

后缀自动机浅谈

Legendgod, Chilly_Bird, Enrosecongratulation

顺便推销一下自己的 [Blog](#)

什么是后缀自动机

和 AC 自动机比较，后者是处理匹配问题的，而前者更多的情况是用来处理子串的问题的。

我们先来看一道例题吧。

给定一个只包含小写字母的字符串 S ,

请你求出 S 的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。

我会 AC 自动机！
诶等等，这个子串数量是不是太多了呀？
我们能不能对于这个子串进行压缩一下？
后缀自动机干的就是这件事情。

定义

- `endpos` : `endpos(S)` 表示字符串 `S` 在原来串中每次出现的结束位置的集合。
- 举个例子 `T = silhouetteisoursun`
- `endpos("s") = {1, 12, 16}`
- `endpos("ou") = {6, 14}`

Theorem 1: 如果两个字符串的 *endpos* 相同，其中一个肯定是另一个的后缀。

证明：比较显然，也就是其最后一次出现的位置是相同的。

Theorem 2: 对于两个子串 S 和 T 他们的 *endpos* 等价类要么是包含关系要么是交集为空。

证明：如果是后缀关系的话肯定是包含关系，如果不是后缀关系的话，考虑 T 出现的位置 S 肯定不可能出现，也就是交集为空。

定义

- endpos 等价类：对于 endpos 相同的字符串会将归于相同的 endpos 等价类。

Theorem 3: 对于同一个 *endpos* 等价类的串，长度一定是连续递增的。

证明：都是后缀关系，比较显然。

Theorem 4: *endpos* 等价类的个数是 $O(n)$ 的。

考虑对于每一个 *endpos* 等价类中的最长串加入一个新的字符字符可以得到新的串，而且得到的 *endpos* 肯定是原来子集，可以考虑成一个子集分割的问题，每次可以隔开变成两个不交的子集。

我们最优秀的方法就是线段树的割法等价类的个数显然是 $O(n)$ 。

对于我们刚才进行分割的集合，我们考虑让子集和距离自己最近的超集合连边，可以得到一颗树，称其为 *parent tree*。

Theorem 5: 对于两个相互连边的集合(不妨设在 *parent tree* 上的边叫做**后缀链接**)，父亲的最长子串 $\text{maxlen}(fa) + 1 = \text{minlen}(u)$ 。

考虑是通过父亲最长的子串在前面加入一个字符得到的。

Theorem 6: 后缀自动机的边数是 $O(n)$ 的。

证明：考虑先找出一颗后缀自动机的生成树，设其根节点为 $root$ ，如果有 M 个状态，显然边数只有 $M - 1$ 条。构造生成树上从 $root \rightarrow a \rightarrow b \rightarrow \forall S$ 的路径，既然说这个是一个包含了所有后缀的自动机，那么这条路径必然构成了一个后缀，而且既然是最简状态所以每个后缀之后对应一个非树边，那么非树边的个数肯定小于等于后缀的个数，所以边数也是 $O(n)$ 的。

由此我们可以证明这是一个线性大小的结构。

如何构造后缀自动机

后缀自动机上面有两种边，本质上就是一个 *Trie* 树和 *parent* 树合起来的東西。我們設 $trans[u][c]$ 表示 *Trie* 樹上的邊，稱其為轉移邊，而對於 *parent* 樹我們只需要存父親邊即可。

我們考慮加入一個字符 c 之前的原串 S 和新串 T 。

考虑先看看 *Trie* 树上是否有该边，毕竟我们是需要找到一个后缀。

如果没有我们考虑从上次加入的位置跳后缀链接保证**后缀**的性质，并且尝试找到一个符合条件的位置。

Case 1: 如果没有转移边，显然这个字符是从来都没有出现过的，我们直接在根节点加入即可。

Case 2: 如果有转移边我们考虑进入当前转移位置 p 和

$q = \text{trans}[p][c]$ ，如果说 $\text{maxlen}(p) + 1 = \text{minlen}(q)$ 可以发现我们加入的字符是已经被钦定为 c 了，那么很显然这些都是 T 的后缀，我们直接加入节点即可。

```
int nq = ++ tot; d[nq] = d[q];  
d[nq].len = d[p].len + 1;  
d[q].fa = d[np].fa = nq;  
for(; p && d[p].ch[c] == q; p = d[p].fa) d[p].ch[c] = nq;
```

Case 3: 如果说 *Case 2* 不成立说明并非在 q 中的所有串都是 T 的后缀，我们考虑将 q 进行拆点之后连边，具体来说是这样的。

具体代码如下

后缀自动机浅谈

很好你已经学会了后缀自动机
来看看应用吧

本质不同的子串个数

发现后缀自动机构造的时候本来就是没有重复的子串，直接让

$$ans = \sum_i maxlen(i) - maxlen(fa(i))$$

统计子串的出现次数

本质上就是求每一个状态 *endpos* 集合的大小，首先每次新增一个状态会产生 1 的贡献，之后对于所有儿子考虑，其贡献肯定是儿子的贡献之和，根据 *DAG* 的性质我们直接计算即可。

求两个串最长公共子串的长度

[题目链接](#)

考虑对于其中的一个串构造后缀自动机，让另外一个串在上面跑即可。

多个串的最长公共子串

[题目链接](#)

不妨考虑有 k 个长度为 n 的串。

考虑还是按照之前的方法进行匹配，之后对于每一个节点取一个最小值。

但是发现一个节点的贡献可能从儿子中间带来，那么我们直接对于儿子的长度取 \max ，对于自己的长度取 \min 即可。

字典序第 K 大子串

[题目链接](#)

感觉上还是比较简单的，我们可以根据上述的方法求出当前节点某个儿子总共有多少个子串，然后直接进行像二分一样的操作即可。

如果说相同子串位置不同算同一个的话，那么每一个 *endpos* 集合本质就是产生 1 的贡献，不然的话就是像我们之前说的那样的。

长度为 K 的子串的最大出现次数

[题目链接](#)

话说我之前还一直找不到这题。

做法其实和之前是差不多的，就是按照每一个 *endpos* 集合的大小来计算，最后合并的时候注意可以通过更长的合并即可。

所有前缀的最长公共后缀和

[题目链接](#)

Emmm，题目上面的式子显然可以拆成两部分，本质上就是所有后缀的最长公共前缀和。

而如果我们要求所有前缀的最长公共后缀和只需要翻转数组即可。

我们考虑计算后面的东西。

考虑将串 S 的 $parent$ 树给建出来，那么两个点的 lcp 就是 $parent$ 树上的 lca 。如果要统计所有的串我们考虑进行 Dp。

这里再提醒一下两个前缀的**最长公共后缀**是后缀树上的 lca ，所以如果写这题需要翻转一下字符串。

之后直接进行 Dp 即可。

但是本题的式子事实上可以看成两点在后缀树上的距离，考虑每条边产生的贡献就是 $maxlen(p) - maxlen(fa(p))$ ，所以事实上有更加简单的做法。

广义后缀自动机（离线）

网上主流有很多写法，在 *dalao* 的指引下找到了一种正确的写法。

将所有的字符串离线下来建立一颗 *Trie*，之后通过搜索遍历节点，在遍历的同时建立自动机。

注意这里需要 *bfs* 而不是 *dfs* 具体问题可以看 [ix35 的讨论](#)

[代码](#)

广义后缀自动机（在线）

在线的本质就是在线插入，我们考虑每次加入一个新的串的时候将 $last$ 设置为 $root$ ，但是我们发现一些问题，我们为了保证节点的个数是最优的，我们可能还需要进行一些特判。

Case 1: 如果加入这个节点，发现 $last$ 之后已经有一个合法的等价类了，我们就不要再加入节点了。

Case 2: 考虑我们裂开节点的时候如果发现 $len(now) = maxlen(p) + 1$ ，虽然说我们仍然是需要裂开节点的，但是我们下次接上节点的时候只需要连接到 nq 上即可。

[代码](#)

广义：维护不同串的 *endpos* 集合

[题目链接](#)

这个东西和之前是不一样的，因为很多子串是被合并起来的，我们考虑对于每一个子串单独计算 *endpos* 集合即可。

广义：线段树合并维护 *siz*

[题目链接](#)

因为一个节点可能表示了很多的串，所以考虑对于每一个节点开一个线段树进行维护即可。

广义：树上本质不同的路径条数

诸神眷顾的幻想乡

考虑跳父亲的时候路径是直的，不能处理路径是合并的情况，所以我们考虑通过暴力枚举叶子节点得到答案。

本质上就是考虑所有叶子作为**根**，之后拿出所有的前缀串再建立广义 *SAM*。

LCT 维护区间字符串

[题目链接](#)

首先考虑静态区间不同元素的种类数，容易想到考虑一个右端点 i 查找答案，对于加入一个元素 x 我们将 x 上一次出现的位置 -1 ，当前出现的位置 $+1$ ，如果出现 $r = i$ 的询问，我们直接使用线段树区间查询即可。

我们考虑上述本质就是取消了之前的贡献，考虑最近的贡献即可。

我们考虑对于后缀自动机上的节点也是如此。当我们新增一个位置的时候删除当前节点的贡献，再加上新的贡献。我们考虑到 SAM 上贡献是从当前节点通过后缀链接到根节点的，我们考虑将其删除之后再加上贡献。

发现到根节点的这个东西和 LCT 的操作本质一样，所以我们直接使用 LCT 进行维护即可。

[代码链接](#)

广义：关于卡空间的问题

如果你按照我的写法写还被卡空间了，要么是假了，要么就是写法问题了。现在的写法是删除了无用的节点的情况。

谢谢

By legendgod, Chilly_Bird, Enrosecongratulation

[legendgod 的 blog](#)