An artistic illustration of a ginger and white cat lying down in a sun-dappled forest. The cat is positioned in the lower center, resting on a light-colored surface. The background is filled with lush green foliage and trees, with sunlight filtering through the leaves, creating a warm and serene atmosphere. The overall style is soft and painterly.

strings

监督 & 制作进行: *crazy_cloud*

原画: *Werkeytom_FTD, Drin_E, crazy_cloud*

Preface

- 一些约定，在后面的讨论中，若无特殊说明，字符集 Σ 大小为常数。
- 在习题中，所有字符串若无特殊说明，都是由小写拉丁字母组成。



理性愉悦



Hashing

- Hash 算法很简单。要注意的点就是：
 - 单哈希在数据范围较大的时候有大概率被生日攻击冲突，应采用双哈希。
 - 自然溢出是可以简单构造来卡的。



排序列表

From XJOI

- 给定数轴上 n 个形如 (a_i, b_i) 的开区间，令 $C(m)$ 表示所有包含实数 $m + 0.5$ 的区间编号排序而成的列表。
- 一个列表，如果存在一个 m 使得它能够表示成 $C(m)$ ，那么它就是合法的。
- 给定 K ，你需要输出字典序第 K 大的非空合法列表。
- $1 \leq n \leq 10^5, 0 \leq a_i < b_i \leq 10^9$ ，给定的 K 保证有解。



排序列表

From XJOI

- 注意到本质不同的合法的非空列表只有 $O(n)$ 个。
- 考虑将它们全部排列出来然后使用 *nth_element* 找到第 K 大。
- 那么我们要解决两个问题：如何储存这么多列表？如何快速比较两个列表的字典序大小？
- 只需要可持久化线段树就能解决上面的问题。我们开一棵可持久化权值线段树，然后扫描线，每次遇到边界就在对应位置插入删除，然后储存一个新列表只需要储存对应的版本的根节点。可是这样有可能会有一些本质相同的被储存。因此我们需要对这个列表哈希来判重，这个可以使用线段树来维护。
- 至于比较大小，我们可以利用线段树上的哈希值来快速找到两个列表第一个不同的位置判断一下就好了。
- 时间复杂度 $O(n \log n)$ 。

KMP & exKMP

- KMP 大家都比较熟，只简单提一下扩展 KMP：
- 定义 $n = |S|, m = |T|$, $extend_i = S_{i,n}$ 与 T 的最长公共前缀长度。要求在线性算出所有的 $extend$ ，即是对经典 KMP 问题的一个扩充。
- 设辅助函数 $next_i$ 表示 $T_{i,m}$ 与 T 的最长公共前缀长度。
- 设 $extend_{1..k}$ 已经算好，并且在以前的匹配过程中到达的最远位置是 p 。最远位置严格的说就是 $i + extend_i - 1$ 的最大值。
- 设 $a + extend_a - 1 = p$
- 则根据定义有： $S_{a,p} = T_{1,p-a+1}$ ，那么 $S_{k+1,p} = T_{k-a+2,p-a+1}$ 。

KMP & exKMP

- 设 $L = next_{k-a+2}$, 有两种情况:
 - $k + L < p$, 这个时候 $extend_{k+1} = L$, 因为 $k + 1 \sim p$ 这些位置在 $T_{k-a+2, p-a+1}$ 已经出现并且是不匹配的
 - $k + L \geq p$, 这个时候只能知道 $extend_{k+1} \geq p - k$, 因为大于 p 的位置在 $T_{k-a+2, p-a+1}$ 没有出现, 所以要暴力去判断往后的位置是否匹配。
- 每次暴力判断, 每匹配成功一次 p 就会右移一位, 而 p 最多右移 n 位, 所以总复杂度是 $O(n)$ 的。
- 计算 $next$ 则是一个自配的过程。



可持久化KMP

- 问题:一开始有一空串, n 次操作, 每次在串末尾加入一个字符问最小循环节, 要求在线与可持久化。
- 由于一般 KMP 算法是均摊分析复杂度的, 所以不能可持久化。
- 考虑一种更新 KMP 只需要跳 \log 步的算法:
- 假设当前在 j , 我们看 $f_j + 1$ 是否与 i 匹配。如果匹配, 那么 $f_i = f_j + 1$ 。
- 否则分类讨论一下:
 - $2f[j] \leq j$, 直接让 $j = f[j]$ 。
 - $2f[j] > j$, 此时长度为 j 的前缀存在长度为 $j - f_j$ 的循环节, 且有 $j - f_j < \frac{j}{2}$, 且在循环段中 $f_j + 1$ 对应的字符都是一样的, 所以若 $f_j + 1$ 与 i 不匹配, 那么我们就可以直接令 $j = j \bmod (j - f_j)$ 。

例题

- 给定两个字符串 S_1, S_2 。问 S_1 至少要操作多少次才能使 S_2 和 S_1 的前缀完全匹配或至多存在一个字符不相同的次匹配。
- 一次操作定义为将一个字符串的第一个字符移到最后或把最后一个字符移到开头。
- $1 \leq |S_2| < |S_1| \leq 10^6$



- 先把 S_1 复制一遍。
- 利用 exKMP 在 $O(n)$ 内求出 S_1 第 i 个字符，向右最多匹配 S_2 的前缀的长度 a_i ，以及向左最多匹配 S_2 的后缀的长度 b_i 。
- 若对于某一个 i ，满足 $a_i + b_{i+m-1} \geq m - 1$ ，则 $S_{1,i,i+m-1}$ 与 S_2 最多只有一个字符不相同。求操作最小次数即可。



Trie

- Trie 树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计，排序和保存大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。它的优点是：利用字符串的公共前缀来减少查询时间，最大限度地减少无谓的字符串比较，查询效率比哈希树高。
- Trie 树中每条边都有一个字符，根到一个节点路径表示了一个串。



供给侧改革

HAOI2017

- 给一个长度为 n 的 01 字符串，是随机的。
- 对于一个询问 $[l, r]$ ，要求对于每个 $l \leq i < r$ ，求出区间 $[i, r]$ 拿出两个不同的后缀的 LCP 最大值的和。
- $1 \leq n \leq 10^5$ 。



供给侧改革

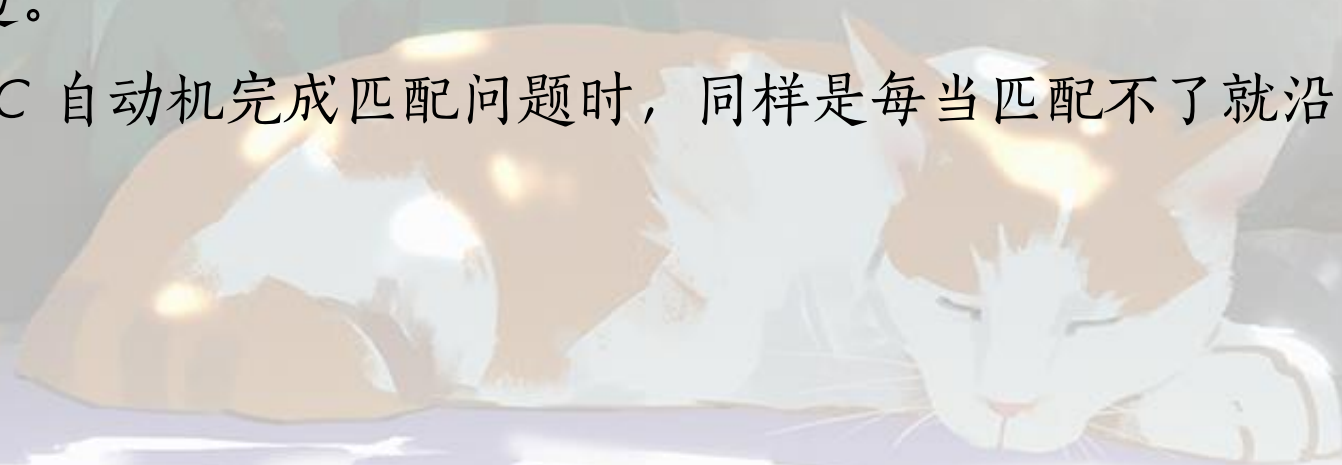
HAOI2017

- 可以注意随机情况下 LCP 不会很长，我们假设是 40。
- 把询问挂在右端点上，然后扫描，线段树对于每个 l 维护询问 $[l, r]$ 的答案。
- 将每个位置开始 40 位扔进 *Trie* 中，对于 *Trie* 中每个节点保存最大的两个位置。
- 那么线段树需要支持区间 \max 和区间和。
- 我们不需要什么吉司机线段树，因为一个位置只会被更新 40 次，因此可以暴力。

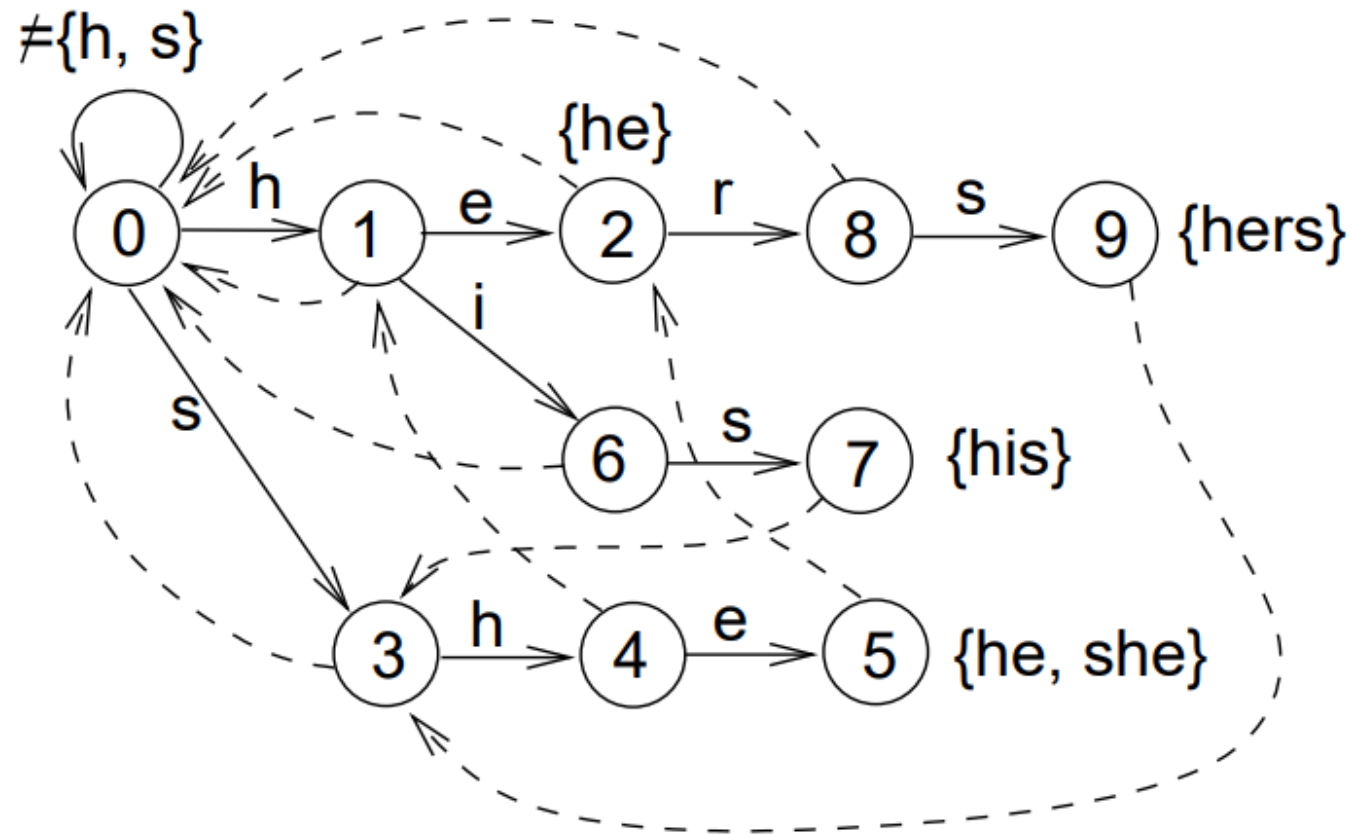


AC 自动机

- AC 自动机是 *Trie* 加上 *fail* 边组成的。
- 我们记 $str(x)$ 表示根到 x 路径代表的字符串。
- *fail* 与 KMP 的 *next* 数组定义类似，一个节点 x 的 $fail = y$ 要满足 $str(y)$ 是 $str(x)$ 的后缀而且最大。即 *fail* 指向 *Trie* 上最长后缀。
- 其建立过程非常简单，建出 *Trie* 后，进行 *bfs*，做类似 KMP 算法的过程即可处理出 *fail* 边。
- 在使用 AC 自动机完成匹配问题时，同样是每当匹配不了就沿 *fail* 边跳。



AC自动机



回忆树

BZOJ 4231

- 给一颗 n 个节点的树，每条边有一个字符。
- 有 q 个询问，每次给定两个节点 x 和 y ，以及一个字符串 S ，问将路径 x 到 y 的字符顺次连接成字符串后， S 在其中的出现次数。
- 询问字符串长度总和在 3×10^5 以内。
- n 和 q 均为 10^5 。



回忆树

BZOJ 4231

- 如果 S 横跨 lca ，我们可以将穿过 lca 的长度为 $2|S|$ 的字符串拿出来直接做 KMP。
- 其余情况，考虑离线，对询问串的正反串都构出 AC 自动机。
- 然后遍历原树，用数据结构搞搞即可。



另一种简单应用

- AC 自动机上的 DP 。
- 大家都会。



Manacher Algorithm

- 解决回文串问题一般有以下两种办法：Manacher 算法和回文树。
- Manacher 算法的思路很简单，主要是利用了回文的性质。
- 维护 mx 表示当前求出的所有回文串能扩展到的最右端点， pos 表示该回文串的回文中心。
- 当 $i \leq mx$ 的时候可以利用对称的性质来更新答案， $r_i = \min(r_{2pos-i}, mx - i + 1)$ 。
- 然后暴力扩展 r_i 。
- 时间复杂度，可以发现每次暴力扩展 r_i 都会使得 mx 向右移动。因此总的时间复杂度是 $O(|S|)$ 。
- 为了方便处理奇偶回文的问题通常在字符之间加上特殊字符。

项链

GDKOI2016, day2 T2, necklace2

- 给你一个长度为 n 字符串 S , 这个字符串连成了一个环。
- 你需要在这个环上删除一段连续的字符, 使得最后剩下的环是对称的。
- 最大化剩余环的长度。
- $1 \leq n \leq 10^5$



项链

GDKOI2016, day2 T2, necklace2

- 题目可以转化为找到这个环的一个最长的子串使其可以由两个回文串拼接得到。
- 考虑倍长字符串，并且在字符之间加上特殊字符，这样我们只需要考虑奇回文的情况。
- 考虑回文中心合法的条件：两个回文中心 $i, j (i < j)$ 合法的条件是 $j - i \leq n$ 以及 $i + r_i \geq j - r_j$ 。你要最大化 $j - i$ 的值。
- 考虑枚举 $x = i + r_i$ 扫描线，处理对应的所有的 i ，所有 $j - r_j \geq x$ 的 j 都可以作为候选，除此之外还要满足 $j \leq i + n$ ，数据结构维护一下就可以 $O(n \log n)$ 。
- 如果从右向左扫还可以使用并查集来维护，可能可以做到 $O(n\alpha(n))$ 。

[Problem Name Lost]

From Codeforces Gym

- 给定一个串 S 。
- 你要选出两个不相交的子串 S_1, S_2 。
- 使得 $S_1 + S_2$ 是一个回文串。
- S_1 和 S_2 可以为空。
- 最大化 $|S_1| + |S_2|$ 。
- $1 \leq |S| \leq 10^5$



[Problem Name Lost]

From Codeforces Gym

- 假设选择的是 A 和 B ，不妨令 $|A| \geq |B|$ 。
- 那么肯定是 $A = \text{rev}(B) + T$ ，其中 B 是一个回文串。
- 枚举 T 的回文中心 t ，枚举半径 r 。
- 相当于在 $[t + r, n]$ 中挑一个 i 满足 $\text{rev}(\text{pre}_{t-r})$ 和 $S_{i,n}$ 的 LCP 最长。
- 由于回文中心是固定的，因此半径肯定是越长越好，所以 r 是固定的。
- LCP 的最大值的话把后缀排一下序二分一下就好了。



Palindrome Tree(Palindrome Automaton)

- 一个串 S 的回文树是两棵树组成的森林，两棵树的根分别是 odd 和 $even$ ，一棵存的是所有的奇回文子串，一棵存的是所有的偶回文子串。特殊地，定义 $even$ 节点对应的回文串长度为 0，即空串；定义 odd 节点对应的回文串长度为 -1，是一个不存在的字符串。
- 每一个节点和 S 中的回文子串一一对应，有若干代表互不相同的字符的转移边，连向两边同时加上该字符形成的回文串对应的节点。同时每一个节点还有一个 $fail$ 指针，指向其最长回文后缀所对应的节点。每个点最多只会被一个转移边指向。
- 由于 S 最多只会有 $|S|$ 个本质不同的回文子串（增量归纳证明，新加入字符只有最长回文后缀是可能有用的），因此节点数和转移数都是 $O(|S|)$ 的。

插入过程时间复杂度

- 构造过程：增量构造，每次在末尾增加一个字符。实时记录当前字符串最长回文后缀对应节点 suf 。每次插入一个字符 c ，找到最长的回文后缀 p 使得 c 是其前驱，如果 p 没有 c 这个转移就新建节点 np 。如果新建了节点就要再找到 p 的最长回文后缀 q 使得 c 是其前驱来计算 $faill_{np}$ 。注意在实现的过程中不要先给 p 增加 c 这个转移边再找 q ，否则会在特殊情况下陷入死循环。插入完之后更新一下 suf 。
- 时间开销在找最长回文后缀满足前驱是 c 的这个过程。一次插入可能要执行两次这种操作，我们只对第一次进行分析，第二次的复杂度可以类似推理得到。
- 考虑定义势能函数 Φ 表示 suf 对应的节点的长度，每一次 suf 往 $fail$ 链跳时长度至少会减少 2，而插入字符长度只会增加 2。因此总的时间复杂度是 $O(|S|)$ 。
- 当字符集较大的时候要带一个 $O(\log \Sigma)$ 的复杂度来维护转移边。

前端插入 & 可持久化 & Trie 上回文树

然而并没有什么卵用

- 前端插入其实和后端插入没什么类似，找到最长的回文前缀满足其后继是 c 即可。
- 注意到回文串的对称性，一个回文串的最长回文前缀其实就是最长回文后缀。
- 因此我们需要多记录的只是整个字符串的最长回文前缀。注意前端插入可能会对最长回文后缀长度产生影响，反之亦然。这个简单更新一下就好了。
- 可以发现我们前面所说的基础插入算法都是基于势能分析的，如果要可持久化的话需要一个不基于势能的插入算法。类似 AC 自动机的实现，我们直接开一个数组来记录失配的转移，每次插入可以通过可持久化的手段来 $O(\log \Sigma)$ 更新。
- Trie 上的回文树直接套用这种不需要势能分析的插入算法即可，树的规模和构造的复杂度依然是 $O(n \log \Sigma)$ 的。

AlphaDog

alphadog, by PhilipsWeng

- 一个串 S 的特殊值 $F(S)$ 定义为

$$\sum_{1 \leq x \leq y \leq |S|} \text{LCP}(x, y)$$

- 这里的 $\text{LCP}(x, y)$ 指的是前缀 x 和前缀 y 的最长回文后缀的长度。
- 一开始 S 是一个空串。接下来 q 次末端插入，每次插入完之后你都需要回答 $F(S)$ 的值。
- 强制在线。
- $1 \leq q \leq 10^5$



AlphaDog

alphadog, by PhilipsWeng

- 建出回文树，将 $fail$ 上的边按照两个端点的回文串长度差分得到权值 val 。
- 每次加入相当于给答案加上一整条树链的 $val \times size$ 的和，并且将这条树链上的 $size$ 加上 1。
- 使用 LCT 维护一下就好了。
- 时间复杂度 $O(q \log q)$ 。



回文串

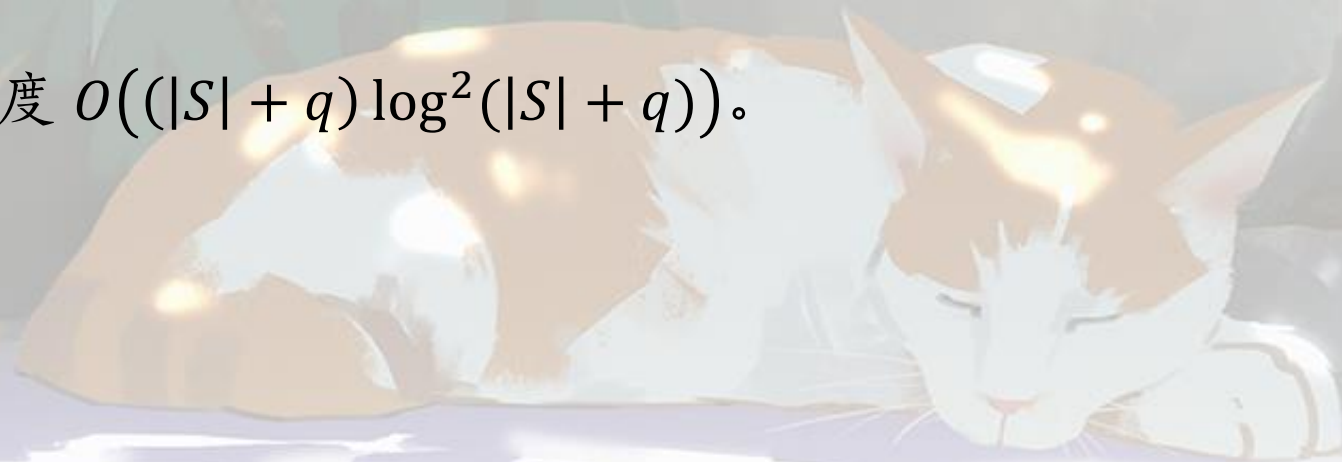
- 给出一个字符串 S ，现在有四种操作共 q 次：
 - $Addl\ c$ ：在当前字符串的左端加入字符 c 。
 - $Addr\ c$ ：在当前字符串的右端加入字符 c 。
 - $Transl\ l_1, r_1, l_2, r_2$ ：取出 S 的两个子串 S_{l_1, r_1} 和 S_{l_2, r_2} ，现在你需要执行若干次操作，形如在前一个字符串的左端插入/删除一个字符，使前者变换为后者，你会用最少的步数达成目标。
 - $Transr\ l_1, r_1, l_2, r_2$ ：取出 S 的两个子串 S_{l_1, r_1} 和 S_{l_2, r_2} ，现在你需要执行若干次操作，形如在前一个字符串的右端插入/删除一个字符，使前者变换为后者，你会用最少的步数达成目标。
- 对一次变换，定义 $S_{i,j}$ 是好的当且仅当 $S_{i,j}$ 是回文的而且在变换过程出现过。假定字符串长度为 n 。一次变换的优美度定义为：

$$\sum_{i=1}^n \sum_{j=i}^n [S_{i,j} \text{ 是好的}] (j - i + 1)$$

- 请输出每一次变换的优美度。
- $1 \leq |S|, q \leq 10^5$

回文串

- 考虑离线。建出最终的回文树，再正反各建一次回文树即可得到任意前（后）缀最终的最长回文后（前）缀，确定子串的回文前后缀只需要在此基础上在回文树上倍增即可。
- 询问其实就是类似于 $fail$ 树上一条路径的 $len_v \times |right_v|$ 的和。
- 树链剖分维护一下。注意特判 LCA。
- 时间复杂度 $O((|S| + q) \log^2(|S| + q))$ 。



塔

- 一开始你有一个空串，然后执行 q 次操作。
 - 每次操作形如左端/右端插入以及连续撤销 k 次最近的插入操作。
 - 每一次插入之后你都需要输出其最长回文子串的长度。
 - 强制在线。
-
- $1 \leq q \leq 10^7, |\Sigma| \leq 100$
 - 时间限制 $1s$
 - 空间限制 $512M$



塔

- 发现直接可持久化的回文树要么 MLE 要么 TLE。
- 似乎察觉到一点不对劲。
- 注意到每次插入最长回文子串长度最多增加 2。
- 直接 *hash* 判断 $len + 1$ 和 $len + 2$ 是否合法就行了。
- 时间复杂度 $O(q)$ 。
- 考察字符串是否学傻。



Suffix Array

- 这年头过气后缀数组都快被后缀自动机完全代替了。
- ~~可能在求 LCP/LCS 的时候还是会比其它字符串数据结构方便一点。~~
- 后缀数组顾名思义就是对所有后缀按字典序排序。
- 常见的两种构造算法：倍增算法（DA）和 DC3 算法，分别是 $O(n \log n)$ 和 $O(n)$ 的。限于篇幅（考虑到在座的都辣么强），这个就不需要细讲了。



后缀数组上树

- 直接将倍增长度改成树上的倍增祖先。
- 求 *height* 数组不能利用一个串的那一种性质，可以倍增，利用处理好的 k 级祖先的排名数组来做。



优秀的拆分

NOI2016, day1 T1, excellent

- 如果一个字符串可以被拆分为 $AABBAABB$ 的形式，其中 AA 和 BB 是任意非空字符串，则我们称该字符串的这种拆分是优秀的。
- 现在给出一个长度为 n 的字符串 S ，我们需要求出在它所有子串的所有拆分方式中，优秀拆分的总个数。
- $1 \leq n \leq 3 \times 10^4$



优秀的拆分

NOI2016, day1 T1, excellent

- 只需要计算每个位置开始/结尾的 AA 串的个数即可。
- 考虑使用关键点的方法，枚举关键点间隔 $L(1 \leq L \leq n)$ ，每隔 L 个字符设置一个关键点。
- 如果有一个长度为 $2L$ 的 AA 串，那么它肯定会经过相邻两个关键点。
- 我们枚举相邻关键点，求一下 LCP/LCS 然后在数组上打 tag 更新一下某些位置开始/结尾的 AA 串个数就好了。
- LCP/LCS 用后缀数组优化一下。
- 时间复杂度是调和级数的形式，即 $O(n \log n)$ 。

后缀自动机

- 后缀自动机是一个 *DAG* 型结构，满足从根出发，任意走法对应的字符串为原串的一个子串，任意不同的走法会走出不同的字符串。
- 其中，后缀自动机的一个节点代表一个等价类，*right* 集相同的字符串会处在同一个等价类内。
- 什么是 *right* 集？即一个字符串在原串出现位置右端点的集合。
- 任意一个节点代表的字符串长度是一段区间，而且这些字符串有后缀关系。
- 通常用 $Min(x)$ 和 $Max(x)$ 表示节点 x 代表字符串长度的极值。

后缀自动机

- 同样，后缀自动机上也有 *fail* 边。
- 假如 $fail_x = y$ ，有 $Min(x) = Max(y) + 1$ ，且 y 代表的字符串都是 x 代表的字符串的后缀。
- 根据 *fail* 边可以建出一颗树，通常称为 *parent* 树。
- 对于一个节点 x ，其 *right* 集为 *parent* 树上儿子的 *right* 集的并集。
- 不同子树间 *right* 集不会有交
- 这就是后缀自动机的基本性质。



Search Engine

Yandex.Algorithm 2016, final round, D

- 有一个长度为 n 的字符串 S 。
- 你现在有一个初始为空串的字符串 T 。
- 你执行 n 次操作，每次在 T 的头部或尾部添加一次字符得到 T' ，然后 ans 加上 T' 在 S 中出现的次数。
- 最大化 ans 。
- $1 \leq |S| \leq 2 \times 10^5$














Search Engine

Yandex.Algorithm 2016, final round, D

- 先对 S 建 SAM, 然后求出每个点出现次数。
- 考虑在尾部添加是什么?
- 沿着转移边走一步。
- 考虑在头部添加是什么?
- 待在自己或者走向 $parent$ 树的一个儿子。
- 注意到我们一定会在一个节点头加直至长度达到 Max 。
- 那么我们随便 dp 一下就好了。

基础SAM练习题

- 如果你秒了这道傻逼题，恭喜你爆踩了许多牛逼选手。

#	Who	=	Penalty	A	B	C	D
1	 tourist	6	427	+ 00:35	+2 01:22	+ 01:36	+1 02:10
2	 Um_nik	5	262	+1 00:53	+ 01:50	+ 00:16	-1
3	 LHiC	5	372	+ 01:07	+1 02:18	+ 01:22	
4	 izban	5	388	+ 01:00	-1	+ 01:37	+1 02:00
5	 snuke	4	210	+ 00:51	-1	+1 01:19	
6	 Petr	4	214	+ 00:12		+ 00:41	
7	 vepifanov	4	223	+ 00:27		+ 01:14	
8	 mikle98	4	267	+ 00:55		+ 01:54	
9	 alexei.zayakin	4	274	+1 01:44		+ 01:07	
10	 aid	4	291	+ 01:47		+1 01:28	
11	 Merkurev	4	338	+4 01:30	-5	+ 00:58	
12	 yutaka1999	4	339	+ 01:57		+ 00:45	
13	 rng.58	3	90	+ 00:24		-1	

Cool Slogans

Codeforces Round #364 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(允许重叠)
- $1 \leq n \leq 2 \times 10^5$



Cool Slogans

Codeforces Round #364 E

- 容易发现若 a_i 不为 a_{i+1} 的后缀显然是不优的，因为这种情况下可以把 a_{i+1} 的末尾直接去掉。
- 设 $f(str)$ 表示字符串 str 作为 a_k 时的最大的 k 。
- 考虑建 SAM，可以发现对于同一节点对应的字符串，由于其 $right$ 集相同，所以同一节点对应的字符串的状态是相同的，我们只需要处理其最长串的 f 值即可。
- 设 F_x 表示 x 到根路径上选若干个节点出来组成 a 数组时的最大值，
- pos_x 则记录 a_{F_x} 具体是哪个节点。
- 判断 F_x 是否能由 $F_{pos_{fa_x}}$ 转移过来，也就是 pos_{fa_x} 所代表的字符串在 x 中是否出现了两次。这个可以通过预处理 $right$ 集的可持久化线段树来判断。

Killjee and k -th letter

CodeChef January Challenge 2018, KILLKTH

- 给一个长度为 n 的字符串，把这个串的所有子串按字典序排序后拼到一起组成一个新串，每次询问新串的某个字符是什么。
- $1 \leq n \leq 2 \times 10^5$ ，强制在线。



Killjee and k -th letter

CodeChef January Challenge 2018, KILLKTH

- 先通过建反串的后缀自动机把原串的后缀树建出来,
- 可以发现把子串按字典序排序等价于求出后缀树的 dfs 序, 在 dfs 序上二分出对应节点, 然后在对应节点计算即可。



广义后缀树

- 就是多个字符串建成的 SAM。
- 先考虑给你一堆字符串，将它们建 SAM。
- 我们每插入完一个字符串后将 *last* 归 0，但是注意我们的添加里要加点东西。

```
node *addchar(node *u, int c)
{
    if (u->trf[c] && u->trf[c]->len==u->len+1)
        return u->trf[c]; //如果是建Trie上SAM可以去掉这两行
    node *x=++tp,*v;
    if (u->trf[c])
    {
        *x=*(v=u->trf[c]),x->len=u->len+1,v->fa=x;
        for (; u && u->trf[c]==v; u->trf[c]=x,u=u->fa);
        return x;
    }
}
```


广义后缀树

- 再来考虑 *Trie* 上建 SAM。
- *Trie* 上建 SAM 和给一堆字符串然后建 SAM 有什么不一样呢？
- *Trie* 的深度和没有保证！
- 我们对 *Trie* 进行 *bfs*，每个节点插入时 *last* 设为父亲，复杂度就有保证了。



重量平衡树速成

- 定义：一次操作所影响的最大子树的大小的最坏/均摊/期望是 $\log n$ 的平衡树称为重量平衡树。
- 常见重量平衡树：替罪羊树、*Treap*、红黑树。
- 重量平衡树能拿来干什么？
- 方便我们在平衡树每个点维护信息，并支持动态更改。



后缀平衡树

- 相当于我们用重量平衡树维护后缀数组。
- 对于平衡树每个节点，定义函数 f 来映射其的排名。
- 即对于两个节点，如果 $f(x) < f(y)$ ，那么 x 的 $rank$ 比 y 低。
- f 值怎么定呢？
- 每个节点有一个实数范围 (L, R) ，其 f 值定义为范围中点，根节点范围是 $[0, 1]$ 。
- 对于实数范围为 (L, R) 的节点 x ，其左儿子范围为 (L, M) ，右儿子范围为 (M, R) 。
- 这样子的 f 函数显然能映射排名。
- 当平衡树形态改变时，暴力重构所有被影响的点。

后缀平衡树

- 有了 f 函数，我们可以实现 $O(1)$ 的比较两个后缀的字典序大小。
- 现在我们考虑原串头部添加一个字符这样的操作，后缀平衡树如何更新。
- 相当于添加了一个后缀，其余后缀没有改变。
- 比较大小非常方便，比如现在在比较后缀 1 和后缀 i ，如果 s_1 和 s_i 不同，大小直接分出来了，否则相当于比较后缀 2 和后缀 $i+1$ ，用 f 快速比较即可。
- 因此有了后缀平衡树，我们可以做到字符串首部加删字符，并能 $O(1)$ 比较两个后缀字典序大小。



后缀平衡树

- 同样，我们可以维护 *height*。
- 每次新加入一个时，要求其与前继，后继与其的 *LCP*。
- 同样可以比较第一位。如果相同，就可以转化为原有的两个后缀求 *LCP*，即一段区间 *height* 的最小值。



后缀平衡树

- 我们接下来考虑一个问题，如何求一个字符串 q 在一个字符串 S 中的出现次数。
- 其中这个 S 会进行动态的头部插入。
- 我们可以用后缀平衡树来解决这个问题，将询问拆成有多少后缀字典序比 $q\#$ 小（ $\#$ 表示一个无穷大字符），以及有多少后缀字典序比 q 小。
- 可以在平衡树上二分，重要的是判断 q 与一个后缀的大小关系。
- 二分 + 哈希： $O(\log^2 n)$ 。
- 直接一位一位比较： $O(|q| \log n)$ 。
- 第二个做法有优化的希望，我们考虑优化。

后缀平衡树

- 假如二分到节点 x ，我们保留下 x 到根和 q 匹配最多的祖先设为 $last$ ，LCP 为 len 。
- 假设 $rank_x < rank_{last}$ ，我们求 x 与 $last$ 的 LCP，如果没有达到 len ，显然直接走向右子树。
- 如果到达了 len ，我们再从 $len + 1$ 位开始比较。
- $rank_x > rank_{last}$ 类似。
- 容易发现这是 $O(|q| + \log^2 n)$ ，好像挺菜的。
- 注意到我们要求的是 $last$ 和 x 的 LCP， $last$ 还是 x 的祖先，不难想到优化。
- 下面仍讨论 $rank_x < rank_{last}$ 。从 $last$ 往下走的过程，每往左子树走可以获得右子树 $height$ 最小值。这样走到 x 时已经维护出了它们的 LCP。
- 那么 $O(|q| + \log n)$ 。

后缀平衡树

- 如果将刚才那题改成区间查询呢？
- 即 q 在 $S_{l,r}$ 的出现次数。
- 我们可以在每个平衡树节点保存一个 *vector*，按顺序存储子树内所有后缀。
- 重量平衡树我们使用 *Treap*，来利用其随机性质。
- 这样我们 *vector* 重构复杂度是很有保证的，为一个 \log 。
- 这样区间查询可以变成在 *vector* 上二分。
- 这样要 $O(\log^2 n)$ ，很不优美，我们可以对于 *vector* 每个元素维护其对应左右儿子的 *vector* 中哪个元素（ \leq 它的最大的），这样就降为 $O(\log n)$ 。

冷静分析



Statistics of strings

HackerEarth October Circuits '17

- 给定字符集大小 a 。
- 定义 $f(s)$ 为 s 这个字符串所有位置 *extend* 函数（扩展 KMP 的那个匹配函数）中的最大值。
- 你要计算所有合法的长度为 n 的字符串的 f 值的和对 mod 取模
- $n \leq 22, 1 \leq a \leq 10^9, 1 \leq mod \leq 10^9$

Statistics of strings

HackerEarth October Circuits '17

- 既然问题是关于最大值，那么我们考虑将 $\max x$ 拆成 $\sum_{i=1}^n [x \geq i]$ 。
- 考虑枚举后缀开始的位置 x 以及上式中的 i ，然后我们相当于要统计有多少字符串满足 x 以前的位置 extend 函数都严格小于 i ，且 x 位置的 extend 大于等于 i 。
- 使用容斥：我们枚举前面的每一个位置的 extend 是否严格小于 i ，如果不是，就用并查集把相同的位置并起来，最后我们会得到若干个连通块，连通块个数结合容斥系数我们就能把答案求出来。

硬币游戏

SDOI2017

- n 个人给出 m 个不同的 01 字符串。
- 设 S 初始为空串，每次等概率令 $S + 0$ 或 1 ，当 S 中出现某个人的字符串时，则此人胜利并结束。
- 求每个人的胜利概率。
- $1 \leq n, m \leq 300$

硬币游戏

SDOI2017

- 假设匹配到一个串后不会停止。
- 设 p_i 表示匹配到的第一个串为 i 的概率, p_x 表示没有匹配到任何串的概率。 p_i 即答案。
- 对于第 i 个串, 我们考虑由一个没匹配到任何串的字符串接上它形成的串, 则出现的概率为 $p_x \times 0.5^m$ 。
- 考虑用另一种方法表示它。
- 当 i 为第一个被匹配到的, 概率为 p_i 。
- 否则, 假设第一个匹配到的串为 j 。由于原串没有匹配到任何串, 所以 i 的一部分前缀一定与 j 的一部分后缀重合。
- 假设重合的长度为 k , 此时概率为 $p_j \times 0.5^{m-k}$ 。
- 即有以下方程 $p_x \times 0.5^m = p_i + \sum p_j \times 0.5^{m-k}$, 结合 $\sum p_i = 1$ 即可高斯消元。

Liblume

The 18th Zhejiang University Programming Contest, B

- 给个 $n \times m$ 的小写字母组成的矩阵。
- 你可以交换任意两行任意次，然后选出一个面积最大的子矩阵，要求这个矩阵的所有行和所有列都回文。
- $1 \leq n \times m \leq 2 \times 10^5$

Liblume

The 18th Zhejiang University Programming Contest, B

- 考虑所求子矩阵每一行长度都是奇数（偶数同理），那么枚举这些行的回文中心是在第 c 列，然后用 Manacher 可以知道第 i 行以第 c 列为回文中心的回文半径 r_i 。
- 假设你知道所求子矩阵有多少列，那么只要把这些半径为 r_i 的回文串，按照列长截断，然后把相同的组成一对，就得到了一个回文子矩阵。
- 容易发现有用的列长只有 $O(n)$ 种，你先把这些回文串排序，同时求出相邻两个串的 LCP。之后，从大到小枚举列长，只会发生两种事件：一个回文串加了进来，或者是相邻两组回文串合并成一组相同的回文串。
- 使用后缀数组优化一下就可以做到 $O(nm \log nm)$ 。

字符串计数

BZOJ 4180

- 给出了一个字符串 T ，字符串 T 中有且仅有 4 种字符 ‘A’，‘B’，‘C’，‘D’。
- 构造一个新的字符串 S ，构造的方法是：进行多次操作，每一次操作选择 T 的一个子串，将其加入 S 的末尾。
- 对于一个可构造出的字符串 S ，可能有多种构造方案，定义构造字符串 S 所需的操作次数为所有构造方案中操作次数的最小值。
- 求对于给定的正整数 n 和字符串 T ，所能构造出的所有长度为 n 的字符串 S 中，构造所需的操作次数最大的字符串的操作次数。
- $1 \leq n \leq 10^{18}$ ， $1 \leq |T| \leq 10^5$

字符串计数

BZOJ 4180

- 最大化操作的次数，需要选取的可拼接的子串的长度尽可能小。
- 考虑构造转移矩阵，设 $f_{i,j}$ 表示以 i 开头的子串后可以允许接 j 开头的子串的所需的最短长度。
- 对模式串建立 SAM，然后按照拓扑逆序更新。
- 具体来说，设 $a_{i,j}$ 表示后缀自动机中的结点 i 向后选择一个最短长度，使其可以接上 j 开头的子串，则有 $a_{i,j} = \min(a_{i,j}, a_{son_{i,k},j} + 1)$ 。
- 最终有 $f_{i,j} = a_{son_{1,i},j}$ ，因为 1 号结点的 i 儿子能到达的所有串都是以 i 开头的子串。
- 然后二分答案 + 矩阵快速幂判断一下得到的最小长度是否 $\geq n$ 即可。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 给定一个字符串 S 。
- 求所有字符串 T 的个数，满足 T 的后缀 Trie 与 S 的后缀 Trie 同构。
- 答案对 42424242 取模。
- $1 \leq |S| \leq 10^5$

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 这是一道十分有趣的题目，需要对字符串的性质有一定的分析。
- 定理 1:
- *Suffix Trie* 中叶子节点数目取决于 S 中最长的满足是某个其它后缀的前缀（即为某个其它后缀的 *border*）的后缀长度。
- 令该后缀长度是 L ，那么 *Suffix Trie* 中的叶子节点数目就是 $n - L$ 。
- 证明:
- 考虑最长的作为另外一个后缀的前缀的后缀。可以发现从这两个后缀左端点开始的每一个后缀都存在相同的关系。也就是较小的后缀都会成为别的叶子到根路径的一部分。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 定理 2:
 - 合法串不同的字符种类数相同。
 - 证明：考虑根节点的儿子。
-
- 定理 3:
 - 所有字符都会在字符串前 $n - L$ 个位置中至少出现一次。
 - 证明：
 - 如果长度为 L 的后缀和其对应的（某个后缀的）前缀没有重叠直接得证。
 - 否则，重叠部分虽然位于后 L 个位置，但是和前缀的一段相同，因此也在前面出现过。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 定理 4:
- 对于 $\forall 1 \leq i, j \leq n - L$, 如果 $S_i = S_j$, 那么在合法串中这两个位置也一定相同。反之亦然。
- 证明:
- 注意到所有叶子节点的深度是不会有重复的, 也就是说一种深度对应一个长度一定的后缀。考虑两个字符所在位置开头的后缀所对应的叶子节点, 是否属于根节点的同一个儿子是由这两个字符的关系确定的。因此在合法串中这两个字符的关系也一定要保持一致。
- 由定理 4 我们可以立即得到: 统计答案时, 前 $n - L$ 个字符只需要和 S 中的保持一致。令不同字符数量为 D , 最后答案只需要乘上 26^D 即可。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 定理 5:
- 最多只有 $n - L$ 种方法来填充后 L 个字符。
- 证明:
- 对于长度为 L 的后缀, 一旦我们固定了其对应的前缀的起点, 我们就可以唯一确定出后面剩余的全部 L 个字符 (并不一定合法)。因此最多只有 $n - L$ 种方案。
- 定理 6:
- 在固定了前面 $n - L$ 个字符的情况下, 字符串合法当且仅当 $\forall 1 \leq i \leq n - L, \text{LCP}(i, n - L)$ 和 S 中的保持一致。
- 证明:
- 首先考虑一个弱一点的结论, 字符串合法当且仅当 $\forall 1 \leq i, j \leq n - L, \text{LCP}(i, j)$ 和 S 中的保持一致, 这个是显然的。然后考虑当 $i \leq j < n - L$ 的时候, 它们的 LCP 没有超过 $n - L$ 的部分是无需检验的, 因为我们固定了前面 $n - L$ 个字符和 S 中的一致。
- 因此只需要比较超出部分, 也就是都是形如 $\text{LCP}(i, n - L)(1 \leq i \leq n - L)$ 的部分。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 至此可以得到一个 $O(n^2)$ 的算法，考虑枚举第 L 个后缀所对应的前缀，然后我们可以直接推出对应的字符串，然后使用扩展 KMP 算法来线性地得前 $n - L$ 个后缀和第 $n - L$ 个后缀的 LCP 长度，只需要和 S 比较一下就可以判定是否合法。
- 使用 *hash* 来对得到的字符串判重。最后答案记上前面提到的字母排列的影响。
- 考虑继续优化。
- 首先我们可以进一步减少需要填充的位数。
- 考虑 $Q = \max\{\text{LCP}(i, n - L) | 1 \leq i < n - L\}$ 。那么我们显然可以确定后 L 个字符中的前 $Q - 1$ 个。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 定理 7:
- 无论我们怎么填充剩余的没有确定的字符, $\forall 1 \leq i \leq n - L$, 新串中的 $\text{LCP}(i, n - L)$ 一定大于等于 S 中的 $\text{LCP}(i, n - L)$ 。
- 证明:
- 考虑 S 原本 $\text{LCP}(i, n - L)$ 和 Q 的关系, 分两种情况讨论:
 - 本来小于 Q 的, 在新串中没有和填充的未确定字符接壤, 因此一定和原来的 LCP 相等。
 - 本来等于 Q 的, 在新串中长度至少为 Q , 于是得证。
- 注意到上面的证明中只有本来等于 Q 的有可能大于原本的长度。因此, $\forall \text{LCP}(i, n - L) = Q (1 \leq i \leq n - L)$, 后 L 个字符的第 Q 个字符是一定不等于第 i 个前缀的第 $Q + 1$ 个字符的, 于是我们可以得到后 L 个字符的第 Q 个字符不能填的字符集合。

Equivalent Suffix Tries

CodeChef July Challenge 2012, EST

- 到了这里，可以发现我们已经能够 $O(1)$ 地判定一个枚举的对应前缀是否合法了。判断我们推算出的后 L 个字符的前 $Q - 1$ 个字符是否和我们枚举的前缀推算出的后 L 个字符的前 $Q - 1$ 个字符相同，可以利用扩展 KMP 算法加上周期的性质来完成，最后再比较一下第 Q 字符是否属于不能选择的字符集合。
- 同样，利用周期的性质我们也可以把哈希的时间复杂度降低到 $O(1)$ 。
- 于是本题线性解决。