回文树及其应用

ZJR

from 翁文涛 2017 年的论文

2023.12

简述

回文树能够高效地处理一些有关回文子串的问题，最初由 Mikhail Rubinchik 和 Arseny M. Shur 在 2015 年发表。

结构

字符串s的回文树是一个由两棵树构成的森林。每个非根节点与s中的回文子串一一对应。边上记录一个字符c，代表如果父节点f对应的字符串为s_f，那么子节点son对应的字符串为$c + s_f + c$。

两棵树的根节点分别为‘ 0 ‘，‘ 1 ‘。根节点为‘ 0 ‘代表s中长度为偶数回文子串的集合，根节点为‘ 1 ‘代表s中长度为奇数回文子串的集合。

结构

对于一个树上节点i，定义len_i为i所对应的字符串s_i的长度。特别的，定义$len_0 = 0$，$len_1 = -1$，因为‘ 0 ‘节点的所有子节点对应的字符串形如cc（两个字符），长度为$0 + 2 = 2$，‘ 1 ‘节点的所有子节点对应的字符串形如c（单个字符），长度为$-1 + 2 = 1$。

对于一个树上节点i，定义其失配指针$fail_i$指向i所对应的字符串s_i的最长回文后缀在树上对应的节点。特别的，定义$fail_0$与$fail_1$均指向‘ 1 ‘。

定理

可以发现,假如一个串s有n个不同的回文子串,那么s对应的回文树的节点数就是$n+2$,所以需要分析s的不同回文子串的个数。

定理：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

使用数学归纳法。

- 当$|s| = 1$时，显然成立。
- 当$|s| > 1$时，设$s = s' + c$，c为s的最后一个字符，并且结论对s'成立。首先能产生的新的回文串一定都是s的回文后缀。考虑s的最长回文后缀t，其他较短的都是t的回文后缀。它们都会被“对称”到t的最前面，变成t的回文前缀。所以不是最长的回文后缀都在之前出现过。至多产生一个新的回文串，因此结论对于s依然成立。

构造

采用增量法来构造回文树。

依据上述定理，新产生的回文串一定是$s+c$ 的最长回文后缀。令t的初值是s未加入c时的回文后缀，看$s[|s|-len_t]$是否于c，来辨别是否找到字符串s+c的最长回文后缀。如果没到，那么使$t=fail_t$（找到更小的回文后缀或长度为‘ 0 ‘或‘ -1 ‘的字符串），直到找到满足条件的t。此时$c+t+c$就是$s+c$的最长回文后缀。

构造

插入时，假如回文树上不存在一个节点代表$c+t+c$，则需要新建一个新的节点来代表它。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，可以让t顺着$fail$指针向上跳，找到第一个节点v满足 s[$|s|$ - len_v] = c。直接将$fail$指针指向对应节点即可。

分析

根据之前证明的定理，回文树的状态（节点数）是$O(|s|)$级别的。考虑转移，对于一个状态，能转移到他（$fa->son$）的节点是唯一的，所以总转移数也是$O(|s|)$的。

分析

使用势能分析来分析一下跳$fail$的复杂度。假设t是字符串的最长回文后缀所代表的节点，$deep$是其在回文树中的深度（root的深度是1），每次执行上述操作时，t先向上跳（$t=fail_t, deep--$），再向下走一个（$t=c+t+c, deep++$）。而$deep$是一个非负整数，每次加入新字符只会$deep++$一次，所以$deep--$的次数一定不超过$|s|$次。

所以跳$fail$的复杂度也是$O(|s|)$的。

经典问题

给定一个只包含小写字母的字符串s，对于s的每个前缀$s[1..i]$，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。

$|s| \leq 10^5$。

经典问题

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，相当于结尾下标小于i的回文串的数量cnt加上以下标i为结尾的回文串数量num，num这就等于s[1..i]的回文后缀的数量，就等于这个点的$fail$链的长度（注意*根节点*不算入长度），这个量可以在构造回文树时顺便维护。

代码

```
struct Node{
  int to[30],next,len,cnt,fail,num;
}tr[MAXN];
int n,pos,tot;
char c[MAXN];
int Get(int x,int y){
  while(c[y-tr[x].len-1]!=c[y]) x=tr[x].fail;
  return x;
}
void Push(int x){
  int tmp=Get(pos,x);
  if(!tr[tmp].to[c[x]-'a']){
    tr[++tot].fail=tr[Get(tr[tmp].fail,x)].to[c[x]-'a'];
    tr[tmp].to[c[x]-'a']=tot;
    tr[tot].num=tr[tr[tot].fail].num+1;
    tr[tot].len=tr[tmp].len+2;
  }
  pos=tr[tmp].to[c[x]-'a'];
}
```

Palindrome

对于字符串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。

$|s| \leq 3 * 10^5$

Palindrome

建立回文树，对于每个节点，定义其出现次数为cnt。考虑每个前缀，我们给它的最长回文后缀的cnt加1。如果a在s中现，且出现时右端点为r，那么$fail_a$也在s中出现，且$fail_a$的右端点也为r。所以我们将cnt_a贡献给cnt_{fail}。因为每个前缀只会给一个点贡献，所以这样不会重复统计，最终得到了每个回文串的出现次数。

答案是$1 \leq i \leq n, \Sigma cnt_i * len_i$。

例3

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。

例4

假如没有前端插入，那么本题就与模板题完全一样了。

现在需要支持前端插入。一种直观的做法是在每个节点再维护一个指针$fail'$代表其最长回文前缀。但是经过观察现$fail'$与$fail$是一样的，因为节点所代表的字符串是回文的，所以最长回文前缀会被对称到最后面变成最长回文后缀。于是不用fail'了。

直接删

考虑是否能直接删（通过一些途径还原之前的状态）。这样的时间复杂度是错的，因为有可能刚刚向上跳了好多次，结果还原了之前的状态，然后又向上跳了好多次...这样就被卡成$O(n^2)$的了。

所以我们需要一种无需势能分析的做法。

非势能分析

对于节点t，定义其失配转移数组$quick_c$，存储t的最长的满足前驱为c的回文后缀。那么在插入时，我们只需要首先检查当前最长回文后缀t的前缀是否为c，假如不是，新的回文后缀就是t的$quick_c$。

非势能分析

对于一个节点t，t的$quick$与$fail_t$的$quick$几乎没有什么差别，因为t的回文后缀只是在$fail_t$的回文后缀中再加入了$fail_t$而已。
令字符集大小为Σ，则时空复杂度都是$O(|s| * \Sigma)$。
当然也可以用可持久化线段树来做，但是一般字符集较小，并不需要。

例题

给定一个根节点为 1 且有 N 个节点的树，每个节点都被分配一个小写的拉丁字符。从根到某个其他节点的简单路径生成的一个字符串，计算每个如此字符串的最长回文子串的长度加和。

$N \leq 100000$

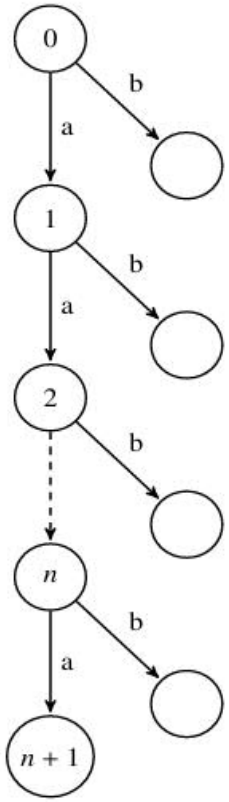
题解

考虑一个节点对应的字符串，相当于在他父亲的字符串后接入一个字符，那么最多就只会新增一个回文子串，并且必然为其对应字符串的最长回文后缀，因此不同的回文子串个数是O(n)的，所以回文树的大小也是O(n)的。

但这里有一个有趣的问题，假如套用最基础的插入算法，复杂度会是多少呢?

题解

在这种情况下，时间复杂度退化为$O(n^2)$，
所以要使用不基于势 能分析的插入算法。



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。

$n \leq 100000$

题解

若对于两个字符串a和b，有a是b的子串。那么，要么a=b，要么a在b的父节点中，要么a在b最长回文后缀中。可以构建一个有向图G，对于一个点i，连一条从i出发到回文树上i的父亲的单向边，以及一条从i出发到$fail_i$的单向边，那么假如点j是i的子串，必然存在一条从i出发到j的路径。

题解

现在的问题转化为：给定一个有向图G，要从G中选出尽量多的节点使得两两不存在路径。这就是一个最长反链问题。根据Dilworth定理，一个有向无环图的最长反链等于这个图的最小链覆盖，直接对图G求最小链覆盖即可。

Palindromeness

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。

$s \leq 100000$

题解

考虑对于回文树上每个节点i再维护一个指针$half_i$，指向最长的长度不超过i的一半的i的回文后缀。假如$half_i$的长度恰好为$\lfloor \frac{|s|}{2} \rfloor$，i的回文指数就等于$half_i$的回文指数加1，否则为1。

题解

考虑如何维护$half$：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。可以用与证明基础插入算法复杂度相同的方法证明这种算法的总复杂度是$O(s)$。

Virus synthesis

初始有一个空串，利用下面的操作构造给定串 S 。

- 串开头或末尾加一个字符。
- 串开头或末尾加一个该串的逆串。

求最小化操作数，$|S| \leq 10^5$，字符集为 $\{A, T, C, G\}$。

题解

反转后相加一定得到回文串，最终的答案一定是通过向一个回文串前后加入字符得到的，于是将问题转为求所有回文串的答案。考虑在回文树上dp，对于一个节点i，$fail_i$，fa_i可以向他贡献（操作1），$half_i$和fa_i形成之前的字符串可以向i贡献（操作2，前提是i是偶回文串）。

基因

给定一个长度为 n 的字符串 s，有 q 组询问，每个询问给定 l，r，询问 $s[l \dots r]$ 中有多少本质不同的回文子串。

$n \leq 100000$

题解

令$L=\text{sqrt }n$，每L个位置设置一个关键点，对于每次询问，假如$r-l\leq L$，可以直接暴力维护回文树，否则必然会经过一个关键点，维护每个关键点到每个右端点的回文树，每次相当于在回文树前插入字符。

总结

回文树将树形结构与字符串有机结合，拥有一些优美的性质。同时，算法本身侧重于处理回文后缀，所以也不适合处理所有的回文串问题（Manacher算法对于部分问题就足够适用了）。这就好比后缀数组与后缀自动机各有自己的优势，回文树与Manacher算法在处理回文问题时也各有侧重。