字符串杂题选讲

范斯喆

浙江省诸暨市海亮高级中学

2025 年 2 月 3 日

前言

本次讲课包含两部分。
第一部分是笔者最近遇到的有关字符串的有趣问题。
第二部分是基本子串结构的介绍，以及相关的一些问题。

前置知识

本次讲课所需要的前置知识如下：

- KMP，扩展 KMP；
- Manacher；
- 字典树，AC 自动机；
- SA，SAM。

没有学过的同学们可以自行学习。

CF1827C

给定一个长度为 n 的字符串 S。
定义一个字符串是好的，当且仅当它可以分解为若干个偶回文串的拼接。
求 S 有多少个子串是好的。
$1 \leq n \leq 5 \times 10^5$。
保证 S 只包含小写字母。

CF1827C

对于一个偶回文串，如果它有一个真后缀也是偶回文串，那它一定可以分解为更短的偶回文串的拼接。

CF1827C

对于一个偶回文串，如果它有一个真后缀也是偶回文串，那它一定可以分解为更短的偶回文串的拼接。
因此一个字符串的“极小”分解方式是唯一的，考虑进行 DP。

CF1827C

对于一个偶回文串，如果它有一个真后缀也是偶回文串，那它一定可以分解为更短的偶回文串的拼接。
因此一个字符串的“极小”分解方式是唯一的，考虑进行 DP。
显然，我们只需要求出以 i 结尾的最短偶回文串长度。设为 f_i。

CF1827C

对于一个偶回文串，如果它有一个真后缀也是偶回文串，那它一定可以分解为更短的偶回文串的拼接。
因此一个字符串的“极小”分解方式是唯一的，考虑进行 DP。
显然，我们只需要求出以 i 结尾的最短偶回文串长度。设为 f_i。
用 Manacher 求出以每个间隔为中心的最大回文半径。从大到小考虑所有间隔，用双向链表维护区间赋值即可。

CF1827C

对于一个偶回文串，如果它有一个真后缀也是偶回文串，那它一定可以分解为更短的偶回文串的拼接。
因此一个字符串的“极小”分解方式是唯一的，考虑进行 DP。
显然，我们只需要求出以 i 结尾的最短偶回文串长度。设为 f_i。
用 Manacher 求出以每个间隔为中心的最大回文半径。从大到小考虑所有间隔，用双向链表维护区间赋值即可。
时间复杂度 $O(n)$。

CF1913F

给定一个长度为 n 的字符串 S。
你可以将至多一个字符替换为任意小写字母，求这样操作一次之后串中回文子串数量的最大值。
$1 \leq n \leq 3 \times 10^5$。
保证 S 仅包含小写字母。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。
减少的回文子串数量就等于跨过 i 且 i 不为回文中心的回文子串数量，可以通过差分简单求出。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。
减少的回文子串数量就等于跨过 i 且 i 不为回文中心的回文子串数量，可以通过差分简单求出。
接下来计算增加的回文子串数量。只考虑奇回文串，偶回文串是类似的。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。
减少的回文子串数量就等于跨过 i 且 i 不为回文中心的回文子串数量，可以通过差分简单求出。
接下来计算增加的回文子串数量。只考虑奇回文串，偶回文串是类似的。
枚举回文中心 x，设其最长回文半径为 len_x。则我们修改的位置必须是 $x - len_x$ 或者 $x + len_x$。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。
减少的回文子串数量就等于跨过 i 且 i 不为回文中心的回文子串数量，可以通过差分简单求出。
接下来计算增加的回文子串数量。只考虑奇回文串，偶回文串是类似的。
枚举回文中心 x，设其最长回文半径为 len_x。则我们修改的位置必须是 $x - len_x$ 或者 $x + len_x$。
于是只要求 $S^T[1, x - len_x - 1]$ 和 $S[x + len_x + 1, n]$ 的 LCP 即可。

CF1913F

枚举替换的位置 i 和替换后的字符 j。则回文子串数量的改变有两部分：增加和减少。
减少的回文子串数量就等于跨过 i 且 i 不为回文中心的回文子串数量，可以通过差分简单求出。
接下来计算增加的回文子串数量。只考虑奇回文串，偶回文串是类似的。
枚举回文中心 x，设其最长回文半径为 len_x。则我们修改的位置必须是 $x - len_x$ 或者 $x + len_x$。
于是只要求 $S^T[1, x - len_x - 1]$ 和 $S[x + len_x + 1, n]$ 的 LCP 即可。
考虑到这是 CF，建议使用 SA。复杂度 $O(n \log n)$。

P10716

给定一个长度为 n 的字符串 S。
q 次询问，每次给定 (i,k)，求有多少个非空字符串 A，使得存在可空字符串 $B_1, B_2, \cdots, B_{k-1}$，满足：

$$S[1,i] = A + B_1 + A + B_2 + A + \cdots + A + B_{k-1} + A$$

$1 \le k \le i \le n \le 2 \times 10^5, 1 \le q \le 2 \times 10^5$。
保证 S 仅包含小写字母。

P10716

特判 $k=1$ 的情况。$k>1$ 时，A 合法的充要条件为：

1. A 为 $S[1,i]$ 的一个 border；
2. A 在 $S[1,i]$ 中不重叠出现了至少 k 次。

P10716

特判 $k=1$ 的情况。$k>1$ 时，A 合法的充要条件为：

1. A 为 $S[1,i]$ 的一个 border；
2. A 在 $S[1,i]$ 中不重叠出现了至少 k 次。

建出 KMP 的失配树，则合法的 A 为根到某个结点 u 的链，其中 u 为 i 的祖先中最近的满足条件的点。因此只要求出 u 即可得到答案。

P10716

特判 $k=1$ 的情况。$k>1$ 时，A 合法的充要条件为：

1. A 为 $S[1,i]$ 的一个 border；
2. A 在 $S[1,i]$ 中不重叠出现了至少 k 次。

建出 KMP 的失配树，则合法的 A 为根到某个结点 u 的链，其中 u 为 i 的祖先中最近的满足条件的点。因此只要求出 u 即可得到答案。

若我们对每个 u 都预处理了其在 S 中不重叠出现的位置（显然，我们需要的空间仅有 $O(n\log n)$），则只需要倍增就可以求出 u。

P10716

考虑如何预处理。枚举 u，则只要从 1 开始暴力往后跳到第一个 p，满足 $S[p, p+u-1] = S[1, u]$ 即可。也就是说，$S[p, n]$ 和 S 的 LCP 长度至少为 u。

P10716

考虑如何预处理。枚举 u，则只要从 1 开始暴力往后跳到第一个 p，满足 $S[p, p+u-1] = S[1, u]$ 即可。也就是说，$S[p, n]$ 和 S 的 LCP 长度至少为 u。
用扩展 KMP 求出每个后缀和 S 的 LCP 长度之后，从小到大枚举 u，利用并查集维护删除即可。

P10716

考虑如何预处理。枚举 u，则只要从 1 开始暴力往后跳到第一个 p，满足 $S[p, p+u-1] = S[1, u]$ 即可。也就是说，$S[p, n]$ 和 S 的 LCP 长度至少为 u。

用扩展 KMP 求出每个后缀和 S 的 LCP 长度之后，从小到大枚举 u，利用并查集维护删除即可。

复杂度 $O(n\alpha(n)\log n + q\log n)$。

P11150

给定一个长度为 n 的字符串 S。
q 次询问，每次给定一个非空字符串 T，求最小的 x 满足 $S[1,x]+T+S[x+1,n]$ 的字典序最小。
$1 \le n, \sum |T| \le 10^6, 1 \le q \le 5 \times 10^5$。
保证字符串仅由数字构成。

P11150

显然，我们要找到一对 (i,j) 使得 $S[1,i]+T[1,j]<S[1,i+j]$，使得 $i+j$ 最小。当然，为了保证存在这样的位置，要在 S 最后添加一个最大的字符。

P11150

显然，我们要找到一对 (i,j) 使得 $S[1,i]+T[1,j]<S[1,i+j]$，使得 $i+j$ 最小。当然，为了保证存在这样的位置，要在 S 最后添加一个最大的字符。

先枚举 j，然后从 $T[j]+1$ 开始往大枚举下一个字符 p，查找 $T[1,j-1]+p$ 在 S 中出现的位置就行了。这里需要使用 SAM 维护子串匹配。

P11150

显然，我们要找到一对 (i,j) 使得 $S[1,i]+T[1,j]<S[1,i+j]$，使得 $i+j$ 最小。当然，为了保证存在这样的位置，要在 S 最后添加一个最大的字符。

先枚举 j，然后从 $T[j]+1$ 开始往大枚举下一个字符 p，查找 $T[1,j-1]+p$ 在 S 中出现的位置就行了。这里需要使用 SAM 维护子串匹配。

然而 $i+j$ 最小时也可能有多个合法的 (i,j)。考虑如何比较它们谁最优。

P11150

显然，我们要找到一对 (i,j) 使得 $S[1,i]+T[1,j]<S[1,i+j]$，使得 $i+j$ 最小。当然，为了保证存在这样的位置，要在 S 最后添加一个最大的字符。

先枚举 j，然后从 $T[j]+1$ 开始往大枚举下一个字符 p，查找 $T[1,j-1]+p$ 在 S 中出现的位置就行了。这里需要使用 SAM 维护子串匹配。

然而 $i+j$ 最小时也可能有多个合法的 (i,j)。考虑如何比较它们谁最优。

设有 $i_1+j_1=i_2+j_2$，且 $S[1,i_1]+T[1,j_1]$ 和 $S[1,i_2]+T[1,j_2]$ 均小于 $S[1,i_1+j_2]$。

P11150

显然，我们要找到一对 (i,j) 使得 $S[1,i]+T[1,j]<S[1,i+j]$，使得 $i+j$ 最小。当然，为了保证存在这样的位置，要在 S 最后添加一个最大的字符。

先枚举 j，然后从 $T[j]+1$ 开始往大枚举下一个字符 p，查找 $T[1,j-1]+p$ 在 S 中出现的位置就行了。这里需要使用 SAM 维护子串匹配。

然而 $i+j$ 最小时也可能有多个合法的 (i,j)。考虑如何比较它们谁最优。

设有 $i_1+j_1=i_2+j_2$，且 $S[1,i_1]+T[1,j_1]$ 和 $S[1,i_2]+T[1,j_2]$ 均小于 $S[1,i_1+j_2]$。

由于 $i+j$ 最小，因此 $S[1,i_1]+T[1,j_1-1]=S[1,i_2]+T[1,j_2-1]$。

P11150

此时只需要比较 T 的后缀 $T[j_1,|T|]$ 和 $T[j_2,|T|]$ 谁更优。用一个后缀数组预处理就行了。

P11150

此时只需要比较 T 的后缀 $T[j_1,|T|]$ 和 $T[j_2,|T|]$ 谁更优。用一个后缀数组预处理就行了。
不难证明，如果 $T[j_2,|T|]$ 恰好是 $T[j_1,|T|]$ 的前缀，则两种方案得到的串相同。

P11150

此时只需要比较 T 的后缀 $T[j_1,|T|]$ 和 $T[j_2,|T|]$ 谁更优。用一个后缀数组预处理就行了。
不难证明，如果 $T[j_2,|T|]$ 恰好是 $T[j_1,|T|]$ 的前缀，则两种方案得到的串相同。
于是，我们得到了一个最优的 i。不过我们要求的是最优的 i 中最小的，怎么求?

P11150

此时只需要比较 T 的后缀 $T[j_1,|T|]$ 和 $T[j_2,|T|]$ 谁更优。用一个后缀数组预处理就行了。
不难证明，如果 $T[j_2,|T|]$ 恰好是 $T[j_1,|T|]$ 的前缀，则两种方案得到的串相同。
于是，我们得到了一个最优的 i。不过我们要求的是最优的 i 中最小的，怎么求?
其实只要找出 T 的周期，然后倍增向前跳就行了。用哈希判断是否相等。

P11150

此时只需要比较 T 的后缀 $T[j_1, |T|]$ 和 $T[j_2, |T|]$ 谁更优。用一个后缀数组预处理就行了。
不难证明，如果 $T[j_2, |T|]$ 恰好是 $T[j_1, |T|]$ 的前缀，则两种方案得到的串相同。
于是，我们得到了一个最优的 i。不过我们要求的是最优的 i 中最小的，怎么求?
其实只要找出 T 的周期，然后倍增向前跳就行了。用哈希判断是否相等。
复杂度为 $O(n + \sum |T| \log \sum |T| + q \log n)$。

CF1975G

给定两个长度分别为 n 和 m 的字符串 S, T，其中包含小写字母，- 和 *。
- 可以替换成任意小写字母。* 可以替换成任意小写字母构成的字符串（可以为空）。
求能否使得 S 和 T 经过替换之后相等。
$1 \leq n, m \leq 2 \times 10^6$。

CF1975G

两个串都没有 * 的情况是简单的。

CF1975G

两个串都没有 * 的情况是简单的。
两个串都有 * 的情况，只要比较没有 * 的前缀或后缀是否相等即可。

CF1975G

两个串都没有 * 的情况是简单的。
两个串都有 * 的情况，只要比较没有 * 的前缀或后缀是否相等即可。
剩下只有 S 有 * 的情况。容易处理成 S 的开头和结尾都是 * 的情况。

CF1975G

两个串都没有 * 的情况是简单的。
两个串都有 * 的情况，只要比较没有 * 的前缀或后缀是否相等即可。
剩下只有 S 有 * 的情况。容易处理成 S 的开头和结尾都是 * 的情况。
设 S 形如 $*s_1*s_2*\cdots*s_k*$。考虑贪心，找到 T 中第一个能匹配 s_1 的位置，然后把这个位置之前的全部删掉，继续匹配。显然这样是对的。

CF1975G

两个串都没有 * 的情况是简单的。
两个串都有 * 的情况，只要比较没有 * 的前缀或后缀是否相等即可。
剩下只有 S 有 * 的情况。容易处理成 S 的开头和结尾都是 * 的情况。
设 S 形如 $*s_1*s_2*\cdots*s_k*$。考虑贪心，找到 T 中第一个能匹配 s_1 的位置，然后把这个位置之前的全部删掉，继续匹配。显然这样是对的。
那么问题变为找到 T 中某个串的第一次出现位置。

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。
那么如何寻找匹配呢？我们有工具：NTT 优化带通配符的字符串匹配。

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。
那么如何寻找匹配呢？我们有工具：NTT 优化带通配符的字符串匹配。
具体地，设我们要将 S 与 T 匹配。构造序列 a, b，如果 s_i 是 - 则 $a_i = 0$，否则 $a_i = s_i$。同理构造 b。

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。
那么如何寻找匹配呢？我们有工具：NTT 优化带通配符的字符串匹配。
具体地，设我们要将 S 与 T 匹配。构造序列 a, b，如果 s_i 是 - 则 $a_i = 0$，否则 $a_i = s_i$。同理构造 b。
那么 s 能匹配 $t_i, \cdots, t_{i+n-1}$，当且仅当

$$\sum_{j=0}^{n-1} (s_j - t_{i+j})^2 s_j t_{i+j} = 0$$

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。
那么如何寻找匹配呢？我们有工具：NTT 优化带通配符的字符串匹配。
具体地，设我们要将 S 与 T 匹配。构造序列 a, b，如果 s_i 是 - 则 $a_i = 0$，否则 $a_i = s_i$。同理构造 b。
那么 s 能匹配 $t_i, \cdots, t_{i+n-1}$，当且仅当

$$\sum_{j=0}^{n-1} (s_j - t_{i+j})^2 s_j t_{i+j} = 0$$

拆括号之后用 NTT 优化差卷积即可。

CF1975G

考虑将 s_1 与 T 中的前 $2|s_1|$ 个字符进行匹配。如果没找到，就删去前 $|s_1|$ 个字符。
那么如何寻找匹配呢？我们有工具：NTT 优化带通配符的字符串匹配。
具体地，设我们要将 S 与 T 匹配。构造序列 a, b，如果 s_i 是 - 则 $a_i = 0$，否则 $a_i = s_i$。同理构造 b。
那么 s 能匹配 $t_i, \cdots, t_{i+n-1}$，当且仅当

$$\sum_{j=0}^{n-1} (s_j - t_{i+j})^2 s_j t_{i+j} = 0$$

拆括号之后用 NTT 优化差卷积即可。
复杂度 $O(n \log n)$。

P11291

给定 n 个字符串 $S_1, S_2, \cdots, S_n$ 以及一个字符串 T。
定义一个字符串 R 是好的当且仅当它是某个 S_i 的前缀。空串也算前缀。
定义一个三元组 (l, r, k) 是好的当且仅当 $1 \le l \le r \le |T|, 1 \le k \le K$ 且存在 k 个好的字符串 $R_1, R_2, \cdots, R_k$ 满足 $R_1 + R_2 + \cdots + R_k = T[l, r]$。
求好的三元组数量，并且对每个 i 求出有多少个好的三元组 (l, r, k) 满足 $l \le i \le r$。
$1 \le n \le 10, 1 \le |S_i| \le 5 \times 10^4, 1 \le |T|, K \le 5 \times 10^5$。
保证字符串仅包含小写字母。

P11291

注意到，当 l, k 固定时，使 (l, r, k) 合法的 r 构成一段从 l 开始的区间。

P11291

注意到，当 l, k 固定时，使 (l, r, k) 合法的 r 构成一段从 l 开始的区间。
设 $dp_{l,k}$ 表示使得 (l, r, k) 合法的最大的 r。转移为：

$$dp_{l,k} = \max_{l \le i \le dp_{l,k-1}+1} to_i$$

其中 $dp_{l,1} = to_l$。

P11291

注意到，当 l, k 固定时，使 (l, r, k) 合法的 r 构成一段从 l 开始的区间。
设 $dp_{l,k}$ 表示使得 (l, r, k) 合法的最大的 r。转移为：

$$dp_{l,k} = \max_{l \le i \le dp_{l,k-1}+1} to_i$$

其中 $dp_{l,1} = to_l$。
第一问的答案可以直接求出。第二问的答案相当于区间加等差数列，差分两次即可。

P11291

注意到，当 l, k 固定时，使 (l, r, k) 合法的 r 构成一段从 l 开始的区间。
设 $dp_{l,k}$ 表示使得 (l, r, k) 合法的最大的 r。转移为：

$$dp_{l,k} = \max_{l \le i \le dp_{l,k-1}+1} to_i$$

其中 $dp_{l,1} = to_l$。
第一问的答案可以直接求出。第二问的答案相当于区间加等差数列，差分两次即可。
时间复杂度 $O(K|T| + n|T|\log|T|)$。需要继续优化。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。
l 固定时这些区间的右端点递增，因此上一次取到区间 max 之前的位置都不用考虑。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。
l 固定时这些区间的右端点递增，因此上一次取到区间 max 之前的位置都不用考虑。
设 pos_i 表示 $[i, to_i+1]$ 中 to_j 最大的 j。建出 $i \to pos_i$ 的边，显然会建出基环树森林。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。
l 固定时这些区间的右端点递增，因此上一次取到区间 max 之前的位置都不用考虑。
设 pos_i 表示 $[i, to_i+1]$ 中 to_j 最大的 j。建出 $i \to pos_i$ 的边，显然会建出基环树森林。
则转移时 $k \leftarrow k+1$ 相当于跳父亲。如果跳到了结点 u，则对应的 dp 值就为 to_u。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。
l 固定时这些区间的右端点递增，因此上一次取到区间 max 之前的位置都不用考虑。
设 pos_i 表示 $[i, to_i+1]$ 中 to_j 最大的 j。建出 $i \to pos_i$ 的边，显然会建出基环树森林。
则转移时 $k \leftarrow k+1$ 相当于跳父亲。如果跳到了结点 u，则对应的 dp 值就为 to_u。
于是通过树上倍增和树上差分就可以求出答案。

P11291

$dp_{l,k}$ 转移时要求 to 在 $[l, dp_{l,k-1}+1]$ 上的区间 max。
l 固定时这些区间的右端点递增，因此上一次取到区间 max 之前的位置都不用考虑。
设 pos_i 表示 $[i, to_i+1]$ 中 to_j 最大的 j。建出 $i \to pos_i$ 的边，显然会建出基环树森林。
则转移时 $k \leftarrow k+1$ 相当于跳父亲。如果跳到了结点 u，则对应的 dp 值就为 to_u。
于是通过树上倍增和树上差分就可以求出答案。
时间复杂度 $O(n|T|\log|T|)$。

CF1975H

给定一个长度为 n 的字符串 S。
你要重排字符串 S，使得其字典序最大的子串的字典序最小。
求这个最小的最大子串。
$1 \leq n \leq 10^6$。
保证 S 仅包含小写字母。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。
要注意子问题中两个串的比较并非是字典序比较，而是在末尾加上 $+\infty$ 之后比较。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。
要注意子问题中两个串的比较并非是字典序比较，而是在末尾加上 $+\infty$ 之后比较。
那么只需知道所有 $s_1, s_2, \cdots, s_k$ 即可。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。
要注意子问题中两个串的比较并非是字典序比较，而是在末尾加上 $+\infty$ 之后比较。
那么只需知道所有 $s_1, s_2, \cdots, s_k$ 即可。
显然，s_i 内部肯定是从小到大排。同时，所有 s_i 的第一个字符必定是原先字符集排序之后的一个前缀。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。
要注意子问题中两个串的比较并非是字典序比较，而是在末尾加上 $+\infty$ 之后比较。
那么只需知道所有 $s_1, s_2, \cdots, s_k$ 即可。
显然，s_i 内部肯定是从小到大排。同时，所有 s_i 的第一个字符必定是原先字符集排序之后的一个前缀。
设我们有 x 个 c，$n-x$ 个其他字符。分类讨论：

- $x > n-x$ 时，把每个其他字符前面加上 $\left\lfloor \frac{x}{n-x+1} \right\rfloor$ 个 c；
- $x \le n-x$ 时，先用字符集排序后的前缀填进每个 s_i，然后在所有最大的 s_i 中继续填。

CF1975H

考虑最大的字符 c。显然，答案应当以 c 开头，以 c 结尾。
设答案为 $cs_1cs_2c\cdots cs_kc$。假设我们已经得到所有 $s_1, s_2, \cdots, s_k$，则求答案只需要递归一次即可。
要注意子问题中两个串的比较并非是字典序比较，而是在末尾加上 $+\infty$ 之后比较。
那么只需知道所有 $s_1, s_2, \cdots, s_k$ 即可。
显然，s_i 内部肯定是从小到大排。同时，所有 s_i 的第一个字符必定是原先字符集排序之后的一个前缀。
设我们有 x 个 c，$n-x$ 个其他字符。分类讨论：

- $x > n-x$ 时，把每个其他字符前面加上 $\left\lfloor \frac{x}{n-x+1} \right\rfloor$ 个 c；
- $x \le n-x$ 时，先用字符集排序后的前缀填进每个 s_i，然后在所有最大的 s_i 中继续填。

每次递归时，问题规模减半。因此复杂度为 $O(n \log n)$。

CF1975H

考虑优化。其实我们可以用迭代的方法干掉递归。

CF1975H

考虑优化。其实我们可以用迭代的方法干掉递归。
维护一个有序的字符串序列，一开始为 $A = (-\infty, s_1, s_2, \cdots, s_n)$
(不妨先将 s 排序)。

CF1975H

考虑优化。其实我们可以用迭代的方法干掉递归。
维护一个有序的字符串序列，一开始为 $A = (-\infty, s_1, s_2, \cdots, s_n)$ (不妨先将 s 排序)。
从左到右扫描序列 A，每次将 A_i 拼接到最靠左的最大值后面，然后删去 A_i，直到 A_n 的最后一个字符是 $-\infty$。

CF1975H

考虑优化。其实我们可以用迭代的方法干掉递归。
维护一个有序的字符串序列，一开始为 $A = (-\infty, s_1, s_2, \cdots, s_n)$ (不妨先将 s 排序)。
从左到右扫描序列 A，每次将 A_i 拼接到最靠左的最大值后面，然后删去 A_i，直到 A_n 的最后一个字符是 $-\infty$。
本质上，每次扫到的 A_i 以 $-\infty$ 结尾时，意味着完成了一次递归。每一次最大值位置跳回前面时相当于又给 s 加完一轮字符。

CF1975H

考虑优化。其实我们可以用迭代的方法干掉递归。
维护一个有序的字符串序列，一开始为 $A = (-\infty, s_1, s_2, \cdots, s_n)$ (不妨先将 s 排序)。
从左到右扫描序列 A，每次将 A_i 拼接到最靠左的最大值后面，然后删去 A_i，直到 A_n 的最后一个字符是 $-\infty$。
本质上，每次扫到的 A_i 以 $-\infty$ 结尾时，意味着完成了一次递归。每一次最大值位置跳回前面时相当于又给 s 加完一轮字符。
复杂度显然是 $O(n)$。

CF2053G

给定两个长度分别为 n 和 m 的字符串 S, T。
对每个 $1 \le k < n$，问 $S[1,k]$ 和 $S[k+1,n]$ 能否拼接得到 T。
$2 \le n, m \le 5 \times 10^6$。保证 S, T 仅包含小写字母。

CF2053G

不妨把 $S[1,k]$ 和 $S[k+1,n]$ 中短的叫做 s_1，长的叫做 s_2。

CF2053G

不妨把 $S[1,k]$ 和 $S[k+1,n]$ 中短的叫做 s_1，长的叫做 s_2。
考虑贪心匹配。设 $s_2 = s_1^k + c$，其中 c 不包含 s_1 作为前缀。我们每次先贪心匹配 s_1，到了匹配不了的地方就撤回 k 次匹配，然后尝试匹配 s_2。

CF2053G

不妨把 $S[1,k]$ 和 $S[k+1,n]$ 中短的叫做 s_1，长的叫做 s_2。
考虑贪心匹配。设 $s_2 = s_1^k + c$，其中 c 不包含 s_1 作为前缀。我们每次先贪心匹配 s_1，到了匹配不了的地方就撤回 k 次匹配，然后尝试匹配 s_2。
这种贪心什么时候是错的呢？比如 $s_1 = aba$，$s_2 = abaab$，$T = abaababa$。贪心匹配了两次 aba 之后需要多反悔一次。

CF2053G

不妨把 $S[1,k]$ 和 $S[k+1,n]$ 中短的叫做 s_1，长的叫做 s_2。
考虑贪心匹配。设 $s_2 = s_1^k + c$，其中 c 不包含 s_1 作为前缀。我们每次先贪心匹配 s_1，到了匹配不了的地方就撤回 k 次匹配，然后尝试匹配 s_2。
这种贪心什么时候是错的呢？比如 $s_1 = aba$，$s_2 = abaab$，$T = abaababa$。贪心匹配了两次 aba 之后需要多反悔一次。
是否会需要多反悔两次以上呢？其实是不可能的。如果要多反悔两次，则 $T = s_1^{k+2} + \cdots = s_1^k + c + s_1 + \cdots$。因此 s_1 和 c 必存在公共循环节。而这种有循环节的情况可以直接枚举，根本不需要贪心。

CF2053G

不妨把 $S[1,k]$ 和 $S[k+1,n]$ 中短的叫做 s_1，长的叫做 s_2。
考虑贪心匹配。设 $s_2 = s_1^k + c$，其中 c 不包含 s_1 作为前缀。我们每次先贪心匹配 s_1，到了匹配不了的地方就撤回 k 次匹配，然后尝试匹配 s_2。
这种贪心什么时候是错的呢？比如 $s_1 = aba$，$s_2 = abaab$，$T = abaababa$。贪心匹配了两次 aba 之后需要多反悔一次。
是否会需要多反悔两次以上呢？其实是不可能的。如果要多反悔两次，则 $T = s_1^{k+2} + \cdots = s_1^k + c + s_1 + \cdots$。因此 s_1 和 c 必存在公共循环节。而这种有循环节的情况可以直接枚举，根本不需要贪心。
因此暴力贪心就行了。复杂度是调和级数，即 $O(n \log n)$。

CF2053G

怎么做到线性?

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n} \log\left(\frac{n}{i}\right)\right) = O(n)$。

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n} \log\left(\frac{n}{i}\right)\right) = O(n)$。
接下来，因为长串的长度为 $O(n)$，所以只会填 $O\left(\frac{m}{n}\right)$ 次。考虑对填充短串进行加速。

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n}\log\left(\frac{n}{i}\right)\right)=O(n)$。
接下来，因为长串的长度为 $O(n)$，所以只会填 $O\left(\frac{m}{n}\right)$ 次。考虑对填充短串进行加速。
直接二分是不可行的。考虑 $s_1=a, s_2=b, T=abab\cdots ab$。这样就能卡到 log。

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n} \log\left(\frac{n}{i}\right)\right) = O(n)$。
接下来，因为长串的长度为 $O(n)$，所以只会填 $O\left(\frac{m}{n}\right)$ 次。考虑对填充短串进行加速。
直接二分是不可行的。考虑 $s_1 = a, s_2 = b, T = abab\cdots ab$。这样就能卡到 log。
可以把二分改为倍增（枚举倍增上界），这样总复杂度就是对的了。

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n}\log\left(\frac{n}{i}\right)\right)=O(n)$。
接下来，因为长串的长度为 $O(n)$，所以只会填 $O\left(\frac{m}{n}\right)$ 次。考虑对填充短串进行加速。
直接二分是不可行的。考虑 $s_1=a, s_2=b, T=abab\cdots ab$。这样就能卡到 log。
可以把二分改为倍增（枚举倍增上界），这样总复杂度就是对的了。
也可以设一个阈值 $B=\frac{n}{|s_1|}$。先二分 s_1^B，剩下的部分不超过 B 个，再二分一遍即可。

CF2053G

怎么做到线性?
首先，这个 k 是可以二分求出的，复杂度 $O\left(\sum_{i=1}^{n}\log\left(\frac{n}{i}\right)\right)=O(n)$。
接下来，因为长串的长度为 $O(n)$，所以只会填 $O\left(\frac{m}{n}\right)$ 次。考虑对填充短串进行加速。
直接二分是不可行的。考虑 $s_1=a, s_2=b, T=abab\cdots ab$。这样就能卡到 log。
可以把二分改为倍增（枚举倍增上界），这样总复杂度就是对的了。
也可以设一个阈值 $B=\frac{n}{|s_1|}$。先二分 s_1^B，剩下的部分不超过 B 个，再二分一遍即可。
这样就可以做到 $O(n+m)$。

基本子串结构

下面是本次讲课的第二部分：尝试把基本子串结构用人话说出来。

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。
也就是说，两个子串同为一个等价类，当且仅当：其中一个每次出现就会有另一个出现，且其中一个是另一个的后缀。

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。
也就是说，两个子串同为一个等价类，当且仅当：其中一个每次出现就会有另一个出现，且其中一个是另一个的后缀。
那假如去掉“其中一个是另一个的后缀”呢?

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。
也就是说，两个子串同为一个等价类，当且仅当：其中一个每次出现就会有另一个出现，且其中一个是另一个的后缀。
那假如去掉“其中一个是另一个的后缀”呢?
显然，每个等价类会有且仅有一个最长的字符串（代表元）。因此每个等价类在平面上都形如阶梯状。

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。
也就是说，两个子串同为一个等价类，当且仅当：其中一个每次出现就会有另一个出现，且其中一个是另一个的后缀。
那假如去掉“其中一个是另一个的后缀”呢?
显然，每个等价类会有且仅有一个最长的字符串（代表元）。因此每个等价类在平面上都形如阶梯状。
同时容易验证，阶梯型网格图的一行（r 相同）代表正串 SAM 上的一个结点，一列（l 相同）代表反串 SAM 上的一个结点。

等价类

首先我们知道 SAM 的思路：将所有子串按出现的右端点集合划分，成为等价类。
也就是说，两个子串同为一个等价类，当且仅当：其中一个每次出现就会有另一个出现，且其中一个是另一个的后缀。
那假如去掉“其中一个是另一个的后缀”呢?
显然，每个等价类会有且仅有一个最长的字符串（代表元）。因此每个等价类在平面上都形如阶梯状。
同时容易验证，阶梯型网格图的一行（r 相同）代表正串 SAM 上的一个结点，一列（l 相同）代表反串 SAM 上的一个结点。
因此每个等价类的行数和列数之和都是 $O(n)$ 的。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。
记正串的 parent 树为 T_0，反串的 parent 树为 T_1。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。
记正串的 parent 树为 T_0，反串的 parent 树为 T_1。
则 T_0 中的边可以看作是从块的左边界连向右边界，T_1 就是从上边界连向下边界。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。
记正串的 parent 树为 T_0，反串的 parent 树为 T_1。
则 T_0 中的边可以看作是从块的左边界连向右边界，T_1 就是从上边界连向下边界。
把树边对应到 SAM 上之后，考虑 SAM 中 DAG 的边。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。
记正串的 parent 树为 T_0，反串的 parent 树为 T_1。
则 T_0 中的边可以看作是从块的左边界连向右边界，T_1 就是从上边界连向下边界。
把树边对应到 SAM 上之后，考虑 SAM 中 DAG 的边。
对于一个正串 SAM 上的点（某一块的一行），如果它不是该块的上边界，则它有且仅有一条出边，连向它所在块的上面一行；否则，它的出边连向这块的所有 T_1 上的点的儿子对应的块的下边界。

连边

划分了等价类之后，要把 SAM 的 parent 树边连到网格图上。
记正串的 parent 树为 T_0，反串的 parent 树为 T_1。
则 T_0 中的边可以看作是从块的左边界连向右边界，T_1 就是从上边界连向下边界。
把树边对应到 SAM 上之后，考虑 SAM 中 DAG 的边。
对于一个正串 SAM 上的点（某一块的一行），如果它不是该块的上边界，则它有且仅有一条出边，连向它所在块的上面一行；否则，它的出边连向这块的所有 T_1 上的点的儿子对应的块的下边界。
对于反串同理。其实 DAG 已经完全被另一个 SAM 的树边替代了。

构建

首先可以 $O(n)$ 构建正反 SAM。

构建

首先可以 $O(n)$ 构建正反 SAM。
接下来分为几个步骤：识别代表元，划分等价类，排序行列。

识别代表元

首先我们要找出所有块的代表元。显然代表元只会是某个 SAM 结点中最长的字符串。

识别代表元

首先我们要找出所有块的代表元。显然代表元只会是某个 SAM 结点中最长的字符串。
因此我们可以对于正串的每个 SAM 结点，找到其最长字符串所在的反 SAM 结点。如果这两个结点的最长串长度恰好相同，则这个最长串就是代表元。

识别代表元

首先我们要找出所有块的代表元。显然代表元只会是某个 SAM 结点中最长的字符串。
因此我们可以对于正串的每个 SAM 结点，找到其最长字符串所在的反 SAM 结点。如果这两个结点的最长串长度恰好相同，则这个最长串就是代表元。
具体实现上，我们可以对正串 SAM 做一遍 dfs，只沿着 $len_v = len_u + 1$ 的边 (u, v) 走。同时维护反 SAM 上对应的结点。
因为每次转移只会向后加一个字符，所以反 SAM 上只会走树边。

识别代表元

首先我们要找出所有块的代表元。显然代表元只会是某个 SAM 结点中最长的字符串。
因此我们可以对于正串的每个 SAM 结点，找到其最长字符串所在的反 SAM 结点。如果这两个结点的最长串长度恰好相同，则这个最长串就是代表元。
具体实现上，我们可以对正串 SAM 做一遍 dfs，只沿着 $len_v = len_u + 1$ 的边 (u, v) 走。同时维护反 SAM 上对应的结点。
因为每次转移只会向后加一个字符，所以反 SAM 上只会走树边。
为了处理这个过程，需要预处理反 SAM 上每个点加上每个字符转移到哪个点，这是容易的。

划分等价类

接下来需要确定不包含代表元的结点分别在哪个等价类。

划分等价类

接下来需要确定不包含代表元的结点分别在哪个等价类。
之前我们证明过，这些结点在 SAM 的 DAG 上只会有一条出边。

划分等价类

接下来需要确定不包含代表元的结点分别在哪个等价类。
之前我们证明过，这些结点在 SAM 的 DAG 上只会有一条出边。
那么其实我们只需要一个合理的枚举顺序，使得枚举到某个未知等价类的结点时，它的唯一一条出边所能到的点所在的等价类已经被确定了。

划分等价类

接下来需要确定不包含代表元的结点分别在哪个等价类。
之前我们证明过，这些结点在 SAM 的 DAG 上只会有一条出边。
那么其实我们只需要一个合理的枚举顺序，使得枚举到某个未知等价类的结点时，它的唯一一条出边所能到的点所在的等价类已经被确定了。
显然按 len 排序是对的。那么这样也是 $O(n)$ 的。

排序行列

接下来我们要将同一个等价类里的结点按照行列排序，这是简单的。

排序行列

接下来我们要将同一个等价类里的结点按照行列排序，这是简单的。
而且也可以直接按照编号顺序插入，因为 SAM 的增量构造性质，同一个等价类内的结点天然按照编号排序。

排序行列

接下来我们要将同一个等价类里的结点按照行列排序，这是简单的。
而且也可以直接按照编号顺序插入，因为 SAM 的增量构造性质，同一个等价类内的结点天然按照编号排序。
至此，我们在 $O(n)$ 的时间复杂度内完成了基本子串结构的构建。

排序行列

接下来我们要将同一个等价类里的结点按照行列排序，这是简单的。
而且也可以直接按照编号顺序插入，因为 SAM 的增量构造性质，同一个等价类内的结点天然按照编号排序。
至此，我们在 $O(n)$ 的时间复杂度内完成了基本子串结构的构建。
接下来分享几道关于基本子串结构的例题。

CF1817F

给定一个长度为 n 的字符串 S。
定义一个非空子串对 (A, B) 合法，当且仅当存在可空子串 C，使得 A 和 B 每次出现都是以 ACB 形式出现。
求有多少对合法的 (A, B)。
$1 \leq n \leq 10^5$。
保证 S 仅包含小写字母。

CF1817F

(A, B) 合法当且仅当它们在同一个等价类中，且在代表元中出现的位置不交。

CF1817F

(A, B) 合法当且仅当它们在同一个等价类中，且在代表元中出现的位置不交。
于是对每个阶梯形做一遍二维偏序即可。

CF1817F

(A, B) 合法当且仅当它们在同一个等价类中，且在代表元中出现的位置不交。
于是对每个阶梯形做一遍二维偏序即可。
时间复杂度 $O(n \log n)$。

P8351

给定一个长度为 n 的字符串 S。
初始有一个串 $T_0 = S$。你要操作 n 次。第 i 次操作，你可以删除 T_{i-1} 的开头或结尾的字符，得到 T_i。
定义 $occ(T)$ 为 T 在 S 中的出现次数。定义操作方案的权值为

$$\prod_{i=0}^{n} occ(T_i)$$

求所有操作方案的权值之和。
$1 \le n \le 10^5$。
保证 S 仅包含小写字母。

P8351

注意到 T 一定是 S 的子串，因此可以对 S 的所有本质不同子串进行 DP。

P8351

注意到 T 一定是 S 的子串，因此可以对 S 的所有本质不同子串进行 DP。
由于本质不同子串数量是 $O(n^2)$ 的，因此复杂度也为 $O(n^2)$。

P8351

注意到 T 一定是 S 的子串，因此可以对 S 的所有本质不同子串进行 DP。
由于本质不同子串数量是 $O(n^2)$ 的，因此复杂度也为 $O(n^2)$。
然而处于相同等价类中的 DP 值可以一起计算。由于阶梯形网格图的性质，只要我们知道右下边界上所有格子的 DP 值，就能用分治和卷积求出左上边界所有格子的 DP 值。

P8351

注意到 T 一定是 S 的子串，因此可以对 S 的所有本质不同子串进行 DP。
由于本质不同子串数量是 $O(n^2)$ 的，因此复杂度也为 $O(n^2)$。
然而处于相同等价类中的 DP 值可以一起计算。由于阶梯形网格图的性质，只要我们知道右下边界上所有格子的 DP 值，就能用分治和卷积求出左上边界所有格子的 DP 值。
于是建出基本子串结构之后，按顺序求解每个等价类即可。

P8351

注意到 T 一定是 S 的子串，因此可以对 S 的所有本质不同子串进行 DP。
由于本质不同子串数量是 $O(n^2)$ 的，因此复杂度也为 $O(n^2)$。
然而处于相同等价类中的 DP 值可以一起计算。由于阶梯形网格图的性质，只要我们知道右下边界上所有格子的 DP 值，就能用分治和卷积求出左上边界所有格子的 DP 值。
于是建出基本子串结构之后，按顺序求解每个等价类即可。
复杂度 $O(n \log^2 n)$。

QOJ5014

给定一个长度为 n 的字符串 S。
第 i 个字符有左权值 wl_i 和右权值 wr_i。定义子串的左权值为所有出现位置左端点的 wl 之和，右权值同理。定义子串的复读程度 $w(l,r)$ 为其左权值与右权值的乘积。
q 次询问，每次给定 l_1, r_1, l_2, r_2，求所有以 $s[l_1, r_1]$ 为前缀、$s[l_2, r_2]$ 为后缀的子串的复读程度之和。
$1 \leq n, q \leq 10^5$。
保证 S 仅包含小写字母。

QOJ5014

一个串的复读程度是其所在正 SAM 节点的右权值乘以所在反 SAM 节点的左权值。

QOJ5014

一个串的复读程度是其所在正 SAM 节点的右权值乘以所在反 SAM 节点的左权值。
以 $s[l_1, r_1]$ 为前缀的子串均在 T_1 的对应结点的子树内，以 $s[l_2, r_2]$ 为后缀的子串均在 T_0 的对应结点的子树内。

QOJ5014

一个串的复读程度是其所在正 SAM 节点的右权值乘以所在反 SAM 节点的左权值。
以 $s[l_1, r_1]$ 为前缀的子串均在 T_1 的对应结点的子树内，以 $s[l_2, r_2]$ 为后缀的子串均在 T_0 的对应结点的子树内。
然而实际上对应的结点上会有一些子串不能被包含，因为长度小于 $r_1 - l_1 + 1$ 或 $r_2 - l_2 + 1$。

QOJ5014

一个串的复读程度是其所在正 SAM 节点的右权值乘以所在反 SAM 节点的左权值。
以 $s[l_1, r_1]$ 为前缀的子串均在 T_1 的对应结点的子树内，以 $s[l_2, r_2]$ 为后缀的子串均在 T_0 的对应结点的子树内。
然而实际上对应的结点上会有一些子串不能被包含，因为长度小于 $r_1 - l_1 + 1$ 或 $r_2 - l_2 + 1$。
于是查询可以分为两部分：求这两棵子树的交集信息，以及在交集中却长度不足的串的信息。

QOJ5014

考虑求交集信息，跑出 dfs 序之后是一个矩形查询。应用莫队二次离线，会变为 $O(q)$ 次询问。每次给定 x_1, x_2, y，查询 $\sum_{1\le l\le r\le n}[x_1 \le dfn(T_0(l,r)) \le x_2, dfn(T_1(l,r)) \le y]w(l,r)$。这里所有 $[x_1, x_2]$ 的区间长度之和为 $O(q\sqrt{n})$。

QOJ5014

考虑求交集信息，跑出 dfs 序之后是一个矩形查询。应用莫队二次离线，会变为 $O(q)$ 次询问。每次给定 x_1, x_2, y，查询 $\sum_{1\le l\le r\le n}[x_1 \le dfn(T_0(l,r)) \le x_2, dfn(T_1(l,r)) \le y]w(l,r)$。这里所有 $[x_1, x_2]$ 的区间长度之和为 $O(q\sqrt{n})$。

对 y 做扫描线，每次添加一个 T_1 上的结点，对 T_0 的影响为一段在网格图某一块中连续的节点。

QOJ5014

考虑求交集信息，跑出 dfs 序之后是一个矩形查询。应用莫队二次离线，会变为 $O(q)$ 次询问。每次给定 x_1, x_2, y，查询 $\sum_{1\le l\le r\le n}[x_1 \le dfn(T_0(l,r)) \le x_2, dfn(T_1(l,r)) \le y]w(l,r)$。这里所有 $[x_1, x_2]$ 的区间长度之和为 $O(q\sqrt{n})$。

对 y 做扫描线，每次添加一个 T_1 上的结点，对 T_0 的影响为一段在网格图某一块中连续的节点。

考虑将所有 T_0 上的点重标号，使得每块中的点编号连续。

QOJ5014

以这个编号为下标，维护序列 A, B。其中 A_i 固定为编号为 i 的 T_0 上的点权。

QOJ5014

以这个编号为下标，维护序列 A, B。其中 A_i 固定为编号为 i 的 T_0 上的点权。
修改时的操作为：对某个编号区间，令其中所有 $B_i \leftarrow B_i + cA_i$，其中 c 为 T_1 上的点权乘以对应的 occ。

QOJ5014

以这个编号为下标，维护序列 A, B。其中 A_i 固定为编号为 i 的 T_0 上的点权。
修改时的操作为：对某个编号区间，令其中所有 $B_i \leftarrow B_i + cA_i$，其中 c 为 T_1 上的点权乘以对应的 occ。
查询时，枚举 $x_1 \leq i \leq x_2$，找到 $dfn(u) = i$ 的 u，查询 B_{id_u} 的值，其中 id_u 表示 u 重标号后的编号。

QOJ5014

以这个编号为下标，维护序列 A, B。其中 A_i 固定为编号为 i 的 T_0 上的点权。
修改时的操作为：对某个编号区间，令其中所有 $B_i \leftarrow B_i + cA_i$，其中 c 为 T_1 上的点权乘以对应的 occ。
查询时，枚举 $x_1 \leq i \leq x_2$，找到 $dfn(u) = i$ 的 u，查询 B_{id_u} 的值，其中 id_u 表示 u 重标号后的编号。
分块维护 $O(\sqrt{n})$ 修改 $O(1)$ 查询即可。这一部分的复杂度为 $O(n\sqrt{q})$。

QOJ5014

考虑减去在交集中却长度不足的串的信息。我们分别算出在 T_0 和 T_1 对应结点上长度不足的串的信息，再减去这两个结点的交集。

QOJ5014

考虑减去在交集中却长度不足的串的信息。我们分别算出在 T_0 和 T_1 对应结点上长度不足的串的信息，再减去这两个结点的交集。
显然，两个结点的交集至多只有一个串，可以 $O(1)$ 求出。

QOJ5014

考虑减去在交集中却长度不足的串的信息。我们分别算出在 T_0 和 T_1 对应结点上长度不足的串的信息，再减去这两个结点的交集。

显然，两个结点的交集至多只有一个串，可以 $O(1)$ 求出。

只考虑其中一侧，问题变为：求有多少个 T_0 上和 $S[l_2, r_2]$ 在同一个结点，但长度不足 $r_2 - l_2 + 1$，且在 $S[l_1, r_1]$ 的 T_1 上的子树中。

QOJ5014

考虑减去在交集中却长度不足的串的信息。我们分别算出在 T_0 和 T_1 对应结点上长度不足的串的信息，再减去这两个结点的交集。

显然，两个结点的交集至多只有一个串，可以 $O(1)$ 求出。

只考虑其中一侧，问题变为：求有多少个 T_0 上和 $S[l_2, r_2]$ 在同一个结点，但长度不足 $r_2 - l_2 + 1$，且在 $S[l_1, r_1]$ 的 T_1 上的子树中。

这是一个二位数点问题，可以直接解决。

QOJ5014

考虑减去在交集中却长度不足的串的信息。我们分别算出在 T_0 和 T_1 对应结点上长度不足的串的信息，再减去这两个结点的交集。

显然，两个结点的交集至多只有一个串，可以 $O(1)$ 求出。

只考虑其中一侧，问题变为：求有多少个 T_0 上和 $S[l_2, r_2]$ 在同一个结点，但长度不足 $r_2 - l_2 + 1$，且在 $S[l_1, r_1]$ 的 T_1 上的子树中。

这是一个二位数点问题，可以直接解决。

总复杂度 $O(n\sqrt{q} + q \log n)$。

总结

篇幅所限，水平不足，抛砖引玉，别的忘了。
感谢大家的聆听，希望大家能学到更多。