后缀相关

hyj

雅礼中学

August 24, 2018

后缀

对于一个字符串，从最后一个字母之前的一个字母开始，一直到最后一个字母所构成的字符串就叫做后缀

后缀

对于一个字符串，从最后一个字母之前的一个字母开始，一直到最后一个字母所构成的字符串就叫做后缀

例如：banana

后缀

对于一个字符串，从最后一个字母之前的一个字母开始，一直到最后一个字母所构成的字符串就叫做后缀

例如：banana

它的后缀有：a,na,ana,nana,anana,banana

后缀数组

SA，后缀数组。就是一个数组而已

后缀数组

SA，后缀数组。就是一个数组而已

$SA[i]$ 代表在所有后缀排好序后第 i 个后缀起始的下标位置

后缀数组

SA，后缀数组。就是一个数组而已

$SA[i]$ 代表在所有后缀排好序后第 i 个后缀起始的下标位置

先只考虑如何得到 SA 数组

后缀数组

SA，后缀数组。就是一个数组而已

$SA[i]$ 代表在所有后缀排好序后第 i 个后缀起始的下标位置

先只考虑如何得到 SA 数组

$O(n^2 logn)$?

后缀数组

SA，后缀数组。就是一个数组而已

$SA[i]$ 代表在所有后缀排好序后第 i 个后缀起始的下标位置

先只考虑如何得到 SA 数组

$O(n^2logn)$?

$O(nlogn)$ 的倍增算法

对于所有后缀，比较它们的第一个字母，进行初步排序

对于所有后缀，比较它们的第一个字母，进行初步排序

拿出两个初步排序后排名相同的后缀 k 和 k'，现在要把它们的大小关系得出来

对于所有后缀，比较它们的第一个字母，进行初步排序

拿出两个初步排序后排名相同的后缀 k 和 k'，现在要把它们的大小关系得出来

由于它们的前一个字母已经相同，那么比较前二个字母

对于所有后缀，比较它们的第一个字母，进行初步排序

拿出两个初步排序后排名相同的后缀 k 和 k'，现在要把它们的大小关系得出来

由于它们的前一个字母已经相同，那么比较前二个字母

如果前两个字母也相同，那么比较前四个字母

对于所有后缀，比较它们的第一个字母，进行初步排序

拿出两个初步排序后排名相同的后缀 k 和 k'，现在要把它们的大小关系得出来

由于它们的前一个字母已经相同，那么比较前二个字母

如果前两个字母也相同，那么比较前四个字母

.....这就是所谓的倍增

对于所有后缀，比较它们的第一个字母，进行初步排序

拿出两个初步排序后排名相同的后缀 k 和 k'，现在要把它们的大小关系得出来

由于它们的前一个字母已经相同，那么比较前二个字母

如果前两个字母也相同，那么比较前四个字母

.....这就是所谓的倍增

但每次比较不一样要 $O(n)$ 扫一遍?

其实并不需要

其实并不需要

考虑后缀之间的联系，后缀 k 的第三个字母到最后一个字母实际上就是后缀 $k+2$

其实并不需要

考虑后缀之间的联系，后缀 k 的第三个字母到最后一个字母实际上就是后缀 $k+2$

那么在倍增长度进行比较的时候，前一半的比较完了，而后一半其实也是比较完了的

其实并不需要

考虑后缀之间的联系，后缀 k 的第三个字母到最后一个字母实际上就是后缀 $k+2$

那么在倍增长度进行比较的时候，前一半的比较完了，而后一半其实也是比较完了的

把前一半的排名看做一个数字，后一半的排名也看做一个数字。所有数字位数相同

其实并不需要

考虑后缀之间的联系，后缀 k 的第三个字母到最后一个字母实际上就是后缀 $k+2$

那么在倍增长度进行比较的时候，前一半的比较完了，而后一半其实也是比较完了的

把前一半的排名看做一个数字，后一半的排名也看做一个数字。所有数字位数相同

基数排序

得到了 SA，然并卵

得到了 SA，然并卵

有用的是接下来的 $height$ 数组

得到了 SA，然并卵

有用的是接下来的 $height$ 数组

$height[i]$ 数组代表 $SA[i-1]$ 和 $SA[i]$ 的LCP（最长公共前缀）长度

得到了 SA，然并卵

有用的是接下来的 $height$ 数组

$height[i]$ 数组代表 $SA[i-1]$ 和 $SA[i]$ 的LCP（最长公共前缀）长度

用处体现在一个结论上：

得到了 SA，然并卵

有用的是接下来的 $height$ 数组

$height[i]$ 数组代表 $SA[i-1]$ 和 $SA[i]$ 的LCP（最长公共前缀）长度

用处体现在一个结论上：

对于两个后缀 j 和 k，$rk[j] < rk[k]$，那么后缀 j 和 k 的LCP长度为：$\min_{i=rk[j]+1}^{rk[k]} height[i]$

其中 $rk[i]$ 代表后缀 i 在后缀排序后的排名

那怎么求这样的数组？暴力是 $O(n^2)$ 的

那怎么求这样的数组？暴力是 $O(n^2)$ 的

再来一个结论：$height[rk[i]] \geq height[rk[i-1]] - 1$

那怎么求这样的数组？暴力是 $O(n^2)$ 的

再来一个结论：$height[rk[i]] \geq height[rk[i-1]] - 1$

考虑证明：

那怎么求这样的数组？暴力是 $O(n^2)$ 的

再来一个结论：$height[rk[i]] \geq height[rk[i-1]] - 1$

考虑证明：

设排在后缀 $i-1$ 前一个的是后缀 k 。后缀 k 和后缀 $i-1$ 分别得到后缀 $k+1$ 和后缀 i，因此后缀 $k+1$ 一定排在后缀 i 前面（这是因为后缀 k 的排名比后缀 $i-1$ 要高），并且最长公共前缀长度为 $height[rank[i-1]]-1$（考虑后缀 k 与后缀 $k-1$ 的 LCP ）

那怎么求这样的数组？暴力是 $O(n^2)$ 的

再来一个结论：$height[rk[i]] \geq height[rk[i-1]] - 1$

考虑证明：

设排在后缀 $i-1$ 前一个的是后缀 k 。后缀 k 和后缀 $i-1$ 分别得到后缀 $k+1$ 和后缀 i，因此后缀 $k+1$ 一定排在后缀 i 前面（这是因为后缀 k 的排名比后缀 $i-1$ 要高），并且最长公共前缀长度为 $height[rank[i-1]]-1$（考虑后缀 k 与后缀 $k-1$ 的 LCP ）

所以 $O(n^2)$ 的第二个枚举完全不需要，于是优化成 $O(n)$

SPOJ 705

给你一个长N的字符串,问你该字符串中出现了多少个不同的子串?

多组数据 $T <= 20$

字符串长度 $n <= 50000$

水题一道

水题一道

每个后缀的前缀都是原串的一个子串

水题一道

每个后缀的前缀都是原串的一个子串

后缀排序后

水题一道

每个后缀的前缀都是原串的一个子串

后缀排序后

$ans = \sum_{i=1}^{n} n - SA[i] + 1 - height[i]$

NOI2015 品酒大会

给出一个长度为 n 的字符串，每一位有一个权值 val 。定义两个位字符为 r 相似，是指分别从这两个字符开始，到后面的 r 个字符都相等。两个 r 相似的字符还有一个权值为这两个字符权值的乘积。问对于 $r = 0, 1, 2, \ldots, n-1$ ，统计出有多少种方法可以选出 2 个“ r 相似 " 的字符，并回答选择 2 个“ r 相似 " 的字符可以得到的权值的最大值。

$n = 300,000$

对于两个位置，它们的最大相似度就是以它们开头的后缀的LCP长度

对于两个位置，它们的最大相似度就是以它们开头的后缀的LCP长度

那么把SA和height求出来，并对于每一个height求一段区间，使得这段区间中当前height值最小，以致在这段区间中，当前位置的左边选一个位置，当前位置的右边选一个位置，这两个位置的相似度必定取决于当前的height

对于两个位置，它们的最大相似度就是以它们开头的后缀的LCP长度

那么把SA和height求出来，并对于每一个height求一段区间，使得这段区间中当前height值最小，以致在这段区间中，当前位置的左边选一个位置，当前位置的右边选一个位置，这两个位置的相似度必定取决于当前的height

于是就可以累加答案，过程中注意对于height值相等的怎么处理，把排在前面的视为更大或更小都可以，但是不能视为完全相同而置于同一段，会算重

那么第二问，既然在算第一问的时候方案数的累计是左边一段的一个位置加上右边一段的一个位置，那么就只要在左边一段选个最大的，乘上右边一段的最大的就好了，当然，负数的处理就是左边最小的乘上右边最小的，两个值取更大。

那么第二问，既然在算第一问的时候方案数的累计是左边一段的一个位置加上右边一段的一个位置，那么就只要在左边一段选个最大的，乘上右边一段的最大的就好了，当然，负数的处理就是左边最小的乘上右边最小的，两个值取更大。

这个最大最小值用ST表预处理

UESTC 1551

给出一个字符串，要求选出子串A，然后自己搞一个长度小于A的子串B，把B接在A的后面，构成一个新的字符串T，要求T为回文串，求能构成的不同回文串的个数。

字符串长 $1 \leq |S| \leq 1e5$

考虑最后回文串长度的奇偶性

考虑最后回文串长度的奇偶性

- 奇数——发现 $abcbc|ba$ 和 $abcb|cba$ 是等价的，所以对于任意一个构造出的长度为 $2n+1$ 的回文串，一定存在一种构造方法，前 $n+1$ 是原串中的子串，即A，后 n 是B，第 $n+1$ 项是对称中心。那么这样的答案就是不同的子串个数

考虑最后回文串长度的奇偶性

- 奇数——发现 $abcbc|ba$ 和 $abcb|cba$ 是等价的，所以对于任意一个构造出的长度为 $2n+1$ 的回文串，一定存在一种构造方法，前 $n+1$ 是原串中的子串，即A，后 n 是B，第 $n+1$ 项是对称中心。那么这样的答案就是不同的子串个数

- 偶数——只存在这一种情况，$abcc|ba$，即对称中心是长度为 $2n$ 的回文串中的 n 和 $n+1$ 项，所有偶数长度的回文串均可以用这种方法构造，前 $n+1$ 是A，后 $n-1$ 是B。那么就是要求原串中有多少种不同的子串，它们的末尾是两个相同的字符，且长度大于 2 。将原串reverse后，对于每种字符，找到每两个这种字符连续出现的后面一个位置，要找可以从这些位置中任意一个位置开头，但必须是其中一个位置开头的不同子串的个数。类似于求不同子串个数

一道NOI冲刺题

给出一个长度为 n 的字符串 s，提出 q 个询问，对于每个询问要求回答：右端点在区间 $[l,r]$ 的所有前缀，最长公共后缀最长的一对前缀的最长公共后缀的长度是多少。

$n,q \leq 1e5$

考虑怎么快速求一堆前缀的最长后缀

考虑怎么快速求一堆前缀的最长后缀

将原串reverse，变成求一堆后缀的最长前缀

考虑怎么快速求一堆前缀的最长后缀

将原串reverse，变成求一堆后缀的最长前缀

将这些后缀按 SA 的顺序构出一个后缀trie树。相邻两个后缀的LCA的深度就是它们最长公共前缀的长度

考虑怎么快速求一堆前缀的最长后缀

将原串reverse，变成求一堆后缀的最长前缀

将这些后缀按 SA 的顺序构出一个后缀trie树。相邻两个后缀的LCA的深度就是它们最长公共前缀的长度

一个新后缀的加入变成了加一条链，一个后缀的剔除变成删一条链

考虑怎么快速求一堆前缀的最长后缀

将原串reverse，变成求一堆后缀的最长前缀

将这些后缀按 SA 的顺序构出一个后缀trie树。相邻两个后缀的LCA的深度就是它们最长公共前缀的长度

一个新后缀的加入变成了加一条链，一个后缀的剔除变成删一条链

对于加链，找到trie树中要加入的后缀 x 的前驱 p 和后继 l，把它们拆开，把当前后继插入，并合并 l 与 x，x 与 l；删链则正好相反

考虑怎么快速求一堆前缀的最长后缀

将原串reverse，变成求一堆后缀的最长前缀

将这些后缀按 SA 的顺序构出一个后缀trie树。相邻两个后缀的LCA的深度就是它们最长公共前缀的长度

一个新后缀的加入变成了加一条链，一个后缀的剔除变成删一条链

对于加链，找到trie树中要加入的后缀 x 的前驱 p 和后继 l，把它们拆开，把当前后继插入，并合并 l 与 x，x 与 l；删链则正好相反

所谓合并，就是把相邻后缀在trie中的LCA的深度算出来

正常情况下，要算LCA的深度，需要把LCA那个点找出来

正常情况下，要算LCA的深度，需要把LCA那个点找出来

但因为我们有 $height$ ，所以只需要维护后缀们的相对顺序，就可以求它们的LCP（LCP的长度就是trie中LCA的深度）

正常情况下，要算LCA的深度，需要把LCA那个点找出来

但因为我们有 $height$ ，所以只需要维护后缀们的相对顺序，就可以求它们的LCP（LCP的长度就是trie中LCA的深度）

将 $height$ 用ST表预处理，于是查询LCP长度变成了 $O(logn)$

那么加链和删链的操作用set维护，面向set的插入和删除 $O(logn)$ ，拆开后缀和合并后缀 $O(1)$

正常情况下，要算LCA的深度，需要把LCA那个点找出来

但因为我们有 $height$ ，所以只需要维护后缀们的相对顺序，就可以求它们的LCP（LCP的长度就是trie中LCA的深度）

将 $height$ 用ST表预处理，于是查询LCP长度变成了 $O(logn)$

那么加链和删链的操作用set维护，面向set的插入和删除 $O(logn)$ ，拆开后缀和合并后缀 $O(1)$

不强制在线区间查询问题——莫队，复杂度 $O(n\sqrt{n})$

正常情况下，要算LCA的深度，需要把LCA那个点找出来

但因为我们有 $height$ ，所以只需要维护后缀们的相对顺序，就可以求它们的LCP（LCP的长度就是trie中LCA的深度）

将 $height$ 用ST表预处理，于是查询LCP长度变成了 $O(logn)$

那么加链和删链的操作用set维护，面向set的插入和删除 $O(logn)$ ，拆开后缀和合并后缀 $O(1)$

不强制在线区间查询问题——莫队，复杂度 $O(n\sqrt{n})$

总复杂度 $O(n\sqrt{n}logn)$

恭喜你

恭喜你

由于莫队常数太大，你还是无法通过此题，除非你“真的很wys”

恭喜你

由于莫队常数太大，你还是无法通过此题，除非你“真的很wys”

真正的正解复杂度可以做到 $O(n\sqrt{n})$

恭喜你

由于莫队常数太大，你还是无法通过此题，除非你“真的很wys”

真正的正解复杂度可以做到 $O(n\sqrt{n})$

可惜我不会

恭喜你

由于莫队常数太大，你还是无法通过此题，除非你“真的很wys”

真正的正解复杂度可以做到 $O(n\sqrt{n})$

可惜我不会

算法是：后缀数组+链表+回滚莫队

恭喜你

由于莫队常数太大，你还是无法通过此题，除非你“真的很wys”

真正的正解复杂度可以做到 $O(n\sqrt{n})$

可惜我不会

算法是: 后缀数组+链表+回滚莫队

链接：https://blog.csdn.net/qq_33330876/article/details/73611464

后缀树

https://www.cnblogs.com/gaochundong/p/suffix_tree.html

后缀平衡树

动态维护 SA，支持在字符串前端插入一个字符，询问后缀的大小关系

后缀平衡树

动态维护 SA，支持在字符串前端插入一个字符，询问后缀的大小关系

用平衡树去维护 SA，如果我们可以快速比较两个后缀的大小，那么直接套用平衡树的插入算法就可以了

后缀平衡树

动态维护 SA，支持在字符串前端插入一个字符，询问后缀的大小关系

用平衡树去维护 SA，如果我们可以快速比较两个后缀的大小，那么直接套用平衡树的插入算法就可以了

两种方法+一个升级版

Method 1

最普通的比较两个字符串的大小——二分哈希，$O(nlogn)$

Method 1

最普通的比较两个字符串的大小——二分哈希，$O(nlogn)$

然后二分新后缀在原来的 SA 中的位置，$O(logn)$

Method 1

最普通的比较两个字符串的大小——二分哈希，$O(nlogn)$

然后二分新后缀在原来的 SA 中的位置，$O(logn)$

总复杂度 $O(nlog^2n)$

Method 2

高级的方法

Method 2

高级的方法

先看一个引例：

Method 2

高级的方法

先看一个引例：

我们现在有一个节点序列，要支持如下两种操作：

[1] 在节点 x 的后面插入一个新节点 y

[2] 询问节点 a, b 的前后关系

Method 2

高级的方法

先看一个引例：

我们现在有一个节点序列，要支持如下两种操作：

[1] 在节点 x 的后面插入一个新节点 y

[2] 询问节点 a, b 的前后关系

一般方法是用简单的平衡树来维护这个序列，插入复杂度 $O(logn)$，询问复杂度 $O(logn)$

好一点的方法。仍用平衡树来维护这个序列，不同的是我们对每个点维护一个标记 tag_i，询问 a,b 的前后关系时只需要比较 tag_a，tag_b 的大小就行了

好一点的方法。仍用平衡树来维护这个序列，不同的是我们对每个点维护一个标记 tag_i，询问 a, b 的前后关系时只需要比较 tag_a，tag_b 的大小就行了

我们让每个点对应一个实数区间，让根对应实数区间 $(0, 1)$。对于对应实数区间 (l, r) 的节点 i，它的 tag_i 是 $\frac{l+r}{2}$，它的左子树的实数区间 (l, tag_i)，右子树是 (tag_i, r)

好一点的方法。仍用平衡树来维护这个序列，不同的是我们对每个点维护一个标记 tag_i，询问 a, b 的前后关系时只需要比较 tag_a，tag_b 的大小就行了

我们让每个点对应一个实数区间，让根对应实数区间 $(0, 1)$。对于对应实数区间 (l, r) 的节点 i，它的 tag_i 是 $\frac{l+r}{2}$，它的左子树的实数区间 (l, tag_i)，右子树是 (tag_i, r)

实际上，这样一个平衡树所有位置的 tag_i 是固定的，一个点如果在某个位置存在，那么它的 tag_i 一定是某一个值

那么我们在比较的时候只需要比较 tag_i 就行了

那么我们在比较的时候只需要比较 tag_i 就行了

只要树是平衡的，也就是最深深度不大，精度是有保证的。使用double就行了

那么我们在比较的时候只需要比较 tag_i 就行了

只要树是平衡的，也就是最深深度不大，精度是有保证的。使用double就行了

插入的时候，可能会对一整颗子树进行重构，以便算它们的 tag ，只要使用重量平衡树，就能做到 $O(logn)$ 的复杂度

那么我们在比较的时候只需要比较 tag_i 就行了

只要树是平衡的，也就是最深深度不大，精度是有保证的。使用double就行了

插入的时候，可能会对一整颗子树进行重构，以便算它们的 tag ，只要使用重量平衡树，就能做到 $O(logn)$ 的复杂度

修改复杂度 $O(logn)$ ，查询复杂度 $O(1)$

把这个东西运用回后缀平衡树

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

如果第一个字符不相同，则直接出结果

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

如果第一个字符不相同，则直接出结果

如果第一个字符相同，那么就是比较两个字符串的第二个字符到最后一个字符对应的子串

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

如果第一个字符不相同，则直接出结果

如果第一个字符相同，那么就是比较两个字符串的第二个字符到最后一个字符对应的子串

实际上这两个子串是旧后缀中的两个

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

如果第一个字符不相同，则直接出结果

如果第一个字符相同，那么就是比较两个字符串的第二个字符到最后一个字符对应的子串

实际上这两个子串是旧后缀中的两个

应用之前的方法，比较已经存在于平衡树中的两个点的前后关系只需要 $O(1)$

把这个东西运用回后缀平衡树

那么我们在比较新的后缀和某个旧后缀的大小关系时，先比较它们的第一个字符

如果第一个字符不相同，则直接出结果

如果第一个字符相同，那么就是比较两个字符串的第二个字符到最后一个字符对应的子串

实际上这两个子串是旧后缀中的两个

应用之前的方法，比较已经存在于平衡树中的两个点的前后关系只需要 $O(1)$

那么现在比较两个后缀的大小就只需要 $O(1)$ 的时间了

Upgraded Version

考虑对平衡树进行可持久化

Upgraded Version

考虑对平衡树进行可持久化

使用Treap，可以将算法扩展成为可持久化算法

Upgraded Version

考虑对平衡树进行可持久化

使用Treap，可以将算法扩展成为可持久化算法

于是可持久化后缀平衡树就诞生了

Upgraded Version

考虑对平衡树进行可持久化

使用Treap，可以将算法扩展成为可持久化算法

于是可持久化后缀平衡树就诞生了

算法时间复杂度 $O(nlogn)$ ，空间复杂度 $O(nlogn)$

BZOJ 3682

给你一个字符串，初始长度为 len，还有一个 n 个元素的序列 P。
接下来 m 个操作，有三种类型，分别是：

1. 在字符串前面加入一个字符

2. 修改 P 中一个元素的值

3. 询问对于所有 $i \in [l,r]$，$S[L-P[i]+1..L]$ 字典序最小的 i（有多个则输出最小的 i，L 是当前字符串长度）

对于 100% 的数据，
$1 \le n \le 500000, 1 \le m \le 800000, 1 \le Pi \le len \le 100000$。

需要动态维护 SA，于是使用后缀平衡树

需要动态维护 SA，于是使用后缀平衡树

然而，使用了后缀平衡树之后，这题就是个裸题了

需要动态维护 SA，于是使用后缀平衡树

然而，使用了后缀平衡树之后，这题就是个裸题了

后缀平衡树维护 SA，再来个线段树维护 P 中对应后缀的排名

需要动态维护 SA，于是使用后缀平衡树

然而，使用了后缀平衡树之后，这题就是个裸题了

后缀平衡树维护 SA，再来个线段树维护 P 中对应后缀的排名

这题就做完了，复杂度 $O(mlogn)$

2013集训队论文 STRQUERY

我们有一个字符串 s，我们需要支持如下4个操作：

[1] 在最左端插入或删除一个字符

[2] 在最右端插入或删除一个字符

[3] 在正中间 ($\frac{|s|}{2}$处) 插入或删除一个字符

[4] 询问字符串 q 出现了几次

一共有 $n(n \leq 150000)$ 个操作，所有询问的长度和 ≤ 1500000

社会猪教你看论文

社会猪教你看论文

首先，我们的后缀平衡树可以支持在最左端插入或删除一个字符，令这个为结构 TL

社会猪教你看论文

首先，我们的后缀平衡树可以支持在最左端插入或删除一个字符，令这个为结构 TL

那么我们将这个整个反过来，并且将询问串也反过来，就能支持在最右端插入或删除一个字符，令这个为结构 TR

社会猪教你看论文

首先，我们的后缀平衡树可以支持在最左端插入或删除一个字符，令这个为结构 TL

那么我们将这个整个反过来，并且将询问串也反过来，就能支持在最右端插入或删除一个字符，令这个为结构 TR

我们现在考虑来实现一个可以在两端删除或插入的结构，我们不妨维护一个结构 TL，一个结构 TR，并让结构 $TB = TL + TR$，那么插入删除分别在 TL 和 TR 做，如果其中一个被删到了0个，那么我们可以把另外一个均匀的分成两半，从均摊意义上这个操作不影响复杂度

同时在 TB 上询问，我们可以先询问 q 在 TL 和 TR 中出现了几次，然后再询问它在跨越的部分出现了几次，注意到跨越部分长度最多为 $2|q|-2$ ，直接取出进行KMP即可

同时在 TB 上询问，我们可以先询问 q 在 TL 和 TR 中出现了几次，然后再询问它在跨越的部分出现了几次，注意到跨越部分长度最多为 $2|q|-2$，直接取出进行KMP即可

然后我们来实现一个可以在正中间插入的结构 TM，我们维护两个 $TB:l,r$，每次插入之后在 l 和 r 之间交换几个字符，使得 l 后面那个位置始终是正中间。这样就能在正中间插入了

同时在 TB 上询问，我们可以先询问 q 在 TL 和 TR 中出现了几次，然后再询问它在跨越的部分出现了几次，注意到跨越部分长度最多为 $2|q|-2$，直接取出进行KMP即可

然后我们来实现一个可以在正中间插入的结构 TM，我们维护两个 $TB: l, r$，每次插入之后在 l 和 r 之间交换几个字符，使得 l 后面那个位置始终是正中间。这样就能在正中间插入了

询问跟在 TB 上询问一样

同时在 TB 上询问，我们可以先询问 q 在 TL 和 TR 中出现了几次，然后再询问它在跨越的部分出现了几次，注意到跨越部分长度最多为 $2|q|-2$ ，直接取出进行KMP即可

然后我们来实现一个可以在正中间插入的结构 TM ，我们维护两个 $TB:l,r$ ，每次插入之后在 l 和 r 之间交换几个字符，使得 l 后面那个位置始终是正中间。这样就能在正中间插入了

询问跟在 TB 上询问一样

那么我们就得出了一个支持此题的结构 TM ，复杂度是 $O(nlogn)$

后缀自动机

为什么要把 SAM 放后缀平衡树的后面?

后缀自动机

为什么要把 SAM 放后缀平衡树的后面?

因为它的思想更难

后缀自动机

为什么要把 SAM 放后缀平衡树的后面?

因为它的思想更难

见丽洁姐的课件

BZOJ 2555

给你一个字符串 $init$，要求你支持两个操作

(1):在当前字符串的后面插入一个字符串

(2):询问字符串 s 在当前字符串中出现了几次？(作为连续子串)

你必须在线支持这些操作。

没有修改时，查询一个串在另一个串中出现次数，只要先在SAM中匹配完前一个串，匹配完的节点的parent树size大小就是出现次数

没有修改时，查询一个串在另一个串中出现次数，只要先在SAM中匹配完前一个串，匹配完的节点的parent树size大小就是出现次数

考虑有修改

没有修改时，查询一个串在另一个串中出现次数，只要先在SAM中匹配完前一个串，匹配完的节点的parent树size大小就是出现次数

考虑有修改

其实有修改就是动态加边，然后要维护子树size

没有修改时，查询一个串在另一个串中出现次数，只要先在SAM中匹配完前一个串，匹配完的节点的parent树size大小就是出现次数

考虑有修改

其实有修改就是动态加边，然后要维护子树size

用LCT维护SAM

CF 364 Div.1 E

给你一个长度为 n 的字符串 S，求最长的一个字符串序列 $a[1..k]$ 满足序列中的每一个字符串都是 S 的子串，且对于任意的 $1 < i \leq k$ 都有 $a[i-1]$ 在 $a[i]$ 中至少出现两次。两次出现允许重叠。

问最大满足条件的 k 是多少。

$n \leq 200000$

定义一个字符串的 $tail$ 为它的出现次数 $>=2$ 的最长的后缀。对于结束节点来说，它的 $tail$ 就是它的 pre 。但是对于一般的点，我们需要不断沿着 pre 向上找，找到第一个在原串出现 2 次的节点才能得到 $tail$ 。具体做法是，我们可以记录 $spre[i]$ 表示最上面一个没有在 i 中出现两次的点，$f[i]$ 代表从沿着 $spre$ 走到根的路径长度（ $spre[i]$ 也是最上面一个 f 不变的点）。那么，如果 i 包含 $sp[pre[i]]$ ，则 $sp[i]=i, f[i]=f[pre[i]]+1$ ；否则 $sp[i]=sp[pre[i]], f[i]=f[pre[i]]$ 。

定义一个字符串的 $tail$ 为它的出现次数 $>=2$ 的最长的后缀。对于结束节点来说，它的 $tail$ 就是它的 pre 。但是对于一般的点，我们需要不断沿着 pre 向上找，找到第一个在原串出现 2 次的节点才能得到 $tail$ 。具体做法是，我们可以记录 $spre[i]$ 表示最上面一个没有在 i 中出现两次的点，$f[i]$ 代表从沿着 $spre$ 走到根的路径长度（ $spre[i]$ 也是最上面一个 f 不变的点）。那么，如果 i 包含 $sp[pre[i]]$ ，则 $sp[i]=i, f[i]=f[pre[i]]+1$ ；否则 $sp[i]=sp[pre[i]], f[i]=f[pre[i]]$ 。

那么如何检查串 a 是否在串 b 中出现 2 次呢？可以用线段树维护 $right$ 集合，如果串a在指定范围内出现过，则说明 a 在 b 中出现了 2 次。线段树合并维护 $right$ 集合。

Thanks