字符串（一）

程思元

南京外国语学校

2023 年 5 月

字符串

- 字符串 S 是由一些字符 $s_1 s_2 \cdots s_n$ 依次组成的有序序列。称 n 为 S 的长度，记作 $|S| = n$。
- S 中的第 i 个字符记作 s_i。其中 $s_i \in \Sigma$。Σ 被称作字符集，$|\Sigma|$ 一般是常数。
- 满足 $|S| = 0$ 的串 S 称为空串，记作 $S = \varnothing$。

子串与前后缀

- S 中的第 l 个字符到第 r 个字符依次组成的串称为 S 的一个子串，记作 $S_{l,r} = s_l, \cdots, s_r$。这里要求 $1 \le l \le r \le |S|$。
- 若 $l = 1$，则称 $S_{l,r}$ 是 S 的一个前缀，记作 $Pre_r = S[1, r]$。
- 若 $r = |S|$，则称 $S_{l,r}$ 是 S 的一个后缀，记作 $Suf_l = S[l, |S|]$。
- 若串 T 既是 S 的前缀，也是 S 的后缀，则称 T 是 S 的一个 border。

确定有限状态自动机

- 一个确定有限状态自动机（DFA）由状态集合 Q，输入字符集 Σ，转移函数 $\delta : Q \times \Sigma \to Q$，起始状态 $s \in Q$，接受状态 $F \subseteq Q$ 组成。
- 一个 DFA 可以简单看作一张边带权有向图，点集是 Q，每个转移是一条边。

Aho-Corasick 自动机的概念

- Aho-Corasick 自动机（简称 AC 自动机）由两部分组成，trie 树和 fail 树。其可以用于求出一个文本串的每个前缀有哪些模式串作为后缀。
- 对于一个模式串集合 $T = \{T_1, T_2, \cdots, T_m\}$，建立其 trie 树。AC 自动机的状态集合 Q 和 trie 树相同。
- 对于每个状态 $q \in Q, q \neq s$，q 在 fail 树上指向其存在的最长后缀。容易证明 fail 树是一棵树。
- 每个点在 trie 树上的祖先是它的前缀，在 fail 树上的祖先是它的后缀。

Aho-Corasick 自动机的构建

- Trie 树的构建是容易的，于是我们只需计算出每个点在 fail 树上的父亲。
- 按照 trie 树的 bfs 序依次计算。设当前点为 x，其在 trie 树上的父亲为 y，若 x 的最长存在后缀非空，则 x 的最长存在后缀去掉最后一个字符一定存在且是 y 的后缀。于是只需在 fail 树上跳找到 y 祖先中最深的的存在与 $y \to x$ 字符相同的转移边的即可。
- 对于给定模式串集合（而不是给定 trie）T 建立的 AC 自动机，这样构建的复杂度为 $O(\sum |T_i|)$。

利用 AC 自动机进行匹配

- 给定一个文本串 S，对于 S 的每个前缀，求出哪些 T_i 是该前缀的后缀。
- 依次求出 S 的每个前缀的所有后缀中，在 AC 自动机上有对应的最长的那个对应的状态 p_i。每次添加一个字符 c_{i+1}，找到 p_i 在 fail 树的祖先中，trie 树上存在权值为 c_{i+1} 的出边的点中最深的一个 p'，然后 p_{i+1} 为 trie 树上 p' 的权值为 c_{i+1} 的儿子。
- 对于一个给定的 S，时间复杂度为 $O(|S|)$。

Trie 图

- 由于这些复杂度是均摊的，如果直接给定 T 构成的 Trie，或者给定一个 Trie 对每个节点求以它为 S 的答案，复杂度就不对了。
- 考虑类似记忆化的思想，对每个状态 p，预处理出它后面添加字符 c 时跳到的祖先 p'，这样就能够 $O(1)$ 添加字符。建 Trie 图的时间复杂度为 $O(n|\Sigma|)$，其中 n 为 Trie 大小。
- 当 $|\Sigma|$ 很大时，可以用可持久化线段树维护，每个状态继承它在 fail 树上的父亲的线段树，这样一共只要进行 n 次修改，时间复杂度为 $O(n \log |\Sigma|)$。

均摊分析

引理

当 T_i 互不相同时，*fail* 树上一个点的所有祖先包含的接受状态（和某个 T_i 对应）个数不超过 $O(\sqrt{\sum |T_i|})$。

证明.

fail 树上一个点 p 的所有祖先都是它的后缀。对于每个长度 l，T 中至多一个长度为 l 的 T_i 是 p 的祖先。于是 p 的祖先中所有接受状态对应的串长度互不相同，而它们的长度之和又不超过 $\sum |T_i|$，所以至多 $O(\sqrt{\sum |T_i|})$ 个。 □

企鹅游戏[1]

例

给定若干模式串 T_i 和若干文本串 S_i。对每个 S_i，求 $\sum_j 3^{jc_{i,j}}$，其中 $c_{i,j}$ 是 T_j 在 S_i 中出现的次数。
$|S|, |T| \leq 2 \times 10^5$，$\sum |S_i|, \sum |T_i| \leq 2 \times 10^6$，$T_i$ 互不相同。

[1] uoj772

企鹅游戏

- 令 $L = \sum |T_i| + \sum |S_i|$。

引理

当 T_i 互不相同时，每个文本串包含的本质不同模式串个数之和为 $O(L^{\frac{4}{3}})$。

证明.

分两类讨论：

- 对于长度不超过 $L^{\frac{1}{3}}$ 的模式串，fail 树上每个点的祖先中至多包含 $O(L^{\frac{1}{3}})$ 个，于是总个数不超过 $O(L^{\frac{4}{3}})$。
- 对于长度超过 $L^{\frac{1}{3}}$ 的模式串，至多只有 $O(L^{\frac{2}{3}})$ 个这样的串，而长度超过 $L^{\frac{1}{3}}$ 的文本串只有 $O(L^{\frac{2}{3}})$。所以总个数不超过 $O(L^{\frac{4}{3}})$。

企鹅游戏

- 建出 T 的 AC 自动机。对于每个不是接受状态的状态，删去这个状态，将它 fail 树上的所有儿子挂到它的父亲上。这样，新的 fail 树上的每个点都对应一个模式串。
- 对于每个 S_i，求出其每个前缀在 AC 自动机上匹配到的状态 p_i。对于所有 p_i 的祖先的并，计算出其出现次数，然后用光速幂计算答案。由之前的结论，这样时间复杂度是 $O(L^{\frac{4}{3}})$ 的。

企鹅游戏

- 我们要计算的东西形如，给定 x, c，求 fail 树上 x 到根的链上所有模式串的 $3^{i\times c}$ 之和。
- 考虑对每个 c 分开计算。那么需要计算这个 c 对应的所有 x 的链并中的所有 $3^{i\times c}$。
- 考虑这样做的复杂度。总共的复杂度是每个模式串在所有文本串的出现次数去重后的不同个数，即 $\sum_{i=1}^{|T|}|\{c_{j,i}|1\le j\le |S|\}|$。

企鹅游戏

- 我们记 c_i 表示长度为 i 的模式串的个数，那么有 $\sum_i i \times c_i \leq L$。长度为 i 的模式串的出现次数和最多为 L，最坏情况下它们的出现次数是平均分配的，那么每个串出现了 $\frac{L}{c_i}$ 次，于是每个串不同的出现次数最多有 $\Theta(\sqrt{\frac{L}{c_i}})$ 个，一共有 c_i 个串，所以总的复杂度是 $\sum_i c_i \times \Theta(\sqrt{\frac{L}{c_i}}) = \Theta(\sqrt{L} \times \sum_i \sqrt{c_i})$。

企鹅游戏

- 设 $d_i = i \times c_i$，那么限制就是 $\sum_i d_i \le L$，复杂度是 $\Theta(\sqrt{L} \times \sum_i \sqrt{\frac{d_i}{i}})$。记 $F(d_1, d_2, .., d_n) = \sum_i \sqrt{\frac{d_i}{i}}$。在最坏情况下 F 对每个 d_i 的偏导应该是相等的，也就是 d_i 正比于 $\frac{1}{i}$。
- 于是此时 $d_i = \Theta(\frac{L}{i \log L})$，$c_i = \Theta(\frac{L}{i^2 \log L})$，$F = \Theta(\sum_i \sqrt{c_i}) = \Theta(\sum_i \sqrt{\frac{L}{i^2 \log L}}) = \Theta(\sum_i \frac{\sqrt{L}}{i\sqrt{\log L}}) = \Theta(\sqrt{L \log L})$。再乘上前面的 $\Theta(\sqrt{L})$，总复杂度为 $O(L\sqrt{\log L})$。

维护文本串子串的匹配信息

- 对于一个文本串 S，我们要求出 S 的每个前缀 Pre_i 的每个后缀在 AC 自动机上的匹配信息。
- 例如每次查询 S 的一个子串 $S_{l,r}$ 中一共出现了多少次模式串。

维护文本串子串的匹配信息

- 方便起见，我们把 S 也插进 AC 自动机。
- 考虑已经求出了 Pre_{i-1} 时的答案，插入一个新的字符 S_i 造成的影响。记 g_j 为 $S_{j,i}$ 在 AC 自动机上的匹配信息。
- 对于一个 AC 自动机上的状态 x，记 $|x|$ 表示它对应串的长度。记 p_i 表示 Pre_i 匹配到的状态，那么对于 p_i 的每个祖先 x，它会对 $g_{[1,|p_i|-|x|+1]}$ 造成贡献。

维护文本串子串的匹配信息

- 对于 p_i 的每个祖先 x，它会对 $g_{[1,|p_i|-|x|+1]}$ 造成贡献。
- 发现 $|p_i| - |fail_{p_i}|$ 单调不降。
- 于是，$g_{[2,|p_i|-|fail_{p_i}|]}$ 在此之后受到的修改一定是一样的，可以一起维护。而 g_1 的值就是前缀的匹配信息，这也是容易维护的。于是我们只需考虑如何维护 $|p_i| - |fail_{p_i}| + 1$ 之后的部分。

维护文本串子串的匹配信息

- 考虑如何维护 $|p_i| - |fail_{p_i}| + 1$ 之后的部分。
- 如果 S 的两个子串相等，那它们在 AC 自动机上匹配的结果也相等。根据 fail 的定义，$S_{|p_i|-|fail_{p_i}|+1,i}$ 是 Pre_i 的一个 border，于是 g 在 $|p_i| - |fail_{p_i}| + 1$ 之后的部分就等于 $Pre_{|fail_{p_i}|}$ 的全部 g 值。
- 于是 g 可以维护。

novel[2]

例

给定 n 个字符串 $s_1, s_2, \cdots, s_n$。每个字符串都有一个权值 w_i。
记 $f(S, T)$ 为字符串 T 在字符串 S 中的出现次数。
记 $g(S) = \sum_{i=1}^{n} w_i \times f(S, s_i)$，$h(S) = \max_{1<l<r<|S|} \frac{g(S_{l,r})}{r-l+1}$。
对于所有 $1 < i < n, 1 < j < |s_i|$，输出 $h((s_i)_{1,j})$。有理数对 998244353 取模。
$n \leq 2 \times 10^5$，$\sum_i |s_i| \leq 5 \times 10^6$，$|\Sigma| = 4$。

[2] loj3396

novel

- 建立所有字符串的 AC 自动机。
- 对于每个串，使用刚才的方法维护它的子串的信息。
- 考虑 g 数组的维护方式。对于前一半，相当于维护一个结构，支持 push back、全局加正数、查询 $\min_i \frac{a_i}{x-i}$。考虑维护凸壳，全局加可以打 tag 解决。查询时发现最优的点具有单调性，可以直接 two pointers。

novel

- 对于后一半，计算对答案的贡献是容易的，可以直接和 $fail_{Pre_i}$ 的答案求 $\max$。
- 但是由于我们没有显式地维护出 g，一个位置从后一半变为前一半时，我们需要计算出它的 g 值。
- 每次我们要求出 $[i-|fail_{p_{i-1}}|+1, i-|fail_{p_i}|+1]$ 的 g 值。考虑对反串建 AC 自动机，只要我们求出 $S_{i-|fail_{p_i}|+1,i}$ 在反串 AC 自动机上的对应状态和 g 值，就可以在反串 AC 自动机上继续匹配算出 $[i-|fail_{p_{i-1}}|+1, i-|fail_{p_i}|+1]$ 中的所有 g 值。

novel

- $S_{i-|fail_{p_i}|+1,i}$ 在 AC 自动机上正好对应 $fail_{p_i}$ 这个状态，于是它的 g 值可以直接利用原串 AC 自动机得到。
- 考虑对于原串 AC 自动机的每个状态，求出它在反串 AC 自动机上匹配到的状态。原串 AC 自动机的每个状态是一个串的前缀，也就是一个反串的后缀。于是所求的状态一定是这个反串对应状态在 fail 树上的祖先，找到最深的长度不超过原状态长度的祖先即可。实现时可以对于每个串用 two pointers 预处理。
- 时间复杂度为 $O(\sum |s_i|)$。

回文自动机的概念

- 回文自动机（简称 PAM）是可以求出一个串的所有回文子串的自动机。每个状态对应一个回文子串。
- 回文自动机由转移边和 fail 树构成。经过一条转移边对一个串的影响是在前后均添加一个该字符。一个状态在 fail 树上指向它的最长回文 border。
- 回文自动机的转移边构成了两棵 trie 树。两棵 trie 树分别对应长度为奇数的回文子串和长度为偶数的回文子串。

回文自动机的构建

- 设 Pre_{i-1} 的最长回文后缀对应的状态为 x，考虑插入 S_i。
- 在 fail 树上找到 x 的祖先找到最深的 y 使得 $S_i = S_{i-|y|-1}$。那么对应 S_i 的最长回文后缀的状态就是 y 经过 S_i 的出边。
- 如果这次操作新建了节点，需要用类似的方法求出这个新建节点的 fail。
- 时间复杂度为 $O(|S|)$。且这种构建方式可以强制在线。

一道例题

例

有一个字符串的可重集 D，一开始为空。有 n 次操作，每次操作要么往 D 中加一个字符串，要么给定一个回文串 s，求 D 的所有串中，以 s 为真后缀的回文子串的个数之和。强制在线。$\sum |s| \leq 3 \times 10^5$。

一道例题

- 考虑对于一次查询，如果我们已经建出了 D 中的串的 PAM，相当于把每个状态的出现次数 c_i 在 fail 树上做两遍子树和之后，查询 s 对应状态的值。这可以通过维护 c_i 和 $\sum dep_i \times c_i$ 的子树和解决，其中 dep_i 是 i 在 fail 树上的深度。
- 考虑如何进行修改。我们发现，如果往 PAM 里插入多个串，这些串是独立的。换句话说，每个串的所有回文子串在两棵 fail 树上分别构成包含根的连通块。于是可以每次暴力重新 pushup 有影响的部分。复杂度为 $O(\sum |s|)$。
- 如果不用这个性质（例如把这题的 PAM 改成 ACAM 或 SAM 使其没有这个性质），也可以使用 lct 维护 fail 树（ACAM 不行）或二进制分组在多一个 $\log$ 的时间复杂度内解决。

双端插入

- 现在我们不仅要支持从后面添加字符，还要支持从前面添加字符。
- 考虑分别维护当前串的最长回文前缀和最长回文后缀，并使用原先的插入算法。
- 唯一会造成问题的是，从后/前端插入字符后，最长回文前/后缀长度发生了变化。发现这只会在整个串是回文串时发生，此时将最长回文前缀和最长回文后缀都设为整个串即可。

PAM 图

- 类似 trie 图，我们也可以定义 PAM 图使得这些操作的时间复杂度不用均摊。
- 和 trie 图一样，我们使用类似记忆化的方法。对每个状态 p，预处理出它后面添加字符 c 时跳到的祖先 p'，这样就能够 $O(1)$ 添加字符。建 PAM 图的时间复杂度为 $O(|S||\Sigma|)$。

Palindromic Tree[3]

例

给定一棵 n 个点的树，每条边上有一个字符。q 次查询两个点之间最短路上的所有边拼起来形成的字符串的本质不同回文子串个数。

$n, q \leq 10^5, |\Sigma| = 2$。

[3] nowcoder33189F

Palindromic Tree

- 首先，本质不同回文子串个数等于 PAM 节点数。
- 对原树进行树分块。对于每一对界点，预处理出它们的最短路对应的 PAM。先不考虑怎么预处理。
- 对于一次查询，如果两个端点在同一个块中，它们的距离一定不超过 $\sqrt{n}$，直接暴力建 PAM。否则找到路径上的第一个和最后一个界点，在预处理出的 PAM 前后插入一些字符即可。

Palindromic Tree

- 考虑如何预处理。枚举一个端点，以这个端点为根进行 dfs，就可以求出这个端点到每个点的路径对应的 PAM。
- dfs 弹栈时，需要支持撤销操作，于是需要使用 PAM 图来保证单次复杂度为 $O(|\Sigma|)$。
- 时间复杂度为 $O((n+q)\sqrt{n}|\Sigma|)$。离线处理可以做到 $O((n+q)|\Sigma|)$ 空间。

endpos 集合

- 对于一个串 S 和 S 的一个子串 T，记 $endpos(T) = \{r | S_{l,r} = T\}$。
- 类似地，记 $beginpos(T) = \{l | S_{l,r} = T\}$。

上下文

- 定义 T 的上文是最长的 T'，使得 $endpos(T) = endpos(T')$。
- 定义 T 的下文是最长的 T'，使得 $beginpos(T) = beginpos(T')$。
- 定义 T 的上下文是 T 的上文的下文。可以证明 T 的上文的下文等于 T 的下文的上文。

后缀自动机的概念

- 后缀自动机（简称 SAM）是可以接受一个串的所有后缀的自动机。
- 后缀自动机由转移边和 fail 树构成。每个状态对应一组 endpos 相同的子串。经过一条转移边对一个串的影响是在其末尾加一个该字符。一个状态在 fail 树上指向 endpos 异于它的后缀中最长的一个。容易证明 fail 树是一棵树。
- 每个状态 x 维护一个 len_x，表示其中最长的一个串的长度。

后缀自动机的构建

- 对于一个字符串 S，我们要构建出它的后缀自动机。
- 考虑当前已经构建出 Pre_{i-1} 的后缀自动机，Pre_{i-1} 对应的状态为 p。
- 新建节点 np 表示 Pre_i 的状态。
- 找到 p 在 fail 树上的祖先中存在权值为 S_i 的出边的状态中最深的一个 p'，将 p 以上 p' 以下的链上所有状态的权值为 S_i 的出边指向 np。若 p' 不存在，则直接令 p 的所有祖先的权值为 S_i 的出边指向 np。
- 记 q 为 p' 权值为 S_i 的出边指向的状态。将 q 以 $len_p + 1$ 为界 split 成两个。
- 时间复杂度为 $O(|S|)$，空间复杂度为 $O(|S||\Sigma|)$。

后缀树的概念

- 对于一个字符串 S，将其所有后缀插入一个 trie，得到的 trie 称为后缀字典树。
- 将后缀字典树进行压缩（收缩出度为 1 的非接受状态），得到的树称为后缀树。
- S 的反串 SAM 的 fail 树即为 S 的后缀树。这也是后缀树比较方便的构建方式。

压缩后缀自动机

- 我们知道，后缀树是后缀字典树压缩的结果。而后缀自动机是后缀字典树最小化的结果。
- 如果对后缀字典树先后进行压缩和最小化，得到的结果称为压缩后缀自动机。

压缩后缀自动机

- 后缀自动机上的每个状态所代表的最长串，均满足上文是自身。
- 由于压缩后缀自动机上的每个状态出度都超过 1（或是原串的后缀），均满足下文是自身。
- 于是，一个点存在于压缩后缀自动机上，当且仅当其上下文是自身。

对称压缩后缀自动机

- 于是，一个串 S 的压缩后缀自动机和其反串的压缩后缀自动机的状态集合相同，均是上下文是自身的串。
- 我们建立一个自动机，将正串压缩后缀自动机和反串压缩后缀自动机的转移边均插入自动机，得到的自动机称为对称压缩后缀自动机。
- 基本字串结构是对称压缩自动机结构的进一步刻画。

Sasha And Swag Strings[4]

例

给定一个字符串 S。求 S 的后缀树的每条边上字符串的本质不同的子串的个数之和。
$|S| \leq 10^5$。

[4] Petrozavodsk Summer-2015. Moscow IPT Contest H

Sasha And Swag Strings

- 建出压缩后缀自动机。由于压缩后缀自动机是后缀树最小化后的结果，只需求出压缩后缀自动机的每条边上字符串的本质不同的子串的个数，再乘上从起始状态到这条边起点的路径条数即可。

Sasha And Swag Strings

- 后缀自动机的一个状态上的所有串之间一定是后缀关系，所以压缩后缀自动机上每个状态的所有入边上的字符串也是后缀关系。
- 找出每个状态最长的入边，求出它的每个后缀的本质不同子串个数，就求出了该状态所有入边的本质不同子串个数。
- 时间复杂度为 $O(|S|)$。

谢谢大家！