[1 \ldots i]$，求出其不同的回文子串个数以及位置不同的回文子串个数。
- $|s| \leq 10^5$
- 由于回文树是增量构造，因此可以在加入每个字符后得到对应的回文树结构。不同的回文子串个数就是当前回文树上的节点数量减2（有两个根）。
- 位置不同的回文子串数量，相当于之前的数量加上以当前位置为结尾的回文串数量，这就等于这个串的回文后缀的数量，就等于这个点的$fail$链的长度（注意odd与$even$不算入长度）。在回文树上另外维护一个节点$fail$链上的节点数量。

APIO 2014 PALINDROME

- 考虑一个只包含小写字符的字符串s，定义s的一个子串t的出现值为t在s中的出现次数乘上t的长度。求出s的所有回文子串中的最大出现值。
- $|s| \leq 3 \times 10^5$

$right$集合

- 首先建出s的回文树。对于回文树上的一个节点t，定义$right[t]$为一个端点集合，表示在s中的出现位置的右端点集合，那么t的出现值就是$len[t] \times |right[t]|$。
- 只需要对每个节点求出$|right[t]|$即可。对于s的每个前缀i，可以求出其最长回文后缀，设为u，将i加入u当中。
- 做完每个前缀后，每个节点的$right$就是$fail$树上他的儿子的$right$的并集，其$right$的大小就是所有儿子的$right$的大小的和。

回文树的拓展

nyg

- 前端插入
- 不基于势能分析的插入算法
- 双端插入删除
- 在Trie上建回文树
- 可持久化

前端插入

- 与末端插入类似的，可以发现在前端插入时，比如当前字符串为s，要求出cs的回文树，那么以c为开头的产生的新的回文串最多只有l个，并且是cs的最长回文前缀。
- 由之前的过程可知，只需要找到s的一个最长的回文前缀t，满足$s[|t|+1]=c$，那么新的最长回文前缀就是ctc。
- 直接的想法是，在回文树上对每个节点再维护一个$fail'$指针，表示这个节点的最长回文前缀指向的节点，那么每次加入新字符的时候，只需要沿着s的最长回文前缀的$fail'$链找到一个最长的合法的t。

前端插入

- 然而由于回文串的性质，有$fail[u] = fail'[u]$。
- 因此只需要维护出字符串的最长回文前缀与最长回文后缀，就能支持字符串的前端和末端插入了。
- 需要注意在前（后）端插入时可能会对最长回文后（前）缀造成影响，另外维护一下就好了。

HDU5421 VICTOR AND STRING

- 一开始有一个空串s。接下来进行n次操作，操作有以下4种：
- 在s的前端加入一个给定的字符。
- 在s的末端加入一个给定的字符。
- 询问当前s中有多少不一样的回文串。
- 询问当前s中有多少位置不同的回文串。
- $n \leq 10^5$
- 一道可以交的裸题而已。

支持删除操作？

- 一种很简单的思路就是，对于s串的每个前缀都记录下它的最长回文后缀，回文树上每个节点记录下这个节点作为一个前缀的最长回文后缀的次数。
- 当删除s 的最后一个字符的时候，相当于删掉其最长回文后缀（因为其他回文后缀依然在s中存在），直接在回文树上对应的节点将其次数减1，假如一个节点的次数变为0了就将这个节点从回文树上删去。
- 然而最坏情况下，一次插入操作会消耗$O(n)$的势能。如果又进行删除则会恢复消耗的势能，复杂度变为$O(n^2)$。
- 于是我们需要另一种插入算法。

不基于势能分析的插入算法

- 之前的插入算法的瓶颈在于，每次插入一个字符c，都要沿着当前最长回文后缀t的$fail$链往上找到第一个v使得v在s中的前驱（即v的前一个字符）为c。
- 注意到除去$s[|s| - len[t]] = c$的情况，我们找到的v将是确定的。
- 于是我们特判前一种情况，对每个节点v记录$go[v][c]$，表示最长的满足前驱为c的回文后缀。

不基于势能分析的插入算法

- 如何维护这个$go[v][c]$呢？
- 注意到v与$fail[v]$的$go[]$几乎没有什么差别，于是我们直接把$go[fail[v]]$复制到$go[v]$，再考虑$fail[v]$的前驱c，直接令$go[v][c] = fail[v]$即可。
- 这样单次插入时间复杂度为$O(\Sigma)$。
- 当然，在字符集较大时，可使用可持久化线段树维护$go[v]$。单次时间复杂度为$O(log\Sigma)$。

双端插入删除

- 首先引入一些概念
- 一个s中的子串$s[l \ldots r]$被称为重要的，当且仅当$s[l \ldots r]$是回文的，且不存在一个$r' > r$，使得$s[l \ldots r']$是回文的；不存在一个$l' < l$，使得$s[l' \ldots r]$是回文的。
- 对于每个位置r，显然最多只有一个$l \leq r$，满足$s[l \ldots r]$是重要的，称$s[l \ldots r]$为r的重要后缀，r可以没有重要后缀；同样的，可以对l定义重要前缀。

在末端插入新的字符

- 不妨设r表示原来末端的位置，那么r必然会有一个重要后缀，而这个后缀就是当前s的最长回文后缀。
- 直接套用末端插入的算法求出新的回文树并得到新的最长回文后缀，令l为新最长回文后缀的左端点，则$s[l \dots r+1]$是重要的。
- 注意可能会让$s[l \dots r+1]$的最长回文前缀变得不重要。另外维护一个桶$cnt[l][r]$表示$s[l \dots r]$被标记了多少次“不重要”，$cnt[l][r]$可以用哈希来实现。

在末端删除字符

- 令l为r的重要后缀的左端点，将$s[l \ldots r]$的最长回文前缀对应的cnt减1，假如被减为0，意味着他再次变成重要的。
- 现在还有一个问题是，我们需要知道$s[l \ldots r]$对应的回文树的节点是否需要被删掉。不妨对回文树上每个节点t定义两个权值$imp[t], child[t]$。
- $imp[t]$表示有多少$1 \le l' \le r' \le |s|, s[l' \ldots r'] = t$，且$s[l' \ldots r']$是重要的，$child[t]$表示在$fail$树上$t$的儿子数。
- 当改变一个串的重要情况时，同时改变对应点的imp值。假如一个点的imp与$child$都是0了，这个点就不可能在s中出现，直接把他删掉，在删掉一个点时其$fail$的$child$也减1。

前端操作

- 在前端插入新的字符
- 与在末端插入类似的，只需要找到新的最长回文前缀，并对其最长回文后缀的重要情况与做出修改即可。
- 在前端删除字符同样与末端删除类似

TRIE上建回文树

- 首先对于一棵节点数为n的Trie，其不同的回文串数至多为 n。
- 对Trie上每个节点对应的字符串，记录其最长回文后缀。每次拓展到儿子时，直接套用不基于势能分析的插入算法即可。
- 注意如果使用基础插入算法，复杂度最坏为$O(n^2)$。

CODECHEF TREEPAL

- 给定一棵以1为根的n个点的树，树上每个点都有一个小写字母。对于树上每个点，求出从根到这个点形成的字符串的最长回文子串长度，输出所有点的答案的和。
- $n \leq 10^5$
- 又一道裸题，直接在Trie上建回文树即可。

可持久化

- 首先显然只能使用不基于势能分析的插入算法。
- 对于一个版本的回文树，用一棵可持久化线段树存储节点信息，每个叶子节点存一个回文树中节点的信息。
- 而一个节点的信息则包括其长度，$fail$指针对应的节点，以及$ch[]$和$go[]$，字符集较大时，后两个数组都可以用可持久化线段树来维护。单次时间复杂度为$O(\log|s|+\Sigma)$或$O(\log|s|+log\Sigma)$。
- 如果同时需要支持前后插入删除呢？$cnt[l][r]$可用可持久化$treap$维护，每个节点还需记录imp和$child$。

由于笔者也未曾实现这个版本的回文树，所以无法估计最终的代码实现复杂度

回文树的应用

- 其实就是一些例题

GDKOI 2013 COUNTRY

- 给定一个长度为n的字符串s，现在要从s中选出尽量多的子串，满足选出的子串都是回文的，并且不存在一个串是另外一个串的子串的情况。
- $n \leq 10^5$

GDKOI 2013 COUNTRY

- 定理 3.1 若对于两个回文串a和b，有a是b的子串。那么，要么$a = b$，要么a是b的最长回文后缀的子串，要么a是b缩去两侧字符后形成的回文串的子串。
- 因此，对于一个节点u，我们只需要连边$u \to fail[u]$, $u \to fa[u]$，这样就可以得到一个DAG。问题转化为在DAG上求一个最大的点集，其中的点两两不存在路径。
- 这实际上是经典的最长反链问题，由Dilworth定理，它等于这个图的最小链覆盖。
- Dinic即可，$O(n\sqrt{n})$

CODECHEF PALPROB

- 定义一个字符串的回文指数如下：
- 若一个字符串不是回文串，则回文指数为0。
- 仅有一个字符的字符串回文指数为1。
- 递归定义其他回文字符串s，等于$1+$前$\lfloor\frac{|s|}{2}\rfloor$个字符所组成的子串的回文指数。
- 给定一个字符串s，求出s的所有子串的回文指数的和。

CODECHEF PALPROB

- 建出回文树，维护出每个节点的$right$集合大小，那么剩下的问题就是求出每个回文串的回文指数。
- 考虑对于回文树上每个节点u再维护一个指针$half[u]$，指向最长的长度不超过u的一半的u的回文后缀。
- 直接的想法是$fail$树上倍增，$O(|s|\log|s|)$。
- 考虑另一种方法。令j为i在回文树上的父亲。$half[i]$必然是$half[j]$的某个回文后缀的儿子。直接沿着$half[j]$的$fail$链找到第一个合法的节点。用势能分析可证明复杂度为$O(|s|)$。

CENTRAL EUROPE REGIONAL CONTEST 2014 VIRUS SYNTHESIS

- 给定一个由大写字符*A*,*G*,*T*,*C*组成的长度为n的字符串s，一开始有一个空串t，每一步你可以对t进行以下操作中的一种：
- 在t的开头或结尾加入一个字符
- 将t的翻转接在t的开头或末尾
- 问最少需要多少步才能使得t变成s。
- $n \leq 10^5$

VIRUS SYNTHESIS

- 注意到第二种操作后的字符串必然是一个回文串，考虑枚举s的一个回文子串$s[i \ldots j]$，求出构造出$s[i \ldots j]$的最小代价，用其加上$i - 1 + n - j$来更新答案即可。
- 考虑由短到长地计算每个回文串的代价，它一定为这个回文串的半边的代价+1。我们发现它的半边要么是它父亲的串的半边加一个字符，要么是它的一个长度不超过一半的回文后缀补上若干个字符。
- 直接套用上题的算法，$O(|s|)$。

集训队互测2017 基因

- 给定一个长度为n的只包含小写字母的字符串s，有q组询问l, r，询问$s[l \ldots r]$中本质不同的回文串数量。
- $n, q \leq 10^5$
- 莫队？
- 强制在线。

集训队互测2017 基因

- 令$L=\sqrt{n}$，每L个位置设置一个关键点，维护每个关键点到右边每个位置的回文树，对于每次询问:
- $r-l\leq L$，可以直接暴力维护回文树。
- $r-l>L$，则必然会经过一个关键点。每次相当于在回文树前插入字符。
- 然而如果把所有节点的信息都存下来，可持久化也需要$O((n+q)\sqrt{n}log\Sigma)$的时空复杂度。

集训队互测2017 基因

- 依旧考虑到不同的回文串只有$O(n)$个，同一个回文串的$go[], fail$均是相同的，因此节点只需要$O(n)$个。
- 预处理时，从一个关键点向右建回文树，对每一个右端点只需记录此时的最长回文前缀以及此时的答案。顺便记录每个回文串从某个关键点开始第一次出现的位置。
- 查询时，由预处理时的信息向前插入，每插一个字符可以得到一个最长回文前缀，需要判断这个回文串是否已经被统计过。
- 最终时间复杂度为$O((n+q)\sqrt{n}+n\Sigma)$，空间则为$O(n\sqrt{n})$。

ALPHADOG

- 对于一个串s的特殊值$F(s)$ 定义为
- $F(s) = \sum_{1 \le x \le y \le |s|} LCP(x, y)$
- $LCP(x, y)$表示最长的字符串t的长度，其中t要满足三个条件：
- t是回文的
- 存在一个$i \le x$，满足$s[i \ldots x] = t$
- 存在一个$j \le y$，满足$s[j \ldots y] = t$
- 一开始有一个空串s。接下来进行q次插入，每次在串的末尾加入一个新的字符。每次加入后，询问当前串的F值。强制在线。
- $q \le 10^5$

ALPHADOG

- 首先考虑如何计算$LCP(x,y)$，令a为$s[1\ldots x]$的最长回文后缀，b为$s[1\ldots y]$的最长回文后缀，那么$LCP(x,y)$就等于a,b在$fail$树上的LCA的长度。
- 每次插入一个新的字符，需要求出这个字符带来的贡献，相当于每次可能会在$fail$树上接入一个新的节点，并查询某点与其他所有点的LCA的长度总和。
- LCT维护即可，$O(qlogq)$。

- Thank you