后缀自动机在字典树上的拓展

湖南省长沙市雅礼中学 刘研绎

摘 要

本文详细分析了能够识别Trie树所有后缀的确定有限状态自动机。给出了状态数是线性、转移函数总数不超过Trie的节点数乘以字符集大小的结论及其证明，并给出了一种可行的、在允许离线时复杂度是线性的构造算法。同时对在线构造算法复杂度进行了分析，介绍了后缀自动机的相关运用，希望本文能够起到一个抛砖引玉的作用。

1 前言

在算法竞赛中，字符串处理问题是相当重要的一部分。一般有固定模式串的字符串处理问题和固定主串的字符串处理问题两大类问题。当固定模式串时，可能有不同的主串，如果模式串仅有一个，有著名的KMP算法[2,3]来处理这类问题，也可以用字符串哈希[13] 来解决模式串的出现次数问题，如果模式串较多，大家熟知的AC自动机算法[1]便可以胜任这类问题。如果主串固定，一般采用对主串构造后缀数组[11]、后缀树[4,5]、后缀自动机[7]来解决这一类问题，这一类问题已经在算法竞赛界得到了充分的研究，其中陈立杰在他的冬令营营员交流讲稿[8]提到了很多关于给串建立的后缀自动机的相关运用，罗穗骞在他的集训队论文[12]详细对后缀数组在信息学竞赛中的运用进行了讲解。当题目可以采用离线算法时，这两类问题往往可以互换，这也体现了字符串处理算法的灵活多变。

而在今年(2015)的浙江省省选中，出现了这么一个题目[1]：给出一棵$n(\leq 10^5)$个节点叶节点个数不超过20的树，树上的每条边上有一个字符，树上两点之间的路径形成了一个字符串，询问总共有多少互不相同的子串。这个题显然

[1]ZJOI2015 Day1 诸神眷顾的幻想乡，作者：陈立杰

属于固定主串的类型，但主串不再是一个串，而是一棵树。考虑直接枚举所有点对，再利用Trie树[1,13]判断字符串是否相等，复杂度达到了$O(n^3)$，即便使用字符串哈希算法[13]，时间复杂度也有$O(n^2)$，均无法通过此题。考虑将题目进行转化，因为只有20个叶节点，不妨以每个叶节点为根分别建立一棵Trie树，再将所有Trie树合并成一棵，这样一个出现在树上的串一定会出现在新的Trie树中，主串便是一棵Trie树。

为了解决上面的问题，本文提出了对能够识别Trie树所有后缀的确定有限状态自动机进行了仔细分析并提出一种构造方法，主要分为五个部分：1.将一些串上的概念定义到了Trie树上，并且对Trie树的后缀自动机进行了简单介绍；2.主要对后缀链接进行了分析，并构造出了Parent树这一结构；3.证明了状态数是线性的，也说明了自动机中转移函数总数的上界；4.详细描述了一个给Trie树构造后缀自动机的算法，当允许离线时，它的复杂度是线性的；5.阐述了Trie树后缀自动机的一些运用，加强了对后缀自动机的理解。

2 Trie树的后缀自动机

在本文中，所有字符串中的字符以及Trie树[1,13]边上的字符都属于一个有限的字符集A，由这个字符集生成的所有字符串记做A^*，一个空串将被表示为ϵ，不包含空串的集合$A^+ = A^* - \{\epsilon\}$，用$|s|$表示字符串s的串长，s_i 表示第i个字符，用$s_{i,j}$表示由字符串s第i个和第j个之间的字符组成的字符串。用$a \cdot b$表示字符串a, b 首尾相接得到的字符串（乘号可能省略）。

一个串s的所有后缀为：

$$S(s) = \{s_{i,|s|} \mid 1 \le i \le |s|\}$$

令$Minl(S), Maxl(S)$分别为一个字符串集合S中最短的和最长的串的长度。

用T表示一颗Trie树，用$T_x, T_y, T_z, \ldots$表示T中的一个节点。令T中某个节点T_x到其子树中某个节点T_y路径上经过的边上的字母顺次连接起来组成的字符串为$T_{x,y}$，下文中使用$T_{x,y}$在没有特殊声明的情况下T_y一定是T_x的子树中的节点。

令T的一个前缀为：

$$P_{T_x} = T_{root,x} \ (root \text{ is the root of } T, T_x \in T)$$

那么T的所有子串为：

$$F(T) = \{T_{x,y} \mid T_x, T_y \in T, T_y \text{ is in the subtree of } T_x\}$$

同理T的所有后缀为：

$$S(T) = \{T_{x,y} \mid T_x, T_y \in T, T_y \text{is a leaf node}\}$$

给Trie树建立后缀自动机，即建立一个有限状态自动机能够识别所有的$S(T)$，并尽可能的缩小自动机的状态数和转移数。类似给一个串建立后缀自动机，我们定义一个串在Trie树上的出现位置：

$$Right(s) = \{T_y \mid T_y \in T, \exists T_x \in T, T_y \text{ is in the subtree of } T_x, T_{x,y} = s\}$$

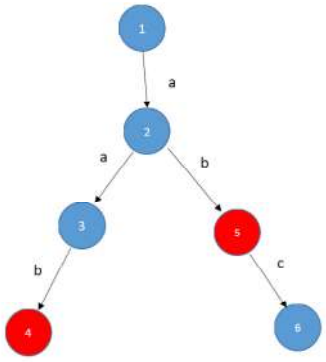
定义一种等价关系R_T（在没有歧义的时候记做R），

$$aRb \iff Right(a) = Right(b)\ (a, b \in A^*)$$

令$R(s)$表示与s等价的所有的串，即一个等价类。那么在自动机中一个状态就对应着一个等价类。令由T建成的后缀自动机为$\Phi(T)$，$\Phi(T)_\epsilon$为其初始状态，用$\Phi(T)_s(s \in F(T))$表示读入s后所处于的状态，那么转移函数被定义为$\Phi(T)_s.a(a \in A, sa \in F(T))$，有$\Phi(T)_s.a = \Phi(T)_{sa}(sa \in F(T), a \in A)$，同理$\Phi(T)_{s_1}.s_2 = \Phi(T)_{s_1 s_2}(s_1, s_2, s_1 s_2 \in F(T))$。根据定义，有：

$$\Phi(T)_a = \Phi(T)_b \iff aRb\ (a, b \in F(T))$$

并且称a是$\Phi(T)_b$这个状态对应的字符串，那么$\Phi(T)_b$对应的字符串集合就是$R(b)$。



*Picture a.*在图中所示的*trie*树中，bR等价于ab，红色节点为他们的*Right*集合。

为了分析出整个结构的状态数（即$F(T)$中R等价类的数量），我们需要对后缀自动机状态的性质进行一些分析。我们希望状态数能够尽可能的少。

引理2.1. 令$x, y, z, u \in F(T)$，有：

(i).$xRy \Rightarrow (x \in S(y)\ or\ y \in S(x))$

(ii).$x \in S(z) \Rightarrow Right(z) \subseteq Right(x)$

(iii).$(x \in S(z), z \in S(u), xRu) \Rightarrow zRuRx$

证明： (i) 不妨假设$|x| \leq |y|$，又$Right(x) = Right(y)$，考虑任意一个$T_v \in Right(x)$，有$x \in S(P_{T_v})$ and $y \in S(P_{T_v})$，因此$x \in S(y)$。

(ii) $\forall T_v \in Right(z)$，有$z \in S(P_{T_v})$，又$x \in S(z)$，因此$x \in S(P_{T_v})$，因此$T_v \in Right(x)$，原命题得证。

(iii) 由(ii)，$z \in S(u) \Rightarrow Right(u) \subseteq Right(z)$，同理$x \in S(z) \Rightarrow Right(z) \subseteq Right(x)$，又$xRu$，因此$Right(z) = Right(u) = Right(x)$即$zRuRx$。

上面个的(i)同时也说明了R等价类中长度相同的串必定唯一。

推论2.2. 对于一个等价类$R(a)$，令x为$R(a)$中长度最长的串，y为$R(a)$中长度最短的串，那么有：

$$R(a) = \{z \mid z \in S(x)\ and\ |y| \leq |z| \leq |x|\}$$

这个可以由引理2.1轻易推出。

3 后缀链接

在给串建立后缀自动机的构造算法中，后缀链接起了关键的作用[7,6]，而且整个算法与KMP算法[3]十分类似。而在给Trie建立后缀自动机时不妨保留后缀链接，并将构造过程与能够识别$A^*\{P_{T_x}\}$的确定最简状态自动机的构造算法，也就是我们熟知的AC自动机[1]的构造算法相类比。和AC自动机中的fail函数类似的，我们定义给Trie树构造的后缀自动机中后缀链接：

后缀链接是一个$A^* - \{\epsilon\}$到A^*的函数s_T（没有歧义的时候记做s），表示：

$$s_T(x) = \{z \mid z \in S(x), |z| = Minl(R_T(x)) - 1\}$$

即x的后缀中不满足xR_Ty的最长的y。

后缀链接也能帮助我们证明$\Phi(T)$的状态数是线性的，在此之前我们需要分析$s(x)$本身的性质。

引理3.1. 对于$x, y \in F(T)$，有：

(i) $xRy \Rightarrow s(x) = s(y)$

(ii) $Right(x) \subset Right(s(x))$

(iii) $Minl(R(x)) - 1 = |s(x)| = Maxl(R(s(x)))$

证明： (i) 不妨假设$|x| \leq |y|$。根据定义有$|s(x)| = Minl(R(x)) - 1, |s(y)| = Minl(R(y)) - 1$，又$R(x) = R(y)$，因此有$|s(x)| = |s(y)| < |x| \leq |y|$。由引理2.1(i)，$x \in S(y)$，又$s(x) \in S(x), s(y) \in S(y)$，因此$s(x) = s(y)$。

(ii) 由引理2.1(ii)，$Right(x) \subseteq Right(s(x))$，又$xRs(x)$为假，因此$Right(x) \neq Right(s(x))$，所以原命题得证。

(iii) 因为$s(x) \in R(s(x))$，有$s(x) \leq Maxl(R(s(x)))$，不妨令y为和$s(x)R$等价并且长度最长的串，那么有$Right(y) = Right(s(x))$且$s(x) \in S(y)$。不妨运用反证法，假设$|y| > |s(x)|$，考虑任意一个$T_v \in Right(x)$，由(ii)可知$T_v \in Right(s(x))$，那么$T_v \in Right(y)$，有$y \in S(P_{T_v})$并且$x \in S(P_{T_v})$。

若$|y| > |x|$，则$x \in S(y)$，根据引理2.1(ii)，势必有$Right(s(x)) = Right(y) \subseteq Right(x)$，与(ii) 矛盾；若$|y| \leq |x|$，因为$xRy$不成立，此时有$s(x) = y$，与$|y| > |s(x)|$矛盾，因此原命题得证。

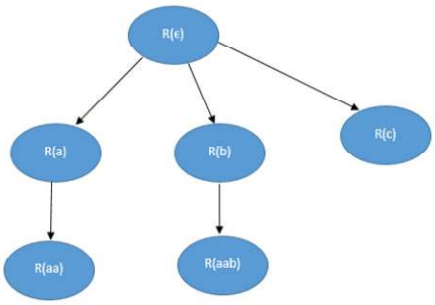
由上面的(i)可知，对于一个等价类$R(a)$，他们的后缀链接指向了相同的串，因此我们可以对后缀自动机的每个状态也建立后缀链接。不妨令$R(a)$的后缀链接指向$R(s(a))$。由(ii)或(iii)可知，这将形成一个树结构（如Picture c）。不妨称其为parent树[8]。因为有了树结构，定义$Fa(\Phi(T)_s)$为状态$\Phi(T)_s$的后缀链接，那么：

$$Fa(\Phi(T)_a) = \Phi(T)_b \iff s(a)Rb$$

引理3.2. 对于$x \in F(T)$，任取$T_v \in Right(x)$。

若$|P_{T_v}| > Maxl(R(x))$，不妨令$a = P_{T_v}$，那么$s(a_{|a|-Maxl(R(x)),|a|})Rx$。

证明： 不妨令$b = a_{|a|-Maxl(R(x)),|a|}$，根据已知可得$x \in S(b)$，根据推论2.2，因为$|b| > Maxl(R(x))$，所以$not\ bRx$，而$|b_{2,|b|}| = Maxl(R(x))$因此有$b_{2,|b|}Rx$，可知$not\ bRb_{2,|b|}$，根据后缀链接定义可知$s(b) = b_{2,|b|}$，因此$s(b)Rx$。



*Picture b.*图中的树对应了*Picture a*中的*Trie*构建出的后缀自动机的*Parent*树

4 线性的状态数

在我们给一个串建立的后缀自动机中[7,6]，对x建立的后缀自动机大小是$O(|x|)$的。我们希望对于Trie建出的后缀自动机的状态规模是线性的。在定义了后缀链接后，我们需要对$\Phi(T)$进行更仔细的分析。

引理4.1. 令$x, y \in F(T)$，有：

(i) $\forall y, not\ xRs(y) \Rightarrow |Right(x)| = 1$。

(ii) $|Right(x)| = 1 \Rightarrow P_{T_v}Rx\ (Right(x) = \{T_v\})$

(iii) $\forall T_v \in Right(x), not\ xRP_{T_v} \Rightarrow \exists u, v, s(u)Rs(v)Rx\ and\ not\ uRv$

证明： (i) 反证法。不妨假设$Right(x) > 1$，那么任取$T_{v_1}, T_{v_2} \in Right(x)$，令$a = P_{T_{v_1}}, b = P_{T_{v_2}}$，那么有$|a|, |b| \geq Maxl(R(x))$（若$|a| < Maxl(R(x))$，则$R(x)$中长度为$Maxl(R(x))$的串不可能出现在$T_{v_1}$，矛盾），且$|a|, |b|$不能都等于$Maxl(R(x))$。

若$a \notin R(x)$，则$|a| > Maxl(R(x))$，那么引理3.2，$s(a_{|a|-Maxl(R(x)),|a|})Rx$；若$a \in R(x)$，又$T_{v_1} \neq T_{v_2}$，因此$a \neq b$，又$|b| \geq Maxl(R(x))$，那么可得$|b| > Maxl(R(x))$（否则两串会相等），同理可得$s(b_{|b|-Maxl(R(x)),|b|})Rx$。原命题得证。

(ii) 可知$x \in S(P_{T_v})$，那么$|Right(P_{T_v})| \leq |Right(x)|$，又$|Right(x)| = 1$，可得$Right(P_{T_v}) = \{T_v\} = Right(x)$，即$P_{T_v}Rx$。

(iii) 由(ii)的逆否命题可推得：$|Right(x)| \neq 1$，那么$|Right(x)| \geq 2$。根据已知可得$\forall T_v \in Right(x), |P_{T_v}| > Maxl(R(x))$，并且$\exists T_{v_1}, T_{v_2}$，使得$P_{T_{v_1}}, P_{T_{v_2}}$的最长公共后缀长度为$Maxl(R(x))$。

假设$\nexists T_{v_1}, T_{v_2}$，由已知可得$P_{T_v}, T_v \in Right(x)$这些串的最长公共后缀的长度至少为$Maxl(R(x))$，而任意两个$P_{T_v}$的公共后缀都不为$Maxl(R(x))$，又$|P_{T_v}| >$

$Maxl(R(x))$，因此他们的最长公共后缀的长度一定会大于$Maxl(R(x))$，根据引理2.1(ii)，可知他们的最长公共后缀一定与xR等价，而长度又大于$Maxl(R(x))$，因此假设不成立。即一定存在T_{v_1}, T_{v_2}，使得$P_{T_{v_1}}, P_{T_{v_2}}$的最长公共后缀长度为$Maxl(R(x))$。

不妨令$a = (P_{T_{v_1}})_{|P_{T_{v_1}}|-Maxl(R(x)),|P_{T_{v_1}}|}$，$b = (P_{T_{v_2}})_{|P_{T_{v_2}}|-Maxl(R(x)),|P_{T_{v_2}}|}$，有上面的结论可知$a \neq b$，因此$a \notin S(b)\text{and } b \notin S(a)$，根据引理2.1(i)可知$not\ aRb$。根据引理3.2可知$s(a)Rx, s(b)Rx$，至此原命题得证，$a, b$即为所求的$u, v$。

有了以上分析，我们可以对状态数的规模进行计算了。

定理4.2. *给一棵Trie树建立的后缀自动机，其状态数是线性的。*

证明： 令$T_v \in T$。

考虑由后缀链接构成的Parent树，由引理4.1(i)&(ii)可知对于所有Parent树上的叶节点$\Phi(T)_s$一定存在$P_{T_v}Rs$，由引理4.1(iii)可推知任意一个状态$\Phi(T)_s$只要满足对所有的T_v均有$not\ sRP_{T_v}$，那么$\Phi(T)_s$一定有两个以上的子节点。也就是说Parent树上叶节点和只有1个子节点的节点均对应着某个P_{T_v}，其他节点均有两个以上的子节点。而T_v的数量是$|T|$并且$\Phi(T)_1.P_{T_v}$所达到的状态是唯一的。那么Parent树上叶节点和只有1个子节点的节点数之和不会超过$|T|$，整棵树的节点数也就不会超过$2|T|$，因此是线性的。

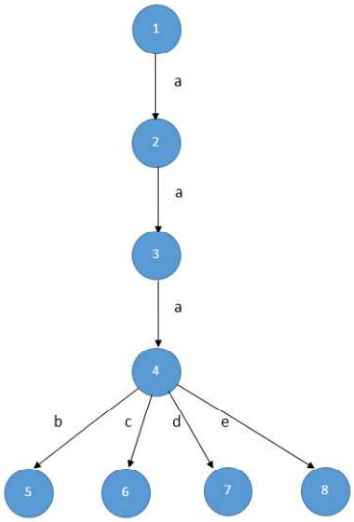
有了线性的状态数，如果转移函数（即边数）的总数也是线性的，那么我们就能推得整个自动机是线性的。但事与愿违，与给字符串建立后缀自动机不同，给Trie树所建立的后缀自动机转移函数的总数将是状态数乘以字符集大小级别。下面将证明不会超过这个数量级并且给出一个达到这个数量级的Trie树的构造方法。

定理4.3. *给一棵Trie树建立的后缀自动机，其转移函数的上界是O((Trie的节点数)×(字符集大小))的。*

证明： 不妨令Trie树为T。

由定理4.2，$|\Phi(T)|$不会超过$2|T|$，再考虑对于每一个$\Phi(T)_s$，其转移函数最多有$|A|$个。因此转移函数的总数不会超过$2|T| \times |A|$。命题得证。

构造一个Trie树使得其后缀自动机转移函数规模达到上界。考虑Trie树T，显然对于A中没有出现在T的树边上的字符我们不需要将其考虑进来，那么$|A| \leq |T|$。T中的一部分是一条长度为$|T|/2$的边上的字母均为同一个字母的链，剩下的点全部以两两不同的字符连接这条链的最后一个点。不妨令链的边上的字母均为a，剩下的点与链的连边上的字符集合为B，那么$a^ib \in S(T)\ (0 \leq i \leq |T|/2, b \in B)$，又显然有$not\ a^iRa^j\ (i \neq j)$，因此对于每一个$\Phi(T)_{a^i}$，所有的$b \in B$都是其一个转移函数，因此$\Phi(T)$的转移函数总数达到了$|T|/2 \times |B|$也就是$|T|/2 \times |A|/2$，和定理4.3中的上界是同一个规模。



Picture c.一个能使转移函数总数达到上界的例子

虽然给Trie建立后缀自动机的转移数并不是线性的，但是在大多数题目以及实际问题中字符集都不会很大，一般字符集都是小写英文字母或者阿拉伯数字，在字符集是常数的情况下其转移数仍是线性的，仍不失为一个有效的处理Trie树有关字符串问题的方法。

5 构造算法

在给串建立后缀自动机的算法[7,8]中，我们使用的是一个在线的增量算法，即在建立好s的后缀自动机上建立出$sa(a \in A)$的后缀自动机，那么不妨在

给Trie建立的过程中也考虑在线的增量算法。有了上面对状态数和转移数的分析，我们知道构造算法的复杂度下界是$|T| \cdot |A|$。

令Trie树T在节点T_x加入一个字符为a的边连向子节点T_y得到的新的Trie树是TN，$s = P_{T_x}$。考虑$\Phi(TN)$在$\Phi(T)$的基础上有哪些需要修改的地方。

我们先对状态数进行分析。

5.1 状态的变化

对于任何$s \in F(T)$，因为s继续延伸可能是T的一个后缀，因此一定存在$\Phi(T)_s$这个状态。反之对于$s \notin F(T)$，s延伸不可能得到T的后缀，因此一定不存在$\Phi(T)_s$这个状态。

如果一个等价类在加入T_y之后没有改变，那么他们对应的后缀自动机上的状态也不会改变，有$\Phi(TN)_s = \Phi(T)_s$。

Case 1：如果T_x已经存在一个儿子使得其与T_x的连边上的字符为a，那么我们不需要对$\Phi(T)$进行任何修改。

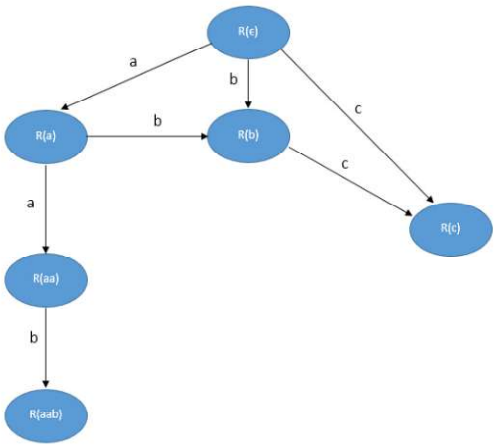
Case 2：如果$sa \in F(T)$，那么有$\Phi(T)_s.a = \Phi(T)_{sa}$，我们可能把$R_T(sa)$分成两个等价类。

考虑如果$x, y \in F(T), xR_{TN}y$，那么如果x, y的$Right_{TN}$集合中同时出现T_y就一齐删除，因此肯定有xR_Ty。也就是说$\forall xR_{TN}sa, yR_{TN}sa$，均有$xR_Ty$，即与$saR_{TN}$等价的原本都是同一个$R_T$等价类$R_T(sa)$中的。当$\forall x \in R_T(sa), xR_{TN}sa$时，此时不会有新的状态出现。当$\exists x \in R_T(sa), not\ xR_{TN}sa$时，需要把$R_T(sa)$分成两个等价类，一类与$saR_{TN}$等价，一类不与其等价即可（如果不与$saR_{TN}$等价，那么他们的$Right_{TN}$集合中一定不存在$T_y$，即$Right_{TN}$和原来的集合$Right_T$相等）。

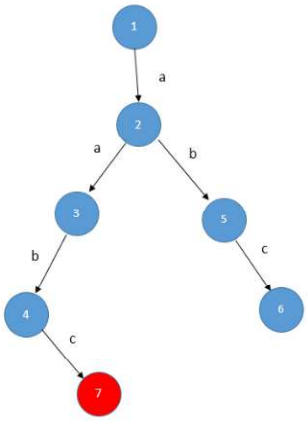
不妨考虑两个新状态的后缀链接。

令$u, v \in F(T), uR_{TN}sa, not\ vR_{TN}sa$且$uR_Tsa, vR_Tsa$，那么在$T_y$加入后，$u$将被分进第一类，$v$则是第二类。因为$sa = P_{T_y}$，又有$uR_{TN}sa$，可知$|u| \le |sa|$（如果$|u| > |sa|$则$u$不可能出现在$T_y$，$uR_{TN}sa$也就不成立了），因此对于任意$x \in S(u)$的$Right$集合中都新加入了$T_y$，其后缀链接不会改变，$s_{TN}(u) = s_T(u)$。再考虑$v$的情况，$vR_Tsa$却$not\ vR_{TN}sa$，考虑$Right_T(sa) \to Right_{TN}(sa)$只多出了$T_y$，那么必然有$T_y \notin Right_{TN}(v)$，根据引理2.1(i)，可推知$sa \in S(v)$，对所有$x \in S(v), |x| > |sa|$的串$x$，因为$vR_Tsa$，根据推论2.2，$xR_Tv$，又他们的$Right$集合在加入$T_y$前后没有变化（$Right_T(x) = Right_{TN}(x), Right_T(v) = Right_{TN}(v)$），因此$vR_{TN}x$，此时就能

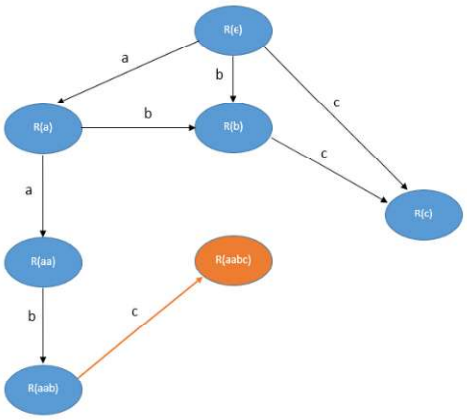
得出结论：$s_{TN}(v) = sa$。



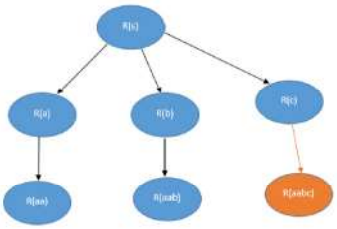
Picture d.为Picture a中的Trie建立的后缀自动机



Picture e.



Picture f.



Picture g.

Picture e&f&g.分别为给Picture a的Trie树加入一个节点的Trie，新Trie树的Parent树及后

缀自动机。红色的节点为新加入*Trie*的节点，橙色的节点和边表示*Parent*树和后缀自动机的变化。

既然已经知道sa所属的等价类$R_{TN}(sa)$，那么sa的所有后缀已经在这之前加入到$R(s_{TN}(sa)), R(s^2_{TN}(sa)), \ldots$中去了，而那些状态并不会改变，因此不需要再考虑。

Case 3：如果$sa \notin F(T)$，我们将新建一个等价类$R_{TN}(sa)$，因为出现了新的后缀，因此有一个等价类可能分裂。

有一个比较显然的结论：$s_{TN}(sa)$等于$S(sa)$中一个出现在$F(T)$中的最长的串。证明也比较简单，若$x \in S(sa), x \notin F(T)$，那么$x$只会在$T_y$处出现，而$sa$亦只会在$T_y$出现，即$Right(sa) = Right(x) = \{T_y\}$，即$xRsa$；若$x \in S(sa) \& x \in F(T)$，那么$|Right(x)|$势必大于1，即$not\ xRsa$。

这个结论将给我们的构造过程带来很多方便。$\forall y \in S(sa), y \notin S(s_{TN}(sa))$将组成一个新的$R_{TN}$等价类，我们为其在自动机中新建一个状态，对于$y \in S(s_{TN}(sa))$，也就是剩下的那些串，我们可以将$s_{TN}(sa)$看做$sa$带入*Case 2*（虽然*Case 2*中有利用到$sa = P_{T_y}$的性质，但s_{TN}本身包含了极长的性质，因此不会造成错误），那么所有串都找到所属的状态。现在的问题就是怎么求$s_{TN}(sa)$了。根据上面的结论，我们要找sa中在$F(T)$出现至少一次的最长后缀，不妨先找到$\Phi(T)_s$，那么我们需要找到s中最长的后缀使得在其之后添上a仍然出现在$\Phi(T)$中，这时候就可以使用我们的转移函数了。根据推论2.2，我们只需在$\Phi(T)_s, \Phi(T)_{s_T(s)}, \Phi(T)_{s^2_T(s)}, \ldots$中找到第一个存在$a$转移的状态$\Phi(T)_{s^i_T(s)}$，那么$s^i_T(s)a$即为$sa$中最长的出现2 次的后缀，即$s_{TN}(sa) = s^i_T(s)a$。

从上面三种情况的分析可以发现，当Trie上新增了一个字符状态最多只会多出2个，这也证明了其状态是线性的，同时也讨论了给后缀链接带来的变化。但我们并没有讨论转移函数的修改情况，接下来将对转移函数的修改情况进行分析。

5.2 转移函数的变化

在*Case 2*中，如果没有状态新建就肯定不会有转移函数的变化。否则原来的等价类$R_T(sa)$将会拆成$R_{TN}(sa)$和$R_{TN}(v)$，考虑他们的转移函数，因为加入的T_y是叶节点，所以不会对原有的转移函数产生影响，$\Phi(TN)_v$和$\Phi(TN)_{sa}$的转移函数

都和$\Phi(T)_{sa}$的一致，将他们复制过来即可。由于之前的等价类$R_T sa$删除那么可能存在$\Phi(T)_x.a = \Phi(T)_{sa}$的状态，这些状态的转移函数肯定需要修改。根据*Case 2*中的分析，在新建的两个状态$\Phi(TN)_u$和$\Phi(TN)_v$中，有$Fa(\Phi(TN)_v) = \Phi(TN)_u$，而所以这些$\Phi(T)_x.a = \Phi(T)_{sa}$的状态，要么$\Phi(TN)_x.a = \Phi(TN)_u$，要么$\Phi(TN)_x.a = \Phi(TN)_v$，由*Case 3*，即所有$\Phi(T)_s, \Phi(T)_{s_T(s)}, \Phi(T)_{s_T^2(s)}, \ldots$直到$\Phi(T)_{s_T^i(s)}.a \neq \Phi(T)_{sa}$这些状态的$a$转移一定均为$\Phi(TN)_u$，同理可知其他$\Phi(T)_x.a = \Phi(T)_{sa}$的状态新的$a$转移都会到$\Phi(TN)_v$。因为其他的转移我们很难依依找到具体的状态是什么，我们保留$\Phi(TN)_v = \Phi(T)_{sa}$，新建一个状态代表$\Phi(TN)_u$，暴力修改所有$\Phi(T)_s, \Phi(T)_{s_T(s)}, \Phi(T)_{s_T^2(s)}, \ldots$直到$\Phi(T)_{s_T^i(s)}.a \neq \Phi(T)_{sa}$这些状态的$a$转移。由推论2.2和引理2.1可推知，$s_T^i(s)a = s_T(sa)$。

在*Case 3*中，新建了一个状态$R_{TN}(sa)$，其余部分和*Case 2*一致。因为T_y是叶节点，因此新建的状态的转移函数一定为空。由于新建了一个状态，可能有$\Phi(T)_s$等状态新增加了转移的边，由*Case 3*的分析，我们只需要将$\Phi(T)_{s_T^i(s)}$ $(i \geq 0, s_{TN}(sa) \neq s_T^i(s)a)$这些状态的转移函数新增转移$a$指向$\Phi(TN)_{sa}$即可。

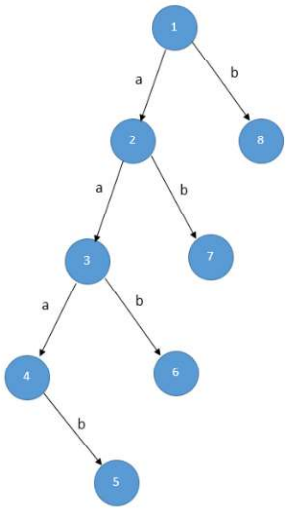
5.3 构造算法复杂度分析

和串类似，整个构造算法是一个在线的增量算法，对于T，他的执行次数是$|T|$次。每次只会新增两个状态，这里的复杂度是$O(|T|)$。每个新增的状态的后缀链接只会修改一次，这部分也是$O(|T|)$的。考虑转移函数的情况，在*Case 2*中可能存在一个转移函数的复制，一个状态的转移函数最多$|A|$个，因此这一步是$O(|T||A|)$的，比较复杂的是我们暴力修改其他转移函数的时间复杂度，可以证明他的时间复杂度不超过在trie树中叶节点深度之和。不妨令T中叶节点深度之和为$G(T)$。

证明的思路类似势能分析。不妨令$Dep(s), s \in F(T)$表示节点$\Phi(T)_s$在Parent树中的深度，考虑$\Phi(TN)_s$和$\Phi(TN)_{sa}$在Parent树中到根的路径，根据引理2.1(ii)，如果存在$R(ba)$就一定存在$R(b)$，因此$Dep(sa) \leq Dep(s) + 1$。令$c = s$，我们在暴力修改的时候会不断的让$c = s(c)$，同时修改c的转移函数，同时$Dep(c)$会减1。当$\Phi(T)_c.a$这个转移存在时，$Dep(c) \geq Dep(s(sa))$，因为$R(s(sa))$这个状态会被拆成两个，因此还有可能c有可能继续在Parent树上往上跳同时修改$\Phi(TN)_c.a$，直到不在往上时，有$\Phi(TN)_c.a = Fa(\Phi(TN)_{s(sa)}) = Fa(Fa(\Phi(TN)_{sa}))$，因此$Dep(sa) \leq Dep(s) + 3$并且每一次修改一定会造成深度的减少，可以将$Dep(s)$看做当前状态

的势能函数，不妨考虑对于每个T中的叶节点，单独做一次从根到该点的插入算法，一次插入树上的路径，每一次插入操作$Dep(s)$最多增加3，每一次暴力修改转移函数$Dep(s)$会减少1，那么我们暴力修改转移函数的次数实际上是深度级别的，将所有叶节点对应的操作次数加起来就能得到上面的结论。

上面这个复杂度并不优秀，但在线构造的复杂度是可以达到这个级别的。详见*Picture h*。



*Picture h.*如果按照节点的编号顺序进行加入，复杂度会达到$O(G(T))$。

当没有要求在线时，我们可以按照FIFO序，即宽度优先搜索顺序将Trie上的节点加入。这样在J. A. Blumer等[9,10]给出的对于串的构造算法复杂度的证明同时也对Trie树适用。只有固定的子串组合会使我们需要暴力修改转移函数[6]，这提示我们这一部分的复杂度不会超过$O(|T|)$。

综上所述，如果要求在线，那么构造算法的复杂度将是$O(G(T))$，如果允许离线，我们可以按照FIFO序构造得到一个复杂度是$O(|T||A|)$的算法，这已经是我们的复杂度下界了。

6 运用

6.1 ZJOI 2015 诸神眷顾的幻想乡

这道题的题面已经出现在前言之中，这里就不再赘述了。

题面已经被我们转化为：统计一个Trie树有多少本质不同的子串。利用上面介绍的给Trie树建立后缀自动机的方法，我们可以得到一个能够识别一个Trie树所有后缀的自动机。而Trie树的一个子串必定是某个后缀的前缀，这和给串建立后缀自动机有着相同的性质。同理我们可以与之相类比，对于一个状态$\Phi(T)_s$，它代表的本质不同的子串数为$|R(s)|$，那么我们将所有状态的$|R(s)|$加起来就能得到答案了。

当然还有另一种解决方案，在建立出自动机后，我们直接统计有多少不同的串不会被自动机拒绝，这就相当于统计不同的子串个数，也等价于从自动机初始状态开始有多少种不同的转移路径使得存在，把自动机看成一个有向无环图，在上面进行简单动态规划就能算出答案。

根据上面的复杂度分析，试题中并没有要求在线，因此我们可以采用离线构造算法，复杂度为$O(n|A|)$。实际上因为叶节点总数是常数这个条件非常强，可以证明按照上面的构造方法构造出来后缀自动机的时间复杂度是$O(n)$的，因为我们可以将根到每个叶子单独拆成字符串，之后与串的复杂度分析一致[7]，因为与本文内容不相关，因此省略详细证明过程。

至此我们圆满解决了前言中的问题。

由这个试题扩展开，我们可以知道对于大部分可以用后缀自动机处理的字符串问题均可以拓展到Trie上，比如求Trie树的第k小子串，多个Trie树的最长公共子串，求循环串在Trie树上的出现次数等等，还有一些串上很容易处理但推广到Trie上就变得复杂的题目，比如说求有多少本质不同的串的前缀是给定的串，这些问题都可以在给Trie树构建出后缀自动机后解决。

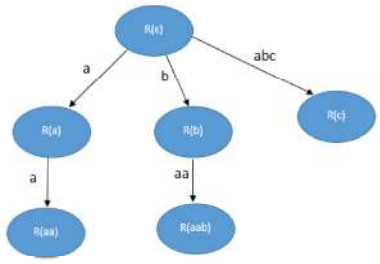
6.2 与AC自动机

我们熟悉的字符串算法：AC自动机，他将n个串建立成Trie树之后再建立好fail指针，然后可以识别对于输入的串每个模式串的出现次数。我们可以用给Trie建立后缀自动机来代替这个用法：将n个串建立成Trie之后再为这个Trie树建立后缀自动机，我们在读入识别串的时候，如果当前状态$\Phi(T)_s$不存在转移a，那么我们跳到$Fa(\Phi(T)_s)$继续进行识别，显然这样跳跃的次数不会超过识别串长。

考虑复杂度，我们用n个串建立出来的Trie树的T，其$G(T)$一定不会超过n个串的总串长，也就是说复杂度不会劣于AC自动机的复杂度。但我们的构造算法

是一个可以是在线算法，而AC自动机的构造算法只能是离线的。

6.3 Parent树



*Picture h.*将*Picture b*边上的字符补全。

这实际上是将所有Trie树中的前缀翻转过来再构成一棵Trie树，类似字符串的后缀树。我们给每个状态$\Phi(T)_s$在parent树中的儿子排序，即对于$Fa(\Phi(T)_x) = \Phi(T)_s$的$\Phi(T)_x$按照$x_{|x|-Maxl(R(s))}$的大小排序，我们将得到一个Trie树的前缀数组（即按照所有前缀翻转过来排序）。我们需要给$\Phi(T)_s$维护一个$T_v \in Right(s)$，配合树上倍增我们就能快速找到$x_{|x|-Maxl(R(s))}$了。

有了这个性质，我们可以快速找到在当前识别串之前加上一个字符$a \in A$，应该对应的状态位置。即假设当前位于状态$\Phi(T)_s$，我们在其Parent树的子节点中二分查找出$\Phi(T)_{as}$，或者判断出$\Phi(T)_{as} = \Phi(T)_s$。

7 致谢

感谢雅礼中学屈运华老师长期以来在学习生活上的指导和关心。
感谢父母对我的养育之恩。
感谢雅礼中学朱全民，汪星明老师的教导。
感谢CCF提供的交流平台和机会。
感谢国家集训队教练的辛勤付出。
感谢清华大学黄志翱，上海交通大学黄宇扬学长对我的帮助。
感谢雅礼中学杨定澄，匡正非，刘剑成同学为本文审稿。
感谢所有对我有过帮助的同学。

参考文献

[1] A.V. Aho and MJ. Corasick, Efficient string matching: An aid to bibliographic research

[2] R-S. Boyer and J.S. Moore, A fast string searching algorithm

[3] D.E. Knuth, J.H. Morris and V.I Pratt, Fast pattern-matching in strings

[4] P. Weiner, Linear pattern-matching algorithms

[5] E.M. McCreight, A space-economical suffix-tree construction algorithm

[6] M. Mohri, P. Moreno and E. Weinstein General Suffix Automaton Construction Algorithm and Space Bounds

[7] M. Crochemore, Transducers and repetitions

[8] 陈立杰，后缀自动机

[9] J. A. Blumer, Algorithms for the directed acyclic word graph and related structures

[10] A. Blumer, J. Blumer, D. Haussler, R. M. McConnell, A. Ehrenfeucht, Complete inverted files for efficient text retrieval and analysis

[11] U. Manber and G. Myers, Suffix arrays: A new method for on-line string searches

[12] 罗穗骞，后缀数组——处理字符串的有力工具

[13] Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L. and Stein, Clifford, ”Introduction to Algorithms”