浅谈字符串匹配的几种方法

绍兴市第一中学 王鉴浩

摘 要

字符串匹配问题是信息学竞赛中的经典问题。本文对于这类问题的解法进行了分类归纳。本文先简要总结了一些常用的结构和算法，然后着重介绍了一个新颖的字符串匹配结构 *Border Tree*，并推出了一类字符串匹配问题的通用解法。本文中每种算法和技巧都有例题，便于读者理解与分析。

1 引言

字符串匹配问题经常应用于文本编辑中。现在处于大数据时代的我们，在许多领域中需要解决一些操作多样、数据量大的问题。信息学竞赛中字符串匹配问题便是这类问题的模型，本文将会对这类问题的一些解法进行介绍。

在本文的第二节中，为了之后描述的方便，将会对一些字符串的定义进行回顾。

然后本文会对一些结构和算法进行简要总结。本文中把这些结构和算法分为了两类：前缀类解法和后缀类解法。

在本文第三节中作者将介绍一些前缀类结构和算法：*Trie*，*KMP*，*AC* 自动机。第四节中会介绍一些后缀类结构：后缀树、后缀数组、后缀仙人掌和后缀自动机。第五节中作者将会介绍一个新颖的结构：*Border Tree*，并进行详细分析。第六节中作者将会基于第五节中的 *Border Tree* 推出一类字符串匹配问题的通用解法。

其中第三节和第四节中的内容比较普及，有些算法以前的国家集训队论文已有介绍，所以作者只对其进行简要总结，并对于一些经典的问题进行分析讨论。

作者会对于第五节和第六节花大量篇幅介绍。其中 *Border Tree* 是比较新颖的结构，作者会提出一类应用抛砖引玉，有兴趣的读者可以继续研究。

2 定义

为了之后描述的方便，先给出如下定义：

定义 2.1 (子串，前缀，后缀，*Border*，*LBorder*)**.**

设一个长度为 n 的字符串 $s = s_1s_2s_3...s_n$。

对于 $1 \le i \le j \le n$，称 $s_is_{i+1}s_{i+2}...s_j$ 为 s 的一个子串，记成 $s[i:j]$。特别地，如果 $i > j$，则 $s[i:j]$ 表示空串，记作 $\emptyset$，空串也是 s 的一个子串。

对于 $1 \le i \le n$，称 $s[i:n]$ 为 s 的一个后缀，记成 $suf[i]$；称 $s[1:i]$ 为 s 的一个前缀，记成 $pre[i]$。

对于 $1 \le i < n$，如果 $s[1:i] = s[n-i+1:n]$，则称 $s[1:i]$ 为 s 的一个 *Border*，特别地，空串 $\emptyset$ 也称为是一个 *Border*；在 s 的 *Border* 中，称最长的 *Border* 为 s 的 *LBorder*。

我们又称 $pre[i]$ 的 *LBorder* 为 $LBorder_i$。

定义 2.2 (周期串，基，周期的长度)**.**

设一个长度为 n 的字符串 $s = s_1s_2s_3...s_n$。

如果串 s 的 *LBorder* 长度为 L，把 $s[L+1:n]$ 称作是 s 的基。

如果 $2L \ge n$，那么 s 串被称为是周期串。

如果 s 串是周期串，那么称 s 串的基的长度是周期的长度。

定义 2.3 (重复串)**.**

如果某个串 t 是由一个串 s 重复 k $(k > 1)$ 次后得到的，那么称 t 是重复串，记为 s^k。

显然，重复串是一类特殊的周期串。

3 前缀类解法

在信息学竞赛的字符串匹配解法中，有一些结构和算法是根据高效地利用前缀的信息来实现对字符串快速匹配。在本文中，我们称这类解法为前缀类解法。在本节中，本文会介绍 3 种前缀类解法及其变种。

3.1 Trie

Tire 是一棵单词查找树，在下文中我们称之为字母树。

字母树是能支持对多个模版串进行查找和匹配的数据结构。关于字母树的定义和构造方法请详见朱泽园 2004 年的集训队论文或董华星 2009 年的集训队论文。

字母树是一棵有根树，每条树边表示一个字符，把从根到点 i 路径上的边按遍历顺序排列，能构成一个串 s_i。所以，我们称字母树上每个点都表示一个串 s_i，其中根表示的串为 $\emptyset$。

通常地，对于多个模版串匹配，会把每个模版串加进字母树中，根据字母树的定义，可以发现每个模版串的前缀都能被字母树中一个点表示。那么，就可以利用树的性质高效地解决一些字符串匹配问题了。比如，快速地计算两个模版串的最长公共前缀等等。

接下来，让我们通过一个例子更好地理解字母树。

3.1.1 例题

例 1 (USACO 2012 Dec gold 2 *First*)**.**

给出 n 个由小写字母组成的字符串，第 i 个串表示为 s_i。

现在，26 个小写字母的字典顺序任意，问每个串是否能在某种字典顺序中成为字典序最小的串。

字符串保证不重复。

数据范围：$1 \le n \le 30000, 1 \le$ 字符串总长度 ≤ 300000

时限：$1s$

对于此题，可以先把这 n 个串建字母树。对于第 i 个串，设字母树中点 j 表示的串为 s_i。如果要使得第 i 个串在某种字典顺序中字典序最小，那么首先要满足在这 n 个串中没有一个别的串为 s_i 的前缀，其次要求在根到点 j 的路径上，每一个字符都要比同层的其他字符的字典顺序小。

那么对于每个串，可以把这些限制拿出来，然后判断这些限制是否会矛盾。具体实现的时候，可以把限制看作是有向边，然后判断这张有向图是否会有环。

此题的时间复杂度为 $O(26^2n + 26 \times$ 字符串总长度$)$。

3.2 KMP

信息学竞赛中会出现模式串和主串匹配的问题。在暴力字符串匹配过程中，会从第一位开始匹配，如果相等则匹配下一个字符，直到出现不相等的情况。此时人们会简单的丢弃前面的匹配信息，然后以主串的下一位和模式串的第一位开始重新匹配，循环进行，直到主串结束，或者出现匹配的情况。这种简单地丢弃前面的匹配信息的做法，造成了极大的浪费和低下的匹配效率。

可以使用 *KMP* 算法来加速模式串与主串匹配的速度，*KMP* 算法具体请参见朱泽园 2004 年的集训队论文。

下面给出模式串每一位失配的数组的定义：

定义 3.1 (失配数组)**.**

设一个长度为 n 的模式串 $s = s_1 s_2 s_3 ... s_n$。

令 $next[i] = |LBorder_i|$，并称 *next* 数组为 s 的失配数组。

KMP 算法的核心思想就是先在模式串中预处理出失配数组。然后对于主串根据模式串的失配数组，计算出以每一位结束最多能与模式串匹配的长度。其中 *KMP* 算法的时间复杂度为线性。

3.2.1 KMP 自动机

当建出 *next* 数组后，可以发现如果把 $next[i]$ 向 i 连边后，整张图就构成了一颗 $n+1$ 个点的树。其中，0 号点为根，i 号点表示为 $pre[i]$。从根到点 i 路径上的点都为 $a[1:i]$ 的 *Border*。于是，就可以根据这棵树建出每个点的转移数组，表示每个串后加一个字符形成的新串最多能匹配模式串几位。在串匹配时，就可以根据转移数组进行快速计算了。我们把上述结构称为是 *KMP* 自动机。

KMP 自动机的具体树的形态可以见图 1 (图片在第 51 页)。

3.2.2 例题

例 2 (NOI 2014 动物园)**.**

有 T 组数据，每组数据给出一个长度为 n 的模式串 $s = s_1 s_2 s_3 ... s_n$。

在 $s[1:i]$ 的非空 *Border* 中，满足 $2|Border| \le i$ 的 *Border* 数量记为 $num[i]$。

要求求出 *num* 数组。

数据范围：$1 \le T \le 5, 1 \le n \le 1000000$

时间限制：1s

一种解决此题方法就是，先建出 *next* 数组后，再根据 *next* 数组，建出 KMP 自动机中的树。对于每个点，我们只需要知道根到这个点的路径上有多少标号不超过当前点标号一半的点。由于标号是单调的，所以直接二分就好了。又由于建 *next* 数组的时间是线性的。那么此题总共的时间复杂度为 $O(Tn\log n)$。

3.3 AC 自动机

AC 自动机可以看成是通过 $Tire$ 和 KMP 自动机相互结合得到的新结构。构建 *AC* 自动机是对于多个模式串先建 $Tire$，然后像 KMP 自动机一样建出 *next* 数组来构建树和转移数组。不过这个 *next* 数组是针对多个串的，不同于 KMP 自动机的 *next* 数组。我们称在 *AC* 自动机中，像 KMP 自动机那样用 *next* 数组构成的树叫 $fail$ 树。可以发现在 $fail$ 树中，每个点 i 都能表示一个串 v_i。而且对于点 i 子树中的点，每个点表示的串都有 v_i 这个前缀。于是，根据 $fail$ 树我们就可以构出每个点的转移数组。在串匹配时，就可以通过转移数组进行快速匹配。

AC 自动机具体性质和构造请参见朱泽园 2004 年的集训队论文。

如此，*AC* 自动机就结合了 $Tire$ 和 KMP 自动机的性质，能解决更复杂的多模式串匹配的问题。

接下来，本文给出一个例题让读者更好地理解 *AC* 自动机。

3.3.1 例题

例 3 (Coci 2015 *Divljak*)**.**

一开始给出 n 个字符串：$s_1, s_2, s_3, ..., s_n$。有一个字符串集合 T，T 一开始是空的。

有 q 个操作，有两种操作：

1. 向字符串集合 T 中添加一个字符串 p。
2. 选择一个串 s_x，问集合 T 中有多少个串，使得 s_x 是其子串。

设 sum_1 为 T 中的字符总数，$sum_2 = \sum_{i=1}^{n} |s_i|$。

字符为小写英文字母。

数据范围：$1 \le n, q \le 100000, 1 \le sum_1, sum_2 \le 2000000$

时限：$1s$

首先对于 $s_1, s_2, s_3, ..., s_n$，可以先建出 AC 自动机来得到 $fail$ 树和转移数组。设在 $fail$ 树中点 d_i 表示的串为 s_i。

接下来，对操作进行一些转化。对于操作一，可以认为是在 AC 自动机中，把那些能表示为 p 串的前缀的点染成一种颜色。对于操作二，可以认为答案就是在 $fail$ 树中点 d_i 的子树的颜色种数。对于这个转化，读者可以通过 AC 自动机的性质来自行理解。

实际实现中，可通过 $fail$ 树的 dfs 序统计答案。一个基本的思想是：由于树上的每个子树对应到 dfs 序都是一个区间，那么每次染色之后，要使得子树中有这个颜色的点，其 dfs 序的区间权值和会加一。那么对于操作二，可以把直接点 d_i 的子树对应到 dfs 序上得到一个区间，然后用树状数组统计区间权值和来得到答案。

对于操作一，根据上述的基本思想，具体的实现是：把那些需要染色的点拿出来，显然这个点数是 $O(|p|)$ 级别的。对于这些点，先把它们去重，然后根据 dfs 序进行排序；接下来，把每个点的权值加一，并在相邻两个点的树上最近公共祖先 (lca) 处把权值减一。经过上述操作，就可以满足：在树上的每个子树中，如果有染色的点的话，只有 dfs 序最小的那个点会有 1 的染色贡献。读者可以结合虚树的形态来更好地理解上述实现。

于是此题就可以在 $O(26sum_1 + sum_2 \log sum_1 + sum_2 \log sum_2)$ 的时间复杂度内解决。

4 后缀类解法

在信息学竞赛的字符串匹配解法中，后缀类解法有着重要的地位。与前缀类解法相比较，后缀类解法显得更加高端和复杂，能解决更加复杂的问题。在本节中，本文将介绍 4 种后缀类数据结构：后缀数组、后缀树、后缀仙人掌和后缀自动机。

4.1 后缀数组

后缀数组是字符串处理中非常优秀的数据结构，是一种处理字符串的有力

工具，在不同类型的字符串问题中有广泛的应用。

后缀数组的核心思想是对于一个模式串 b，把 b 串中的每个后缀按照字典序排序，用 sa 数组记录。其中 sa_i 表示：在 b 串的所有后缀中字典序第 i 小的后缀的位置。相对地，还能得到一个 $rank$ 数组，$rank_i$ 表示 b 串中的 $suf[i]$ 在 sa 数组中的位置。得到 sa 数组和 $rank$ 数组后，还能通过后缀数组的性质用线性复杂度得到 $height$ 数组。其中，$height_i$ 表示 sa_{i-1} 和 sa_i 这两个后缀的最长公共前缀 (LCP)的长度。运用 $height$ 数组，可更高效地计算一些问题。比如，求 b 串两个子串的 LCP 等等。

对于构建后缀数组，可以运用倍增算法或 $DC3$ 算法。但由于篇幅有限，本文就不详细介绍后缀数组具体性质和构造了，有兴趣的读者可以参见许智磊 2004 年的集训队论文或罗穗骞 2009 年的集训队论文。

接下来，本文将对于关于后缀数组的 3 个经典问题进行分析。

4.1.1 问题一

例 4 (最长公共前缀)**.**

给出一个长度为 m 的串 b。

有 q 个询问，每个询问在 b 串上给出两个子串 (b', b'')，问这两个子串的最长公共前缀的长度。

对于这个问题，可以先用倍增算法在 $O(m \log m)$ 的时间复杂度内把串 b 做后缀数组，得到数组 sa，$rank$ 和 $height$。然后对于每次询问两个子串的 LCP，可以把这两个子串加长成为两个后缀。先求出这两个后缀的 LCP，然后再对 $|b'|$ 和 $|b''|$ 取 min 就可以得到答案了。而对于询问两个后缀的 LCP，可以把这两个后缀对应到 sa 数组上，根据 $height$ 数组的性质，把问题转化为在 $height$ 数组上的 rmq 问题。对于在 $height$ 数组上的 rmq 问题，我们可以先在时间复杂度为 $O(m \log m)$ 内建出 ST 表，然后对于每个询问，在 $O(1)$ 的时间复杂度内计算答案。

对此问题，上述解法的时间复杂度是 $O(m \log m + q)$。

4.1.2 问题二

例 5 (子串拼合问题)**.**

给出一个长度为 m 的串 b。

有 q 个询问，每个询问在 b 串上选择两个子串 (b', b'')，令 $s = b'b''$。询问串 s 是否是 b 串的子串。如果是子串的话，给出一个合法的位置。

对此问题，可以先用倍增算法在 $O(m \log m)$ 的时间复杂度内把串 b 做后缀数组，得到数组 *sa*，*rank* 和 *height*。然后可以发现，如果串 s 是 b 串的子串的话，那么串 s 就可以表示为 b 串中某一个后缀的前缀。于是，就先把所有以 b' 为前缀的后缀都找出来。根据 *sa* 数组的定义，可以发现这些后缀对应到 *sa* 数组上是连续的一段，这一段可以通过二分得到。现在要在这个区间中找到一个以 s 串为前缀的后缀。由于 *sa* 数组是把那些后缀按照字典序排序，那么又可以根据字典序，在这个区间中继续二分，寻找以 s 串为前缀的后缀。在二分的过程中，可以结合 *rank* 数组和 *LCP* 来进行快速判断。由于在问题一中，我们已经可以支持在 $O(1)$ 的时间复杂度内进行 *LCP* 的计算，那么二分的时间复杂度还是 $O(\log m)$。

上述解法的时间复杂度为 $O(m \log m + q \log m)$。

4.1.3 问题三

例 6 (区间最大前缀问题)**.**

给出一个长度为 m 的串 b。

有 q 个询问，每个询问在 b 串上选择给一个区间 $[l, r]$，计算：

$$\max_{i=l}^{r} LCP(suf[i], b)$$

对此问题，可以先用倍增算法在 $O(m \log m)$ 的时间复杂度内把串 b 做后缀数组，得到数组 *sa*，*rank* 和 *height*。对于此题，我们可以先对每个 i 计算 $LCP(suf[i], b)$。于是，该题就转化成了 *rmq* 问题了。与问题一同理，可以先在时间复杂度为 $O(m \log m)$ 内建出 ST 表，然后对每个询问，在 $O(1)$ 的时间复杂度内计算答案。那么对于这个问题，上述解法的时间复杂度是 $O(m \log m + q)$。

此外，此题还有一个拓展：我们可以在计算最大值的同时统计最大值的个数。但是如果要统计最大值的个数，就不能使用 ST 表了。我们可以通过倍增来计算答案，在 $O(\log m)$ 的时间复杂度内完成对单次询问的计算。这个解法的时间复杂度是 $O(m \log m + q \log m)$。

4.2 后缀树

后缀树是处理字符串的经典数据结构。后缀树的本质是字母树。对于一个长度为 m 的字符串 b，如果把它的所有后缀建字母树，就得到了 b 串的后缀树了。但由于这样建字母树的话，字母树中的点会达到 $O(m^2)$ 级别，所以需要把这棵后缀树的路径进行压缩。原本的字母树一条边只表示一个字符，现在让每条边表示一个字符串。可以发现如果有 m 个后缀要加入后缀树中的话，那么这个字母树的边数就变成 $O(m)$ 级别了。简单来说，就是先在串 b 后加一个终止符，建出一棵点数是 $O(m^2)$ 的字母树后，把那些只有一个儿子的点和它的儿子合并，我们就得到一棵点数为 $O(m)$ 级别的后缀树了。

关于后缀树具体的性质分析和构造方法请参见朱泽园 2004 年的集训队论文。

当构建出后缀树之后，可以发现在上一节中的 sa 数组其实就是后缀树的 dfs 遍历数组。于是，就可以使用后缀树建后缀数组了，那么显然地，所有后缀数组能做的问题，后缀树都是可以支持的。而且在下一节中，本文就会介绍一种用后缀数组建后缀树的方法。由于这两个数据结构是可以互通的，在本节中，本文就不对后缀树进行举例分析了，有兴趣的读者可以根据后缀数组的经典问题自行理解分析。

4.3 后缀仙人掌

后缀仙人掌其实和后缀数组一样都是基于后缀树的。且后缀仙人掌和后缀树类似的，是一个树状结构。在后缀树中，我们是把那些只有一个儿子的点和它的儿子合并，但在后缀仙人掌中，我们是把每个非叶子结点都与一个儿子合并，所形成的连接体称为树枝。

对于构建后缀仙人掌，可以先建出后缀数组，然后根据 sa 数组和 $height$ 数组的定义，在 sa 数组升序枚举的同时维护一个 $height$ 数组的递减单调栈。对于每次枚举到的位置 i，把 sa_i 和目前栈顶元素连边。

由于篇幅有限，上述构造的正确性本文在这里就不再详述。关于后缀仙人掌的具体性质请参见王悦同，徐毅和徐子涵 2014 年的冬令营营员交流。

那么，根据后缀数组，即可得到后缀仙人掌。接下来，思考后缀仙人掌的定义，可以发现，如果把后缀仙人掌的每个树枝根据连边位置断开，就可得到后缀树。所以，只需要在后缀仙人掌连边的同时记录连边的位置，就可以根据

后缀数组得到后缀树了。而且这个根据后缀数组通过后缀仙人掌构建后缀树的方法的时间复杂度是线性的。

4.4 后缀自动机

后缀自动机的树是一个模式串逆序的后缀树。由于后缀树是字母树，所以后缀自动机也支持对多个模式串进行维护。于是，可以得到构建一个模式串后缀树的另一种方法：可以先建出后缀自动机，然后通过后缀自动机建后缀树。我们可以通过增量法构建后缀自动机。由于篇幅有限，关于后缀自动机的构造和性质分析请参见陈立杰 2012 年的冬令营营员交流。

接下来，本文给出一个例题让读者更好地理解后缀自动机。

4.4.1 例题

例 7 (ZJOI 2015 诸神眷顾的幻想乡)**.**

给一棵 n 个点的树。每个点上有一个字符。

每次选择树上两个点 x, y，把点 x 到点 y 路径上的字符按遍历顺序排列得到一个字符串。对树上每一对点都计算出字符串。问这 n^2 个字符串中有多少个不同的字符串。

树的叶子个数不超过 20 个，字符集大小不超过 10。

数据范围：$1 \le n \le 100000$

时限：$1s$

考虑暴力的做法。由于只有 20 个叶子，可以把每对叶子之间的字符串拿出来，可以发现，叶子之间的字符串的每个子串都能对应一条路径，而且每条路径都至少被一个子串对应。所以，可以把这 400 个字符串建后缀自动机，然后在后缀自动机上统计不同的字符串个数。由于每个字符串长度是 $O(n)$ 级别的，那么上述做法的时间复杂度为 $O(20^2 n \times 10)$。

更优地，可以考虑每个叶子当起点时的情况。把这个叶子当根的话，这个叶子到其他所有叶子之间的字符串我们可以理解为把整棵树遍历了一遍。那么，每个叶子当根得到的总字符串长度是 $O(n)$ 级别的。此题的时间复杂度为 $O(20n \times 10)$。

5 Border Tree

在本节中，本文将会介绍 *Border Tree* 这个新颖的数据结构。这个数据结构基于 *KMP* 自动机，能高效地处理一类单模版串匹配问题。

设模版串是长度为 m 的串 b。

如果把串 b 做 *KMP* 自动机，那么 *KMP* 自动机中树的深度是可以到达 $O(m)$ 级别的。*Border Tree* 是 *KMP* 自动机中树的变种，它能通过合并一些同样类型的前缀，使得把 *KMP* 自动机中树的深度降到 $O(\log m)$。也就是每个点表示的并非是一个前缀而是一组前缀了。这一组前缀称为一个 *Border Group*。

Border Tree 具体形态见图 2 (图片在第 52 页)。

5.1 Border Group

考虑 $b[1:i]$ $(1 \le i \le n)$ 的 *Border*。*Border* 的数量是可以达到 $O(i)$ 级别的，例如：$b[1:i] = \underbrace{111...1}_{i\text{ 个 }1}$。

我们需要压缩这些信息。基本的思想是我们把这些 *Border* 分类，把 $b[1:i]$ 的 *Border* 的种类数控制在 $O(\log i)$ 的级别。

定义 5.1 (*Border Group*)**.**

$b[1:i]$ 的 *Border* 可以分类得到序列 g：

$$g = \left[B_1, B_2, B_3, ..., B_{|g|}\right] \ (1 \le |g| \le O(\log i))$$

把 $b[1:i]$ 的 *Border* 按照长度递减排序，令 B_i 是保存连续的一段 *Border* 的序列。并且要满足：在 B_i $(1 \le i < |g|)$ 内的 *Border* 要比 B_{i+1} 内的 *Border* 长。

其中，B_i 内的 *Border* 序列是：

$$\left[\pi'_i\pi_i^{k_i}, ..., \pi'_i\pi_i^3, \pi'_i\pi_i^2\right] \ (k_i \ge 2)$$

或是：

$$\left[\pi'_i\pi_i^{k_i}, ..., \pi'_i\pi_i^3, \pi'_i\pi_i^2, \pi'_i\pi_i\right] \ (k_i \ge 1)$$

满足：π'_i 是 $b[1:i]$ 的某一个前缀，π_i 是 $b[1:i]$ 的基。

那么则称 B_i 是 *Border Group*，称 g 为 *BG* 序列。

下面给出一个例子来更好地理解 *Border Group*。

例 8.

设串 b = bababbababcbababbabab。

那么可以发现有 4 个 *Border*：[bababbabab, babab, bab, b]。

其中可以把这 4 个 *Border* 分成 3 类：[bababbabab], [babab, bab], [b]。

其中第一类为：π = babab, π' = $\emptyset$，第二类为：π = ab, π' = b，第三类为：π = b, π' = $\emptyset$。

现在来构造 *BG* 序列 g，其主要思想是：用 *Border Group* 把所有的 *Border* 分配进不同的序列中。每个 *Border Group* 中的 *Border* 含有相同的 π' 和 π。具体方法如下：

1. 令 $\pi'\pi^k = LBorder_i$，把序列 $\left[\pi'\pi^k, ..., \pi'\pi^3, \pi'\pi^2\right]$ 加入 B_1。
2. 令 $s = \pi'\pi$。若 s 不是周期串，那么把 s 加入 B_1，并且把 s 的 *LBorder* 作为 B_2 的第一个元素，否则把 s 作为 B_2 的第一个元素。
3. 当 π' 和 π 的 *LBorder* 同时为 $\emptyset$ 的时候结束构造。

可以发现这样构造可以保证 $1 \le |g| \le O(\log m)$：

Proof.

在第 2 步中，如果 s 是周期串，那么 s 是下一个 *Border Group* 的第一个元素。根据定义，我们可以得到 $|\pi'| < |\pi|$，于是 $|s| \le \frac{2}{k+1}\left|\pi'\pi^k\right|$ $(k \ge 2)$。由于当 $k = 2$ 时，$\frac{2}{k+1}\left|\pi'\pi^k\right|$ 最大，所以对于任意的 k，都有 $|s| \le \frac{2}{3}\left|\pi'\pi^k\right|$。

否则 s 不是周期串，那么 $LBorder_{|s|}$ 是下一个 *Border Group* 的第一个元素。根据 *LBorder* 的定义，$2\left|LBorder_{|s|}\right| < |s|$，又因为 $|s| = |\pi'\pi| \le \left|\pi'\pi^k\right|$，所以 $2\left|LBorder_{|s|}\right| < \left|\pi'\pi^k\right|$。

于是 $\left|\pi'_{i+1}\pi^{k_{i+1}}_{i+1}\right| \le \frac{2}{3}\left|\pi'_i\pi^{k_i}_i\right|$，所以这样构造可以保证 $1 \le |g| \le O(\log m)$。 □

5.2 构建 Border Tree

对于构建 *Border Tree*，首先要构建出 *KMP* 自动机，然后把自动机中的树转化成 *Border Tree*。

构建 *KMP* 自动机请参见本文的第三节。

对于 *KMP* 自动机的每个点 i，定义一个三元组 (x_i, y_i, z_i)：

1. 如果 $b[1:i]$ 是周期串，即 $b[1:i] = \pi'\pi^k$ $(k > 1)$，那么 $(x_i, y_i, z_i) = (|\pi'|, |\pi|, k)$。
2. 如果 $b[1:i]$ 不是周期串，即 $b[1:i] = \pi'\pi$，那么 $(x_i, y_i, z_i) = (|\pi'|, |\pi|, 1)$。
3. 如果 $b[1:i]$ 不是周期串，但 i 有一个儿子 j，满足 $z_j = 2$，那么 $(x_i, y_i, z_i) = (x_j, y_j, 1)$。

在情况三中，有如下定理，保证 i 最多只会有一个儿子 j，满足 $z_j = 2$：

定理 5.1. 每个非周期串 $s = b[1:i]$ 一定不会有两个及以上的儿子是周期串。

Proof.

我们使用反证法。

假设 s 有两个及以上的儿子是周期串，则 s 可以根据它的某两个周期串儿子，表示成：$s = \pi_1'\pi_1 = \pi_2'\pi_2$，满足：$|\pi_1| < |\pi_2|, |\pi_2| \bmod |\pi_1| \neq 0$。

因为当 $|\pi_2| \bmod |\pi_1| = 0$ 时，s 一定能表示为 $\pi_2'\pi_1^x$ $(x > 1)$ 的形式，于是 s 就是周期串了，和我们的定理矛盾。

设 $d = \gcd(|\pi_1|, |\pi_2|)$，那么 s 一定存在一种表示方式：$s = \pi_3'\pi_3$，其中 $|\pi_3| = d$。

那么将导致 $|\pi_2| \bmod |\pi_3| = 0$，于是结论和我们的条件矛盾，所以假设不成立，原命题正确。 □

计算 (x_i, y_i, z_i) 是可以经过两次遍历做到线性复杂度的：

1. 第一次遍历时，对满足情况一和情况二的点，根据 *Border Group* 中的定义，令：

$$x_i = |\pi| = i - |LBorder_i|$$
$$y_i = |\pi'| = i \bmod |\pi|$$
$$z_i = \lfloor \frac{i}{|\pi|} \rfloor$$

2. 第二次遍历时，对满足情况三的点修改 (x_i, y_i, z_i)。

其中，情况三存在的意义在于维护 *Border Group* 中的 $\pi'\pi$。下面给出一个例子来理解情况三。

例 9.

设串 b = abaaba。

本来 $(x_3, y_3, z_3) = (1, 2, 1)$，但由于 $(x_6, y_6, z_6) = (0, 3, 2)$，于是 (x_3, y_3, z_3) 被修改成了 $(0, 3, 1)$。

于是就把 (x_i, y_i, z_i) 维护好了。

现在已经给了 *KMP* 自动机的每个点一个三元组。接下来，我们要通过修改自动机中的树来构造 *Border Tree*。

对于 x_i, y_i 都相同的点，根据 *Border Group* 的定义，它们属于同一类，就只保留 z_i 最小的那个点，并在那个点上记录这些点的信息。比如，如果一类点中有能表示为 $\pi'\pi$ 的点，那么保留这个点，否则保留能表示为 $\pi'\pi^2$ 的点。

最后再经过一次遍历，就可以把 *Border Tree* 构出来了。通过图 2 (图片在第 52 页)，我们可以更好地理解 *Border Tree* 。

5.3 分析性质

对于 *Border Tree* 中每个点 i 都对应了一个 *Border Group*。

这个 *Border Group* 可以表示为：

$$\left[\pi'_i\pi_i^{l_i}, \pi'_i\pi_i^{l_i+1}, \pi'_i\pi_i^{l_i+2}, ..., \pi'_i\pi_i^{r_i}\right]$$

但实际上，对于点 i，根到其路径上每个点对应的 *Border Group*，并不能准确地表示 $b[1:i]$ 的 *Border*。因为已经把边缩起来了。

例如图 1 中，$b[1:7]$ 并非是 $b[1:13]$ 的 *Border*，但在图 2 的 *Border Tree* 中，$b[1:7]$ 是被分在了点 3 的类型中。而点 3 是点 13 的父亲，处于根到点 13 的路径中。

设 $fa[i]$ 表示在 *Border Tree* 中点 i 的父亲。在 $fa[i]$ 这个 *Border Group* 中，$a[1:i]$ 的 *Border* 表示为：

$$\left[\pi'_i\pi_i^{l_i}, \pi'_i\pi_i^{l_i+1}, \pi'_i\pi_i^{l_i+2}, ..., \pi'_i\pi_i^{x}\right]$$

其中 $x = \min(z_{next[i]}, r_i)$。($z_{next[i]}$ 指的是在 *KMP* 自动机中，点 i 的父亲 $next[i]$ 所对应的三元组中的 z)

所以当需要查找 $b[1:i]$ 所有的 *Border* 时，需要对从根到 i 的路径上的点实时维护 x 值。

5.4 例子

设串 b = bababababcbabab。串 b 的长度是 13。下图为串 b 的 *KMP* 自动机中的树和 *Border Tree* 的形态。本来自动机中的树有 14 个点，深度为 5，经过把 *Border* 分类后建成 *Border Tree* 后变成有 10 个点，深度为 3 的树了。构造 *Border Tree* 时，图 1 中方形的点会缩起来，读者可以通过图 2 来理解构造。

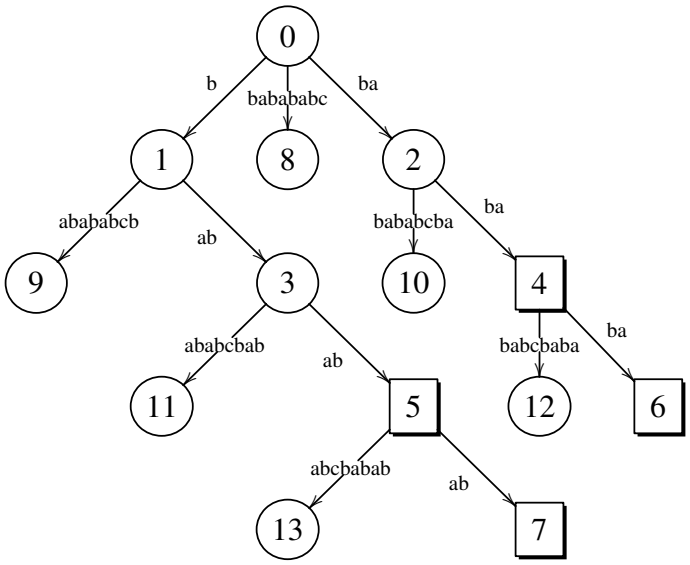

图1：*KMP* 自动机

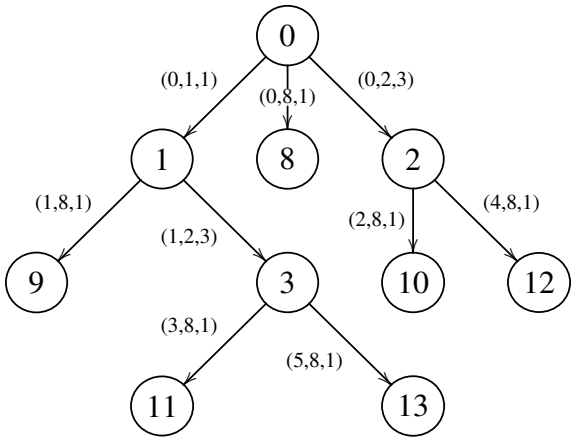

图2：*Border Tree*

读者可以根据下表来更好地对 *KMP* 自动机和 *Border Tree* 进行比较。

KMP 自动机和 *Border Tree* 中各个点的状态

点编号/	*KMP* 自动机中的 (x_i, y_i, z_i)			*Border Tree* 中的 (x_i, y_i, z_i)
前缀长度	π'	π	k	
1		b	1	(0, 1, 1)
2		ba	1	(0, 2, 3)
3	b	ab	1	(1, 2, 3)
4		ba	2	
5	b	ab	2	
6		ba	3	
7	b	ab	3	
8		bababab**c**	1	(0, 8, 1)
9	b	ababab**cb**	1	(1, 8, 1)
10	ba	babab**cba**	1	(2, 8, 1)
11	bab	abab**cbab**	1	(3, 8, 1)
12	baba	babc**baba**	1	(4, 8, 1)
13	babab	abc**babab**	1	(5, 8, 1)

6 一类文本匹配问题

众所周知，动态文本串与静态模式串的匹配问题是字符串匹配的经典问题。本节将对这类文本匹配提出一种基于 *Border Tree* 来实现的高效通用解法。其中，关于 *Border Tree* 的相关内容请详见上一节。

6.1 子问题

设模式串是长度为 m 的 b 串。

处理这一类问题之前，需要先解决 3 个在 b 串上的子问题：

1. 最长公共前缀 (LCP)：在 b 串上给两个子串 (b', b'')，问这两个子串的最长公共前缀长度。

   解法详见例 4[1]，预处理时间复杂度为 $O(m \log m)$。单次询问时间复杂度为 $O(1)$。

2. 子串拼合问题：在 b 串上两个子串 (b', b'')，令 $s = b'b''$，问串 s 是否是 b 串的子串。如果是子串的话，给出一个合法的位置。

   解法详见例 5，预处理时间复杂度为 $O(m \log m)$。单次询问时间复杂度为 $O(\log m)$。

3. 区间最大前缀问题：在 b 串上给一个区间 $[l, r]$。询问：

$$\max_{i=l}^{r} LCP(suf[i], b)$$

   解法详见例 6，预处理时间复杂度为 $O(m \log m)$。单次询问时间复杂度为 $O(1)$。如果还需要计算答案最大化时 i 合法的个数，那么单次询问时间复杂度为 $O(\log m)$。

6.2 模型转换

首先，把动态文本串与静态模式串的匹配问题简化为一个基础问题：给一个文本串 a 和一个模式串 b，每次修改 a 串后都与 b 串匹配一遍。修改操作可以是 a 串中修改某一位或是插入或删除一位。

[1]例题在第 43 页

然后转化模型，把 a 串与 b 串匹配问题变成求如下函数的值：

定义一个关于 a 串和 b 串的函数 $F(a,b)$，$F(a,b)$ 的值是一个二元组 (x,y)，x,y 的含义如下：

- $x = \max\limits_{i=1}^{|a|} LCP(a[i:|a|],b)$
- y 表示当 x 最大化时，合法的 i 的个数。

问题就转化成了：给出 a 串和 b 串，求 $F(a,b)$，其中 a 串为动态文本串，b 串为模式串。

6.3 覆盖

在本小节中，将提出覆盖这个概念。

设模式串为长度为 m 的 b 串，文本串为长度为 n 的 a 串。

覆盖的基本思想为：用模式串 b 中的子串去精确覆盖 a 串。

下面给出覆盖的定义：

定义 6.1.

一个关于 b 串和 a 串的覆盖 $\mathcal{G}$，是一个字符串的序列 $[v_1,v_2,v_3...v_t]$，而 v_i 被称为是覆盖的一个元素。

覆盖需满足 $v_1v_2v_3...v_t = a$，且对于每个元素 $v_i\ (1 \le i \le t)$，均满足以下两条性质：

- 子串性质：v_i 是 b 串的子串。
- 极大化性质：若 $i < t$，则 v_iv_{i+1} 不是 b 串的子串。

特别地，对于 a 串的第 x 位，若 a_x 这个字符没有出现在 b 串中，我们也把 a_x 作为一个 v_i，不过这个 v_i 在 b 串中对应的是空串。

由于覆盖方式可能有多种，所以合法的 $\mathcal{G}$ 也可能有多种，具体实现时，只需取任意一种合法的 $\mathcal{G}$ 即可。

对于 $\mathcal{G}$ 可以简单地理解为 v_i 把 a 串分裂成了 t 个小块。

现在对于每个 v_i 用一个三元组 (s_i,l_i,r_i) 来表示：

- l_i, r_i 表示 v_i 在 b 串中的位置：$v_i = b[l_i : r_i]$。

  特殊地，若 v_i 在 b 串中表示的是空串，那么令 $l_i = r_i = 0$。

- s_i 表示 v_i 在 a 串中的位置，令 $s_i = 1 + \sum_{j=1}^{i-1} |v_j|$。

那么先来考虑如何初始化 $\mathcal{G}$。具体操作如下：

1. 新建一个数组 c，大小为字符集大小。对于第 i 这个字符，如果在串 b 中的第 x 位出现了，那么 $c_i = x$，否则 $c_i = 0$。

2. 对于 $1 \le i \le |a|$，我们认为 a_i 在 b 串中出现的位置是 c_{a_i}。如果 $c_{a_i} = 0$ 表示 a_i 并没有在串 b 中出现。我们令 $v_i = (i, c_{a_i}, c_{a_i})$。

3. 这样构造 v_i 并不满足极大化性质。那么要通过合并相邻的元素来满足极大化性质。根据 s_i 递增的顺序依次判断 $v_i v_{i+1}$ 是否是 b 串的子串。如果是的话，那么合并，否则不合并。对于这个判断，可以使用子问题 2[2] 来解决。

现在已经初始化 $\mathcal{G}$ 了。接下来考虑如何动态维护 $\mathcal{G}$。

修改操作可以是在 a 串中插入、删除或是修改某一位，在下文分析中只考虑修改某一位，插入、删除一位的做法类似。

对于修改某一位，现在需要动态维护 $\mathcal{G}$，具体做法如下：

1. 对于把第 x 位改为 y，可以先找到覆盖 x 的 v_i。由于是精确覆盖，所以 v_i 有且只有一个。

2. 根据第 x 位，把 v_i 分裂成左中右 3 个子元素：$v_{i'}, v_{i''}, v_{i'''}$，并用三元组来表示它们：

$$v_{i'} = (s_i, l_i, l_i + (x - 1 - s_i))$$
$$v_{i''} = (x, l_i + (x - s_i), l_i + (x - s_i))$$
$$v_{i'''} = (x + 1, l_i + (x + 1 - l_i), r_i)$$

   那么现在，第 x 位恰好被 $v_{i''}$ 精确覆盖。根据修改，令 $v_{i''} = (x, c_y, c_y)$，再在这 3 个子元素中，把在 a 串中表示为 $\emptyset$ 的元素删掉。

[2]子问题 2 在第 53 页

3. 根据 v_i 的极大化性质，当 v_i 分裂后，只有 v_{i-1} 和 v_{i+1} 是有可能改变的。那么也把这两个元素提出来。

4. 现在，最多有 5 个元素未满足极大化性质，根据 s_i 递增的顺序依次用子问题 2 判断相邻两个元素是否能合并。于是就可以重构这些元素使得其满足极大化性质了。

于是，就可以动态维护 $\mathcal{G}$ 了。接下来通过一个例子更好地理解 $\mathcal{G}$ 的维护：

例 10.

设串 a = cbabbabbbacb，串 b = abbacb，初始化后 $\mathcal{G}$ 为 [cb, abba, bb, bacb]。现在要把 a 串中的第 4 位修改成 c，做法是：

1. 找到覆盖这一位的 v_2 = abba；

2. 把 v_2 分裂：$v_{2'}$ = a, $v_{2''}$ = b, $v_{2'''}$ = ba；

3. 修改 $v_{2''}$ = c。

4. $v_{2'}$ 和 $v_{2''}$ 是可以合并的，将其合并。

于是此时的 $\mathcal{G}$ 为 [cb, ac, ba, bb, bacb]，满足定义中的性质，完成了重构。

6.3.1 时间复杂度分析

在本小节中，我对上述做法的时间复杂度做一些分析：

在子问题 2 中，需要进行时间复杂度为 $O(m \log m)$ 的预处理，然后使得询问单次时间复杂度为 $O(\log m)$。

在初始化阶段，会进行 n 次子问题 2 的判断，时间复杂度为 $O(n \log m)$

设修改次数为 q。那么每进行一次动态维护 $\mathcal{G}$，需要进行常数次的子问题 2 的判断，时间复杂度为 $O(q \log m)$。

此时，发现需要一个能支持如下操作的数据结构：插入、删除、查找前驱、后继和下标不小于 k 的第一个值的位置。如果使用二叉搜索树来维护 $\mathcal{G}$ 的话，就可以做到单次操作的时间复杂度为 $\log n$。由于每次修改，二叉搜索树的操作数是常数级别的，于是就可以得到动态维护 $\mathcal{G}$ 的时间复杂度是 $O(q \log n + q \log m)$ 的。

在预处理中，二叉搜索树建树是时间复杂度是 $O(n)$ 的，由于这个复杂度并非是瓶颈，可以忽略不记。

于是综合上述分析，可以在 $O((n+m+q)\log m+q\log n)$ 的时间复杂度内维护 $\mathcal{G}$。

同时，可以直接记录 s_i 作为下标，于是就可以运用线段树代替二叉搜索树，虽然时间复杂度不变，但常数可以减小。

更优地，可以使用 *van Emde Boas tree* 来代替二叉搜索树。由于在 *vEB Tree* 上单次操作时间复杂度为 $O(\log\log m)$，我们可以继续优化这个算法。

6.4 计算技巧

在上一小节中，已经可以高效地动态维护 $\mathcal{G}$ 了。

在本节中，作者会通过简化原问题，来介绍一种基于 *Border Tree* 和 $\mathcal{G}$ 的高效计算 $F(a,b)$ 的技巧。

6.4.1 简化问题

由于 $F(a,b)$ 函数的本质是把 $LCP(a[i:|a|],b)\ (1\le i\le|a|)$ 的最大值及最大值的个数表示成一个二元组，$\mathcal{G}$ 函数也可以简单地理解为把 a 串分裂成了 t 个小块 $v_j\ (1\le j\le t)$。对于每个 v_j，我们令 $s_j=v_jv_{j+1}v_{j+2}..v_t$，并把 $LCP\left(s_j\left[i:|s_j|\right],b\right)\ \left(1\le i\le|v_j|\right)$ 的最大值及最大值的个数也表示成一个二元组w_j。于是我们就可以通过合并部分的最大值及最大值的个数 w_j，来得到整体的最大值及最大值的个数 $F(a,b)$。

而且根据 v_j 的极大化性质，可以直观地发现当 $s_j=v_jv_{j+1}v_{j+2}$ 时，w_j 的值并不会变。于是 w_j 只和 v_j,v_{j+1},v_{j+2} 有关。在上一节中，已经可以用 $O(\log n)$ 的时间复杂度找到 v_j 的前驱和后继，于是，每当需要计算 w_j 时，只需要耗费 $O(\log n)$ 的时间就能找到 v_{j+1} 和 v_{j+2}。

小结一下，我们把计算 $F(a,b)$ 的问题简化成了计算 w_j。而对于计算 w_j，我们又可以把问题转化为：

给出 b 串的 3 个子串 s_1,s_2,s_3，令 $s=s_1s_2s_3$，把 $LCP(s[i:|s|],b)\ (1\le i\le|s_1|)$ 的最大值及最大值的个数合并。

6.4.2 新问题

设 s_1 串长度为 d，$s'' = s_2 s_3$。

现在我们来求 $LCP(s[i:|s|], b)$ $(1 \le i \le d)$ 的最大值及最大值的个数。令最大值为 Max，最大值的个数为 Num。

对于 $1 \le i \le d$，不妨令 $j = i + LCP(s[i:|s|], b) - 1$，则 $s[i:j]$ 便是 b 串的一个前缀。

特殊地，如果 s_1 是一个字符且这个字符没有在 b 串中出现，那么 $Max = 0$，$Num = 1$。

否则，可以分两种情况分别计算答案：当 $j < d$ 时和当 $j \ge d$ 时。

6.4.3 情况一 ($j < d$)

对于这种情况 $j < d$，可以把问题转化为询问：

$$\max_{i=1}^{d} LCP(s_1[i:d], b)$$

由于 s_1 是 b 串的一个子串，那么我们可以在串 b 中找到一个区间 $[L, R]$，使得 $s_1 = b[L:R]$。又因为有 $j < d$ 限制的存在，我们把串 $s_1[i:d]$ $(1 \le i \le d)$ 加长成为 $b[L+i-1:|b|]$，然后计算答案，答案是不变的。

于是，这个问题就转化成了子问题 3[3]，根据子问题 3，就可以得到 Max 和 Num 了。

但是，问题来了：该如何控制 $j < d$ 呢？

有一个简单的想法：在计算子问题 3 时，保证在 b 串中，后缀匹配的最远位置是严格小于 R 的。

考虑 $b[1:R]$ 的所有 $Border$，要求一个最大的 x，使得 $b[1:x]$ 是 $b[1:R]$ 的一个 $Border$，且满足 $0 \le x < R - L + 1$。

于是，只要对区间 $[L, R-x]$ 做子问题 3，就可以保证在 b 串中，后缀匹配的最远位置是严格小于 R 的。而对于区间 $[R-x+1, R]$，根据 $j < d$，在做子问题 3 时，答案显然是小于 x 的，所以可以忽略这种情况。

对于最大化 x，可以先建出 KMP 自动机，然后从 R 这个点开始向根用倍增算法寻找 x。关于 KMP 自动机，本文已在第二节中介绍，具体形态见图 ref1。

[3]子问题 3 在第 53 页

显然地，也可以建出 *Border Tree* 来计算答案。由于 *Border Tree* 的深度是 $O(\log m)$ 的，所以直接枚举 $b[1:R]$ 所在的点到根路径上的点计算 x 也是能保证时间复杂度的。关于 *Border Tree* 的性质请参见第六节。

现在我们给出一个例子来理解情况一：

例 11.

设串 a = babaca，串 b = abcabacabc。

现在 $\mathcal{G}$ 为：[b, abaca]，其中 v_1 = b, v_2 = abaca，v_2 对应的三元组为 $(2,4,8)$。

如果现在需要计算 w_2，可以发现 $LCP(v_2,b)$ 为 2，所对应的前缀为 ab。此时属于情况一。

分析 b 串中的子串 $[4,8]$。

如果直接对于区间 $[4,8]$ 做子问题 3 的话，答案会是 3，所对应的前缀为 abc，匹配的区间为 $[8,10]$。这个是错误的。

需要先找到 x，关于 x 的定义见上文*(*在子串 $[4,8]$ 中 $x=1$*)*。现在对于区间 $[4,8-x]$ 做子问题 3，可以得到答案为 2，所对应的前缀为 ab，匹配的区间为 $[3,5]$，匹配的最远距离是小于 8 的。

于是对于计算 w_2，就可以得到 $Max = 2, Num = 1$。

6.4.4 情况二 ($j \geq d$)

由于 s_1 是 b 串的一个子串，那么我们可以在串 b 中找到一个区间 $[L,R]$，使得 $s_1 = b[L:R]$。

因为 $j \geq d$，所以 $s_1[i:d]$ 是 $s[i:j]$ 的前缀。可以显然地发现，由于 $s_1[i:d] = b[L+i-1:R]$，所以 $s_1[i:d]$ 一定是 $b[1:R]$ 的一个 *Border*。

考虑一种暴力的方法：先建出 *KMP* 自动机，然后在根到点 R 的路径上暴力枚举点 R 的 *Border*，对于一个长度合法的 *Border*，可以把这个 *Border* 与 s'' 拼接起来得到串 s'，再计算 $LCP(s',b)$。由于串 s' 是由 3 个 b 串的子串拼起来的，所以只需计算 3 次子问题 1[4] 就可以得到 *Max* 了，且 $Num = 1$。

但由于 *KMP* 自动机深度是可以达到 $O(m)$ 的，所以单次暴力枚举的时间复杂度是可以达到 $O(m)$ 的。现在，让我们来优化上述暴力做法。可以很自然地想

[4]子问题 1 在第 53 页

到，我们可以根据 *Border Tree* 深度是 $O(\log m)$ 这一性质来优化暴力。如果能在 $O(1)$ 的时间复杂度内完成对一整个 *Border Group* 的询问，那么就可以把单次暴力枚举的时间复杂度降到 $O(\log m)$ 了。

考虑如何在 $O(1)$ 的时间复杂度内完成对一整个 *Border Group* 的询问。首先来回顾一下我们暴力枚举的 *Border Group* 的表现形式，设这个 *Border Group* 为 v：

$$\left[\pi'_v\pi_v^{l_v}, \pi'_v\pi_v^{l_v+1}, \pi'_v\pi_v^{l_v+2}, ..., \pi'_v\pi_v^{x}\right]$$

具体详见 5.3 中的性质。

这个转化的本质是：在 *KMP* 自动机中，只需考虑一个 i $(1 \le i \le d)$，而现在要考虑的是一组具有相同性质的 i。

特殊地，如果这个 *Border Group* 中有 $\pi'_v\pi_v$ 这个串，那么用暴力判断，并且令 $l_v = 2$。然后，根据周期串和基的定义，这个 *Border Group* 中剩下的串都是周期串，它们周期的长度为 $|\pi_v|$。

现在对 b 串进行如下的分类讨论：

1. 当串 b 是周期串且周期的长度为 $|\pi_v|$ 时，先令

$$len = LCP\left(s'', b\left[\left|\pi'_v\right| + 1 : |b|\right]\right)$$

对于这一个 *Border Group*，可以得到匹配长度为：

$$\min\left(|b|, \min_{k=l_v}^{x}\left\{\left|\pi'_v\right| + k\,|\pi_v| + len\right\}\right)$$

可以这样理解：当 *LCP* 长度不超过 $|b|$ 时，由于串 b 和 *Border Group* 中的串都是周期串且周期的长度为 $|\pi_v|$，那么可以发现 *len* 的值是固定的，与 k 的取值无关。其中，*len* 的意义为：选择一个 *Border* 后，s'' 拼在后面能继续匹配的长度。

于是，可以先判断出最大的匹配长度是否为 $|b|$，如果是的话，$Max = |b|$，*Num* 为合法的 k 的数量。否则，*Max* 为最大匹配长度，$Num = 1$。

2. 否则，一定存在一个 q，满足：

$$\left|\pi'_v\pi^{r_v}\right| < q \le \left|\pi'_v\pi^{r_v+1}\right|, b_q \ne b_{q-|\pi_v|}$$

目前，已知 r_v，那么可以通过计算 LCP 来求出最小的 q。

由于已知 *Border Group* 中的串为周期串，直观地，可以发现 $b[1:q-1]$ 也是周期串且周期的长度为 $|\pi_v|$。

设 d_1 为这个 *Border Group* 中最长串长度：$\left|\pi'_v\pi_v^x\right|$，$d_2$ 为这个 *Border Group* 中原来最长串长度：$\left|\pi'_v\pi_v^{r_v}\right|$ (具体详见 5.3 中的性质)。再设 k 为使得 π_v^k 为串 s'' 的前缀的最大整数 (可以通过多次计算子问题 1 来得到 k)，并令

$$
\begin{aligned}
len_1 &= LCP\left(s'', b\left[|\pi_v|+1:|b|\right]\right)\\
len_2 &= LCP\left(s''\left[k|\pi_v|+1:|s''|\right], b\left[d_2+1:|b|\right]\right)\\
rem &= \left(q-1-\left|\pi'_v\right|\right) \bmod |\pi_v|
\end{aligned}
$$

现在会有 3 种情况：

(a) 当 $x+k<r_v$ 时，那么在这个 *Border Group* 中的 $\pi'_v\pi_v^x$ 后面拼接 s'' 能使得匹配的长度最长。所以 $Max=d_1+len_1$，$Num=1$。

(b) 当 $l_v+k>r_v$ 时，那么可以把 b 串简化为 $b[1:q-1]$，然后通过分类 1 来解决 (其实这时 $Max=q-1$)。

(c) 当 $l_v+k\le r_v$ 且 $x+k\ge r_v$ 时，可以发现 b 串的前 d_1 位是一定能匹配的 (即 $Max\ge d_1$)。此时还是有两种情况：

如果 $len_2>rem$，那么 $Max=d_2+len_2$，$Num=1$。

如果 $len_2\le rem$，那么又可以把 b 串简化为 $b[1:q-1]$，然后通过分类 1 来解决 (这时 $Max\le q-1$)。

现在本文给出一个例子来更好地理解分类 1。

例 12.

设串 $b=$ abababababa，子串 $s_1=$ ababab, $s_2=$ ab, $s_3=$ ba，那么 $s''=$ abba。

由于 s_1 是 b 串的一个子串，那么可以在串 b 中找到一个区间 $[L,R]$，使得 $s_1=b[L:R]$。*(在这个例子中令 $[L,R]=[3,8]$)*

如果在 *KMP* 自动机中计算答案，枚举 $b[1:8]$ 的 *Border*，可以发现 ababab, abab, ab 这三个 *Border* 的长度都是符合条件的 *(没有超过 s_1 的长度)*。那么可以在这个两个 *Border* 后接上 s'' 成为 abababababba, abababba, ababba。其中，

第一个串匹配的长度为 8，第二个串匹配的长度为 6，第三串匹配的长度为 4，那么 $Max = 8, Num = 1$。

而在 *Border Tree* 中计算答案时，可以发现 $b[1:8]$ 的所有 *Border* 是属于一个 *Border Group* 的，这个 *Border Group* 的三元组为 $(0, 2, 4)$。由于 b 串是周期串且周期的长度为 2，所以这个例子是属于分类 1 的。根据上文的定义，这个 *Border Group* 中包含了 3 个 *Border*：ababab, abab, ab。

对于 ab 这个 *Border*，据上文所述，通过暴力计算，可以得到匹配长度为 5。接下来，由上文公式，就能算出 $len = 2$。对于 ababab, abab 这两个 *Border*，就可以直接算出匹配长度：8, 6。于是，我们可以得到 $Max = 8, Num = 1$。

分类 2 中有 4 种情况要讨论，但由于篇幅有限，所以本文在这里只根据一种情况进行举例分析：

例 13.

设串 b = abababababacde，子串 s_1 = ababab, s_2 = ababac, s_3 = de，那么 s'' = ababacde。

由于 s_1 是 b 串的一个子串，那么可以在串 b 中找到一个区间 $[L, R]$，使得 $s_1 = b[L:R]$。*(在这个例子中令 $[L, R] = [3, 8]$)*

在 *Border Tree* 中计算答案时，可以发现 $b[1:8]$ 的所有 *Border* 是属于一个 *Border Group* 的，这个 *Border Group* 的三元组为 $(0, 2, 4)$。由于 b 串并不是周期串，所以这个例子是属于分类 2 的。根据上文的定义，这个 *Border Group* 中包含了 3 个 *Border*：ababab, abab, ab。

对于 ab 这个 *Border*，据上文所述，通过暴力计算，可以得到匹配长度为 4。然后根据上文定义，我们可以得到：$q = 10, l = 2, x = 3, r = 4$。接下来，由上文公式，又可以得到：$d_1 = 6, d_2 = 8, k = 2, len_1 = 5, len_2 = 4, rem = 1$。

于是，可以发现 $l+k \le r$ 且 $x+l \ge r$，是属于情况 2c。而且又满足 $len_2 > rem$，那么在这个例子中 $Max = 12, Num = 1$。*(使用 abab 这个 Border)*

6.4.5 时间复杂度分析

根据上文中对于情况一和情况二的分析，可以发现，如果运用 *Border Tree* 来实现计算的话，暴力枚举 *Border Group* 的个数是 $O(\log m)$ 级别的。对于每个

Border Group 的计算，都可以把问题分类讨论，根据具体情况进行不同的转化。从时间复杂度角度来说，计算一个 *Border Group* 的时间复杂度实际上等于进行常数次子问题 1 的运算的时间复杂度。由于子问题 1 单次是可以 $O(1)$ 询问的，所以计算一个 w_j 的时间复杂度是 $O(\log m)$。

那么联系上一节的时间复杂度分析，对于动态修改文本串 a 并对静态模式串 b 求 $F(a,b)$ 这个问题，是可以在时间复杂度为 $O((n+m+q)\log m + q\log n)$ 内解决的。

当解决这个动态文本串与静态模式串匹配的基础问题之后，就可以把这个问题推广，从而解决以这个问题为基础的一类问题了。在下节中，我们将对这一类问题进行举例讨论。

6.5 例题

现在本文给出这类问题的一个例题：

例 14 (2015 国家集训队互测 *Sone2*)**.**

给一个长度为 n 的主串 a 和一个长度为 m 的模式串 b。

有 q 个操作，操作有 4 种：

1. 修改 a 串的一位，求 $F(a,b)$。
2. 选取 a 串中的一个区间 $[L,R]$，求 $F(a[L:R],b)$。
3. 求子问题 1。
4. 求子问题 2。

数据范围：$1 \le n,m,q,$ 字符集大小 ≤ 100000

时限：$2s$

在这道题目中，除了第二种操作，其他操作的计算方法都已在上文中提及。对于第二种操作，一个简单的想法就是维护 $a[L:R]$ 这个串的覆盖 $\mathcal{G}'$，然后进行计算。由于 $a[L:R]$ 是 a 串的子串，可以发现，如果把 $\mathcal{G}'$ 序列的头尾两个元素去掉，剩下的序列就是 $\mathcal{G}$ 序列的子序列了。于是对于第二种操作，只需要对于 $\mathcal{G}'$ 的头尾两个元素进行特殊计算，而中间的部分可以运用数据结构直接从 $\mathcal{G}$

序列中提取。那么就能在单次 $O(\log n + \log m)$ 的时间复杂度内进行构造 $\mathcal{G}'$ 了。然后在维护 v'_j 的同时维护 w'_j，就可以直接对 $\mathcal{G}'$ 计算来得到答案。

于是，此题就可以在 $O((n+m+q)\log m + q\log n)$ 的时间复杂度内解决了。

此题还有一个部分分，在这类数据中 b 串是随机的。由于 b 串随机，那么就可以发现如果用 b 串建 KMP 自动机，KMP 自动机的深度就是 $O(\log m)$ 级别的。于是，在上述算法中，就不需要运用 $Border\ Tree$ 了，可以直接用 KMP 自动机代替。在这类数据中，这个解法并没有能降低时间复杂度，但大大降低了实现此题的难度。

此外，此题可以加强一下：建一棵 n 个点的树，树上每个点都有一个字符。对于树上的一条链，把链上点的字符按顺序拼接起来能得到一个字符串 a。再给一个长度为 m 的模式串 b。有 q 个操作，每个操作可以改变树上一个点的字符，或是给定树上的一条链，得到串 a，求 $F(a,b)$。

再加强的话，还可以把这棵树变成动态树。

对于解法的话，可以用 $Link\text{-}Cut\ Tree$ 来维护覆盖 $\mathcal{G}$。每个 $splay$ 内维护这一条重链构成字符串的覆盖 $\mathcal{G}'$。每次 $access$ 操作把维护重链的 $splay$ 拼接或分裂，只会造成常数个 v_j 和 w_j 重构。所以可以在 $O((n+m)\log m + q\log n\log m)$ 的时间复杂度内解决这个问题。

7 感谢

感谢中国计算机学会提供学习和交流的平台。

感谢绍兴一中的陈合力老师、董烨华老师、游光辉老师、邵红祥多年来给予的关心和指导。

感谢国家集训队教练余林韵和陈许旻的指导。

感谢清华大学的俞鼎力、董宏华、何奇正学长对我的帮助。

感谢绍兴一中的张恒捷、任之洲、贾越凯对我的帮助和启发。

感谢绍兴一中的杨嘉诚、郭雨、邵杰晶、翁天东、陶渊政、洪华敦等同学对我的帮助和启发。

感谢绍兴一中的贾越凯同学为本文审稿。

感谢其他对我有过帮助和启发的老师和同学。

感谢父母对我的关心和照顾。

8 参考文献

[1] 许智磊：《后缀数组》

[2] 朱泽园：《多串匹配算法及其启示》

[3] 杨弋：《Hash在信息学竞赛中的一类应用》

[4] 罗穗骞：《后缀数组——处理字符串的有力工具》

[5] 董华星：《浅析字母树在信息学竞赛中的应用》

[6] 陈立杰：《后缀自动机Suffix Automaton》

[7] 王悦同, 徐毅, 徐子涵：《Suffix Cactus》

[8] Amihood Amir,Gad M. Landauy,Moshe Lewensteinz,Dina Sokolx,“Dynamic Text and Static Pattern Matching”

[9] Mokhtar S. Bazaraa, John J. Jarvis, and Hanif D. Sherali, “Linear Programming and Network Flows (4th ed.)”, John Wiley & Sons Inc.

[10] 刘汝佳, 黄亮, 《算法艺术与信息学竞赛》, 清华大学出版社。

[11] 刘汝佳, 《算法竞赛入门经典》, 清华大学出版社。