字符串速通

xay5421

2025 年 2 月 10 日

目录

目录

定义 1.1

(回文串). 满足 $\forall 1 \le i \le |S|, S[i] = S[|S| + 1 - i]$ 的字符串 S 是回文串。

定义 1.2

(回文中心). 对于 S 的一个回文子串 $S[l \dots r]$，称它的回文中心是 $\frac{l+r}{2}$。

定义 1.3

(回文半径). 字符串 S 中，位置 i 的回文半径是最长的 x 满足 $S[i - x + 1; i + x - 1]$ 回文。

manacher

在处理回文子串问题时，有时要求出字符串的回文半径，manacher 算法可以做到线性时间处理出字符串每个位置的回文半径。

manacher

在处理回文子串问题时，有时要求出字符串的回文半径，manacher 算法可以做到线性时间处理出字符串每个位置的回文半径。
为了避免分奇偶讨论和边界问题，我们在字符串相邻字符之间加入一个特殊字符，再在字符串开头和末尾添加一个不同的特殊字符。
如 aabbacab 变成 $#a#a#b#b#a#c#a#b#@。

manacher

在处理回文子串问题时，有时要求出字符串的回文半径，manacher 算法可以做到线性时间处理出字符串每个位置的回文半径。
为了避免分奇偶讨论和边界问题，我们在字符串相邻字符之间加入一个特殊字符，再在字符串开头和末尾添加一个不同的特殊字符。
如 aabbacab 变成 $\$\#a\#a\#b\#b\#a\#c\#a\#b\#@$。
由此我们得到了一个新的字符串 S，我们要求出对每个 i 求出 $R[i]$ 表示位置 i 的回文半径。

manacher

在处理回文子串问题时，有时要求出字符串的回文半径，manacher 算法可以做到线性时间处理出字符串每个位置的回文半径。
为了避免分奇偶讨论和边界问题，我们在字符串相邻字符之间加入一个特殊字符，再在字符串开头和末尾添加一个不同的特殊字符。
如 aabbacab 变成 $#a#a#b#b#a#c#a#b#@。
由此我们得到了一个新的字符串 S，我们要求出对每个 i 求出 $R[i]$ 表示位置 i 的回文半径。

表 1: S 和对应的 R

S	#	a	#	a	#	b	#	b	#	a	#	c	#	a	#	b	#
R	1	2	3	2	1	2	5	2	1	2	1	6	1	2	1	2	1

manacher（算法流程）

我们从小到大枚举 i 并计算 $R[i]$，维护两个辅助变量 r 和 pos，表示已求出的回文串覆盖到的最右边界和对应的回文中心。
计算 $R[i]$ 时，我们可以给 $R[i]$ 一个下界，分以下两种情况讨论：

1. $r < i$ 时，有 $R[i] \geq 1$。
2. $r \geq i$ 时，有 $R[i] \geq \min(R[2pos - i], r - i + 1)$。因为 i 和 $2pos - i$ 关于 pos 对称，如果只考虑 $S[pos - R[pos] + 1; pos + R[pos] - 1]$，它们的回文半径是相同的，而在这个子串中，$2pos - i$ 的回文半径是 $\min(R[2pos - i], r - i + 1)$。

有了 $R[i]$ 的下界，我们可以不断增加 $R[i]$，直到 $S[i - R[i]; i + R[i]]$ 不回文，由此得到正确的 $R[i]$。
再用当前的 $i + R[i] - 1$ 和 i 更新 r 和 pos。

manacher

```
void manacher(char *s) {
    int n = strlen(s), r = 0, pos = 0;
    for (int i = 1; i <= n; i++) {
        if (r > i) R[i] = min(R[2 * pos - i], r - i);
        else R[i] = 1;
        while (i - R[i] >= 1 && i + R[i] <= n && s[i - R[i]] == s[i + R[i]]) R[i]++;
        if (i + R[i] - 1 > r) r = i + R[i] - 1, pos = i;
    }
}
```

回文树（结构）

一个字符串 S 的回文树/自动机由两棵树组成，设两棵树的根分别为奇根和偶根，回文树上除了根每个节点表示一个回文串，每个节点对应的字符串是父亲对应的字符串两端添加边对应的字符。特别的，偶根对应的字符串是空串，奇根对应的字符串是长度为 -1 的不存在的字符串，它的孩子对应的字符串是边对应的字符。同时，回文树每个节点有失配指针 $fail_i$，指向这个节点对应字符串的最长回文后缀对应节点。

回文树（构造）

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

回文树（构造）

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

引理 1.1

Sc 中不在 S 中出现的最多只有一个回文串，且是最长回文后缀。

回文树（构造）

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

引理 1.1

Sc 中不在 S 中出现的最多只有一个回文串，且是最长回文后缀。

证明

Sc 比 S 多的回文串肯定含有 c，因此必然是回文后缀，考虑 Sc 的一个回文后缀，若这个回文后缀不是最长的，那么这个回文后缀以最长回文后缀的中心对称必然出现在 S 中。

回文树（构造）

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

引理 1.1

Sc 中不在 S 中出现的最多只有一个回文串，且是最长回文后缀。

证明

Sc 比 S 多的回文串肯定含有 c，因此必然是回文后缀，考虑 Sc 的一个回文后缀，若这个回文后缀不是最长的，那么这个回文后缀以最长回文后缀的中心对称必然出现在 S 中。

因此我们只要找最长回文后缀，如果没有这个回文串对应节点，为这个回文串新建一个节点，并维护出其回文树的父亲和失配指针。

回文树（构造）

我们使用增量构造的方法构造回文树，假设现在已经构造出 S 的回文树，现在我们要在 S 的末尾增加一个字符 c，并维护出 Sc 的回文树。

引理 1.1

Sc 中不在 S 中出现的最多只有一个回文串，且是最长回文后缀。

证明

Sc 比 S 多的回文串肯定含有 c，因此必然是回文后缀，考虑 Sc 的一个回文后缀，若这个回文后缀不是最长的，那么这个回文后缀以最长回文后缀的中心对称必然出现在 S 中。

因此我们只要找最长回文后缀，如果没有这个回文串对应节点，为这个回文串新建一个节点，并维护出其回文树的父亲和失配指针。

我们沿着 S 最长回文后缀对应节点的失配指针走，直到找到节点 u 满足 $S_{|S|-\text{len}(u)-1} = c$。如果 u 节点没有字符 c 对应孩子，新建节点 v 表示 u 在两端加上字符 c 得到的回文串，$fail_v$ 为 $ch_{w,c}$ 其中 $w \neq u$ 是沿着 u 的 $fail$ 指针走找到的第一个节点使得 $S_{|S|-\text{len}(w)-1} = c$。

回文树（构造）

```
int node(int l) {  // 建立一个新节点，长度为 l
  sz++;
  memset(ch[sz], 0, sizeof(ch[sz]));
  len[sz] = l, fail[sz] = 0;
  return sz;
}

void clear() {  // 初始化
  sz = -1, last = 0, s[tot = 0] = '$', node(0), node(-1), fail[0] = 1;
}

int getfail(int x) {  // 找后缀回文
  while (s[tot - len[x] - 1] != s[tot]) x = fail[x];
  return x;
}

void insert(char c) {  // 建树
  s[++tot] = c;
  int now = getfail(last);
  if (!ch[now][c - 'a']) {
    int x = node(len[now] + 2);
    fail[x] = ch[getfail(fail[now])][c - 'a'];
    ch[now][c - 'a'] = x;
  }
  last = ch[now][c - 'a'];
}
```

回文树（前端加入）

回文树除了可以支持后端加入，还能支持前端加入。

回文树（前端加入）

回文树除了可以支持后端加入，还能支持前端加入。
前端加入我们只需要维护最长回文前缀对应节点 u，由于每个节点对应的都是一个回文串，$fail_u$ 指向 u 的最长回文后缀节点，同时也指向最长回文前缀节点，并且根据引理，最多只会新增一个本质不同的回文串。因此与后端加入过程几乎相同。

APIO2014 回文串

例题 1.1

给定一个字符串 S，求 S 中出现次数乘长度最大的回文串。
$|S| \leq 3 \times 10^5$。

APIO2014 回文串

例题 1.1

给定一个字符串 S，求 S 中出现次数乘长度最大的回文串。
$|S| \leq 3 \times 10^5$。

题解

我们建出 S 的回文树，要维护每个节点代表的回文串的出现次数，我们对每个位置 i，找到 i 最长回文后缀对应节点，然后将这个节点沿着失配指针到根路径上节点的出现次数加一。将这个过程用树上前缀和优化即可。最后遍历每个不同的回文串，更新答案。

回文后缀理论

定理 1.1

字符串 S 的所有回文后缀按照长度排序后，可以划分成 $O(\log |S|)$ 段等差数列。

证明

任意回文后缀是最长回文后缀的 *border*，而任意最长回文后缀的 *border* 是回文后缀。而 *border* 理论部分讲过 *border* 可以划分成 $O(\log |S|)$ 段等差数列，因此回文后缀也能这样划分。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。
新串的 S' 的回文后缀将会是:

1. 单独一个字符 c。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。
新串的 S' 的回文后缀将会是:

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。
新串的 S' 的回文后缀将会是:

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

第一种情况新增的回文后缀就是单独一个字符 c。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。
新串的 S' 的回文后缀将会是:

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

第一种情况新增的回文后缀就是单独一个字符 c。
第二种情况，考虑原先的一个极长的等差数列，将这个等差数列表示的串表示成: $A, BA, BBA, B^3A, \dots B^kA$。此时如果 B 的最后一个字符是 c，那么这个等差数列除了 B^kA 都出现在新串的回文后缀中；如果不是 c，那么这个等差数列除了 B^kA 都不出现在新串的回文后缀中。而最后一项只需要检验它左边的字符是否是 c 即可判断它是否出现在新串回文后缀中。

回文后缀理论

因此我们可以通过 $O(\log)$ 个等差数列的形式显示的维护出回文后缀。
而这个 $O(\log)$ 个等差数列是可以增量维护的。(即支持末尾加并维护)
我们用等差数列的形式记录了字符串 S 的所有回文后缀，现在我们要在 S 的末尾加入字符 c，求新的字符串 $S' = Sc$ 的所有回文后缀。
新串的 S' 的回文后缀将会是:

1. 单独一个字符 c。
2. 原先的回文后缀在两端同时增加字符 c。

第一种情况新增的回文后缀就是单独一个字符 c。
第二种情况，考虑原先的一个极长的等差数列，将这个等差数列表示的串表示成: $A, BA, BBA, B^3A, \dots B^kA$。此时如果 B 的最后一个字符是 c，那么这个等差数列除了 B^kA 都出现在新串的回文后缀中；如果不是 c，那么这个等差数列除了 B^kA 都不出现在新串的回文后缀中。而最后一项只需要检验它左边的字符是否是 c 即可判断它是否出现在新串回文后缀中。
做完这个，再把相邻的公差相同的等差数列合并成一个等差数列，以保证等差数列是 $O(\log |S|)$ 段。

回文后缀理论

往前加入字符是更简单的，新增回文后缀只可能是整个串，只需判断是否回文，然后合并等差数列即可。

回文后缀理论

往前加入字符是更简单的，新增回文后缀只可能是整个串，只需判断是否回文，然后合并等差数列即可。
如果只是末尾加入字符并维护等差数列，可以直接用回文树，每个点维护最后一个等差数列和跳过这个等差数列会到达的点。

CF932G Palindrome Partition

例题 1.2

给定一个串 $S(|S| \leq 10^7)$，要求把串分为偶数段，设分为了 $S = S_1 S_2 \dots S_k$。
求满足 $S_1 = S_k, S_2 = S_{k-1} \dots S_i = Sn + 1 - i$ 的方案数，对 $10^9 + 7$ 取模。

CF932G Palindrome Partition

例题 1.2

给定一个串 $S(|S| \leq 10^7)$，要求把串分为偶数段，设分为了 $S = S_1 S_2 \dots S_k$。
求满足 $S_1 = S_k, S_2 = S_{k-1} \dots S_i = Sn + 1 - i$ 的方案数，对 $10^9 + 7$ 取模。

题解

将 S 的后半部分翻转，然后逐个插入 S 的前一半的相邻两个之间，那么字符串将变为：$S[1]S[n]S[2]S[n-1] \dots$。我们发现，原划分的方案恰好对应了对新串进行偶数长度的回文划分的方案，于是问题转化为求偶数长度的回文划分方案。

$$f_i = \sum_{j=0}^{i-1} f_j \times [S[j+1;i] \text{ 回文}] \quad (f \text{ 的所有奇数位置为 } 0)$$

CF932G Palindrome Partition

例题 1.2

给定一个串 $S(|S| \leq 10^7)$，要求把串分为偶数段，设分为了 $S = S_1 S_2 \dots S_k$。
求满足 $S_1 = S_k, S_2 = S_{k-1} \dots S_i = Sn + 1 - i$ 的方案数，对 $10^9 + 7$ 取模。

题解

将 S 的后半部分翻转，然后逐个插入 S 的前一半的相邻两个之间，那么字符串将变为：$S[1]S[n]S[2]S[n-1] \dots$。我们发现，原划分的方案恰好对应了对新串进行偶数长度的回文划分的方案，于是问题转化为求偶数长度的回文划分方案。

$$f_i = \sum_{j=0}^{i-1} f_j \times [S[j+1;i] \text{ 回文}] \quad (f \text{ 的所有奇数位置为 } 0)$$

根据转移方程，我们需要维护的是一个等差数列位置的 f 之和，注意到这个等差数列去掉末项是维护过的，直接继承在加入末项的贡献即可，如果回文树写法上就是从父亲继承，如果是显示维护等差数列写法就是从等差数列开头的位置继承。

CF1738H Palindrome Addicts

例题 1.3

你要维护一个字符串 S，初始为空，有 $Q(Q \leq 10^6)$ 次操作，每次可以：

1. *push* c：往 S 末尾插入字符 c。
2. *pop*：删除 S 的第一个字符，保证 S 非空。

每次操作后询问 S 有多少本质不同的回文串。

CF1738H Palindrome Addicts

例题 1.3

你要维护一个字符串 S，初始为空，有 $Q(Q \leq 10^6)$ 次操作，每次可以：

1. *push* c：往 S 末尾插入字符 c。
2. *pop*：删除 S 的第一个字符，保证 S 非空。

每次操作后询问 S 有多少本质不同的回文串。

题解

我们发现一次操作答案改变量不超过 1，对于 *push* 我们只需要找到最长回文后缀，查询是否在加入前的字符串中出现过。对于 *pop* 我们只需要找到最长回文前缀，查询是否在删除后的字符串中出现过。

用回文树维护，每个节点维护这个点对应回文串最后出现位置，加入时更新一个点到根。每次更新到根复杂度是错的，但是可以懒惰更新，只有在删除时才把儿子的信息更新到父亲上。

区间最长回文串

例题 1.4

给出一个长度为 n 的字符串 S，有 Q 个询问，每次给定区间 $[l, r]$，查询 $S[l; r]$ 中的最长回文串。

$1 \leq n \leq 2 \times 10^5, Q \leq 2 \times 10^5$

区间最长回文串

例题 1.4

给出一个长度为 n 的字符串 S，有 Q 个询问，每次给定区间 $[l, r]$，查询 $S[l; r]$ 中的最长回文串。
$1 \leq n \leq 2 \times 10^5, Q \leq 2 \times 10^5$

引理 1.2

找到 $S[l \dots r]$ 的最长回文前缀 $S[l; i]$ 和最长回文后缀 $S[j; r]$，如果一个回文串的中心 c 小于 $\frac{l+i}{2}$ 或者大于 $\frac{j+r}{2}$，则它不可能成为区间最长回文串。

证明

根据中心位置可判断这些字符串的长度没有最长回文前缀/后缀长，因此不可能成为区间最长回文串。

区间最长回文串

例题 1.4

给出一个长度为 n 的字符串 S，有 Q 个询问，每次给定区间 $[l, r]$，查询 $S[l; r]$ 中的最长回文串。
$1 \leq n \leq 2 \times 10^5, Q \leq 2 \times 10^5$

引理 1.2

S 的任意一个中心位置在区间 $[\frac{l+i}{2}, \frac{j+r}{2}]$ 内的回文子串完全在 $[l, r]$ 之间。

证明

反证，假设有一个回文子串 $S[L; R]$ 中心在区间 $[\frac{l+i}{2}, \frac{j+r}{2}]$ 内，且 $\frac{L+R}{2} \leq \frac{l+r}{2}$，且 $L < l$，那么 $S[l; r + l - L]$ 必然是回文前缀，且长度比 $S[l; r]$ 的最长回文前缀长(因为中心位置更靠右)，矛盾。

区间最长回文串

例题 1.4

给出一个长度为 n 的字符串 S，有 Q 个询问，每次给定区间 $[l, r]$，查询 $S[l; r]$ 中的最长回文串。
$1 \leq n \leq 2 \times 10^5, Q \leq 2 \times 10^5$

题解

有了这两个引理，我们只需要求出 $S[l; r]$ 的最长回文前缀和最长回文后缀，用 manacher 算法预处理出回文半径 $R[i]$，查询 $R[i]$ 对应区间最大值即可。
求最长回文前后缀可以离线回文树，然后倍增。或者树上线性并查集做到线性时间复杂度。
查询区间 $R[i]$ 最大值是经典的 RMQ 问题，可以直接使用 ST 表做到 $O(n \log n)$ 预处理 $O(1)$ 查询，也可以按照 $\log n$ 大小分块，然后外层使用 ST 表，内层状压单调栈做到 $O(n)$ 预处理，$O(1)$ 查询。
所以此题可以做到线性！

区间带修回文后缀查询

例题 1.5

给定一个只含小写字母的字符串 S，有 Q 次操作，操作有以下两种：

1. 给出两个整数 i 和 c：将 S_i 修改为 c（$1 \le i \le |S|$，c 是小写字母）。
2. 给出两个整数 l 和 r：查询 $S[l...r]$ 有多少回文后缀（$1 \le l \le r \le |S|$）。

$1 \le |S| \le 2 \times 10^5, 1 \le Q \le 2 \times 10^5$。

区间带修回文后缀查询

例题 1.5

给定一个只含小写字母的字符串 S，有 Q 次操作，操作有以下两种：

1. 给出两个整数 i 和 c：将 S_i 修改为 c（$1 \le i \le |S|$，c 是小写字母）。
2. 给出两个整数 l 和 r：查询 $S[l...r]$ 有多少回文后缀（$1 \le l \le r \le |S|$）。

$1 \le |S| \le 2 \times 10^5, 1 \le Q \le 2 \times 10^5$。

这是我的集训队互测，是一个显示维护回文等差数列战胜回文树的例子。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S + T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S + T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S + T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。
3. 处于 S, T 之间：只有一个串。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S + T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。
3. 处于 S, T 之间：只有一个串。
4. 处于 S：关键的是 S 后缀的等差数列。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S+T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。
3. 处于 S, T 之间：只有一个串。
4. 处于 S：关键的是 S 后缀的等差数列。

考虑情况 2，根据条件，新串回文后缀的中线肯定也是 T 的某个回文前缀的中线，因此新串回文后缀肯定是从 T 中回文前缀两边同时扩展得到的。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S+T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。
3. 处于 S, T 之间：只有一个串。
4. 处于 S：关键的是 S 后缀的等差数列。

考虑情况 2，根据条件，新串回文后缀的中线肯定也是 T 的某个回文前缀的中线，因此新串回文后缀肯定是从 T 中回文前缀两边同时扩展得到的。
设 $T = BA^kU$，A 不是 U 的前缀。

区间带修回文后缀查询

用线段树维护回文前缀，回文后缀的等差数列，如何合并？
当合并 S, T 时，讨论 $S+T$ 的回文后缀的中线处于何处。

1. 处于 T 且完全包含于 T：继承 T 的信息。
2. 处于 T 且包含 S 的一个后缀：此时关键的是 T 前缀的等差数列。
3. 处于 S, T 之间：只有一个串。
4. 处于 S：关键的是 S 后缀的等差数列。

考虑情况 2，根据条件，新串回文后缀的中线肯定也是 T 的某个回文前缀的中线，因此新串回文后缀肯定是从 T 中回文前缀两边同时扩展得到的。
设 $T = BA^kU$，A 不是 U 的前缀。
分类讨论：

- U 不是 A 的前缀：新回文串只可能以等差数列的其中一项 $BA^{k'}$ 为中心。
- U 是 A 的前缀，新回文串可以以一个等差数列的后缀为中心（只取决于 S）。后缀长度可以二分哈希得到。

情况 4 与加单个字符类似，新串的回文后缀会从 S 的回文后缀中向两端扩展得到。原先是向两端加入单个字符 c，现在开头加入 T 的反串，末尾加 T。

情况 4 与加单个字符类似，新串的回文后缀会从 S 的回文后缀中向两端扩展得到。
原先是向两端加入单个字符 c，现在开头加入 T 的反串，末尾加 T。
设 S 等差数列的一段为 A^kB，只需讨论 T 是否形如 $\hat{A}^k\hat{A}'$（$\hat{A}$ 定义为 A 的反串，$\hat{A}'$ 表示 $\hat{A}$ 的一个前缀）。

情况 4 与加单个字符类似，新串的回文后缀会从 S 的回文后缀中向两端扩展得到。
原先是向两端加入单个字符 c，现在开头加入 T 的反串，末尾加 T。
设 S 等差数列的一段为 A^kB，只需讨论 T 是否形如 $\hat{A}^k\hat{A}'$（$\hat{A}$ 定义为 A 的反串，$\hat{A}'$ 表示 $\hat{A}$ 的一个前缀）。

- 如果不是，只可能以等差数列的一项 $A^{k'}B$ 为中心。
- 如果是，可以以一段等差数列的前缀为中心。也需要二分哈希求这个前缀长度。

情况 4 与加单个字符类似，新串的回文后缀会从 S 的回文后缀中向两端扩展得到。原先是向两端加入单个字符 c，现在开头加入 T 的反串，末尾加 T。
设 S 等差数列的一段为 A^kB，只需讨论 T 是否形如 $\hat{A}^k\hat{A}'$（$\hat{A}$ 定义为 A 的反串，$\hat{A}'$ 表示 $\hat{A}$ 的一个前缀）。

- 如果不是，只可能以等差数列的一项 $A^{k'}B$ 为中心。
- 如果是，可以以一段等差数列的前缀为中心。也需要二分哈希求这个前缀长度。

枚举等差数列，再在等差数列上二分，精细实现复杂度是 $O(\log n)$（二分使用倍增的二分）。

情况 4 与加单个字符类似，新串的回文后缀会从 S 的回文后缀中向两端扩展得到。
原先是向两端加入单个字符 c，现在开头加入 T 的反串，末尾加 T。
设 S 等差数列的一段为 A^kB，只需讨论 T 是否形如 $\hat{A}^k\hat{A}'$（$\hat{A}$ 定义为 A 的反串，$\hat{A}'$ 表示 $\hat{A}$ 的一个前缀）。

- 如果不是，只可能以等差数列的一项 $A^{k'}B$ 为中心。
- 如果是，可以以一段等差数列的前缀为中心。也需要二分哈希求这个前缀长度。

枚举等差数列，再在等差数列上二分，精细实现复杂度是 $O(\log n)$（二分使用倍增的二分）。
通过分块 $O(\sqrt{n})$ 修改，$O(1)$ 查询维护区间哈希值，单组询问可以做到 $O(\sqrt{n}+\log^2 n)$。

目录

序列自动机

将 i 的字符 c 的指针指向 i 后第一个 c 的位置，并加入一个根，指向每个字符第一次出现位置，这样就得到了序列自动机。

引理 2.1

从根出发的一条路径对应一个本质不同子序列。

证明

很显然，子序列到路径，路径到子序列，形成一个双射。

本质不同子序列

例题 2.1

给定一个字符串 S，求 S 的本质不同子序列个数。
$|S| \leq 10^5$。

本质不同子序列

例题 2.1

给定一个字符串 S，求 S 的本质不同子序列个数。
$|S| \leq 10^5$。

题解

序列自动机，数从根出发的路经数。

本质不同子序列

例题 2.1

给定一个字符串 S，求 S 的本质不同子序列个数。
$|S| \leq 10^5$。

题解

定义 f_i 表示 $S[1;i]$ 有多少本质不同子序列。（含空子序列，$f_0 = 1$）

$$f_i = 2f_{i-1} - f_{\mathrm{lst}_{s_i}}$$

其中 lst_c 表示上一个 c 出现的位置。
意思是 $S[1;i]$ 的本质不同子序列，等于 $S[1;i-1]$ 的子序列，加上 $S[1;i-1]$ 的每个子序列末尾加上 S_i，再减去那些后边第一个 S_i 不是位置 i 的子序列。

本质不同子序列

例题 2.1

给定一个字符串 S，求 S 的本质不同子序列个数。
$|S| \leq 10^5$。

值得一提的是，本质不同子序列是支持合并的，通过矩阵乘法，$A[x][y]$ 表示从当前串第一个字符 x 出发，在当前串走了若干步，走到下个串的第一个字符 y 的系数。（可能还需要额外维护一个所有的和）

PKUWC2020 Day1 T1

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

PKUWC2020 Day1 T1

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

题解

显然需要通过矩阵乘法维护本质不同子序列。

PKUWC2020 Day1 T1

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

题解

显然需要通过矩阵乘法维护本质不同子序列。
预处理 *1 2 3 ... i x x x* 的本质不同序列矩阵 A_i。

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

题解

显然需要通过矩阵乘法维护本质不同子序列。
预处理 *1 2 3 ... i x x x* 的本质不同序列矩阵 A_i。
注意到将 1 到 i 替换成其它互不相同的数的矩阵可直接从矩阵 A_i 得到。

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

题解

显然需要通过矩阵乘法维护本质不同子序列。
预处理 *1 2 3 ... i x x x* 的本质不同序列矩阵 A_i。
注意到将 1 到 i 替换成其它互不相同的数的矩阵可直接从矩阵 A_i 得到。
那么求 A_i 其实就是枚举第 $i+1$ 位是什么，然后通过 A_{i+1} 算出矩阵是什么，求乘法。

例题 2.2

给你一个长度为 $n(n \leq 50)$ 的排列，让你把编号比小于等于它的排列列出来，例如给你 *3 1 2*，列出来的就是 *1 2 3 1 3 2 2 1 3 2 3 1 3 1 2*
让你求出列出来的本质不同的子序列的数量。

题解

显然需要通过矩阵乘法维护本质不同子序列。
预处理 *1 2 3 ... i x x x* 的本质不同序列矩阵 A_i。
注意到将 1 到 i 替换成其它互不相同的数的矩阵可直接从矩阵 A_i 得到。
那么求 A_i 其实就是枚举第 $i+1$ 位是什么，然后通过 A_{i+1} 算出矩阵是什么，求乘法。
枚举 i，再枚举 $i+1$ 位置的值，再计算矩阵、矩阵乘法（瓶颈是矩阵乘法），总复杂度 $O(n^5)$。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

题解

令 f_i/g_i 表示 i 位置结尾/开头有多少个 AA 串。答案为 $\sum_{i=1}^{n-1} f_i g_{i+1}$。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

题解

令 f_i/g_i 表示 i 位置结尾/开头有多少个 AA 串。答案为 $\sum_{i=1}^{n-1} f_i g_{i+1}$。
要求 f，枚举 $|A|$，将所有 $|A|$ 的倍数位置视为关键点。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

题解

令 f_i/g_i 表示 i 位置结尾/开头有多少个 AA 串。答案为 $\sum_{i=1}^{n-1} f_i g_{i+1}$。
要求 f，枚举 $|A|$，将所有 $|A|$ 的倍数位置视为关键点。
这样 AA 必然恰好经过两个相邻的关键点，因此我们枚举关键点算这对关键点对答案的贡献，设关键点为 i, j。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

题解

令 f_i/g_i 表示 i 位置结尾/开头有多少个 AA 串。答案为 $\sum_{i=1}^{n-1} f_i g_{i+1}$。
要求 f，枚举 $|A|$，将所有 $|A|$ 的倍数位置视为关键点。
这样 AA 必然恰好经过两个相邻的关键点，因此我们枚举关键点算这对关键点对答案的贡献，设关键点为 i, j。
求出 i, j 的 LCS 和 LCP，如果 $\mathrm{LCS} + \mathrm{LCP} - 1 \geq |A|$，那么就有合法的 AA 串，并且贡献是个区间（知道 LCS 和 LCP 后区间是好求的）。

" 优秀的拆分"

太经典了以至于用这题的名字表示这个 trick。

例题 2.3

求字符串 S 的所有子串拆成 $AABB$ 形式的方案数的总和（A, B 是非空字符串）。$|S| \leq 3 \times 10^5$。

题解

令 f_i/g_i 表示 i 位置结尾/开头有多少个 AA 串。答案为 $\sum_{i=1}^{n-1} f_i g_{i+1}$。
要求 f，枚举 $|A|$，将所有 $|A|$ 的倍数位置视为关键点。
这样 AA 必然恰好经过两个相邻的关键点，因此我们枚举关键点算这对关键点对答案的贡献，设关键点为 i, j。
求出 i, j 的 LCS 和 LCP，如果 $\mathrm{LCS} + \mathrm{LCP} - 1 \geq |A|$，那么就有合法的 AA 串，并且贡献是个区间（知道 LCS 和 LCP 后区间是好求的）。
如果用后缀数组处理 LCS 和 LCP，那么复杂度是枚举 $|A|$ 再枚举关键点的复杂度，是调和级数 $O(n \log n)$。

LG P6629 [ZJOI2020] 字符串

例题 2.4

给定一个字符串 S，有 Q 次询问，每次给定一个区间，问区间有多少本质不同的形如 AA 的串。
$|S|, Q \leq 2 \times 10^5$，字符集大小是 2。

注意要求是本质不同的，不然就是优秀的拆分加二维数点了。

LG P6629 [ZJOI2020] 字符串

例题 2.4

给定一个字符串 S，有 Q 次询问，每次给定一个区间，问区间有多少本质不同的形如 AA 的串。
$|S|, Q \leq 2 \times 10^5$，字符集大小是 2。

通过这道题来讲一下 runs 理论。

runs 理论

定义 2.1

(run). 定义字符串 S 中的一个 run，指其内部一段两侧都不能扩展的周期子串，且周期至少完整出现过两次。
形式化的定义：一个 run 是一个三元组 (i, j, p)，满足 p 是 $S[i; j]$ 的最小周期，$j - i + 1 \geq 2p$，且满足如下两个条件：

1. 如果 $i > 1$，则 $S[i-1] \neq S[i-1+p]$。
2. 如果 $j < n$，则 $S[j+1] \neq S[j+1-p]$。

runs 理论

定义 2.1

(run). 定义字符串 S 中的一个 run，指其内部一段两侧都不能扩展的周期子串，且周期至少完整出现过两次。
形式化的定义：一个 run 是一个三元组 (i, j, p)，满足 p 是 $S[i; j]$ 的最小周期，$j - i + 1 \geq 2p$，且满足如下两个条件：

1. 如果 $i > 1$，则 $S[i - 1] \neq S[i - 1 + p]$。
2. 如果 $j < n$，则 $S[j + 1] \neq S[j + 1 - p]$。

举个例子，$S =$ aabcabcabb，则 abcabcab 是一个 run，而 abcab 和 abcabca 都不是一个 run。

runs 理论

怎么求出所有的 runs?

runs 理论

怎么求出所有的 runs?

如果我们知道有一个周期是 $S[l;r]$，则可以向左向右扩展得到一个极长的 run。

runs 理论

怎么求出所有的 runs?

如果我们知道有一个周期是 $S[l;r]$，则可以向左向右扩展得到一个极长的 run。

那就是" 优秀的拆分" 的 trick，枚举周期扩展，最后去重，就得到了所有的 runs，复杂度 $O(n\log^2 n)$。

runs 理论

怎么求出所有的 runs?

如果我们知道有一个周期是 $S[l;r]$，则可以向左向右扩展得到一个极长的 run。

那就是" 优秀的拆分" 的 trick，枚举周期扩展，最后去重，就得到了所有的 runs，复杂度 $O(n\log^2 n)$。

```
int cnt = 0;
for (int k = 1; k <= n; k++)
    for (int i = k; i + k <= n; i += k) {
        if (a[cnt].p == k && a[cnt].r >= i + k)
            continue;
        int LCS = lcs(i, i + k);
        if (LCS > k)
            continue;
        int LCP = lcp(i, i + k);
        if (LCS + LCP - 1 < k)
            continue;
        cnt++, a[cnt].l = i - LCS + 1, a[cnt].r = i + k + LCP - 1, a[cnt].p = k;
    }
sort(a + 1, a + cnt + 1, cmpnode);
int newcnt = 0;
for (int i = 1; i <= cnt; i++)
    if (a[i].l != a[i - 1].l || a[i].r != a[i - 1].r)
        a[++newcnt] = a[i];
cnt = newcnt;
```

runs 理论

怎么求出所有的 runs?

如果我们知道有一个周期是 $S[l;r]$，则可以向左向右扩展得到一个极长的 run。

那就是" 优秀的拆分" 的 trick，枚举周期扩展，最后去重，就得到了所有的 runs，复杂度 $O(n \log^2 n)$。

```
int cnt = 0;
for (int k = 1; k <= n; k++)
    for (int i = k; i + k <= n; i += k) {
        if (a[cnt].p == k && a[cnt].r >= i + k)
            continue;
        int LCS = lcs(i, i + k);
        if (LCS > k)
            continue;
        int LCP = lcp(i, i + k);
        if (LCS + LCP - 1 < k)
            continue;
        cnt++, a[cnt].l = i - LCS + 1, a[cnt].r = i + k + LCP - 1, a[cnt].p = k;
    }
sort(a + 1, a + cnt + 1, cmpnode);
int newcnt = 0;
for (int i = 1; i <= cnt; i++)
    if (a[i].l != a[i - 1].l || a[i].r != a[i - 1].r)
        a[++newcnt] = a[i];
cnt = newcnt;
```

如果用后缀数组处理 LCP 和 LCS，并且去重部分使用哈希表，复杂度 $O(n \log n)$。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

定义 2.2

(lyndon 串). 如果一个串 w 小于它的所有真后缀，则称 w 是一个 lyndon 串。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

定义 2.2

*(*lyndon 串*).* 如果一个串 w 小于它的所有真后缀，则称 w 是一个 lyndon 串。

任意一个最小周期的循环移位扩展都能得到对应的 run，我们从其中最小的循环移位 $w = S[i; j]$ 入手。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

定义 2.2

(lyndon 串). 如果一个串 w 小于它的所有真后缀，则称 w 是一个 lyndon 串。

任意一个最小周期的循环移位扩展都能得到对应的 run，我们从其中最小的循环移位 $w = S[i; j]$ 入手。
由于最小周期至少完整出现了 2 次，可推出 w 是一个 lyndon 串（否则 w 有更小后缀，说明不是最小循环移位，或者有更小周期）。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

定义 2.2

(lyndon 串). 如果一个串 w 小于它的所有真后缀，则称 w 是一个 lyndon 串。

任意一个最小周期的循环移位扩展都能得到对应的 run，我们从其中最小的循环移位 $w = S[i; j]$ 入手。
由于最小周期至少完整出现了 2 次，可推出 w 是一个 lyndon 串（否则 w 有更小后缀，说明不是最小循环移位，或者有更小周期）。
所以对于任意 $k \in [i + 1, j]$ 都有 k 后缀要大于 i 后缀，那么 $j + 1$ 后缀是 i 后面第一个可能比 i 小的后缀，而这个大小关系只取决于 $S[j + 1]$ 与 $S[j + 1 - |w|]$ 的大小关系。

runs 理论

但这不是传统写法，传统写法基于 lyndon 串的性质。

定义 2.2

*(*lyndon 串*).* 如果一个串 w 小于它的所有真后缀，则称 w 是一个 lyndon 串。

任意一个最小周期的循环移位扩展都能得到对应的 run，我们从其中最小的循环移位 $w = S[i; j]$ 入手。
由于最小周期至少完整出现了 2 次，可推出 w 是一个 lyndon 串（否则 w 有更小后缀，说明不是最小循环移位，或者有更小周期）。
所以对于任意 $k \in [i + 1, j]$ 都有 k 后缀要大于 i 后缀，那么 $j + 1$ 后缀是 i 后面第一个可能比 i 小的后缀，而这个大小关系只取决于 $S[j + 1]$ 与 $S[j + 1 - |w|]$ 的大小关系。
通过找到所有 i 后缀后面第一个比它小的后缀，并且检查对应区间，可以找到所有 $S[j + 1] < S[j + 1 - |w|]$ 的 runs。同理的，将大小关系反转，也能找出所有 $S[j + 1] > S[j + 1 - |w|]$ 的 runs。

runs 理论

因此，只需要二分哈希加单调栈维护 i 后缀后第一个小/大的后缀，然后扩展对应区间即可。时间复杂度 $O(n \log n)$，可用后缀数组优化到线性。

```cpp
int go[N], st[N], top;
void lyndon(int o) {
    go[n] = n, st[top = 1] = n;
    per(i, n - 1, 1) {
        st[++top] = i;
        while (top > 1 && cmp(o, i, st[top], st[top] + 1, st[top - 1]))
            --top;
        go[i] = st[top];
    }
}
vector<tuple<int, int, int>>runs; // 三元组 (i,j,p)
void sol() {
    rep(o, 0, 1) {
        lyndon(o);
        rep(l, 1, n) {
            int r = go[l];
            int L = l - LCS(l - 1, r), R = r + LCP(l, r + 1);
            if ((R - L + 1) >= (r - l + 1) * 2)
                runs.emplace_back(L, R, r - l + 1);
        }
    }
    sort(runs.begin(), runs.end());
    runs.erase(unique(runs.begin(), runs.end()), runs.end());
}
```

runs 理论

定理 2.1

run 的个数是 $O(n)$。

runs 理论

定理 2.1

run 的个数是 $O(n)$。

证明

根据代码可知。

runs 理论

定理 2.1

run 的个数是 $O(n)$。

证明

根据代码可知。

实际上还有一个更紧的界 $n-1$。(似乎是证明每个 lyndon 根不交)

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

定理 2.2

每个位置结尾的本原平方串个数为 $O(\log n)$。

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

定理 2.2

每个位置结尾的本原平方串个数为 $O(\log n)$。

证明大概是，如果存在三个串 u, v, w，并且 uu 是 vv 的后缀，vv 是 ww 的后缀，则 $|u| + |v| \leq |w|$。细节比较复杂。

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

定理 2.2

每个位置结尾的本原平方串个数为 $O(\log n)$。

推论 2.1

本原平方串的个数是 $O(n \log n)$。即所有 runs 的 $r - l + 1 - 2p$ 之和是 $O(n \log n)$。

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

定理 2.2

每个位置结尾的本原平方串个数为 $O(\log n)$。

推论 2.1

本原平方串的个数是 $O(n \log n)$。即所有 runs 的 $r - l + 1 - 2p$ 之和是 $O(n \log n)$。

定理 2.3

所有 runs 的指数 $\frac{r-l+1}{p}$ 之和是 $O(n)$ 的。(证明略)

runs 理论

定义 2.3

(本原平方串). 如果一个串 w 的周期是 $\frac{|w|}{2}$，那么称 w 为本原平方串。

定理 2.2

每个位置结尾的本原平方串个数为 $O(\log n)$。

推论 2.1

本原平方串的个数是 $O(n \log n)$。即所有 runs 的 $r - l + 1 - 2p$ 之和是 $O(n \log n)$。

定理 2.3

所有 runs 的指数 $\frac{r-l+1}{p}$ 之和是 $O(n)$ 的。(证明略)

定理 2.4

本质不同的本原平方串的个数是 $O(n)$。(听说不需要本原也是对的，证明略)

LG P6629 [ZJOI2020] 字符串

例题 2.5

给定一个字符串 S，有 Q 次询问，每次给定一个区间，问区间有多少本质不同的形如 AA 的串。
$|S|, |Q| \leq 2 \times 10^5$，字符集大小是 2。

LG P6629 [ZJOI2020] 字符串

例题 2.5

给定一个字符串 S，有 Q 次询问，每次给定一个区间，问区间有多少本质不同的形如 AA 的串。
$|S|, |Q| \leq 2 \times 10^5$，字符集大小是 2。

题解

要求本质不同的个数，多次出现我们只在最后一次出现算答案。
用三元组 $[l, r, v]$ 表示对所有区间 $I \supseteq [l, r]$ 贡献 v。
那么对于一个 run 的三元组 (i, j, p)，它的所有长为 p 的倍数的子串整周期均为 p。
它长为 $2p$ 的串的贡献三元组为：

- $[a, a + 2p - 1, 1], \forall a \in [i, j - 2p + 1]$
- $[a, a + 3p - 1, -1], \forall a \in [i, j - 3p + 1]$

LG P6629 [ZJOI2020] 字符串

例题 2.5

给定一个字符串 S，有 Q 次询问，每次给定一个区间，问区间有多少本质不同的形如 AA 的串。
$|S|, |Q| \leq 2 \times 10^5$，字符集大小是 2。

题解

要求本质不同的个数，多次出现我们只在最后一次出现算答案。
用三元组 $[l, r, v]$ 表示对所有区间 $I \supseteq [l, r]$ 贡献 v。
那么对于一个 run 的三元组 (i, j, p)，它的所有长为 p 的倍数的子串整周期均为 p。
同理，长度为 $2kp$ 的串的贡献三元组为：

- $[a, a + 2kp - 1, 1], \forall a \in [i, j - 2kp + 1]$
- $[a, a + (2k + 1)p - 1, -1], \forall a \in [i, j - (2k + 1)p + 1]$

枚举 k 的复杂度是不超过 runs 的指数次数和 $O(n)$。

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

我们只需要记录所有长度为 $2kp$ 的平方串的最后一次出现位置，在枚举到的时候根据当前的出现位置扣掉重复的就行。这是瓶颈，要枚举一个 $i \in [l, l+p)$ 和 k 满足 $i + 2kp - 1 \leq r$，复杂度是每个 runs 的 $r - l + 1 - 2p$ 之和，是 $O(n \log n)$ 的。

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

我们只需要记录所有长度为 $2kp$ 的平方串的最后一次出现位置，在枚举到的时候根据当前的出现位置扣掉重复的就行。这是瓶颈，要枚举一个 $i \in [l, l+p)$ 和 k 满足 $i + 2kp - 1 \leq r$，复杂度是每个 runs 的 $r - l + 1 - 2p$ 之和，是 $O(n \log n)$ 的。

写成贡献三元组就是 [最后一次的 l, 当前出现的 r, -1]。

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

我们只需要记录所有长度为 $2kp$ 的平方串的最后一次出现位置，在枚举到的时候根据当前的出现位置扣掉重复的就行。这是瓶颈，要枚举一个 $i \in [l, l+p)$ 和 k 满足 $i + 2kp - 1 \le r$，复杂度是每个 runs 的 $r - l + 1 - 2p$ 之和，是 $O(n \log n)$ 的。

写成贡献三元组就是 [最后一次的 l, 当前出现的 r, -1]。

我们需要支持的操作是：

- $[a, a + \text{len}, v], \forall a \in [l, r]$（将 $(a, a + \text{len})$ 视为一个点，点会形成一个斜线）
- 单点查询。

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

我们只需要记录所有长度为 $2kp$ 的平方串的最后一次出现位置，在枚举到的时候根据当前的出现位置扣掉重复的就行。这是瓶颈，要枚举一个 $i \in [l, l+p)$ 和 k 满足 $i + 2kp - 1 \le r$，复杂度是每个 runs 的 $r - l + 1 - 2p$ 之和，是 $O(n \log n)$ 的。

写成贡献三元组就是 [最后一次的 l, 当前出现的 r, -1]。

我们需要支持的操作是：

- $[a, a + \text{len}, v], \forall a \in [l, r]$（将 $(a, a + \text{len})$ 视为一个点，点会形成一个斜线）
- 单点查询。

单个贡献三元组其实就是到右上角的矩形加。（设 $(1, 1)$ 是最左上角）

LG P6629 [ZJOI2020] 字符串

题解

还有一个小问题，我们只是让在一个 run 内的一个 AA 只贡献一次，不同的 run 之间的贡献还要减去。

我们只需要记录所有长度为 $2kp$ 的平方串的最后一次出现位置，在枚举到的时候根据当前的出现位置扣掉重复的就行。这是瓶颈，要枚举一个 $i \in [l, l+p)$ 和 k 满足 $i + 2kp - 1 \le r$，复杂度是每个 runs 的 $r - l + 1 - 2p$ 之和，是 $O(n \log n)$ 的。

写成贡献三元组就是 [最后一次的 l, 当前出现的 r, -1]。

我们需要支持的操作是：

- $[a, a + \text{len}, v], \forall a \in [l, r]$（将 $(a, a + \text{len})$ 视为一个点，点会形成一个斜线）
- 单点查询。

单个贡献三元组其实就是到右上角的矩形加。（设 $(1, 1)$ 是最左上角）

可转化为：

- 左上-右下斜线加。$O(n \log n)$ 个。
- 到左上角矩形求和（原本是到左下角，为了方便转化成到最左上角求和）。$O(n)$ 个。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。
转成斜射线加（从某个点到右下加），左上矩形查。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。
转成斜射线加（从某个点到右下加），左上矩形查。
如果斜射线的起点在矩形内，射线肯定会穿过与矩形的下边界或者右边界中的恰好一个。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。
转成斜射线加（从某个点到右下加），左上矩形查。
如果斜射线的起点在矩形内，射线肯定会穿过与矩形的下边界或者右边界中的恰好一个。
两个分别统计，穿过下边界的，其矩形内长度是起点和查询点 x 坐标的差。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。
转成斜射线加（从某个点到右下加），左上矩形查。
如果斜射线的起点在矩形内，射线肯定会穿过与矩形的下边界或者右边界中的恰好一个。
两个分别统计，穿过下边界的，其矩形内长度是起点和查询点 x 坐标的差。
按照 x 顺序枚举，加斜射线 (x, y) 在起点的 $x - y$ 位置加，查询 (x, y) 就是在 $x - y$ 前缀求和（贡献是 x 的差，因此要维护两个值，x 坐标之和和个数）。

LG P6629 [ZJOI2020] 字符串

题解

现在是斜线加，左上矩形查。
转成斜射线加（从某个点到右下加），左上矩形查。
如果斜射线的起点在矩形内，射线肯定会穿过与矩形的下边界或者右边界中的恰好一个。
两个分别统计，穿过下边界的，其矩形内长度是起点和查询点 x 坐标的差。
按照 x 顺序枚举，加斜射线 (x, y) 在起点的 $x - y$ 位置加，查询 (x, y) 就是在 $x - y$ 前缀求和（贡献是 x 的差，因此要维护两个值，x 坐标之和和个数）。
右边界同理，总复杂度 $O(n \log^2 n)$。

P5211 [ZJOI2017] 字符串

例题 2.6

维护一个长度为 n 的动态字符串，字符集大小为 2×10^9，m 次询问，支持：

- 区间加减。（保证依然在字符集内）
- 询问某个子串的最小后缀的开头位置。

$n \leq 2 \times 10^5, m \leq 3 \times 10^4$。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。
  反证：设字符串 U, V 满足 $|U| < |V| < 2|U|$。根据上面两个结论，U 是 V 的 *border*，因此存在长度为 $|V| - |U|$ 的周期 T，记 $U = T^k C, V = T^{k+1} C$。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。
  反证：设字符串 U, V 满足 $|U| < |V| < 2|U|$。根据上面两个结论，U 是 V 的 *border*，因此存在长度为 $|V| - |U|$ 的周期 T，记 $U = T^k C, V = T^{k+1} C$。
  根据定义（U 末尾加一个串可能成为最小后缀），存在可空串 R 满足 $UR < VR \Rightarrow T^k CR < T^{k+1} CR \Rightarrow CR < TCR \overset{\forall k, T^k CR < T^{k+1} CR}{\Rightarrow} CR < T^k CR = UR$。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。
  反证：设字符串 U, V 满足 $|U| < |V| < 2|U|$。根据上面两个结论，U 是 V 的 *border*，因此存在长度为 $|V| - |U|$ 的周期 T，记 $U = T^k C, V = T^{k+1} C$。
  根据定义（U 末尾加一个串可能成为最小后缀），存在可空串 R 满足 $UR < VR \Rightarrow T^k CR < T^{k+1} CR \Rightarrow CR < TCR \overset{\forall k, T^k CR < T^{k+1} CR}{\Rightarrow} CR < T^k CR = UR$。
  此时 CR 比 UR 优，这对任意 R 都满足，因此 UR 末尾加一个串后不可能成为最小后缀。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。
  反证：设字符串 U, V 满足 $|U| < |V| < 2|U|$。根据上面两个结论，U 是 V 的 *border*，因此存在长度为 $|V| - |U|$ 的周期 T，记 $U = T^k C, V = T^{k+1} C$。
  根据定义（U 末尾加一个串可能成为最小后缀），存在可空串 R 满足 $UR < VR \Rightarrow T^k CR < T^{k+1} CR \Rightarrow CR < TCR \overset{\forall k, T^k CR < T^{k+1} CR}{\Rightarrow} CR < T^k CR = UR$。
  此时 CR 比 UR 优，这对任意 R 都满足，因此 UR 末尾加一个串后不可能成为最小后缀。
  其实类似 *ababa* 中的 *aba* 不可能成为最小后缀，因为有更长的 *ababa* 和更短的 *a* 比它优秀。

题解

考虑子串的所有末尾加一个串可能会成为最小后缀的后缀。

- 它们必然满足短的是长的后缀，这是由于它们都是子串的后缀。
- 它们必然满足短的是长的前缀，否则可以直接比出大小，有一个必然不可能是最小后缀。
- 它们必然满足长的是短的的至少两倍长度。
  反证：设字符串 U, V 满足 $|U| < |V| < 2|U|$。根据上面两个结论，U 是 V 的 *border*，因此存在长度为 $|V| - |U|$ 的周期 T，记 $U = T^k C, V = T^{k+1} C$。
  根据定义（U 末尾加一个串可能成为最小后缀），存在可空串 R 满足 $UR < VR \Rightarrow T^k CR < T^{k+1} CR \Rightarrow CR < TCR \overset{\forall k, T^k CR < T^{k+1} CR}{\Rightarrow} CR < T^k CR = UR$。
  此时 CR 比 UR 优，这对任意 R 都满足，因此 UR 末尾加一个串后不可能成为最小后缀。
  其实类似 *ababa* 中的 *aba* 不可能成为最小后缀，因为有更长的 *ababa* 和更短的 *a* 比它优秀。

因此，子串内可能成为最小后缀的个数不超过 $O(\log |S|)$。

题解

显然用线段树维护这个集合，假设维护出了区间哈希值，怎么合并?

题解

显然用线段树维护这个集合，假设维护出了区间哈希值，怎么合并?
现在合并两个串 S_1, S_2，在线段树上有 $|S_2| \leq |S_1| \leq |S_2| + 1$（两侧长度相近），根据至少两倍长度的性质至多有 S_1 中的一个后缀会被保留在最终结果中。

题解

显然用线段树维护这个集合，假设维护出了区间哈希值，怎么合并?
现在合并两个串 S_1, S_2，在线段树上有 $|S_2| \leq |S_1| \leq |S_2| + 1$（两侧长度相近），根据至少两倍长度的性质至多有 S_1 中的一个后缀会被保留在最终结果中。
对于 U, V 是 S_1 中可能成为最小后缀的后缀，不妨设 $US_2 < VS_2$，若 US_2 不是 VS_2 的前缀，则 VS_2 可排除，若 US_2 是 VS_2 的前缀则也是 *border*，故 US_2 可排除。(证明与前面同理，找到周期 T，得到更小的后缀，US_2 注定不可能是最小后缀)

题解

显然用线段树维护这个集合，假设维护出了区间哈希值，怎么合并?
现在合并两个串 S_1, S_2，在线段树上有 $|S_2| \le |S_1| \le |S_2| + 1$（两侧长度相近），根据至少两倍长度的性质至多有 S_1 中的一个后缀会被保留在最终结果中。
对于 U, V 是 S_1 中可能成为最小后缀的后缀，不妨设 $US_2 < VS_2$，若 US_2 不是 VS_2 的前缀，则 VS_2 可排除，若 US_2 是 VS_2 的前缀则也是 *border*，故 US_2 可排除。(证明与前面同理，找到周期 T，得到更小的后缀，US_2 注定不可能是最小后缀)
根据这个规则可以使得 S_1 中只留下后缀 X，记 $P = XS_2$。
初始令合并后的后缀为 S_2 的后缀，按照长度枚举其中的串 V 进行如下操作:

- 若 V 是 P 的前缀，不动。
- 若 V 不是 P 的前缀，且 $V < P$，抛弃 P。停止。
- 若 V 不是 P 的前缀，且 $V > P$，抛弃 V。

其实直接令合并后的后缀为 S_2 可能成为最小后缀的后缀的集合加入 X 也是正确的，因为大小是 $O(\log n)$ 的。
最终，通过 $O(\sqrt{n})$ 修改，$O(1)$ 查询区间哈希值可得总复杂度是 $O(n \log^2 n + m \log^2 n + m\sqrt{n})$ 的做法。

todo

以上，就速通了字符串的大部分内容。

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。
还有一些内容，作者不是很熟悉，或者十分冷门，留给感兴趣的同学们课后自行学习：

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。
还有一些内容，作者不是很熟悉，或者十分冷门，留给感兴趣的同学们课后自行学习：

- lyndon 分解：Duval 算法。

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。
还有一些内容，作者不是很熟悉，或者十分冷门，留给感兴趣的同学们课后自行学习：

- lyndon 分解：Duval 算法。
- bitset 求最长公共子序列（没研究过，复杂度 $O(\frac{n^2}{\omega})$）

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。
还有一些内容，作者不是很熟悉，或者十分冷门，留给感兴趣的同学们课后自行学习：

- lyndon 分解：Duval 算法。
- bitset 求最长公共子序列（没研究过，复杂度 $O(\frac{n^2}{\omega})$）
- 后缀平衡树（这个比较简单，但过于冷门，不讲了）

todo

以上，就速通了字符串的大部分内容。
个人感觉最重要的是后缀自动机，因为后缀自动机能整的花活比较多，而且并不难写，科技点点深一点有时候能够偷鸡。
还有一些内容，作者不是很熟悉，或者十分冷门，留给感兴趣的同学们课后自行学习：

- lyndon 分解：Duval 算法。
- bitset 求最长公共子序列（没研究过，复杂度 $O(\frac{n^2}{\omega})$）
- 后缀平衡树（这个比较简单，但过于冷门，不讲了）
- ...

目录

OCPC2024 Day2 Suspicious Submissions

例题 3.1

给定 n 个串 $S_1, \ldots, S_n$，和整数 K，我们认为一对字符串 S_i 和 S_j 是可疑的当且仅当 S_i 删除一个长度不超过 K 的子串，再在删去的位置加入一个长度不超过 K 的子串能够得到 S_j。
求有多少对 (i, j) 是可疑的。
$n \leq 2 \times 10^5, K, \sum |S_i| \leq 10^6$。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。如果 (i,j) 是可疑的，必然是有一个位置 p，满足 S_i 和 S_j 前 p 位相等，并且 S_i 和 S_j 的最长公共后缀大于等于 $|S_i| - p - K$。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。
如果 (i,j) 是可疑的，必然是有一个位置 p，满足 S_i 和 S_j 前 p 位相等，并且 S_i 和 S_j 的最长公共后缀大于等于 $|S_i| - p - K$。
但是可能有很多个 p 是合法的，发现我们算的是 $\leq K$ 的方案数，因此我们只需要用 $\leq K$ 的方案减去 $\leq K-1$ 的方案，那么一个可疑对只会被算一次。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。
如果 (i,j) 是可疑的，必然是有一个位置 p，满足 S_i 和 S_j 前 p 位相等，并且 S_i 和 S_j 的最长公共后缀大于等于 $|S_i| - p - K$。
但是可能有很多个 p 是合法的，发现我们算的是 $\leq K$ 的方案数，因此我们只需要用 $\leq K$ 的方案减去 $\leq K-1$ 的方案，那么一个可疑对只会被算一次。
现在算 $\leq K$ 的方案数，枚举 S_i 后在 Trie 树上走（相当于枚举了 p），求之前经过这个 Trie 树上点的 S_j 中有多少与它的 LCS 大于等于 $|S_i| - K - p$，等于找到 S_i 在后缀自动机上长度为 $|S_i| - K - p$ 的后缀节点，求子树内的和。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。
如果 (i,j) 是可疑的，必然是有一个位置 p，满足 S_i 和 S_j 前 p 位相等，并且 S_i 和 S_j 的最长公共后缀大于等于 $|S_i| - p - K$。
但是可能有很多个 p 是合法的，发现我们算的是 $\leq K$ 的方案数，因此我们只需要用 $\leq K$ 的方案减去 $\leq K-1$ 的方案，那么一个可疑对只会被算一次。
现在算 $\leq K$ 的方案数，枚举 S_i 后在 Trie 树上走（相当于枚举了 p），求之前经过这个 Trie 树上点的 S_j 中有多少与它的 LCS 大于等于 $|S_i| - K - p$，等于找到 S_i 在后缀自动机上长度为 $|S_i| - K - p$ 的后缀节点，求子树内的和。
将每个 Trie 树上节点的所有操作离线，变成了加入一个节点，查询子树内节点数。

题解

打乱顺序不影响答案，不妨设 S_i 按长度排序，考虑 $(i,j), \forall j < i$ 中有多少对可疑的。
如果 (i,j) 是可疑的，必然是有一个位置 p，满足 S_i 和 S_j 前 p 位相等，并且 S_i 和 S_j 的最长公共后缀大于等于 $|S_i| - p - K$。
但是可能有很多个 p 是合法的，发现我们算的是 $\leq K$ 的方案数，因此我们只需要用 $\leq K$ 的方案减去 $\leq K-1$ 的方案，那么一个可疑对只会被算一次。
现在算 $\leq K$ 的方案数，枚举 S_i 后在 Trie 树上走（相当于枚举了 p），求之前经过这个 Trie 树上点的 S_j 中有多少与它的 LCS 大于等于 $|S_i| - K - p$，等于找到 S_i 在后缀自动机上长度为 $|S_i| - K - p$ 的后缀节点，求子树内的和。
将每个 Trie 树上节点的所有操作离线，变成了加入一个节点，查询子树内节点数。
还有细节，如果查询 u 子树长度 $\geq l$ 的串，可能会查到 u 节点中长度 $< l$ 的串（因为后缀自动机节点表示一个区间，l 可能落在这个区间中），此时如果我们保证 $|S_j| \geq |S_i| - K$ 就没有问题，因为这时 $|S_j| \geq |S_i| - K - p = l$，而且只有这些 S_j 可能会有贡献。因此还需要双指针删去 $|S_j| < |S_i| - K$ 的 S_j，不影响复杂度。

UOJ429 集训队作业 2018 串串划分

例题 3.2

给定一个长度为 $n(n \leq 2 \times 10^5)$ 的字符串 S，求有多少划分 $S = S_1 \cdots + S_k$，满足：

- 每个 S_i 不循环（循环定义存在 T，$S_i = T^k$，$k \geq 2$）。
- $S_i \neq S_{i+1}$。

UOJ429 集训队作业 2018 串串划分

例题 3.2

给定一个长度为 $n(n \leq 2 \times 10^5)$ 的字符串 S，求有多少划分 $S = S_1 \cdots + S_k$，满足：

- 每个 S_i 不循环（循环定义存在 T，$S_i = T^k$，$k \geq 2$）。
- $S_i \neq S_{i+1}$。

题解

设 $C(S)$ 表示 S 最小循环节出现次数，那么 S_i 容斥系数就是 $(-1)^{C(S_i)}$。

UOJ429 集训队作业 2018 串串划分

例题 3.2

给定一个长度为 $n(n \leq 2 \times 10^5)$ 的字符串 S，求有多少划分 $S = S_1 \cdots + S_k$，满足：

- 每个 S_i 不循环（循环定义存在 T，$S_i = T^k$，$k \geq 2$）。
- $S_i \neq S_{i+1}$。

题解

设 $C(S)$ 表示 S 最小循环节出现次数，那么 S_i 容斥系数就是 $(-1)^{C(S_i)}$。
证：假设一个划分，如果相同的小段有相同的循环节，我们合并成一个大段。
设循环节在大段总共出现了 k 次，那么它划分成小段的贡献之和是：

$$\sum_{i=0}^{k-1} \binom{k-1}{i} (-1)^{k+1-i} = [k = 1]$$

这里 i 枚举的是切了几刀。而划分方案合法等价于所有大段循环节次数为 1。

UOJ429 集训队作业 2018 串串划分

例题 3.2

给定一个长度为 $n(n \leq 2 \times 10^5)$ 的字符串 S，求有多少划分 $S = S_1 \cdots + S_k$，满足：

- 每个 S_i 不循环（循环定义存在 T，$S_i = T^k$，$k \geq 2$）。
- $S_i \neq S_{i+1}$。

题解

设 $C(S)$ 表示 S 最小循环节出现次数，那么 S_i 容斥系数就是 $(-1)^{C(S_i)}$。
证：假设一个划分，如果相同的小段有相同的循环节，我们合并成一个大段。
设循环节在大段总共出现了 k 次，那么它划分成小段的贡献之和是：

$$\sum_{i=0}^{k-1} \binom{k-1}{i} (-1)^{k+1-i} = [k = 1]$$

这里 i 枚举的是切了几刀。而划分方案合法等价于所有大段循环节次数为 1。
因此，一个大段合法时容斥系数之和是 1，不合法时是 0。

UOJ429 集训队作业 2018 串串划分

题解

写出 *dp* 式子：

$$\begin{aligned} f_i &= \sum_{j=0}^{i-1} (-1)^{C(S[j+1;i])-1} f_j \\ &= \sum_{j=0}^{i-1} f_j - 2 \times \sum_{j=0}^{i-1} [C(S[j+1;i]) \text{ 是偶数}] f_j \end{aligned}$$

这个暴力转移是 $O(n^2)$ 的。
注意到 C 是偶数说明是一个 AA 串。对于一个 run，内部两个距离为 $2p$ 的倍数的位置能转移，因此只需要记录一个 $\bmod 2p$ 的前缀和（这样就能对 i 维护出当前 run 内部的 f_{i-2kp} 之和）。
转移均摊到每个 run 上复杂度是 $r-l+1-2p$ 之和，之前说过，这个值是 $O(n \log n)$ 的。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

题解

与区间本质不同子串类似，肯定也是扫描线右端点 r，在左端点的位置算答案。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

题解

与区间本质不同子串类似，肯定也是扫描线右端点 r，在左端点的位置算答案。
那么只需要考虑新出现回文后缀，它们的左端点会改变。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

题解

与区间本质不同子串类似，肯定也是扫描线右端点 r，在左端点的位置算答案。
那么只需要考虑新出现回文后缀，它们的左端点会改变。
枚举回文后缀的一个等差数列，设左端点为 $a, a + k, a + 2k \ldots, a + lk$。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

题解

与区间本质不同子串类似，肯定也是扫描线右端点 r，在左端点的位置算答案。
那么只需要考虑新出现回文后缀，它们的左端点会改变。
枚举回文后缀的一个等差数列，设左端点为 $a, a + k, a + 2k \ldots, a + lk$。
这个等差数列去掉末项其实也在右端点为 $r - k$ 的时候出现过，并且每个回文串最后出现的位置向右移动了 k，那么再加上新增的左端点为 a 的回文后缀，其实左端点的改变量只是多一个末项，还需要去掉最长串（首项）上一次出现的贡献，给 $(上一次出现的\ l, 这次出现的\ l]$ 这个区间加一。

LOJ6070「2017 山东一轮集训 Day4」基因

例题 3.3

给定长度为 n 的字符串 S，有 Q 组询问，每个询问给定 l, r，问 $S[l; r]$ 有多少本质不同的回文子子串。
强制在线。$n, Q \leq 10^5$。

题解

与区间本质不同子串类似，肯定也是扫描线右端点 r，在左端点的位置算答案。
那么只需要考虑新出现回文后缀，它们的左端点会改变。
枚举回文后缀的一个等差数列，设左端点为 $a, a + k, a + 2k \ldots, a + lk$。
这个等差数列去掉末项其实也在右端点为 $r - k$ 的时候出现过，并且每个回文串最后出现的位置向右移动了 k，那么再加上新增的左端点为 a 的回文后缀，其实左端点的改变量只是多一个末项，还需要去掉最长串（首项）上一次出现的贡献，给 $(上一次出现的\ l, 这次出现的\ l]$ 这个区间加一。
因此枚举等差数列，加入末项的贡献，强制在线通过主席树维护，复杂度 $O(n \log^2 n)$。

例题 3.4

给定字符串 S，还有初始为空字符串 T。
Q 次对 T 的修改，每次修改为以下两种之一：

- *1 U*，U 是一个字符串：表示在 T 的末尾接上 U。
- *2 l k*，$1 \le l \le |T|, k \ge 1$，表示重复 k 次，第 i 次操作为，将 T_{l+i-1} 接在 T 的末尾。

请在每次修改后求出 S 在 T 中出现次数。
$|S| \le 2 \times 10^5, Q \le 10^4, |T| \le 10^{15}, \sum |U| \le 2 \times 10^5$，字符集为小写字母。

题解

如果 $|S| = 1$ 那么就是可持久化平衡树，复杂度可以做到 $O(Q \log |T|)$。

InfOJ53 NIT 学 KMP

题解

如果 $|S| = 1$ 那么就是可持久化平衡树，复杂度可以做到 $O(Q \log |T|)$。
考虑在平衡树上维护，需要支持合并操作，合并两个区间的答案其实是左区间的答案加上右区间的答案加上左右区间新产生的贡献。那么要维护哪些信息？

题解

如果 $|S| = 1$ 那么就是可持久化平衡树，复杂度可以做到 $O(Q \log |T|)$。
考虑在平衡树上维护，需要支持合并操作，合并两个区间的答案其实是左区间的答案加上右区间的答案加上左右区间新产生的贡献。那么要维护哪些信息？
对于每个区间，我们需要维护最长的前缀，使得这个前缀在 S 中作为子串 $S[l_1; r_1]$ 出现了；维护最长的后缀，使得这个后缀在 S 中作为子串 $S[l_2; r_2]$ 出现了。

InfOJ53 NIT 学 KMP

题解

如果 $|S| = 1$ 那么就是可持久化平衡树，复杂度可以做到 $O(Q \log |T|)$。
考虑在平衡树上维护，需要支持合并操作，合并两个区间的答案其实是左区间的答案加上右区间的答案加上左右区间新产生的贡献。那么要维护哪些信息？
对于每个区间，我们需要维护最长的前缀，使得这个前缀在 S 中作为子串 $S[l_1; r_1]$ 出现了；维护最长的后缀，使得这个后缀在 S 中作为子串 $S[l_2; r_2]$ 出现了。
设左孩子前缀信息为 $[l_1, r_1]$，右孩子前缀信息为 $[l_2, r_2]$，现在考虑合并后的最长前缀，若 $[l_1, r_1]$ 长度不等于左孩子长度，显然整个的前缀信息就是 $[l_1, r_1]$。否则，预处理出后缀数组，可以二分出 $S[l_1; r_1] + S[l_2; r_2]$ 在原串中的排序，进而得到新的信息。

题解

如果 $|S| = 1$ 那么就是可持久化平衡树，复杂度可以做到 $O(Q \log |T|)$。
考虑在平衡树上维护，需要支持合并操作，合并两个区间的答案其实是左区间的答案加上右区间的答案加上左右区间新产生的贡献。那么要维护哪些信息？
对于每个区间，我们需要维护最长的前缀，使得这个前缀在 S 中作为子串 $S[l_1; r_1]$ 出现了；维护最长的后缀，使得这个后缀在 S 中作为子串 $S[l_2; r_2]$ 出现了。
设左孩子前缀信息为 $[l_1, r_1]$，右孩子前缀信息为 $[l_2, r_2]$，现在考虑合并后的最长前缀，若 $[l_1, r_1]$ 长度不等于左孩子长度，显然整个的前缀信息就是 $[l_1, r_1]$。否则，预处理出后缀数组，可以二分出 $S[l_1; r_1] + S[l_2; r_2]$ 在原串中的排序，进而得到新的信息。
现在我们只需要算合并后新产生的贡献。那么问题即为 S 在 $S[l_1; r_1] + S[l_2; r_2]$ 中出现了多少次。

题解

那么 S 的前缀就是 $S[l_1; r_1]$ 的后缀，S 的一个后缀就是 $S[l_2; r_2]$ 的前缀。

题解

那么 S 的前缀就是 $S[l_1; r_1]$ 的后缀，S 的一个后缀就是 $S[l_2; r_2]$ 的前缀。
此时在正串 *kmp* 的 *fail* 树（就是 $next$ 数组形成的树）上找到 $[l_1, r_1]$ 对应的节点，反串的 *fail* 树上 $[l_2, r_2]$ 对应的节点。那么就是求两个点到根路径（正 *fail* 树上到根路径上一个点满足是 S 的前缀且是 $S[l_1; r_1]$ 的后缀）有多少对节点长度之和为 n。

InfOJ53 NIT 学 KMP

题解

那么 S 的前缀就是 $S[l_1; r_1]$ 的后缀，S 的一个后缀就是 $S[l_2; r_2]$ 的前缀。
此时在正串 *kmp* 的 *fail* 树（就是 $next$ 数组形成的树）上找到 $[l_1, r_1]$ 对应的节点，反串的 *fail* 树上 $[l_2, r_2]$ 对应的节点。那么就是求两个点到根路径（正 *fail* 树上到根路径上一个点满足是 S 的前缀且是 $S[l_1; r_1]$ 的后缀）有多少对节点长度之和为 n。
将问题离线后（注意到左右之间的贡献不急着算），*dfs* 其中一棵树，*dfs* 到 x 时，把另一棵树上 $n - x$ 对应点的子树权值加一，询问就是 *dfs* 到 x 时，另一棵树上 y 的权值和。

InfOJ53 NIT 学 KMP

题解

那么 S 的前缀就是 $S[l_1; r_1]$ 的后缀，S 的一个后缀就是 $S[l_2; r_2]$ 的前缀。
此时在正串 *kmp* 的 *fail* 树（就是 $next$ 数组形成的树）上找到 $[l_1, r_1]$ 对应的节点，反串的 *fail* 树上 $[l_2, r_2]$ 对应的节点。那么就是求两个点到根路径（正 *fail* 树上到根路径上一个点满足是 S 的前缀且是 $S[l_1; r_1]$ 的后缀）有多少对节点长度之和为 n。
将问题离线后（注意到左右之间的贡献不急着算），*dfs* 其中一棵树，*dfs* 到 x 时，把另一棵树上 $n - x$ 对应点的子树权值加一，询问就是 *dfs* 到 x 时，另一棵树上 y 的权值和。
因此合并部分也只需要额外一个 $\log$，总复杂度 $O(Q \log |T| \log |S|)$。