Trivial String Techniques

Falsyta

2018 年 12 月 18 日

Deep Purple

给定一个长度为 n 的字符串 S，q 次询问，每次询问 $S[l:r]$ 的 $LBorder$（即最大的 i 使得 $S[l:l+i-1]=S[r-i+1:r]$）。
$n,q\leqslant 2\times 10^5$

Sol

考虑后缀树，求最小的 i 使得 $l < i \leqslant r \leqslant i + LCP(l, i) - 1$。
扫描线，扫到 l 的时候把 l 加进去，到 i 的时候暴力找符合条件的 l 并删掉即可。
用树剖维护，时间复杂度 $O(n \log^2 n)$。

Quasi Template

定义一个串 s 能匹配 S 当且仅当 s 能可超出头尾的覆盖 S。
给定 S，问不同的 s 的个数和字典序最小的 s。
$|S| \leqslant 2 \times 10^5$

Sol

考虑后缀树，唯一问题是匹配开头，匹配其他的可以用平衡树启发式合并。
考虑使用 KMP。
考虑每个后缀树节点到其父亲节点的边上的每一个位置到根的路径组成的字符串，令我们当前考虑的边中的可行字符串是 $S[p:l], S[p:l+1], \ldots S[p:r]$，其中 p 是当前后缀树节点的第一次出现位置。
那么该边的贡献为 $\sum_{i=l}^{r}[next_i \geq p-1]$。
该条件的必要性显然，充分性的话，
因为若 $i - next_i + 1 <= p$，则显然 p 不是该后缀节点的第一次出现位置，不可能。
时间复杂度 $O(n\log^2 n)$。

一道简单的字符串题

有一个字符串 s，每次给出 $[l, r]$，询问 $s[l : r]$ 的后缀树的节点个数。

Sol

后缀树上的点可以分为 (1) 在原树中的节点且在 $[l, r]$ 的后缀树中还有两个儿子存在，(2) 后缀节点。

考虑枚举左端点并用线段树维护右端点的答案，维护一个向左加字符的后缀树，我们要知道的是每个节点在右端点大于等于多少的时候有两个儿子存在，记为 t_u ，在后缀树上的每个节点记录 $last_u$ 表示子树中最新加入的节点是谁，每次新加入一个点 x 的时候，暴力扫描根到 x 的每一段 last ，每一段 last 中只有深度最大的一个点 t_u 会改变，暴力改一下就好。所以算第一类点总复杂度是 $O(n \log^2 n)$ 的。

第二类点很容易统计，唯一要处理的就是既在 (1) 里也在 (2) 里的，注意到这种后缀节点是可以二分的（一个后缀在第一类中则比它短的也在第一类中），二分即可。

CTT18 close

例 1 (北大集训 2018 Day3 close) 维护一个字符串 s，支持 push-front/push-back，在每次操作后，输出 s 中本质不同的子串个数。

Sol

想要同时进行 push-front 和 push-back ，我们不妨想一想如何把构建后缀自动机的方法和构建后缀树的方法拼在一起。
Ukkonen's Algorithm 告诉我们，在只有向后加字符的时候，一个后缀树节点一旦成为了叶节点，就永远会是叶节点。
我们考虑什么样的后缀所对应的节点是叶节点，就是在 s 中出现了至少两次的后缀。由此我们还可以知道 Ukkonen's Algorithm 能告诉我们的第二个性质：假设 s[p:] 是最长的出现至少两次的后缀，那么所有 s[k:](k>p) 对应的节点都是非叶节点。
容易看出在统计本质不同的子串的过程中，我们只需要维护出所有叶节点并算出它们的虚树边长和即可，而叶节点在向后加字符的时候大小相对关系是不会改变的，因此我们可以用一个平衡树来维护它们。

Sol

向前加字符是不会改变后缀之间的相对顺序的，只会增加一个新的后缀，增加一个新的后缀意味着后缀树上至多一个叶节点变成了非叶节点。
在向前和向后加字符的时候我们只需要知道最长的出现两次的后缀 s[p:] 即可，我们在平衡树中维护所有 s[k:](k<p) ，并维护在平衡树中所有相邻后缀的 LCP 之和，就可以算出我们所需要的答案。
注意我们的平衡树中维护的并不是所有后缀，因此不能用重量平衡树来优化，比较用二分哈希复杂度就是 $O(n \log^2 n)$ 。