字符串基础

TQX

2024 年 8 月 13 日

基础知识

字符集

用 Σ 表示字符集，它是一个建立了全序关系的集合，也就是任意两个不同元素都可以比较大小，其中的元素称为字符。一般可以认为字符集是一些符号的集合，大小关系为字典序。

基础知识

字符集

用 Σ 表示字符集，它是一个建立了全序关系的集合，也就是任意两个不同元素都可以比较大小，其中的元素称为字符。一般可以认为字符集是一些符号的集合，大小关系为字典序。

字符串

Σ 上的字符串 s 是 Σ 中的字符组成的序列。$|s|$ 表示 s 的长度。$s[i]/s_i$ 表示 s 的第 i 个字符。Σ 上的所有字符串集合写作 Σ^*。

基础知识

字符串 s 和 t 的连接是 t 接在 s 后面得到的字符串，写作 st。k 个串 s 相连写作 s^k。

基础知识

字符串 s 和 t 的连接是 t 接在 s 后面得到的字符串，写作 st。k 个串 s 相连写作 s^k。

如果存在串 u 和 v 使 $t = usv$，则称 s 是 t 的子串。也可用下标表示：$s = t_{i...j} = t_i t_{i+1} \cdots t_j$。

基础知识

字符串 s 和 t 的连接是 t 接在 s 后面得到的字符串，写作 st。k 个串 s 相连写作 s^k。

如果存在串 u 和 v 使 $t = usv$，则称 s 是 t 的子串。也可用下标表示：$s = t_{i...j} = t_i t_{i+1} \cdots t_j$。

如果存在串 u 使 $t = su$，则称 s 是 t 的前缀。如果 u 非空，则称 s 是 t 的真前缀。t 的长为 x 的前缀记作 $pre(t, x)$。

如果存在串 u 使 $t = us$，则称 s 是 t 的后缀。如果 u 非空，则称 s 是 t 的真后缀。t 的长为 x 的后缀记作 $suf(t, x)$。

基础知识

通过字典序可以将 Σ^* 中的所有字符串排序。首先需要有一个 Σ 中所有字符的大小顺序（比如 $a < b < \cdots < z$），定义 $s < t$ 当且仅当 s 是 t 的真前缀，或者存在 i 使 $s_{1\cdots i-1} = t_{1\cdots i-1}$ 且 $s_i < t_i$。

基础知识

通过字典序可以将 Σ^* 中的所有字符串排序。首先需要有一个 Σ 中所有字符的大小顺序（比如 $a < b < \cdots < z$），定义 $s < t$ 当且仅当 s 是 t 的真前缀，或者存在 i 使 $s_{1\cdots i-1} = t_{1\cdots i-1}$ 且 $s_i < t_i$。

串 s 反过来写，得到它的反串，用 s^R 表示。如果 $s = s^R$，则称 s 为回文串。

基础知识

通过字典序可以将 Σ^* 中的所有字符串排序。首先需要有一个 Σ 中所有字符的大小顺序（比如 $a < b < \cdots < z$），定义 $s < t$ 当且仅当 s 是 t 的真前缀，或者存在 i 使 $s_{1\cdots i-1} = t_{1\cdots i-1}$ 且 $s_i < t_i$。

串 s 反过来写，得到它的反串，用 s^R 表示。如果 $s = s^R$，则称 s 为回文串。

对于整数 $p = 1, 2, \cdots, |s|$，如果对任意 $i \in [1, |s|-p]$ 都有 $s_i = s_{i+p}$，就称 p 是 s 的周期。

对于整数 $r = 0, 1, \ldots, |s|-1$，若 $pre(s, r) = suf(s, r)$，则称 $pre(s, r)$ 是 s 的 border。

如果存在串 u 和 v 使 $s = uv$ 且 $t = vu$，则称 s 是 t 的循环移位，或称 s 和 t 循环同构。

字符串哈希

字符串哈希是一种通过算术运算将字符串映射到数的方法。我们希望通过哈希函数将字符串映射到一个值域较小、可以方便比较的范围，来便捷地判断两个字符串是否相同。

设哈希函数为 h，那么对于字符串 s 和 t，如果 $h(s) = h(t)$，就认为 $s = t$。

字符串哈希

字符串哈希是一种通过算术运算将字符串映射到数的方法。我们希望通过哈希函数将字符串映射到一个值域较小、可以方便比较的范围，来便捷地判断两个字符串是否相同。

设哈希函数为 h，那么对于字符串 s 和 t，如果 $h(s) = h(t)$，就认为 $s = t$。

而考虑真实答案时，$h(s) \neq h(t)$，则 s 一定不等于 t；但 $h(s) = h(t)$ 时 s 可能不等于 t，此时称为哈希冲突。在字符串哈希中，一般会假定哈希值互不相同，这样如果发生冲突，可能导致答案错误。

字符串哈希

字符串哈希是一种通过算术运算将字符串映射到数的方法。我们希望通过哈希函数将字符串映射到一个值域较小、可以方便比较的范围，来便捷地判断两个字符串是否相同。

设哈希函数为 h，那么对于字符串 s 和 t，如果 $h(s) = h(t)$，就认为 $s = t$。

而考虑真实答案时，$h(s) \neq h(t)$，则 s 一定不等于 t；但 $h(s) = h(t)$ 时 s 可能不等于 t，此时称为哈希冲突。在字符串哈希中，一般会假定哈希值互不相同，这样如果发生冲突，可能导致答案错误。

通常采用多项式 Hash 的方法做字符串哈希：对于长度为 l 的字符串，定义 $h(s) = \sum_{i=1}^{l} s_i \cdot base^{l-1} \pmod{M}$。其中 $base$ 和 M 是自己选择的数字，通常 base 取一些小质数，M 取一些大质数。

使用多项式 Hash 的好处在于预处理出字符串每个前缀的 hash 值，以及 $base^k$ 的 hash 值后，可以 $\mathcal{O}(1)$ 计算任意子串的 hash 值：
$h(s_{l,r}) = pre_r - pre_{l-1} \cdot base^{r-l+1}$。

错误率分析

可以近似的认为此 hash 函数会在 $0 \sim M-1$ 间等概率生成，设要计算的字符串量为 n，那么 hash 冲突（存在相同 hash 值）的概率为：

$$
\begin{aligned}
1-\prod_{i=1}^{n-1}(1-\frac{i}{M}) &\approx 1-\prod_{i=1}^{n-1} e^{\frac{-i}{M}} \\
&= 1-\exp(-\sum_{i=1}^{n-1}\frac{i}{M}) \\
&= 1-\exp(-\frac{n(n-1)}{2M})
\end{aligned}
$$

约等号：根据 e^x 的泰勒展开，当 $|x|$ 足够小时，$e^{-x} \approx 1-x$。

错误率分析

可以近似的认为此 hash 函数会在 $0 \sim M-1$ 间等概率生成，设要计算的字符串量为 n，那么 hash 冲突（存在相同 hash 值）的概率为：

$$
\begin{aligned}
1-\prod_{i=1}^{n-1}(1-\frac{i}{M}) &\approx 1-\prod_{i=1}^{n-1} e^{\frac{-i}{M}} \\
&= 1-\exp(-\sum_{i=1}^{n-1}\frac{i}{M}) \\
&= 1-\exp(-\frac{n(n-1)}{2M})
\end{aligned}
$$

约等号：根据 e^x 的泰勒展开，当 $|x|$ 足够小时，$e^{-x} \approx 1-x$。

$n=10^5, M=10^9$: 99.3%, 所以说单哈希 10^9+7 模数不需要卡就容易自己挂掉。
$n=10^5, M=2^{64}$: $\frac{n(n-1)}{2M}$ 足够小，小于 10^{-10}。这是一般使用的自然溢出哈希，直接使用 unsigned long long 维护哈希值，一般情况下可以认为不会冲突，但可以被针对性卡：$s_1=abaab$，$!s_i$ 表示将 s_i 中所有字符反转，构造 $s_i=s_{i-1}+!s_{i-1}$。s_{12} 和 $!s_{12}$ 就会得到相同哈希值。

错误率分析

可以近似的认为此 hash 函数会在 $0 \sim M-1$ 间等概率生成，设要计算的字符串量为 n，那么 hash 冲突（存在相同 hash 值）的概率为：

$$
\begin{aligned}
1-\prod_{i=1}^{n-1}(1-\frac{i}{M}) &\approx 1-\prod_{i=1}^{n-1} e^{\frac{-i}{M}} \\
&= 1-\exp(-\sum_{i=1}^{n-1}\frac{i}{M}) \\
&= 1-\exp(-\frac{n(n-1)}{2M})
\end{aligned}
$$

约等号：根据 e^x 的泰勒展开，当 $|x|$ 足够小时，$e^{-x} \approx 1-x$。

$n=10^5, M=10^9$: 99.3%, 所以说单哈希 10^9+7 模数不需要卡就容易自己挂掉。
$n=10^5, M=2^{64}$: $\frac{n(n-1)}{2M}$ 足够小，小于 10^{-10}。这是一般使用的自然溢出哈希，直接使用 unsigned long long 维护哈希值，一般情况下可以认为不会冲突，但可以被针对性卡：$s_1=abaab$，$!s_i$ 表示将 s_i 中所有字符反转，构造 $s_i=s_{i-1}+!s_{i-1}$。s_{12} 和 $!s_{12}$ 就会得到相同哈希值。
实际实现时，通常使用双哈希，即维护两组 $(base, M)$，要求两个哈希函数结果都相同才认为两个字符串相同。

loj 10035. Power Strings

Description

给定字符串 s，询问 s 最多是由多少个相同的字符串重复连接而成的。如：$ababab$ 最多由 3 个 ab 组成。$|s| \leq 10^6$

loj 10035. Power Strings

Description

给定字符串 s，询问 s 最多是由多少个相同的字符串重复连接而成的。如：$ababab$ 最多由 3 个 ab 组成。$|s| \leq 10^6$

答案是真的调和级数暴力枚举子串，哈希判断就能过。

loj 6287. 诗歌

Description

一个 $1 \sim n$ 的排列 h, 问是否存在三元组 (i, j, k) 满足 $1 \leq i < j < k \leq n$ 且 $h_i - h_j = h_j - h_k$。$n \leq 3 \times 10^5$。

loj 6287. 诗歌

Description

一个 $1 \sim n$ 的排列 h, 问是否存在三元组 (i, j, k) 满足 $1 \leq i < j < k \leq n$ 且 $h_i - h_j = h_j - h_k$。$n \leq 3 \times 10^5$。

枚举中间点 j，转化为判断是否存在 m 使得 $h_j + m$ 和 $h_j - m$ 在 j 两侧。

用一个 01 字符串表示每个数字是否在 j 左侧，那么只要以 h_j 为中心的 len 最大的字符串不是回文子串就有解。回文子串可以维护正串与反串的哈希值判断。

枚举 j 相当于单点修改，查询区间哈希值，线段树维护。

洛谷 P5537 【XR-3】系统设计

Description

小 X 需要你设计一个系统。
这个系统首先需要输入一棵 n 个点的有根树和一个长度为 m 的序列 a，接下来需要实现 q 个操作。
操作分两种：

- ‘1 x l r‘ 表示设定起点为有根树的节点 x，接下来依次遍历 $l \sim r$。当遍历到 i 时，从当前节点走向它的编号第 a_i 小的儿子。如果某一时刻当前节点的儿子个数小于 a_i，或者已经遍历完 $l \sim r$，则在这个点停住，并输出这个点的编号，同时停止遍历。
- ‘2 t k‘ 表示将序列中第 t 个数 a_t 修改为 k。

$n, m, q \leq 5 \times 10^5$

洛谷 P5537 【XR-3】系统设计

Description

小 X 需要你设计一个系统。
这个系统首先需要输入一棵 n 个点的有根树和一个长度为 m 的序列 a，接下来需要实现 q 个操作。
操作分两种：

- '1 x l r' 表示设定起点为有根树的节点 x，接下来依次遍历 $l \sim r$。当遍历到 i 时，从当前节点走向它的编号第 a_i 小的儿子。如果某一时刻当前节点的儿子个数小于 a_i，或者已经遍历完 $l \sim r$，则在这个点停住，并输出这个点的编号，同时停止遍历。
- '2 t k' 表示将序列中第 t 个数 a_t 修改为 k。

$n, m, q \leq 5 \times 10^5$

树的形态是不变的，将到父亲的边顺序作为这个节点的字符。问题转化为找最长的 $[l, r]$ 前缀满足是从 x 到子孙的一条链。

二分序列长度，查询的 hash 值确定，考虑维护树上链的 hash 值。多项式 hash 可减，因此只需要查询是否有 x 的子孙的 hash 值为特定值，因为假设不会冲突所以只需要维护一个哈希表。序列哈希那边是单点修改区间查询，线段树维护。

CF1063F String Journey

Description

给定字符串 S，构造有序非空字符串 $t_1, t_2, \dots t_k$ 和可以为空的字符串 $u_1, u_2, \dots u_{k+1}$，使 $S = u_1 t_1 u_2 t_2 \dots t_k u_{k+1}$，且 k 最大。求最大的 k。有序是指 t_i 是 t_{i-1} 的真子串且 $t_i \neq t_{i-1}$。$|S| \leq 50000$。

CF1063F String Journey

Description

给定字符串 S，构造有序非空字符串 $t_1, t_2, \dots t_k$ 和可以为空的字符串 $u_1, u_2, \dots u_{k+1}$，使 $S = u_1 t_1 u_2 t_2 \dots t_k u_{k+1}$，且 k 最大。求最大的 k。有序是指 t_i 是 t_{i-1} 的真子串且 $t_i \neq t_{i-1}$。$|S| \leq 50000$。

记 $n = |S|$，答案不超过 $\mathcal{O}(\sqrt{n})$，考虑求出 f_i 表示当前答案的 t_1 以 s_i 作为开头时的最大答案。
从后往前枚举，注意到 $f_i \leq f_{i+1} + 1$，因此从 $f_{i+1} + 1$ 开始从大到小枚举 len 进行判断，当前长度 len 合法当且仅当 $s[i, i + len - 2]$ 或者 $s[i + 1, i + len - 1]$ 在 $i + len - 1$ 后出现过且是一个合法的开始串。用哈希表动态维护所有合法开始串的哈希值即可，对于在 $i + len - 1$ 后出现这个限制，因为 $i + len - 1$ 是单调不增的，所以在每次 len 减少时加入原 $i + len - 1$ 开头的合法子串。

复杂度 $\mathcal{O}(n\sqrt{n})$，实际上可以通过原题的 $5e5$ 数据范围。

KMP

对于序列上的字符串 $s_{1,n}, i \in [1, n]$，定义 $fail_i$ 表示前缀 pre_i 的最长 border 长度。KMP 算法可以 $\mathcal{O}(n)$ 求解字符串的所有 $fail$。
关键性质：s 的所有 border 长度：$fail[n], fail[fail[n]], fail[fail[fail[n]]], \cdots$。

从左到右求所有的 $fail$，pre_i 的 border 必然是 pre_{i-1} 的 border 增加一位或者是长为 1 的字符串。用跳 $fail$ 的方法从大到小遍历 pre_{i-1} 的所有 $border\ pre_k$，判断 s_{k+1} 是否与 s_i 相同。

复杂度为什么正确：遍历过程 $fail_1 \to$ 往回跳 $\to fail_2 \to$ 往回跳 $\to fail_3$，这里每次找到 fail 时数值 $+1$，每次往回跳数值至少减少 1。因此总跳的次数是 $\mathcal{O}(n)$ 的。

字符串匹配

洛谷 P3375 字符串匹配

给定文本串 t，模式串 s，$\mathcal{O}(|t|)$ 判断 s 是否是 t 的一个子串，找到所有出现位置。

字符串匹配

洛谷 P3375 字符串匹配

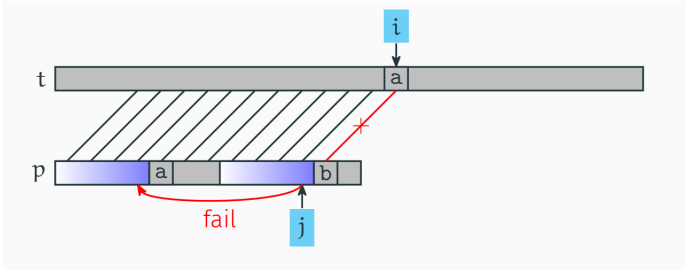
给定文本串 t，模式串 s，$\mathcal{O}(|t|)$ 判断 s 是否是 t 的一个子串，找到所有出现位置。

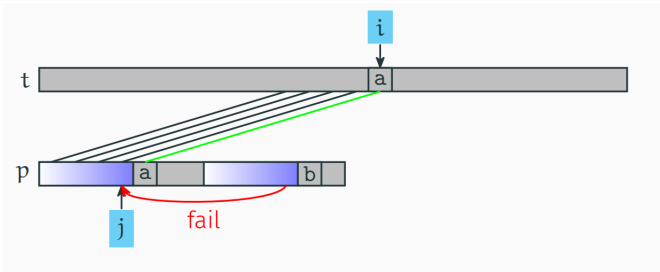
一个朴素的做法：枚举起始点 l，逐字符判断 $t_{l,l+|s|-1}$ 是否等于 $s_{1,|s|}$。

为什么这个数组叫 $fail$？失配指针。

字符串匹配

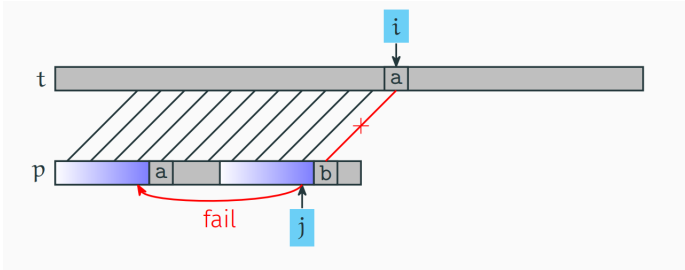
首先找到最长的前缀满足 $t_{1,l} = p_{1,l}$，但 $t_{l+1} \neq p_{l+1}$。此时我们不再从头开始枚举，而是利用已有信息：下一个可匹配的位置应当从 $t_{1,l}$ 的最长 border 处开始，且可以直接从 $fail_l + 1$ 开始逐字符对比。

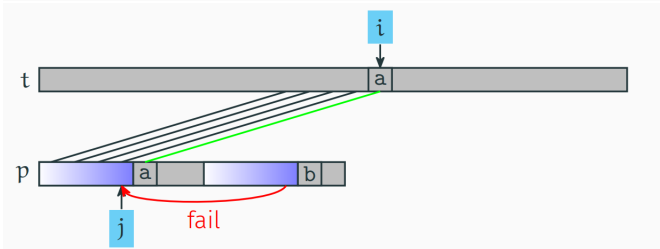




字符串匹配

首先找到最长的前缀满足 $t_{1,l} = p_{1,l}$，但 $t_{l+1} \neq p_{l+1}$。此时我们不再从头开始枚举，而是利用已有信息：下一个可匹配的位置应当从 $t_{1,l}$ 的最长 border 处开始，且可以直接从 $fail_l + 1$ 开始逐字符对比。





如图用两个指针 i, j 维护逐字符对比的位置。i 单调增加，j 和跳 fail 复杂度相同，复杂度 $\mathcal{O}(|t|)$。

border

KMP 用到的匹配和失配的思想在字符串题目中相当常见，在复杂度分析时要小心。

border

KMP 用到的匹配和失配的思想在字符串题目中相当常见，在复杂度分析时要小心。

border 与周期

给定一个串求它每一个前缀的最大周期。

border

KMP 用到的匹配和失配的思想在字符串题目中相当常见，在复杂度分析时要小心。

border 与周期

给定一个串求它每一个前缀的最大周期。

border 和周期是一一对应的：$pre(s, r)$ 是 s 的 border 当且仅当 $|s| - r$ 是 s 的周期。

border

KMP 用到的匹配和失配的思想在字符串题目中相当常见，在复杂度分析时要小心。

border 与周期

给定一个串求它每一个前缀的最大周期。

border 和周期是一一对应的：$pre(s, r)$ 是 s 的 border 当且仅当 $|s| - r$ 是 s 的周期。

转化为对每个前缀求最小 border。KMP 之后维护 mn_l 表示 $pre(1, l)$ 的最小 border，递归 $mn_l = mn_{fail_l}$ 即可。

洛谷 P4824

Description

给定字符串 s, t，找到 t 在 s 中作为子串的第一次出现位置，将其删掉，将左右拼接起来，重复以上流程直到 t 不再是 s 的子串。$|s| \leq 10^6$。

洛谷 P4824

Description

给定字符串 s, t，找到 t 在 s 中作为子串的第一次出现位置，将其删掉，将左右拼接起来，重复以上流程直到 t 不再是 s 的子串。$|s| \leq 10^6$。

在 KMP 的过程中，将 s 中逐字符匹配正确的部分加入栈中，匹配成功时将栈顶 $|l|$ 个字符弹出，从新的栈顶出发匹配。在 KMP 时，维护与 s 的每个前缀匹配的最大 t 前缀 f，将 (i, j) 设为 $stk[top], f[stk[top]]$。

【HNOI 2008】GT 考试

Description

给定 n, m, k，以及长为 m 的字符串 t，询问有多少个长为 n 的字符串 s 没有 t 这个子串。$n \leq 10^9, m \leq 20, k \leq 1000$。

【HNOI 2008】GT 考试

Description

给定 n, m, k，以及长为 m 的字符串 t，询问有多少个长为 n 的字符串 s 没有 t 这个子串。$n \leq 10^9, m \leq 20, k \leq 1000$。

dp 模拟 KMP，$f_{i,j}$ 表示文本串匹配到 i，模式串匹配到 j 的方案数，根据 s_i 与 t_{j+1} 相等转移到 $f_{i+1,j+1}$ 或 $f_{i,fail_j}$。矩阵乘法优化转移。

bzoj4974 字符串大师

Description

给定长为 n 的字符串 s 的 fail 数组，还原一个合法的字符串。$n \leq 10^5$。

bzoj4974 字符串大师

Description

给定长为 n 的字符串 s 的 fail 数组，还原一个合法的字符串。$n \leq 10^5$。

逐个考虑 $fail_i$ 如何变成 $fail_{i+1}$ 的，可以还原出 kmp 时往回跳然后 $+1$ 的过程。每次往回跳意味值一个不等关系，$+1$ 意味着一个相等关系。满足这些关系就行了。

并查集。由于 fail 数组连的边性质比较好可以从前往后填，每个位置在仅考虑与已填字符的相等关系下，取符合条件的最小字符。

【JLOI 2016】字符串覆盖

Description

给定字符串 A 的若干子串 $B_1, B_2, \ldots, B_n$，将每个字符串放到恰好一个它在 A 中的出现位置上（可以重叠），问最多/最少能覆盖多少个 A 中的字符。
$T \leq 10, |A| \leq 10000, n \leq 4, \sum_i |B_i| \leq 1000$。

【JLOI 2016】字符串覆盖

Description

给定字符串 A 的若干子串 $B_1, B_2, \ldots, B_n$，将每个字符串放到恰好一个它在 A 中的出现位置上（可以重叠），问最多/最少能覆盖多少个 A 中的字符。
$T \leq 10, |A| \leq 10000, n \leq 4, \sum_i |B_i| \leq 1000$。

先 KMP 出所有可以放的位置，然后最大最小值分别处理。

【JLOI 2016】字符串覆盖

Description

给定字符串 A 的若干子串 $B_1, B_2, \ldots, B_n$，将每个字符串放到恰好一个它在 A 中的出现位置上（可以重叠），问最多/最少能覆盖多少个 A 中的字符。
$T \leq 10, |A| \leq 10000, n \leq 4, \sum_i |B_i| \leq 1000$。

先 KMP 出所有可以放的位置，然后最大最小值分别处理。
最大值：尽可能少交。$4!$ 枚举放置字符串的先后顺序，后一个如果和前一个有交就尽量靠后放，否则尽量靠前放，再上一个 2^3 枚举然后贪心。

最小值：尽可能多交。去掉被其他字符串包含的子串，枚举放置顺序，记 $f_{i,s}$ 表示前 i 位用 s 集合的子串能覆盖的最小位置个数。转移枚举与第 i 位匹配的子串 $p \in S$，朴素转移是

$$f_{i,s} = \min_{p \in S} \min_{0 \leq j < i} (f_{j,s/p} + \min(len_p, i - j))$$

两层都是 $\min$，可以去掉后面的 $\min$，同时贡献 len_p 和 $i - j$ 的转移。前者只需要维护 f 的前缀最小值，后者单调队列优化。

洛谷 P10716 简单的字符串问题

Description

你有一个字符串 S。q 个询问，每次给出 (i,k)，求有多少个非空字符串 A，使得存在可空字符串 $B_1, B_2, \ldots, B_{k-1}$ 满足：

$$S[1,i] = AB_1AB_2A\ldots AB_{k-1}A$$

其中 $S[1,i]$ 表示 S 的长度为 i 的前缀。
$1 \le k \le i \le n \le 2\times 10^5, 1 \le q \le 2\times 10^5$。

注意 A 必须是前缀 i 的一个 border，并且任何满足条件的 A 的 border 也满足条件。连边 $i \to fail_i$ 形成一棵 fail 树，那么合法的 A 是一条到根的链，只需找到深度最大的那个。

洛谷 P10716 简单的字符串问题

考虑倍增，对于前缀 i 的一个 border，只需要 check 它在原串中第 k 次不重叠出现位置是否 $\le i$。border 一定是前缀 $pre(S,l)$，这个数值 $p_{l,k}$ 可以贪心预处理：$pre(s,l)$ 在 $[r-l+1,r]$ 处再次出现当且仅当 $pre(s,l)$ 是它的 $pre(s,r)$ 的 border，即 fail 树上 $pre(s,l)$ 是 $pre(s,r)$ 的祖先 (fail 树最重要的性质)。

预处理时只需要每次贪心找 $pre(s,l)$ 的下一个不重叠出现位置，在子树的所有位置中二分即可，转化为 dfs 序上的序列二分，用主席树完成。询问次数是调和级数的，复杂度 $\mathcal{O}(n\log^2 n)$。

洛谷 P10716 简单的字符串问题

考虑倍增，对于前缀 i 的一个 border，只需要 check 它在原串中第 k 次不重叠出现位置是否 $\le i$。border 一定是前缀 $pre(S,l)$，这个数值 $p_{l,k}$ 可以贪心预处理：$pre(s,l)$ 在 $[r-l+1,r]$ 处再次出现当且仅当 $pre(s,l)$ 是它的 $pre(s,r)$ 的 border，即 fail 树上 $pre(s,l)$ 是 $pre(s,r)$ 的祖先 (fail 树最重要的性质)。

预处理时只需要每次贪心找 $pre(s,l)$ 的下一个不重叠出现位置，在子树的所有位置中二分即可，转化为 dfs 序上的序列二分，用主席树完成。询问次数是调和级数的，复杂度 $\mathcal{O}(n\log^2 n)$。

与处理完成后，查询可以 $\mathcal{O}(1)$ check，再套一个倍增，复杂度 $\mathcal{O}(n\log n)$。

扩展 KMP

对于字符串 S，定义 z_i 为后缀 $suf(s,i)$ 与 s 的最长公共前缀（LCP）的长度。扩展 KMP 就是用于在 $\mathcal{O}(n)$ 复杂度下求解 z 函数的算法。

类似 KMP，扩展 KMP 通过已经求解的数值减少逐字符比对的复杂度。

扩展 KMP

假设已经求好了 $z_1 \sim z_{i-1}$，现在求解 z_i。

扩展 KMP

假设已经求好了 $z_1 \sim z_{i-1}$，现在求解 z_i。

根据 z 函数的定义，对于任意的 i 有 $s(i, i+z_i-1) = s(1, z_i)$，它是一个 border 可以用来加速匹配。考虑维护最长的形如 $[i, i+z_i-1]$ 的匹配段，记为 $[l, r]$，特别注意的是 z_1 显然等于 $|s|$，但它比较特殊，不算作匹配段，于是从 2 开始枚举 z，初始令 $l = r = 0$。

扩展 KMP

假设已经求好了 $z_1 \sim z_{i-1}$，现在求解 z_i。

根据 z 函数的定义，对于任意的 i 有 $s(i, i+z_i-1) = s(1, z_i)$，它是一个 border 可以用来加速匹配。考虑维护最长的形如 $[i, i+z_i-1]$ 的匹配段，记为 $[l, r]$，特别注意的是 z_1 显然等于 $|s|$，但它比较特殊，不算作匹配段，于是从 2 开始枚举 z，初始令 $l = r = 0$。

计算 z_i 时，若 $i \le r$，则 $s(i, r) = s(i-l+1, r-l+1)$，又根据 z_{i-l+1} 的定义

$$s(1, z_{i-l+1}) = s(i-l+1, i-l+1+z_{i-l+1}-1)$$

于是有

$$s(i-l+1, r-l+1) = s(1, min(z_{i-l+1}, r-i+1))$$

因此：$z_i \ge min(z_{i-l+1}, r-i+1)$，直接将它作为 z_i 的初始值，然后暴力的增加 z_i 并比对字符判断是否合法，最后用 z_i 更新 r 即可。

复杂度分析

整个过程中唯一看起来非线性的就是暴力扩展 z_i 了，考虑什么时候我们会暴力扩展 $z[i]$：

- $i > r$ 时，那么此时扩展 z_i 后 $i + z_i - 1$ 一定 $> r$，每次扩展都会导致 r 增加。
- $i <= r$ 且 $z_{i-l+1} > r - i + 1$ 时，也就是说初始 $i + z_i - 1 > i + r - i + 1 = r + 1$, 那么同样的每次扩展也会导致 r 的增加。
- 因为 r 最多从 1 增加到 $|s|$，因此扩展不超过 $|s|$ 次，总复杂度是 $\mathcal{O}(|s|)$ 的。

洛谷 P5410 【模板】扩展 KMP

Description

给定字符串 s, t，求解 t 的 z 函数以及 s 的每个后缀与 t 的 LCP。

洛谷 P5410 【模板】扩展 KMP

Description

给定字符串 s, t，求解 t 的 z 函数以及 s 的每个后缀与 t 的 LCP。

设另一个 LCP 为 p 数组，求解方式类似。

考虑我们已经求得了 $p_{1\sim i-1}$ 求解 p_i，同样维护最长的匹配段 $[i, i+p_i-1]$，记为 $[l, r]$，有 $s[l, r] = t[1, r-l+1]$
当 $i \le r$ 时，有 $s(i, r) = t(i-l+1, r-l+1) = t(1, min(z_{i-l+1}, r-i+1))$，故将 p_i 的初始值设为 $min(z_{i-l+1}, r-i+1)$，然后暴力扩展即可。复杂度显然也是线性的。

HDU4333 Revolving Digits

Description

字符串 s 每次把最后一个字符放到前面（循环位移），求形成的字符串比最初串分别小，相同，大于的个数。$|s| \leq 10^6$。

HDU4333 Revolving Digits

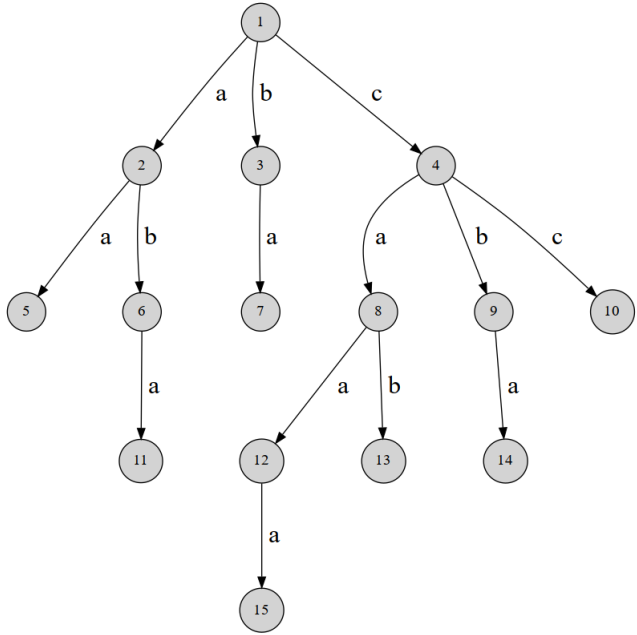
Description

字符串 s 每次把最后一个字符放到前面（循环位移），求形成的字符串比最初串分别小，相同，大于的个数。$|s| \leq 10^6$。

复制一遍字符串，求出每个后缀与 s 的 LCP 后，如果长度 $> n$ 则相等，否则比较后面一个字符大小即可。

Trie

Trie 又被称为字典树，它的每条边上有一个字符，用来储存一个字符串的集合。



Trie

Trie

Trie 的结构很好理解：边表示字母，根到每个节点路径上的所有字母组成这个节点代表的字符串。

Trie

Trie 的结构很好理解：边表示字母，根到每个节点路径上的所有字母组成这个节点代表的字符串。

用 $ch[u][c]$ 表示节点 u 字母 c 的边连向的节点，插入字符串时只需要沿该字符串字符对应的边一直走，如果某个字符 c 对应的边不存在，就从当前节点新建一个儿子连接 c 对应的边。

Trie

Trie 的结构很好理解：边表示字母，根到每个节点路径上的所有字母组成这个节点代表的字符串。

用 $ch[u][c]$ 表示节点 u 字母 c 的边连向的节点，插入字符串时只需要沿该字符串字符对应的边一直走，如果某个字符 c 对应的边不存在，就从当前节点新建一个儿子连接 c 对应的边。

Trie 可以用来快速查询某个字符串是否在字符串集合中出现过：将所有字符集中字符串对应的打上标记，沿查询字符串中的字符在 trie 树上走，最终节点存在且有标记则出现过。(洛谷 P2580)

01 Trie

01 Trie 是字符集为 $\{0,1\}$ 的 Trie，可以用来维护二进制数。通常按高位到低位建立 Trie，可以用来解决一些二进制数的位运算问题，比如求全局最大异或和，虽然似乎和字符串关系不大，但也找不到什么别的好分类，这里提一下。

洛谷 P4551 最长异或路径

Description

给定一个 n 个节点的带权树，寻找树上两个节点使得路径上所有边权异或和最大。$n \leq 10^5, w \leq 2^{31}$。

洛谷 P4551 最长异或路径

Description

给定一个 n 个节点的带权树，寻找树上两个节点使得路径上所有边权异或和最大。$n \leq 10^5, w \leq 2^{31}$。

(i, j) 路径的异或和可以转化为 i 到根路径的异或和异或 j 到根路径的异或和。然后变成全局最大异或和。

洛谷 P4551 最长异或路径

Description

给定一个 n 个节点的带权树，寻找树上两个节点使得路径上所有边权异或和最大。$n \leq 10^5, w \leq 2^{31}$。

(i, j) 路径的异或和可以转化为 i 到根路径的异或和异或 j 到根路径的异或和。然后变成全局最大异或和。

将所有 pre_i （i 到根路径的异或和）作为 01 串加入到从高位到低位的 01 trie 中。枚举其中一个节点 i，从高到低对每一位贪心，如果有字符串在这一位与它不同（trie 树上有对应节点），那么选择这些字符串一定更优，在 trie 树上沿这条边走；否则沿另一条边走。
复杂度 $\mathcal{O}(n)$。

[省选联考 2020 A 卷] 树

Description

给定一棵 n 个结点的有根树 T，每个结点有一个正整数权值 w_i。
请你求出

$$\sum_{u=1}^{n} \otimes_{v \in son(u)} (dis_{u,v} + w_v)$$

其中 son_u 为 u 子树中的点的集合。
$n, v_i \leq 525010$。

[省选联考 2020 A 卷] 树

容易想到使用 $trie$ 树维护一个点子树中的所有 w，那么我们就只需要完成以下操作：

- 单点插入
- 全局加 1
- 合并两棵 trie 树
- 求全局异或和

$1,3,4$ 是好做的，考虑 2 怎么做，如果还是向普通 $trie$ 树一样从高到低维护，一层一层平移非常恶心；考虑改为从低位向高位维护，模拟一般的进位过程，将当前点的左儿子与右儿子交换，然后向左儿子递归下去即可。
那么接下来就是模板 $trie$ 合并，其实现思路与线段树合并完全一致，复杂度也为 $\mathcal{O}(n\log n)$。为了支持 4 操作，我们对 $trie$ 树上的每个点维护 t_i 表示 i 子树中的点的仅考虑从低到高第 $dep_i \sim 20$ 位时的异或和，有：

$$t_i = t_{lc} \otimes t_{rc} | ((siz_{rc} \& 1) << dep_i)$$

这是因为，对于 dep_i 这一位，影响全局异或和是否为 1 的就是这一位 1 的个数，即 siz_{rc}。

洛谷 P9934 绝不能忘记的事

Description

太长了，自行查看。

洛谷 P9934 绝不能忘记的事

Description

太长了，自行查看。

逆天讨论题，充分考察选手的心理素质。

N 在开头：不妨假设原串为 NZH。

- 复制串没有忘了：map
- 复制串后面忘了：中间部分是 ZH 的非空真前缀，在 trie 上求每个点到根的路径点权和。
- 复制串中间忘了：后面部分是 ZH 的非空真后缀，在反串 trie 上求每个点到根的路径点权和。
- 复制串中间后面都忘了：直接累加。

N 在结尾是一样的。

洛谷 P9934 绝不能忘记的事

N 在中间：不妨假设原串为 QNH。

- 复制串没有忘了：map，两个参数
- 复制串后面忘了：map
- 复制串前面忘了：map
- 复制串前面后面都忘了：直接累加

AC 自动机

AC 自动机是以 Trie 的结构为基础，结合 KMP 的思想建立的自动机，用于解决多模式串（作为子串的串）匹配等任务。

AC 自动机

AC 自动机是以 Trie 的结构为基础，结合 KMP 的思想建立的自动机，用于解决多模式串（作为子串的串）匹配等任务。

AC 自动机

AC 自动机是以 Trie 的结构为基础，结合 KMP 的思想建立的自动机，用于解决多模式串（作为子串的串）匹配等任务。

思考

设 $\Sigma = a, b, c, d$，如何判断一个串是否满足：它的前面是若干个字符 a（至少 1 个），然后以一个 c 或者 d 结束？例如，$ad\ aaac$ 满足上述条件，而 $c\ acd\ aaabc$ 不满足上述条件。

限制条件：只允许从前往后扫描一次串。即每个字符仅处理一次，且处理当前字符时，并不知道之后的字符。

AC 自动机

AC 自动机是以 Trie 的结构为基础，结合 KMP 的思想建立的自动机，用于解决多模式串（作为子串的串）匹配等任务。

思考

设 $\Sigma = a, b, c, d$，如何判断一个串是否满足：它的前面是若干个字符 a（至少 1 个），然后以一个 c 或者 d 结束？例如，$ad\ aaac$ 满足上述条件，而 $c\ acd\ aaabc$ 不满足上述条件。

限制条件：只允许从前往后扫描一次串。即每个字符仅处理一次，且处理当前字符时，并不知道之后的字符。

有限状态自动机

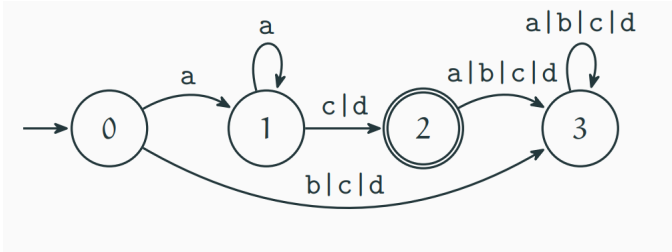
一种做法是记录当前前缀的“状态”。用 0 代表前缀为空，1 代表全为 a，2 代表前面若干 a，最后是一个 c 或者 d，3 代表其他情况。
这样，如果处理完整个串后处于状态 2，这个串就满足要求。

状态转移如下表所示：

字符 \ 状态	0	1	2	3
a	1	1	3	3
b	3	3	3	3
c	3	2	3	3
d	3	2	3	3

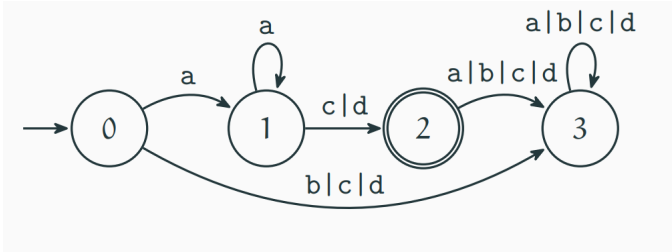
有限状态自动机

也可以用一个有向图表示。状态对应点，转移对应边。
这就是一个有限状态自动机。



有限状态自动机

也可以用一个有向图表示。状态对应点，转移对应边。
这就是一个有限状态自动机。

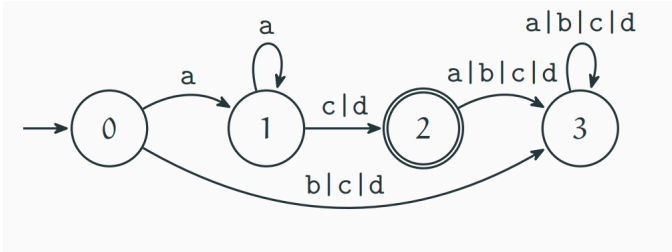


它具有这些性质：

- 它具有有限个状态，不同的状态有不同的意义。
- 它能处理一个字符集上的所有字符串。
- 系统在任何一个状态（当前状态）下，从输入字符串中读入一个字符，根据当前状态和读入的这个字符转到新的状态。当前状态和新的状态可以是同一个状态，也可以是不同的状态。当系统从输入字符串读入一个字符后，它下一次再读时，会读入下一个字符。

有限状态自动机

也可以用一个有向图表示。状态对应点，转移对应边。
这就是一个有限状态自动机。



它具有这些性质：

- 系统中有一个状态，它是系统的开始状态，系统在这个状态下开始进行某个给定串的处理。
- 系统中还有一些状态表示它到目前为止所读入的字符串是合法的，称为终止状态。将系统从开始状态引导到终止状态的所有字符串都是系统所能接受的串。

有限状态自动机

有限状态自动机又分为两类：确定性有限状态自动机（DFA）和非确定性有限状态自动机（NFA）。

所谓确定性就是指对于任何一个状态，输入一个字符都可以转移到另一个确定的状态，具有这种特点的有限状态自动机属于 DFA，如上面的自动机。

有限状态自动机

有限状态自动机又分为两类：确定性有限状态自动机（DFA）和非确定性有限状态自动机（NFA）。

所谓确定性就是指对于任何一个状态，输入一个字符都可以转移到另一个确定的状态，具有这种特点的有限状态自动机属于 DFA，如上面的自动机。

NFA 并不遵从上面的规则，有时甚至不需要读入字符就可以进行转移，这样的转移边称为 ϵ 边。这使得在 NFA 处理串的过程中，可能存在多个当前状态。

AC 自动机

AC 自动机用来在一个文本串中同时搜寻多个模式串。

AC 自动机

AC 自动机用来在一个文本串中同时搜寻多个模式串。

我们的思路非常粗暴，直接同时对所有模式串运行 KMP。遍历文本串的前缀 $pre(s,i)$，在 KMP 时，每个模式串此时都有一个前缀与 $pre(s,l)$ 正在匹配的过程中（这个前缀是 $pre(s,l)$ 的后缀）。我们用这些前缀的集合来作为自动机的状态。

为了能同时寻找多个模式串，需要对所有串的每个不同前缀分别设置一个状态，这正好是 Trie 树上的所有点。于是可以先建出 Trie。
但为了处理 KMP 中的失配，还需得到不在 Trie 上的转移边来处理失配的情况。

AC 自动机

KMP 中，我们定义失配指针 $fail$ 指向前缀的最大 border。类似地定义 trie 上的 $fail$ 指向在自动机里出现的最长真后缀。

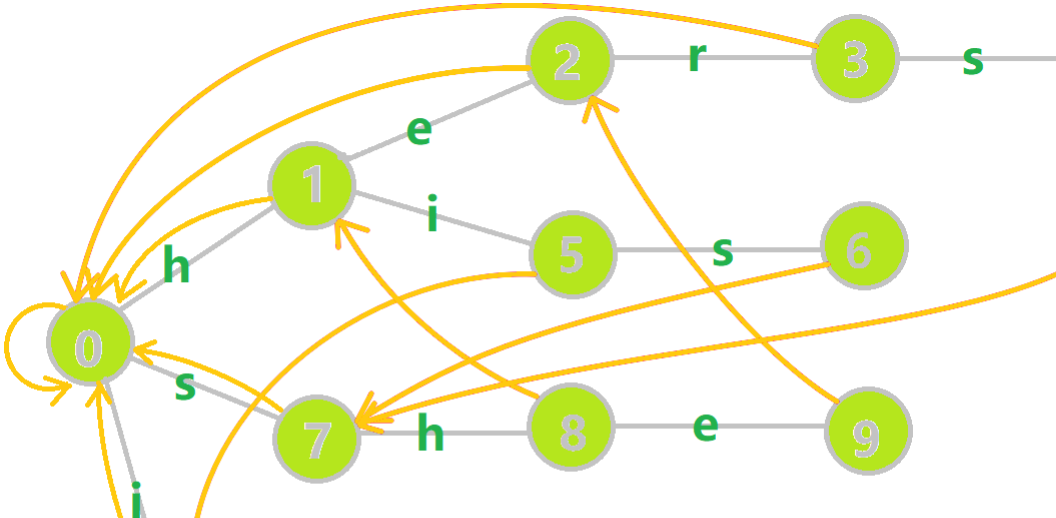
、

AC 自动机

KMP 中，我们定义失配指针 $fail$ 指向前缀的最大 border。类似地定义 trie 上的 $fail$ 指向在自动机里出现的最长真后缀。

、

因为 $fail[u]$ 在 trie 上的深度总是小于 u，可以在 trie 上按 bfs 序递推，根据 $fail[u]$ 的转移边得到 u 的失配转移边：一路跳 fail 直到 $ch[fail[u], c]$ 存在，令 $fail[ch[u, c]] = ch[fail[u], c]$。最终 fail 边构成一棵树。



AC 自动机

按照正常 KMP 的做法，我们在没法向下匹配（不存在 c 的出边）时应当一路跳 fail 直到存在出边。这里常用的 AC 自动机使用了一个转化方法，直接将 $ch[u, c]$ 指向这个失配后的节点，也就是说它看似通过 trie 的边转移，但其实是经过了一个跳 fail 的过程。

用 0 表示直接回到根节点，若 $ch[u, c]$ 存在则 $fail[ch[u, c]] = ch[fail[u], c]$，否则直接令 $ch[u, c] = ch[fail[u], c]$（走到 c 相当于失配）。

AC 自动机

按照正常 KMP 的做法，我们在没法向下匹配（不存在 c 的出边）时应当一路跳 fail 直到存在出边。这里常用的 AC 自动机使用了一个转化方法，直接将 $ch[u, c]$ 指向这个失配后的节点，也就是说它看似通过 trie 的边转移，但其实是经过了一个跳 fail 的过程。

用 0 表示直接回到根节点，若 $ch[u, c]$ 存在则 $fail[ch[u, c]] = ch[fail[u], c]$，否则直接令 $ch[u, c] = ch[fail[u], c]$（走到 c 相当于失配）。

注意这并没有体现我们所说的“前缀集合”，前缀集合实际上是通过从该点出发的 fail 链上的所有节点来体现的，因此在遍历到 $s[1, i]$ 在 AC 自动机上的节点时，与它匹配的不只是这个节点的字符串，还有 fail 链上的所有节点的字符串。

洛谷 P3808 AC 自动机（简单版）

Description

给定 n 个模式串 s_i 和一个文本串 t，求有多少个不同的模式串在文本串里出现过。两个模式串不同当且仅当他们 ** 编号 ** 不同。
$n \leq 10^6, |t \leq 10^6, \sum |s_i| \leq 10^6$。

洛谷 P3808 AC 自动机（简单版）

Description

给定 n 个模式串 s_i 和一个文本串 t，求有多少个不同的模式串在文本串里出现过。两个模式串不同当且仅当他们 ** 编号 ** 不同。
$n \leq 10^6, |t \leq 10^6, \sum |s_i| \leq 10^6$。

预处理每个模式串对应的节点处打上标记。沿文本串在 AC 自动机上走一遍，在沿途打上标记，将标记沿 fail 链传递上去，在这道题上传递可以暴力，遇到被标记的点就不往上传了。

洛谷 P5357 【模板】AC 自动机

Description

给定 n 个模式串 s_i 和一个文本串 t，求每个模式串在 t 中的出现次数。
$n \leq 10^6, |t \leq 10^6, \sum |s_i| \leq 10^6$。

洛谷 P5357 【模板】AC 自动机

Description

给定 n 个模式串 s_i 和一个文本串 t，求每个模式串在 t 中的出现次数。
$n \leq 10^6, |t \leq 10^6, \sum |s_i| \leq 10^6$。

这道题就没法暴力沿 fail 链传标记了，先记录在节点上。由于传递标记都是向上处理，在预处理 fail 时记录节点的 bfs 序，沿 bfs 序倒序传递标记。复杂度 $\mathcal{O}(n)$。

[HNOI2004] L 语言

Description

给定 n 个模式串 s_i 和 m 个文本串 t_i。对每个 t_i 求最长的前缀使得该前缀是由多个模式串首尾拼接而成的。$n \leq 20, m \leq 50, |s| \leq 10, |t| \leq 10^6$。

[HNOI2004] L 语言

Description

给定 n 个模式串 s_i 和 m 个文本串 t_i。对每个 t_i 求最长的前缀使得该前缀是由多个模式串首尾拼接而成的。$n \leq 20, m \leq 50, |s| \leq 10, |t| \leq 10^6$。

对模式串建立 AC 自动机，记录 t 的每个前缀在 AC 自动机上的节点。记 f_i 表示前缀 i 是否可以被拼接而成，暴力做法是枚举所有作为 pre_i 后缀的模式串进行转移，枚举可以通过在 AC 自动机上跳 fail 实现：$f_i = \cup_{j<i} f_j[valid(j+1, i)]$

[HNOI2004] L 语言

Description

给定 n 个模式串 s_i 和 m 个文本串 t_i。对每个 t_i 求最长的前缀使得该前缀是由多个模式串首尾拼接而成的。$n \leq 20, m \leq 50, |s| \leq 10, |t| \leq 10^6$。

对模式串建立 AC 自动机，记录 t 的每个前缀在 AC 自动机上的节点。记 f_i 表示前缀 i 是否可以被拼接而成，暴力做法是枚举所有作为 pre_i 后缀的模式串进行转移，枚举可以通过在 AC 自动机上跳 fail 实现：$f_i = \cup_{j<i} f_j[valid(j+1, i)]$

暴力跳复杂度会炸，但 $|s| \leq 10$，可以状压。对 AC 自动机每个节点预处理 g_i 表示它在 fail 树上祖先的终止节点的长度，预处理很容易完成。

这样 AC 自动机转移是只需要枚举 g_i 中的长度，复杂度 $\mathcal{O}(m|s||t|)$。还可以进一步预处理 $f_{i-10}, f_{i-9}, \ldots, f_{i-1}$ 的状态，通过位运算让转移变成 $\mathcal{O}(1)$，复杂度 $\mathcal{O}(m|t|)$。

[NOI2011] 阿狸的打字机

Description

给定一个长为 m 的操作串，有三种操作：

- 输入一个小写字母
- 删除最后一个字母
- 打印目前的字符串

最终打印出了 n 个字符串，q 次询问第 x 个字符串在第 y 个中出现了多少次。$n, m, q \leq 10^5$。

[NOI2011] 阿狸的打字机

Description

给定一个长为 m 的操作串，有三种操作：

- 输入一个小写字母
- 删除最后一个字母
- 打印目前的字符串

最终打印出了 n 个字符串，q 次询问第 x 个字符串在第 y 个中出现了多少次。$n, m, q \leq 10^5$。

离线询问，建出 AC 自动机（很容易建出）。
第 x 个字符串在第 y 个字符串中出现次数等于有多少个 y 字符串对应的前缀在 x 的 fail 树子树内。
利用 dfs 序变成区间询问，离线询问，记录每个字符串的结束节点。怎样让线段树里区间内只有待询问节点的信息？由于本题的特殊结构很容易实现，将询问离线到字符串 y 被打印的位置，遍历操作串时线段树里只维护当前正在输入的串前缀的信息。

floj3789. 语言学

Description

给定 m 个优美串 $B_1, B_2, \ldots, B_m$，第 i 个优美串有优美度 w_i。如果两个串拼在一起时，前串的一个后缀与后串的一个前缀组合起来形成了一个优美串，就能获得这个优美串的优美度。他认为两个串拼接的优美度为通过这种方法能形成的所有优美串的优美度之和（每个优美串只统计一次）。

为了证明这一结论，B 博士选出了 n 个询问串 $S_1, S_2, \ldots, S_n$，计算这 n 个串两两拼接得到的优美度和。$n, m \leq 10^6, \sum |S_i| \leq 10^6, \sum |T_i| \leq 10^6$。

subtask 1: $n \leq 500$
subtask 2: $\sum |S|, \sum |T| \leq 2 \times 10^5$

$n \leq 500$

考虑枚举前串 A ，对于优美串 T ，我们可以 $\mathcal{O}(|T|)$ 的求出 T 的所有前缀 $T[1,i]$ 满足 $T[1,i]$ 作为 A 的后缀出现，因此所有以 $T[i+1,|T|]$ 作为前缀的串作为后串与 A 配对都能获得 T 的优美度，于是对询问串建立 AC 自动机，在 $T[i+1,|T|]$ 上记录二元组 (A,T) 。问题转化为对于每个后串 B，求它的前缀上标记的互不相同二元组的数量，这等价于在 AC 自动机的 fail 树上到祖先链上的不同颜色数。

于是可以通过在 fail 树上 dfs 一遍求出答案，这部分的复杂度为染色次数，由于一共进行了 $\mathcal{O}(n\sum|T|)$ 次染颜色，因此复杂度也为 $\mathcal{O}(n\sum|T|)$。

$$\sum |S|, \sum |T| \leq 2 \times 10^5$$

考虑枚举优美串 T，对于 T 的每一个前缀 $T[1 \sim i]$，找出所有满足 $T[1 \sim i]$ 是该串最长后缀的前串集合 U（最长后缀是指最长的满足作为 T 前缀出现的后缀），那么 T 作为这些串后缀出现的前缀都是 $T[1 \sim i]$ 的 border，同时任何一个后串如果与 S 中任意一个串组合得到 T，那么一定会与 S 中所有的串组合得到 T，因此可以将 S 中的串与 T 的二元组合并为同一种颜色，暴力下放到 $T[1 \sim i]$ 的 border 上进行染色。

$$\sum |S|, \sum |T| \leq 2 \times 10^5$$

考虑枚举优美串 T，对于 T 的每一个前缀 $T[1 \sim i]$，找出所有满足 $T[1 \sim i]$ 是该串最长后缀的前串集合 U（最长后缀是指最长的满足作为 T 前缀出现的后缀），那么 T 作为这些串后缀出现的前缀都是 $T[1 \sim i]$ 的 border，同时任何一个后串如果与 S 中任意一个串组合得到 T，那么一定会与 S 中所有的串组合得到 T，因此可以将 S 中的串与 T 的二元组合并为同一种颜色，暴力下放到 $T[1 \sim i]$ 的 border 上进行染色。

求 U 集合可以通过对优美串建立 AC 自动机，对于每个询问串 A 在 AC 自动机上走一遍，那么最终节点 fail 树上的祖先就是在 A 中作为后缀出现的询问串前缀。对于“最大后缀”这一限制可以暴力在祖先上全部打标记，最后对于询问串的每个前缀，容斥掉以它为 border 的前缀的被标记次数即可。接下来就只需要暴力染色，染完色后的部分就与上一个 subtask 相同了。

$$\sum |S|, \sum |T| \leq 2 \times 10^5$$

考虑这个算法的复杂度，求 U 集合是 $\mathcal{O}(n)$ 的，对于后面的部分，如果当前枚举优美串的大小 $|T| \leq \sqrt{H}(H = \max(\sum |S|, \sum |T|))$，那么对它暴力做的复杂度应该是 $\mathcal{O}(|T|^2)$，否则该优美串中至多有 $\sqrt{H}$ 个串会被标记到，暴力做的复杂度是 $\mathcal{O}(|T|\sqrt{H})$ 的，因此总复杂度为 $\mathcal{O}(H\sqrt{H})$。

正解

考虑优化上一个做法，先把每个询问串对应的关键点在 AC 自动机上找出，于是之前的染色相当于对某一个优美串前缀，求出有多少个关键点在该前缀的 border 的子树里。那么可以对枚举的这个优美串建立 kmp 自动机，在自动机上 dfs 每一个前缀，那么只需要支持维护一个集合，在集合里增加一个点，动态求有多少个关键点在目前集合中点的子树里。用线段树维护区间加、出现次数 > 0 的点数量即可，复杂度为 $\mathcal{O}(H \log H)$。

完结撒花

谢谢大家！