后缀自动机及其应用

长沙市一中 张天扬

摘 要

后缀自动机作为一□OI□新兴的字符串处理工具，越来越受到广大出题人的欢迎。本文□在对后缀自动机的性□以及一些应用做一个归纳，并通过一些例题来深入对后缀自动机性□的研究。

1 后缀自动机的定义及构造

1.1 后缀自动机的定义

一个串S的后缀自动机（SAM）是一个有限状态自动机（DFA），它能且只能接受所有S的后缀，并且拥有最少的状态与转移。

我们将在下面证明：一个串S的后缀自动机的状态数和转移数都是$O(|S|)$的。

1.2 后缀自动机的构造

几个定义

定义后缀自动机的母串为S。令$S[l,r]$表示S中第l个字符到第r 个字符组成的字串。令从第i 个位置开始的后缀为suf_i，到第i个位置为止的前缀为$pref_i$。

令SAM_{str}表示从初始状态读入字符串str后的状态（或者也可以叫节点）。

对于一个S的一个子串str，令$right_{str}$表示str在S中每一次出现的右端点位置组成的集合。

如果没有特殊说明，我们认为字符集为所有小写字母。并□把字符集大小视为常数，在计算复杂度的时候不予考虑。

复杂度证明

考虑一个串str。如果str是S的一个子串，那么SAM_{str}应该是一个合法状态。这是因为我们可以在str后添加若干字符来变成S 的一个后缀，而这个后缀应该是一个可接受的状态，那么SAM_{str} 就必须是一个合法状态。反之，如果str 不是S的子串，那么SAM_{str} 应该是一个不合法的状态（或者说空状态），因为无论怎么添加字符它都不会变为S的一个后缀。

也就是说，对于S的每一个子串，后缀自动机中都应该有对应的状态。但是显然我们不能对于每个子串单独建立一个状态，因为子串个数是$O(|S|^2)$ 的。

考虑S的一个子串str。如果$right_{str}$和$right_{a+str}$（其中a 是一个字符串）完全一致的话，那么我们可以把它们合并为同一个状态。因为我们从这两个字符串开始添加字符，能够得到的后缀是一样的（或者说除了前面的a字符串以外一样）。换言之，我们从它们的状态开始，能够转移到的接受状态也是一样的。因此，这两个串在后缀自动机中的本质是一样的，它们可以合并为同一个状态。

接下来我们考虑S的两个子串A和B。如果$right_A$ 和$right_B$ 有交，那么显然有一个串是另一个串的后缀，不妨设A是B的后缀，那么容易看出$right_A \subset right_B$。也就是说，对于S 任意两个子串，它们的$right$ 集合要么没有交集，要么一个包含另一个。我们令一个状态s 的父状态（fa_s）为满足$right_s \subset right_{fa_s}$ 且$|right_{fa_s}|$最小的状态。那么所有状态会形成一个树结构，我们把它叫做$parent$树。

显然初始态的$right$集合是$1-|S|$的所有整数。而对每一个叶节点，它的$right$集合大小为1。也就是说叶节点个数是$O(|S|)$的。而一个非叶节点至少有两个子节点（如果只有一个，那么可以把它和它的子节点合并），那么非叶节点至多有$O(|S|)$个，因此**后缀自动机的状态数是$O(|S|)$的**。

考虑转移。注意到状态是$O(|S|)$的，不妨考虑一个从初始状态开始的自动机的树形图：树形图上边的数量显然是$O(|S|)$的。那么只需要考虑非树边。

考虑一条非树边$u \to v$。我们从初始状态沿树边走到u，然后经过这一条非树边走到v，然后继续向下走一定能走到至少一个后缀。也就是说，考虑一个后缀把它经过的第一条非树边覆盖，那么所有非树边一定都会被覆盖。因此，非树边的数量不会超过后缀的数量。所以我们证明了**后缀自动机的转移数是$O(|S|)$的**。

构造方法

后缀自动机有一种非常简单的构造方法：增量法。

我们对每个状态s，记它代表的最长子串的长度为len_s。

考虑我们当前已经有了前$|S|-1$个字符的后缀自动机，且现在的自动机中$[1,|S|-1]$位于状态$last$。现在加入第$|S|$个字符（不妨设为c），那么我们新建一个状态np，显然$len_{np}=|S|$。

之后考虑转移：显然我们应该加入一个$last \to np$的转移，我们也应该加入一个$fa_{last} \to np$的转移……直到我们发现，这个状态已经有了一个字符c的转移。不妨设这个状态是p，设它经过字符c 的转移后的状态为q。

此时，$right_q$会多出一个：右端点为$|S|$的串，且最长的长度是len_p+1。那么有两种情况：

1. $len_q = len_p+1$。此时q代表的所有串的$right$仍然相同，那么我们只需要令$fa_{np}=q$ 即可。

2. $len_q > len_p+1$。这种情况下q代表的串中，长度不超过len_p+1的串的$right$集合会多出一个值$|S|$，而长度超过它的串则不会。那么为了维护一个状态中所有串的$right$相同这一性质，我们需要新建一个状态nq，nq代表的是原来q代表的串中所有长度不超过len_p+1 的串，因此$len_{nq}=len_p+1$，nq的其他属性（fa 和转移）和原来的q 点一致。同时容易发现$fa_q=fa_{np}=nq$。

   然后我们再次从p开始：本来p的c字符转移到的是点q，现在它应该转移到nq。同理，如果fa_p的c字符转移到的是点q，那么它也应该转移到nq……直到发现当前点的c字符转移到的不是q为止。

至此，我们完成了后缀自动机的构造。

1.3 后缀自动机的几□基本性质

上文中提到了后缀自动机的几个基本性质。将它们归纳总结如下：

性质一：每□状态s代表的串的长度是□间$(len_{fa_s},len_s]$。

性质二：每□状态s代表的所有串在原串中的出现次数及每次出现的右端点相同。

性质三：在*parent*树中，每□状态的*right* 集合是它的□状态*right* 集合的子集。

1.4 *right*集合的□法

首先，在$pref_i$所在状态的*right*集合中加入i。

之后，我们按*parent*树中的深度从大到小，依次将每个状态的*right*集合并入它父状态的*right* 集合。

这样就可以求出每个状态的*right*集合了。但是这样做是$O(|S|^2)$的。

根据实际的问题，我们可能需要求出*right*集合的大小或者最大值等数值，可以在$O(|S|)$的时间内求出。如果确实需要每个状态的*right*集合，可以使用平衡树来维护，在合并的时候使用启发式合并[1]，可以做到$O(|S|\log|S|)$ 的复杂度。如果需要在线，则可以使用可持久化平衡树来维护。

1.5 使用后缀自动机□后缀树和后缀数组

除了后缀自动机之外，后缀树和后缀数组是我们通常使用的后缀数据结构。但是求它们一般不太方便：后缀树的线性构造可以说相当麻烦，而后缀数组的线性构造常数比较大。因为后缀自动机的构造常数小而且比较简洁，我们可以考虑使用后缀自动机来求后缀树和后缀数组。

性质四：后缀自动机的*parent*树是原串的反向□缀树。

首先，反向前缀树[2]的定义是：把每一个前缀的反串插入到一个trie中，并把没有分支的链合并（就像后缀树那样）。

这个性质是容易发现的。考虑一个前缀在后缀自动机上的状态，我们一直沿fa 指针走，每次会变成当前串的一个后缀，直到空串。把这个过程反过来看，就是在反向前缀树上从上向下走。

那么，我们求出初始串的反串的后缀自动机。它的*parent* 树就是原串的后缀树！

[1]关于平衡树的启发式合并的问题，因为与本文无关在此不作讨论，有兴趣可以参见《小树苗与集合》解题报告算法八

[2]其实就是反串的后缀树

为了还原后缀树，我们可以求一下后缀自动机上每个状态*right*集合内的任意一个值（比如最大值）。然后就容易找出每个点在后缀树上父边上的字符串了。

最后只需要对后缀树DFS一遍就可以求出后缀数组了。

2 后缀自动机的应用

2.1 一个解释

我们描述“将串放到后缀自动机上运行”，可以类比KMP和AC自动机的运行，也就是：从初始状态开始，每次如果存在对应的转移就转移，否则沿fa指针回跳，直到存在对应的转移或者跳出了后缀自动机为止。

2.2 几个经典问题

首先我们来讨论几个经典问题的解法。

2.2.1 最长公共子串

我们对其中一个串建立后缀自动机。将另一个串放到后缀自动机上运行，并时刻维护当前子串的长度即可。

对于多个（大于2）串的最长公共子串，我们对其中一个建立后缀自动机，将其它的串分别放到后缀自动机上运行，对每个状态维护它对每个串的最长匹配长度。然后每个状态把所有串在这个状态的最长匹配长度取min，所有状态取max即可。

2.2.2 字典序k小子串

我们对后缀自动机做一遍拓扑，就可以求出从每个点开始可以到达多少个子串。然后就可以从初始状态开始DFS，如果当前状态能到的子串不少于k就继续DFS，否则就把k减去当前状态能到的子串个数然后回溯。

如果求不同子串的k小则不用求*right*集合大小。否则需要求出每个点的*right*大小。

2.2.3 字典序最小后缀（最小循环表示）

把原串复制一遍接到后面，然后构造后缀自动机。

从初始状态开始，每次走字典序最小的转移，走$|S|$次之后得到的就是最小循环表示。

如果求的是最小后缀，就在原串后加入一个比字符集中最小的字符更小的字符作为终止，然后再复制一遍即可。

下面我们以一些简单的例题来探讨后缀自动机的基础应用。

例题一：字符串[3]

题目大意

给定n个字符串，询问每个字符串有多少非空子串是所有n个字符串中至少k个的子串。

$1 \leqslant n, k \leqslant 10^5, \sum |S_i| \leqslant 10^5$

题目分析

首先把n个字符串连接起来，中间用一个不在原字符集中的字符隔开。然后构造它的后缀自动机。对后缀自动机中的每个节点，我们需要计算它是原来n个串中多少个串的子串。

基本的想法是：将每个串放到后缀自动机上运行，对于运行到的每个节点，把它沿fa指针走到根的路径上的每个节点的出现次数+1。但是这样做会导致重复。我们可以对每个状态记录一下最后一次被哪一个串到达，那么每当到一个点时，只需要向上走到第一个当前串已经到达过的点就可以了。如果直接暴力的话，复杂度是$O(n\sqrt{n})$的[4]。使用树链剖分可以达到$O(n\log^2 n)$。

之后只要递推一遍求出每个节点开始沿fa走到根的路径上有多少个串是出现了至少k次的，然后再把每个串放到后缀自动机上运行，把每个运行到的节点的值相加即可。

[3]题目来源：Adera 1 杯冬令营模拟赛

[4]需要非常特殊的串才会达到这个复杂度，而且常数非常小

例题二：回文串[5]

题目大意

给一个长度为n的字符串，求它的所有回文子串中出现次数乘以长度的最大值。

$1 \leqslant n \leqslant 3 \times 10^5$

题目分析

构造原串的后缀自动机，并求出每个节点*right*集合的最大值*rmax*。

之后把反串放到后缀自动机上运行，如果当前反串中的匹配串$[l, r]$[6]覆盖了当前节点的*rmax*，那么$[l, rmax]$是一个回文串。

详见参考文献[4]。

例题三：识别子串

题目大意

给一个长度为n的字符串，定义一个位置i的识别子串为包含这个位置且在原串中只出现一次的字符串。求每个位置的最短识别子串长度。

$1 \leqslant n \leqslant 10^5$

题目分析

显然所有的识别子串在后缀自动机中都在那些*right*集合大小为1的节点上。那么我们首先求出所有这样的节点。对于每一个集合大小为1的节点，我们容易知道它唯一一次出现的右端点。

而显然，如果一个串的出现次数为1，那么把它向左扩展一段之后的出现次数也为1。所以我们对每个*right* 集合大小为1的节点，算出这个右端点向左最少扩展多长的出现次数是1。那么每一个*right* 对应的就是一段区间内和一个值

[5]**题目来源：**APIO2014

[6]这里指在原串中的位置

取min，然后一个前缀和一个公差是−1的等差数列取min。这可以排个序之后简单的模拟实现。

复杂度为$O(n \log n)$。

例题四：差异[7]

题目大意

给一个长度为n的字符串，求它的所有后缀两两的最长公共前缀长度之和[8]。

$1 \leqslant n \leqslant 5 \times 10^5$

题目分析

首先我们把这个字符串反过来，那么我们要求的就是：所有前缀两两最长公共后缀长度之和。

性质五：两个串的最长公共后缀，位于这两个串对应的状态在*parent* 树上的最近公共祖先状态

这是因为，一个串对应节点的祖先节点都是它的后缀，且深度越大则长度越长。

那么我们把每个前缀所在的节点染黑，然后问题就变成了每个节点是多少个黑色节点的LCA。这是一个简单的问题，在*parent*树上从下往上递推一遍即可。

后缀自动机也可以与其它数据结构和字符串工具相结合，从而解决一些较难的问题。

[7]**题目来源：**AHOI2013

[8]这里给出的是原题经过简单的转化后的题意

例题五：珠宝商[9]

题目大意

给定一棵n个点组成的树，每个点有一个字符。再给定一个长度为m的字符串S。对每两个点组成的点对，它们之间路径上的点组成了一个字符串，求所有这样的字符串在S中出现次数的和。

$1 \leqslant n, m \leqslant 5 \times 10^4$

题目分析

考虑使用点分治。由于n, m同阶，一下计算复杂度时统一使用n。

1. 分治块大小小于$\sqrt{n}$。此时我们直接DFS 每一条路径，并在S的后缀自动机中统计出现次数。因为这样的块是互不相交的，计算一个块的复杂度是$O(size^2)$。那么复杂度最多是$O(n\sqrt{n})$。

2. 分治块大小大于$\sqrt{n}$。考虑如何计算经过当前分治块重心的路径，对于其它路径直接递归分治即可。

   考虑重心x上的字符c在S中的出现位置。如果一条路径形如$a \to x \to b$，那么$a \to x$这一段在S中的出现位置的$right$必然是c在S中的出现位置的子集。同理，我们把$x \to b$这一段反过来，那么它在S 的反串中出现位置的$right$必然是c在S 中出现位置的子集。

   那么考虑从x开始DFS分治块。每次维护当前串在原串和反串的后缀树上的位置。然后我们将两个后缀树从上向下递推一遍就可以求出每一个后缀（也就是每一个位置）的匹配数量了。最后把两个后缀树上对应位置的匹配数相乘，就可以得到经过重心的路径在原串中的出现次数之和了。

   注意这样做会多算那些a, b在重心的同一棵子树的情况，那么把重心的每一棵子树分别使用相同的方法计算一遍就可以了。

   计算一个分治块的复杂度是$O(size + n)$。注意分治块大小大于$\sqrt{n}$的至多只有$O(\sqrt{n})$ 个[10]，那么复杂度是$O(n\sqrt{n})$。

[9]**题目来源：**CTSC2010

[10]证明的思路是：每次分治子树大小至少缩小一半，那么进行k次分治之后最大块不超过$\frac{n}{2^k}$，此时块数是2^k，然后令最大块大小不超过$\sqrt{n}$即可得出结论。

综上，我们得到了一种复杂度是$O(n\sqrt{n})$的优秀算法。

例题六：str[11]

题目大意

给定两个长度分别为n, m的字符串。定义两个字符串匹配为它们至多有一位不同。求这两个字符串的最长公共子串。

$1 \leqslant n, m \leqslant 10^5$

题目分析

我们考虑最后能匹配出来的串：它中间有一位不匹配，而这一位左边和右边分别匹配。假如我们已经知道了左边的串，那么我们就容易知道它在两个串中的每一次出现位置，然后把相应的右边的串两两取一个LCP[12]就可以了。

那么我们将两个串连接起来，中间加一个不在原字符集中的字符，求出这个串的后缀自动机。考虑后缀自动机中的一个节点：它的*right*集合中，有一部分是出现在第一个串内部的，另一部分是出现在第二个串内部的。比如有一个*right*是a，它在第一个串内部；另一个*right*是b，它在第二个串内部，那么我们求$a+2$和$b+2$的LCP，就是相应的最长匹配长度。如果我们把所有这样的(a, b)对的相应LCP 都求出来，取其最大值加上当前点的*len*，得到的就是这一个节点的答案。把所有节点的答案取最大值就是最后的答案。

但是求所有(a, b)对的LCP时间效率上肯定是不行的。我们考虑把所有后缀排序，那么我们对于一个节点，如果把所有的a和b混合起来按照对应的后缀顺序来排序，那么我们就只要计算所有排序后相邻的出现在两个不同的串内部的LCP。对于这样一个问题，我们按*parent*树自底向上依次考虑每个节点，使用平衡树来维护整个序列就可以了。在平衡树合并的时候维护一下信息，就可以做到$O((n+m)\log(n+m))$的复杂度了。

[11]**题目来源：**Feyat Cup 1.5 **题目作者：黄志翱**

[12]最长公共前缀

3 字母树的后缀自动机

我们将KMP算法推广到字母树上，形成了AC自动机。同样的，我们也可以把后缀自动机推广到字母树上。

我们先来看一道例题：

例题七：pty的字符串[13]

题目大意

给一棵n个节点的有根树，每条边上有一个字符。定义一条路径是从某个节点开始，向下走到某个节点结束，边上的字符组成的字符串。定义两个字符串匹配为它们完全相同。再给一个长为m的字符串，求它的子串和树的路径共有多少对匹配。

$1 \leqslant n \leqslant 8 \times 10^5, 1 \leqslant m \leqslant 8 \times 10^6$

题目分析

基础的做法是，对字符串建立后缀自动机，并求出*right*集合的大小。然后DFS 字母树即可。

但是，在本题中，因为串的长度太大，对串建立后缀自动机的做法空间上无法接受[14]，需要另辟蹊径。

考虑对所给的字母树建立后缀自动机。我们插入一个节点的时候，将它的父节点的前缀[15]在后缀自动机中的状态作为*last* 状态来插入。但是会有一个问题：一个节点有可能有两个子节点的边上的字符是一样的。

其实解决方法很简单：我们把同一个节点下字符相同的子节点合并起来，合并后的节点*right*初始大小是合并的节点个数。

最后将所给字符串放到后缀自动机上运行一遍统计一下答案就可以了。

再来看另一道例题：

[13]题目作者：彭天翼

[14]也许将边使用hash来保存的方法可以。

[15]这里的前缀指从根节点到它的路径。

例题八：诸神眷顾的幻想乡[16]

题目大意

给定一棵n个节点的树，每个节点上有一个字符。定义路径为某两点间的节点上的字符组成字符串。求树上有多少条互不相同的路径。

$1 \leqslant n \leqslant 10^5$，字符集大小为10，树至多有20个叶节点。

题目分析

注意到题目的特点：叶节点很少。那么我们依次把每个叶节点当做根，然后把它们合在一起形成一棵字母树。

接下来的问题就是这棵字母树上有多少条互不相同的从上至下的路径。我们对它建立后缀自动机后，对每个节点将$len - len_{fa}$计入答案即可。

4 总结

后缀自动机作为OI中新兴的字符串处理工具，具有功能全面，代码简洁，复杂度低，常数小等优点。相应的，它需要我们在应用过程中进行更多的思考与研究，才能理解后缀自动机的优美之处。

5 鸣谢

- 感谢CCF提供的学习与交流的平台。
- 感谢引进后缀自动机的陈立杰。
- 感谢教会我后缀自动机的石文斌同学。
- 感谢陈胤伯、吕凯风同学在论文写作过程中的帮助。
- 感谢教练周祖松老师的支持。

[16]**题目来源：ZJOI2015 题目作者：陈立杰**

参考文献

[1] 陈立杰，《后缀自动机》，2012冬令营营员交流。

[2] 范浩强，《后缀自动机与线性构造后缀树》。

[3] 许昊然，《CTSC2010珠宝商 新解》，2013国家集训队作业。

[4] 张天扬，《APIO2014回文串 解题报告》，2015国家集训队作业。

[5] 陈立杰，《ZJOI2015 Day1 题解》。

[6] 黄志翱，《Feyat Cup 1.5 题解》。