字符串相关算法

洪华敦

2020 年 2 月 4 日

目录

基础算法

符号约定

- S 表示字符串，下标从 1 开始，$|S|$ 表示字符串长度，$S[i]$ 表示第 i 个字符
- c 表示一个字符
- AB 表示 A 和 B 直接拼起来
- $S[L:R]$ 表示从 L 到 R 的子串，$L=R+1$ 时表示空串
- S^k 表示 $S^{k-1}S$，$S^1=S$

基础算法

Hash

- 定义一个字符串到整数的映射函数，然后比较字符串时直接用数来比较
- $f(S) = \sum_{i=1}^{|S|} S[i]D^{|S|-i} \ mod \ P$
- 代码实现：$G[i] = G[i-1]D + S[i]$，那么 $f(S[L:R]) = G[R] - G[L-1]D^{R-L+1}$
- 一般取两个模数会稳一点
- <u>不要用 64 位自然溢出</u>

KMP

算法

- 一个字符串的 *Border* 是既为它的后缀又为它的前缀的字符串
- $Fail[i]$ 表示 $S[1:i]$ 的最大的 *Border*
- 求 $Fail[i]$ 时，检查 $S[Fail[i-1]+1]$ 是否等于 $S[i]$，否则检查 $S[Fail[Fail[i-1]]+1]$，以此类推
- 当进行字符串匹配时，也利用了类似的跳 Fail 的过程，由于每次跳 Fail 会使匹配的下标 -1，所以均摊复杂度为线性

KMP

周期

- D 为 S 的一个周期，当且仅当对于所有 $i \leq |S| - D$，有 $S[i] = S[i + D]$
- 若 V 是 S 的一个 Border，则有 $S[i] = S[i + |S| - V]$
- 所以 $|S| - Fail[|S|]$ 是 S 的一个周期
- 如果要判断 S 是否是一个循环串，只要判断 $|S| \% (|S| - Fail[|S|])$ 即可

KMP

关于周期的例题

- 给定一个无限循环小数的前 N 位，定义一个循环节的置信度为 $ap-bl$，其中 a,b 是给定的正整数系数，而 p 为这 N 位里包含该循环节的小数位数，l 为循环节长度，求所有可能的循环节中最大的置信度
- 例如 1.0285714285714 中，0285714285714 和 428571 都是可能的循环节
- $N \le 10^6$

KMP

例题思路

- 枚举 p，之后用 Fail 算出最小的循环节
- 倒着跑一遍 KMP 求出所有 Fail

KMP

RANK-KMP

- 定义两个序列 $A[1...N]$ 和 $B[1...N]$ 相似，当且仅当对于任意 i, j，$(A[i] < A[j]) = (B[i] < B[j])$
- 给定序列 X, Y，求 X 中有几个连续子序列和 Y 相似
- $|X|, |Y| \leq 10^6$

KMP
RANK-KMP

- 与 *KMP* 做法类似，若 A, B 相似，则 Ac 与 Bd 相似的条件是 c 在 A 中的排名与 d 在 B 中的排名相同
- 求 Fail 时，原版 KMP 是直接判断两个数相不相等，RANK-KMP 就利用数据结构判断两个数排名是否相等
- 时间复杂度：$O(N \log N)$
- 更简单但是不靠谱的做法：*Hash*

KMP

动物园

- 给定一个串 S，求每个前缀求出它有几个 Border 满足长度不超过这个前缀的一半
- $N \leq 10^6$

KMP

动物园

- 暴力就是一直跳 Fail，直到满足条件，但是遇到全 a 串会自闭
- 可以用倍增的方法，对每个 i 处理出跳了 2^j 次 Fail 后得到的数，然后通过倍增来求解
- 定义 Fail[x] 表示 x 在树上的父亲，实际上就是找一个最近的祖先使得满足长度限制，可以离线 *DFS* 时对祖先维护一个栈，然后在栈里二分，常数或许会小一点

KMP

周期的性质

- 对于一个串 S，如果它的最长 *Border* 长度为 L，且 $L \geq |S|/2$，则 S 可以写成 AB^K 的形式，且 $|B| = |S| - L$，A 是 B 的一个后缀，此时最长的 *Border* 是 AB^{K-1}
- 这种情况下，如果一直跳 Fail，则能得到的是 $AB^k, AB^{k-1}, AB^{k-2} ... AB$
- 所以如果我们要找 $S[1:i]$ 中最长的长度不超过一半的 *Border* 的话，如果 $fail[i] < i/2$ 则直接就找到了
- 否则就跳到 i%(i-fail[i])+(i-fail[i])，也就是从 AB^k 跳到 AB，这样可以证明每次至少去掉了 $\frac{1}{3}$ 的长度，所以复杂度是 $O(1)$ 的

KMP

树上的 KMP

- 给定一棵 Trie，求每个点到根的路径表示的字符串的最长 Border
- $N \leq 10^6$，字符集还蛮大的

KMP
做法 1

- 维护一下 $go[x][c]$ 表示 x 代表的串后面加上 c 后的 $fail$ 是啥
- 那么 $fail[x] = go[fa[x]][S_x]$
- 而 $go[x]$ 与 $go[fail[x]]$ 之间的区别就只有一个位置，也就是 $fail[x]$ 到 x 的路径上的第一个字符
- 复杂度应该是 $O(N \log N)$

KMP

做法 2

- 暴力跳 Fail 显然是 $O(N^2)$ 的，但是我们考虑一下如果现在的串是 AB^k，那么在他跳到 AB 的过程中，匹配的全是 $B[1]$，所以如果 AB^{k-1} 失配了，就直接跳到 AB，这样复杂度也是 $O(\log N)$

KMP

扩展内容

- 关于 Fail 的更多扩展内容可以参考 sone2 的解题报告

KMP

CF1286E

- 给定一个字符串 S 和权值数组 W
- 定义 S 的一个子串是好的，当且仅当这个子串等于 S 的某个前缀
- 一个子串 $S[L:R]$ 的权值是 $W[L...R]$ 的最小值
- 对于 S 的每个前缀，求他的所有好的子串的权值之和
- $N \leq 10^5$

KMP

CF1286E

- 将答案差分一下，变成询问所有好的后缀的权值之和，也就是所有 *Border* 的权值之和，我们用某种方法维护一下这些 *Border*
- 考虑加入一个字符，那么有些 *Border* 会被扔掉，这个扔的过程是均摊 $O(n)$ 的
- 对于剩下的后缀，它们的权值相当于要对于一个数取 min，这个可以用数据结构维护

AC 自动机

- Trie 树上的 KMP，Fail 不仅限于祖先，也可以到 Trie 上的其他结点上，常用于多模式串匹配
- 求 Fail 的方法与树上 KMP 差不多，用一个 $go[x][c]$ 表示 x 后面加上 c 后会跳到哪之类的

AC 自动机例题

- 给定 n 个字符串 $T_{1\ldots n}$，求有多少长度为 L 的字符串 S，使得任何一个 T_i 都不是 S 的连续子串
- $n, |T_i| \leq 50$，$L \leq 1000$

exKMP

- 对于两个字符串 S, T，线性时间内求出 S 的每个后缀和 T 的最长公共前缀
- 算法步骤是先对每个 $S[i:|S|]$ 求出和 S 的最长公共前缀，假设为 $p[i]$，之后再利用 $p[i]$ 求出每个 $S[i:|S|]$ 和 T 的最长公共前缀，设为 $ex[i]$
- 算法的过程和 Manacher 类似

exKMP

求 $p[i]$

- 假设我们已经求出了 $p[1...i-1]$，现在要求 $p[i]$
- 设 D 是 $i+p[i]-1$ 的最大值，满足这样条件的 i 我们设为 i_D
- 则 $S[1:p[i]]=S[i_D:D]$，所以 $S[i-i_D+1:p[i]]=S[i:D]$
- 那么有 $min(D-i+1,p[i-i_D+1])\le p[i]$
- 然后暴力扩展一下，每次暴力会使得 D 的值变大，所以复杂度也是线性的

exKMP

求 $ex[i]$

- 和求 $p[i]$ 似乎也没什么区别

后缀数据结构

- 后缀数组
- 后缀自动机（在大多数题目中可以替代后缀数组）

定义

- 两个字符串 A, B 的大小比较一般是采用字典序的方式，先比第一个，再比第二个，以此类推，如果 A 是 B 的前缀那么 $A < B$
- 后缀数组是处理字符串的基本工具，它的主要思路是将所有后缀排好序，然后去做各种各样的操作

后缀排序

- 最简单的方法：直接 sort，然后 cmp 时用二分 +Hash
- 考虑倍增的方法，我们对每个后缀只看前 2^K 个字符，假设已经排好了，考虑一下如何扩展到只看前 2^{K+1} 个字符
- 这是个简单的基数排序

H

- 设 $Rank[i]$ 表示 $S[i:|S|]$ 的排名，$Sa[i]$ 表示排名为 i 的后缀的下标
- $H[i]$ 表示 $Sa[i]$ 与 $Sa[i+1]$ 的最长公共前缀 (LCP)
- 我们有 $LCP(Sa[X], Sa[Y]) = Min_{i=X}^{Y-1} H[i]$
- 只要能在较快的时间内求出 $Height$，之后就可以通过 RMQ 去计算任意两个后缀的 LCP

H

- 对于任何 i，我们有 $H[rank[i]] - 1 \le H[rank[i+1]]$
- 所以我们枚举 $i = 1...n$，然后去暴力计算 $H[rank[i]]$

基本操作

- 求 $LCP(Suf[x], Suf[y])$，转换成 RMQ 问题
- 求 $S[L:R]$ 的出现次数
- 求本质不同的子串个数
- 求字符串的最小循环表示
- 求最长的出现了至少 K 次的子串
- 求最长的出现了至少 K 次的子串，要求这 K 次互不重叠
- 求 S, T 的 LCS
- 求第 K 小子串

基本操作 1

- 求 $S[L:R]$ 的出现次数
- 等价于求有多少 X 满足 $lcp(X, L) \geq R - L + 1$
- 这样的 X 在后缀数组里肯定是一个连续的区间，二分一下左右端点即可

基本操作 2

- 求本质不同的子串个数
- 子串就是后缀的前缀
- 一个子串可能是多个后缀的前缀，为了不算重我们规定在最小的后缀里算到他
- 那么对于 $Suf[x]$ 有多少个子串需要被他算到呢?
- $n - x + 1 - H[rank[x] - 1]$
- 答案就是 $\frac{n(n+1)}{2} - \sum_{i=1}^{n-1} H[i]$

基本操作 3

- 求字符串的最小循环表示
- 每个循环表示都是 SS 的一个长度为 $|S|$ 的子串，相当于求最小的长度为 $|S|$ 的子串
- 从 $Suf_1 ... Suf_{|S|}$ 中选出一个最小的后缀即可

基本操作 4

- 求最长的出现了至少 K 次的子串
- 一个子串所在的后缀在后缀数组中一定是一个区间
- 所以答案就是，选一段长度为 K 的连续区间，使得 H 的最小值最大
- 直接 RMQ 即可

基本操作 5

- 求最长的出现了至少 K 次的子串，要求互不重叠
- 我们可以二分答案 L，现在要求的就是长度为 L 的串最多互不重叠地出现了多少次
- 相当于选出尽量多的位置 $P_{1...M}$，使得任意两个 P_i 的差大于等于 L，且 $lcp(P_i, P_j) \geq L$
- 条件 2：我们将 $H[i] < L$ 的所有空隙切断，剩下的在同一段内的都满足这个条件
- 条件 1：对于每一段，我们从左往右能选就选

基本操作 6

- 求 S, T 的 LCS
- 拼出一个新的串 $D = S\#T$
- 相当于从属于 S 的部分选出一个后缀，然后从属于 T 的部分选出一个后缀，使得它们的 LCP 最大
- 求出后缀数组，找前一个和后一个

基本操作 7

- 求第 K 小子串
- 我们知道对于 $Suf[x]$，有 $n-x+1-H[rank[x]-1]$ 个串被它算到
- 我们按照 $Suf[x]$ 从小到大，然后长度也从小到大的顺序遍历这些串，得到的就是从小到大遍历所有子串的效果

品酒大会

- 给定字符串 S 以及权值数组 $a[1...n]$，对于每个 r，求满足 $LCP(Suf_x, Suf_y) \geq r$ 的 (x, y) 中 $a[x]a[y]$ 的最大值
- $n \leq 3 \times 10^5$

品酒大会

- 因为 Suf_x, Suf_y 的 LCP 对应的是一个区间的 H 的最小值，我们可以考虑枚举这个最小值，例如是 H_i
- 考虑找到左边第一个比 H_i 小的 H_L，以及右边第一个比 H_i 小的 H_R，那么对于 $(L, R]$ 中的 X, Y，他们的 LCP 都大于等于 H_i
- 维护个个区间最大值最小值之类的东西就行了

差异

- 求 $\sum_{i=1}^{n} \sum_{j=i+1}^{n} LCP(Suf_i, Suf_j)$
- $n \leq 5 \times 10^5$

差异

- 在求出 H 后，相当于求所有子区间的最小值的和
- 对每个 H_i 求出左边第一个比他小的和右边第一个比他小的，就能知道有多少个区间的最小值等于他了

优秀的拆分

- 对于一个串 S，定义一个优秀的拆分是将一个串表示成 AABB 的形式，求 S 所有子串的优秀的拆分的拆分个数之和
- $|S| \leq 2 \times 10^5$

优秀的拆分

- 等价于对每个 i 求出 Pre_i 和 Suf_i，表示 $S[1:i]$ 的 AA 型后缀个数以及 $S[i:|S|]$ 的 AA 型前缀个数，那么答案显然就是 $\sum_{i=1}^{n} Pre_i Suf_{i+1}$
- 枚举一下 $|A|$，把 S 按 $|A|$ 切段，对每两段去讨论一下，可以发现只要求 lcp 就行了

缺位匹配

- 给定串 S, T，求 S 有几个子串满足改动不超过 K 个字符就可以和 T 相等
- $|S|, |T| \leq 10^5$， $k \leq 20$

缺位匹配

- 枚举 S 的每个长度为 $|T|$ 的子串，每次贪心地往后匹配，可以发现就是进行 K 次 LCP 询问

BZOJ4310

- 给定一个字符串 S，你需要将它分成不超过 m 个连续子串，使得分割后的所有串的子串中字典序最大的尽量小
- $|S| \leq 10^5$

BZOJ4310

- 考虑二分答案，可以二分是第 K 大的子串，然后用 SA 求出
- 从后往前，只要不需要切断，就不切断，切断的判断很容易用判 LCP 实现
- 最后 check 断点是否不超过 m 个即可

总结

- H 数组是后缀数组的灵魂，后缀数组能有这么多功能都是因为他快速求出了 H 数组
- 充分地理解 H 数组的含义能帮你更好地理解后缀数组的各种套路

后缀自动机

- 理解后缀自动机的三步:
- 1. 理解 n^2 版的后缀树，利用该后缀树的性质去思考题目
- 2. 理解增量构造后缀自动机的过程，以及 go 数组的应用
- 3. 将边压缩，得到真正的后缀自动机

后缀树

- 对于一个字符串 S，将它的所有后缀插入 $Trie$ 中后可以得到后缀树
- 插入每个后缀后所得到的那个最终的点我们称为后缀点，表示为 $Key[x]$
- 后缀树上每个结点都对应了恰好一个子串，每个子串也恰好对应到了一个结点
- 后缀数组其实是后缀树的 DFS 序

祖先关系

- 考虑后缀树上点 x 是点 y 的祖先
- 那么 x 代表的串就是 y 代表的串的前缀
- x 的出现次数 = 它是几个后缀的前缀 = 子树里有几个后缀点
- x 在 L 出现 = 它是 $Suf[L]$ 的前缀 = 它是 $key[L]$ 的祖先

LCP 的新姿势

- 考虑如何求 $LCP(Suf_x, Suf_y)$
- 最长公共前缀 = 找一个最长的串，使得它是这两个后缀的前缀
- 等价于找一个最长的串，使得他在后缀树上是 $key[x]$ 和 $key[y]$ 的祖先
- 等价于 $LCA(key[x], key[y])$

基本操作

- 本质不同的子串个数 = 后缀树的结点个数
- 求最长的出现了超过 K 次的子串：计算出每个结点的出现次数即可
- 求最长的不重叠地出现了超过 K 次的子串：与后缀数组做法类似
- 求最长公共子串：拼在一起求后缀树，然后 check 每个结点
- 求第 K 小的子串：显然按出边从小到大 DFS 一遍就是从小到大遍历子串

后缀自动机

- 对于一棵后缀树，如下定义一些东西：
- $len[x]$： x 代表的串的长度，也就是 x 的深度
- $Fail[x]$： x 的父亲
- 为了引入后缀自动机，我们引入后缀自动机的辅助数组：
- $go[x][c]$： x 代表的串往前加一个字符 c 后得到的结点
- 注意为了契合后缀树，本文介绍后缀自动机时是按照反方向来的，与传统认知中的后缀自动机前后可能相反

增量构造后缀自动机

- 考虑我们已经整出了 S 的后缀自动机，如何整出 cS 的后缀自动机
- 第一步：给 cS 建一个结点，然后给它在后缀树上找到父亲，它的父亲可能不存在，需要我们自己建
- 第二步：更新 go[x]

增量构造后缀自动机

找父亲

- 它的祖先一定长成 cS[1:T] 的样子，我们找到一个它已经存在的祖先，然后把剩下的祖先给建出来
- cS[1:T] 等于 go[S[1:T]][c]
- S[1:T] 是 S 的祖先
- 所以算法很明显了：从 S 出发一直往父亲跳，直到 go[x][c] 有值，设 Y=go[x][c]
- 则 cS 的加入导致后缀树从 Y 分叉出了一条链

增量构造后缀自动机

更新 go[x]

- 对于新加入的这些结点，它们的 go 不需要更新，因为它们之前不在 S 中，那前面加了字符后更不可能在
- 对于 S 到 Y 上的这条链，本来它们的 go[x][c] 是没值的，显然现在它们有了
- 做完了！

后缀树的压缩

- 接下来考虑重头戏：后缀树的压缩
- 我们发现虽然后缀树有 $O(n^2)$ 个结点，但它只有 $O(n)$ 个叶子，这显然是不太合理的
- 我们知道，如果一棵树没有度数为 1 的点的话，那么结点个数和叶子个数是同一个数量级的
- 若一个点 x 只有一个儿子，那么我们就将 x 并到它的儿子那里去，我们称这些点为被压缩点
- 同时：由于后缀点记录了很多关键信息，我们规定所有的后缀点不能被并走
- 我们称最后留下来的点为关键点

后缀树的压缩

信息记录的含义的变化

- 既然后缀树被压缩了，那么我们之前记录的信息的含义也会有变化
- x 代表的串：从一个串变成了一堆串，且这些串的形式是某个串的一系列前缀，我们设这个集合为 $Z(x)$
- $len[x]$：表示点 x 代表的最长的串的长度
- $fail[x]$：表示点 x 往上爬的第一个关键点
- $go[x][c]$：这个的表示含义改变很大，需要仔细分析

后缀树的压缩

性质

- 若 x 是被压缩点，则原树的 $go[x][c]$ 一定也是被压缩点
- 证明:
- 设 x 代表的串为 T，假设 cT 是关键点
- 若 cT 是后缀点，则显然 T 是后缀点，矛盾
- 否则 cT 有至少两个儿子，设为 cTx，cTy，则 T 也应该有儿子 Tx，Ty，矛盾
- 这条性质表明了：被压缩点的信息可以由关键点推出来，所以他是可以被压缩的

边的压缩

性质

- 但是若 x 是关键点，$go[x][c]$ 却也可能是被压缩点，这种情况我们令 $go[x][c]$ 变成 $go[x][c]$ 压缩到的那个点中
- 那么我们保证了对于 $Y \in Z(x)$，$cY \in Z(go[x][c])$

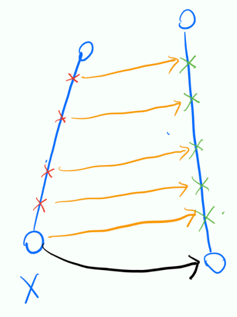

图: go[x][c] 的图示

后缀树的压缩

构造

- 定义完压缩的后缀树后，我们回过头来看看增量构造的过程
- 1. 建立点 cS 以及它的祖先
- 2. 更新其他点的 go[x]

后缀树的压缩

建立 cS

- 考虑 cS 的建立，原本的思路是找到最长的 S[1:T] 使得 go[S[1:T]][c] 有值，我们可以沿用这个思路，找到 S 的最近的祖先 Q，使得 go[Q][c] 有值，设 E=go[Q][c]
- 我们知道 $cS[1:T] \in Z(E)$，它可能是边上的一个点，所以我们需要把它拆出来
- 如何判断这个点是否在边上呢？判断 len[Q]+1 是否等于 len[E]
- 拆出来的新点假设为 K，那么 fail[K]=fail[E]，fail[cS]=fail[E]=K，且 len[K]=len[Q]+1
- 到这里 cS 就建立好了

后缀树的压缩

更新其他点的 go[x]

- 首先我们更新 go[K]，虽然点 K 翻身了，但是本来 E 代表的边 go 映射过去依然是一条边，所以 go[K] 整个就等于 go[E]
- 然后我们考虑 Q，本来 go[Q][c]=E 就是一个权宜之计，现在我们有了点 K，所以 go[Q][c]=K
- 特别地，考虑 Q 的所有祖先 x，如果 go[x][c]=E 的话就令 go[x][c]=K
- 然后考虑 S 到 Q 的这条链，他们的 go[x][c] 本来没值，现在有了，就是点 cS

后缀树的压缩

代码

```
void add(int c,int pos){
    int Q=S;int cS=++tot;
    len[cS]=len[S]+1;    //更新好 len[cS]
    S=cS;                //应该在return前写, 这里偷懒先写好
    for(;Q&&(!go[Q][c]);Q=fail[Q])
        go[Q][c]=cS;     //找 Q, 顺便把 (S,Q) 上的go[x][c]更新了
    if(!Q){
        fail[cS]=1;      //莫得 Q, fail[cS] 直接指向根
        return;
    }
    int E=go[Q][c];
    if(len[Q]+1==len[E]){
        fail[cS]=E;      //go[Q][c] 指向的就是 E, 直接认爹就行了
        return;
    }
    int K=++tot;
    len[K]=len[Q]+1;     //从边 (E,fail[E]) 分裂出点 K
    fail[K]=fail[E];
    fail[E]=fail[cS]=K; //处理好因为分裂了边导致的 fail 变动
    rep(i,0,25)
        go[K][i]=go[E][i];//go[K]=go[E]
    for(;Q&&go[Q][c]==E;Q=fail[Q])
        go[Q][c]=K;      //重新定向 go[Q][c]
}
```

图: 代码实现

时间复杂度

- 最后点和边都是线性的
- 但是实现时因为要开数组，所以可以认为是 $O(n|c|)$
- 证明： 显然法

基本操作

- 我们回头看看 go[x] 数组，可以发现一个有用的性质：
- 如果 $S \in Z(x)$，那么 $cS \in Z(go[x][c])$
- 也就是说，要找到代表 T 的结点很简单，从根出发，按 T 的值去一直走 go[x][c] 即可
- 现在我们回头看看那些基本操作怎么整

基本操作

- 本质不同的子串个数
- 一个串的出现次数
- S[L:R] 定位
- 求最小循环串
- 求长度为 K 的字典序最小的子串
- 求 S,T 有几对子串相等
- 给定 S,T，求 LCS
- 最长的出现了至少 K 次的子串
- 求第 K 小的子串，要求 $O(Qn)$.（EX: 要求 $O(Q\log K)$）

做之前的题

- 品酒大会
- 差异

例题 1

- 设 $f(i)$ 为所有长度为 i 的子串中出现次数的最大值，求 $f(1)...f(|S|)$
- $|S| \leq 250000$，要求线性

例题 2

- 维护一个串 S，支持往后加 a，以及求 T 在 S 中的出现次数
- 离线/在线，$Q \leq 10^5$

例题 3

- 给定 n 个字符串，求有几个不同的字符串是至少 K 个串的子串
- $\sum |S_i| \le 10^5$，$n, k \le 100$

例题 4

- 给定串 S，对于一个位置 K，定义 S[L:R] 是它的识别子串，当且仅当 $L \le K \le R$，且 S[L:R] 只出现了一次
- 对于每个位置，求它最短的识别子串
- $|S| \le 10^5$

例题 5

- 给定串 T，对于一个串 S，定义它的距离为至少需要用几个 T 的子串拼起来，拼不出来就是 -1
- 求长度为 N 的距离最大的串的距离
- $N \leq 10^{18}, 1 \leq |T| \leq 10^5$，字符集为 4
- 提示：二分答案

例题 6

- 给定字符串 S
- 对于每个 k，求把 S[k] 变成 # 后，本质不同 s 的子串个数
- $1 \le |S| \le 10^5$

例题 7

- 给定字符串 S,T
- 从 S 中选出子串 s，从 T 中选出子串 t，组成新的串 st，求能组成的串的个数
- $1 \le |S|, |T| \le 10^5$

例题 8

- 给定串 S，每次询问给定 T,L,R，求 T 有几个不同的子串不是 S[L:R] 的子串
- $|S| \leq 5 \times 10^5$
- $Q \leq 10^5$
- $\sum |T| \leq 10^6$
- NOI2018 《你的名字》

例题 9

- 给定串 S，每次询问 $S[a:b]$ 中和 $S[c:d]$ LCP 最大的子串，求这个 LCP 的值
- $1 \le |S| \le 10^5$

回文自动机

- 回文自动机和后缀自动机非常类似，主要区别如下：
- 因为回文串的性质，所以 go[S][c] 表示的是 cSc
- 因为本质不同的回文串只有 $O(n)$ 个，所以回文自动机没有麻烦得要死的压缩
- fail[S] 的含义和后缀自动机一样，表示的是 S 最大的回文后缀

增量法

- 假设 S 最大的回文后缀为 W，现在往后添加了字符 c
- 相当于要找一个 W 的最长后缀 D，使得 cDc 是 Sc 的回文后缀，显然 D 也是个回文串
- 暴力枚举 W 的 fail 找到 D
- 之后再暴力找到 D 的最长后缀 P，使得 cPc 是 Sc 的回文后缀，令 fail[cDc]=fail[cPc]
- 令 go[D][c]=cDc 即可

应用

- 动态维护一个串的不同的回文子串个数，也就是结点个数
- 回文树 DP
- 区间本质回文子串个数

回文树 DP

- 给定 S，从中选出 k 个互不重叠的回文子串，使得长度和最长
- $|S| \leq 10^5$，$k \leq 20$
- 概括成这种形式 $F[R] = \max_L F[L-1] + W(L, R)$，且 $S[L:R]$ 是个回文串

回文树 DP

- 结论：一个回文串的回文后缀一定是它的一个 Border，且它的所有 Border 都是回文串
- 而一个串的 Border 可以分成 $O(\log n)$ 组等差数列，每组是 $AB...AB^k$ 的形式

回文树 DP

- 我们挑一组等差数列来看
- $L_1....L_m$，公差为 d
- 首先考虑最大的 L_1，可得 L_2 是 $i-d$ 的回文后缀
- 所以在 i-d 时，这组等差数列中的 $L_2...L_m$ 为它提供了贡献
- $i-d-L_2....i-d-L_m$ 等价于 $i-L_1....i-L_{m-1}$
- 所以这组等差数列对 i 的贡献就是之前对 $i-d$ 的贡献再加上了 $i-L_m$
- idea 有了，想个办法维护维护就好了

本质不同回文子串计数

- 与上面的回文树 DP 类似，考虑枚举 $R = 1...n$
- 对于一个长度为 D 的回文后缀，对 $L \leq R - D + 1$ 都有贡献
- 由上面的做法可知，每个等差数列只会增加 $R - L_m + 1$ 的贡献，用线段树维护一下单点加就好了，甚至可以树状数组?