Z 函数、SA、SAM

宋佳兴

2024 年 7 月 22 日

- 欢迎有想法的同学随时上来与我交流。
- 有任何问题，请随时提问。
- 讲课过程中途，我们将会休息 15 到 20 分钟。

Z 函数（又名扩展 KMP）

定义：对于一个长度为 n 的字符串 s（下标从 0 开始），定义函数 z_i 表示 s 和 $s[i, n-1]$ 的最长公共前缀长度（LCP）。z 被称为 s 的 Z 函数，特别地，z_0 无意义。

来源：OI Wiki, Z 函数

Z 函数（又名扩展 KMP）

定义：对于一个长度为 n 的字符串 s（下标从 0 开始），定义函数 z_i 表示 s 和 $s[i, n-1]$ 的最长公共前缀长度（LCP）。z 被称为 s 的 Z 函数，特别地，z_0 无意义。

我们从 1 到 $n-1$ 依次计算 z_i 的值。在计算 z_i 的过程中，我们会利用已经计算好的 $z_1, \ldots, z_{i-1}$。

来源：OI Wiki, Z 函数

Z 函数（又名扩展 KMP）

定义：对于一个长度为 n 的字符串 s（下标从 0 开始），定义函数 z_i 表示 s 和 $s[i, n-1]$ 的最长公共前缀长度（LCP）。z 被称为 s 的 Z 函数，特别地，z_0 无意义。

我们从 1 到 $n-1$ 依次计算 z_i 的值。在计算 z_i 的过程中，我们会利用已经计算好的 $z_1, \ldots, z_{i-1}$。

对于 i，我们称区间 $[i, i+z_i-1]$ 为 i 的**匹配段**。

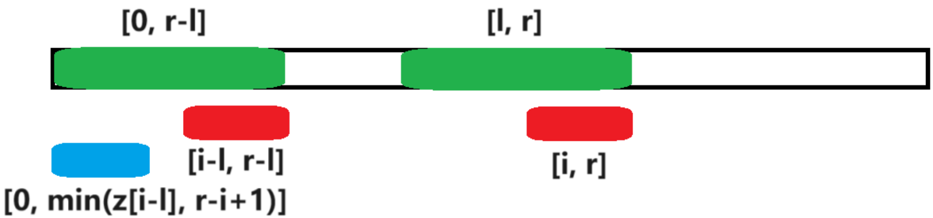
算法中我们维护已求出的匹配段中右端点最靠右的一个，记作 $[l, r]$。根据定义，$s[l, r]$ 是 s 的前缀。初始时 $l = r = -1$。

来源：OI Wiki, Z 函数

Z 函数

在计算 z_i 的过程中：

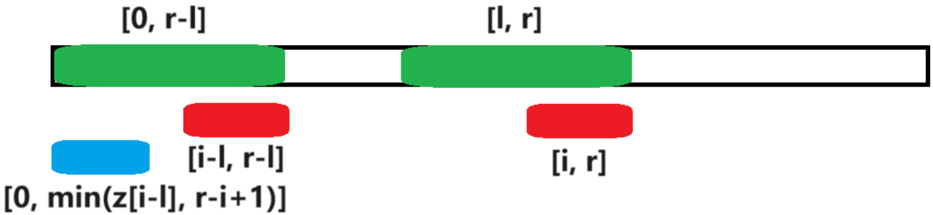
- 如果 $i \le r$，那么根据 $[l, r]$ 的定义有 $s[i, r] = s[i-l, r-l]$，因此 $z_i \ge \min(z_{i-l}, r-i+1)$。我们令 $z_i = \min(z_{i-l}, r-i+1)$，然后暴力枚举下一个字符，直到不能扩展为止。



Z 函数

在计算 z_i 的过程中：

- 如果 $i \le r$，那么根据 $[l, r]$ 的定义有 $s[i, r] = s[i-l, r-l]$，因此 $z_i \ge \min(z_{i-l}, r-i+1)$。我们令 $z_i = \min(z_{i-l}, r-i+1)$，然后暴力枚举下一个字符，直到不能扩展为止。

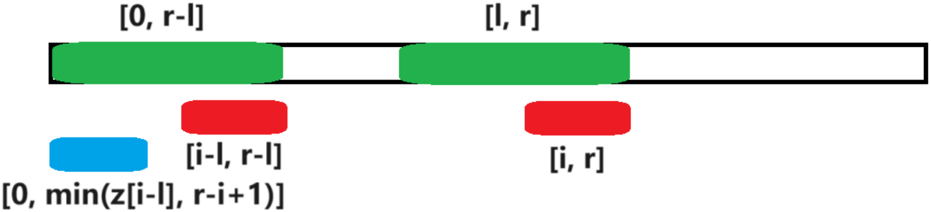


- 如果 $i > r$，那么我们直接暴力求出 z_i。

Z 函数

在计算 z_i 的过程中：

- 如果 $i \le r$，那么根据 $[l, r]$ 的定义有 $s[i, r] = s[i - l, r - l]$，因此 $z_i \ge \min(z_{i-l}, r - i + 1)$。我们令 $z_i = \min(z_{i-l}, r - i + 1)$，然后暴力枚举下一个字符，直到不能扩展为止。



- 如果 $i > r$，那么我们直接暴力求出 z_i。
- 在求出 z_i 后，使用 $[i, i + z_i - 1]$ 更新 $[l, r]$。

Z 函数

```cpp
int l = -1, r = -1;
for (int i = 1; i < n; i++) {
    int& j = z[i] = i <= r ? min(z[i - l], r - i + 1) : 0;
    while (s[i + j] == s[j]) j++;
    if (i + j > r) l = i, r = i + j - 1;
}
```

Z 函数

```
int l = -1, r = -1;
for (int i = 1; i < n; i++) {
    int& j = z[i] = i <= r ? min(z[i - l], r - i + 1) : 0;
    while (s[i + j] == s[j]) j++;
    if (i + j > r) l = i, r = i + j - 1;
}
```

复杂度分析：复杂度不明确的部分是 while 循环次数，我们分两种情况讨论：

Z 函数

```
int l = -1, r = -1;
for (int i = 1; i < n; i++) {
    int& j = z[i] = i <= r ? min(z[i - l], r - i + 1) : 0;
    while (s[i + j] == s[j]) j++;
    if (i + j > r) l = i, r = i + j - 1;
}
```

复杂度分析：复杂度不明确的部分是 while 循环次数，我们分两种情况讨论：

- 若 $z_{i-l} < r - i + 1$，则 $z_i = z_{i-l}$，while 会立刻跳出。

Z 函数

```cpp
int l = -1, r = -1;
for (int i = 1; i < n; i++) {
    int& j = z[i] = i <= r ? min(z[i - l], r - i + 1) : 0;
    while (s[i + j] == s[j]) j++;
    if (i + j > r) l = i, r = i + j - 1;
}
```

复杂度分析：复杂度不明确的部分是 while 循环次数，我们分两种情况讨论：

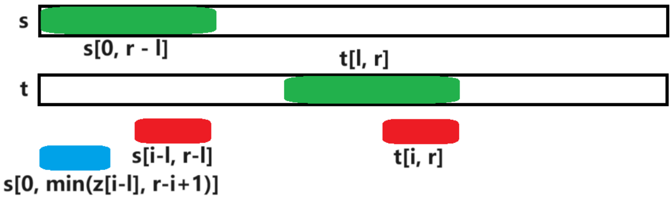
- 若 $z_{i-l} < r - i + 1$，则 $z_i = z_{i-l}$，while 会立刻跳出。
- 若 $z_{i-l} \geq r - i + 1$ 或 $i > r$，则 z_i 会初始化为 $r - i + 1$，此时 $i + z_i - 1 = r$，于是 while 每次执行都会使 r 后移一位，而 $r \leq n - 1$，所以总共执行 $O(n)$ 次。

Z 函数

匹配文本串：对于字符串 s 和 t，定义 zt_i 表示 s 和 $t[i, n-1]$ 的 LCP 长度，求 zt。

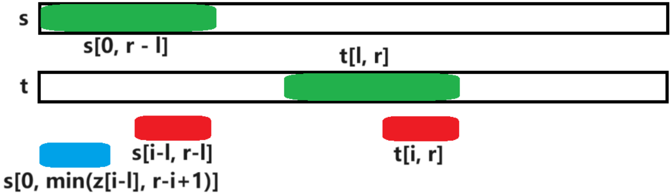
Z 函数

匹配文本串：对于字符串 s 和 t，定义 zt_i 表示 s 和 $t[i, n-1]$ 的 LCP 长度，求 zt。



Z 函数

匹配文本串：对于字符串 s 和 t，定义 zt_i 表示 s 和 $t[i, n-1]$ 的 LCP 长度，求 zt。




```
int l = -1, r = -1;
for (int i = 0; i < m; i++) {
    int& j = zt[i] = i <= r ? min(z[i - l], r - i + 1) : 0;
    while (j < n && t[i + j] == s[j]) j++;
    if (i + j > r) l = i, r = i + j - 1;
}
```

CF30E Tricky and Clever Password

给定一个字符串 s，将 s 分解为 $A+\text{prefix}+B+\text{middle}+C+\text{suffix}$ 满足：

- A, B, C 为任意字符串（可以为空）。
- prefix 和 suffix 互为反串（可以为空）。
- middle 是长度为奇数的回文串。

最大化 $\text{prefix}+\text{middle}+\text{suffix}$ 的长度。

$n \leq 10^5$

题目链接

CF30E Tricky and Clever Password

可以证明 middle 一定是以某个点为中心的极长回文串，否则可以加长 middle，缩短 prefix 和 suffix 并保持不劣。

CF30E Tricky and Clever Password

可以证明 middle 一定是以某个点为中心的极长回文串，否则可以加长 middle，缩短 prefix 和 suffix 并保持不劣。

因此先枚举以 i 为中心的极长回文串 $[l, r]$，接下来的问题为：

- 找到最大的 k，使得 $s[n-k+1, n]$ 的反串为 $s[1, l-1]$ 的子串。

提示：

CF30E Tricky and Clever Password

可以证明 middle 一定是以某个点为中心的极长回文串，否则可以加长 middle，缩短 prefix 和 suffix 并保持不劣。

因此先枚举以 i 为中心的极长回文串 $[l, r]$，接下来的问题为：

- 找到最大的 k，使得 $s[n-k+1, n]$ 的反串为 $s[1, l-1]$ 的子串。

提示：二分答案。

CF30E Tricky and Clever Password

可以证明 middle 一定是以某个点为中心的极长回文串，否则可以加长 middle，缩短 prefix 和 suffix 并保持不劣。

因此先枚举以 i 为中心的极长回文串 $[l,r]$，接下来的问题为：

- 找到最大的 k，使得 $s[n-k+1,n]$ 的反串为 $s[1,l-1]$ 的子串。

提示：二分答案。

先通过两遍 Z 算法预处理 z_i 表示 LCP$(s[i,n],rev(s))$，二分答案为 k，则判定方法为 $\max z_{1..l-k}\geq k$，预处理前缀 max 即可。

CF30E Tricky and Clever Password

可以证明 middle 一定是以某个点为中心的极长回文串，否则可以加长 middle，缩短 prefix 和 suffix 并保持不劣。

因此先枚举以 i 为中心的极长回文串 $[l, r]$，接下来的问题为：

- 找到最大的 k，使得 $s[n-k+1, n]$ 的反串为 $s[1, l-1]$ 的子串。

提示：二分答案。

先通过两遍 Z 算法预处理 z_i 表示 LCP$(s[i, n], rev(s))$，二分答案为 k，则判定方法为 $\max z_{1..l-k} \geq k$，预处理前缀 max 即可。

复杂度 $O(n \log n)$，思考：怎么做到 $O(n)$。

NOIP2020 字符串匹配

小 C 需要找到字符串 S 的所有具有下列形式的拆分方案数：求 $S=(AB)^iC$ 的方案数，其中 $F(A)\le F(C)$，$F(S)$ 表示字符串 S 中出现奇数次的字符的数量，$(AB)^i$ 表示 AB 重复 i 遍。两种方案不同当且仅当拆分出的 A、B、C 中有至少一个字符串不同。

要求复杂度 $O(|S|)$。

提示：可以用到 Z 函数。

题目链接

NOIP2020 字符串匹配

注意到 AB 是 S 的前缀，因此第一个想法是枚举 AB 的长度和 i，然后计算 AB 有多少个前缀 A 满足 $F(A) \leq F(C)$。

NOIP2020 字符串匹配

注意到 AB 是 S 的前缀，因此第一个想法是枚举 AB 的长度和 i，然后计算 AB 有多少个前缀 A 满足 $F(A) \le F(C)$。

具体地，需要维护 cnt_k 表示当前有多少个前缀 A 满足 $F(A) = k$，利用哈希判定 $(AB)^i$ 是否是 S 的前缀，对每个后缀预处理 $F(C)$ 等等。可以做到 $O(|S| \log |S| + 26|S|)$。

NOIP2020 字符串匹配

注意到 AB 是 S 的前缀，因此第一个想法是枚举 AB 的长度和 i，然后计算 AB 有多少个前缀 A 满足 $F(A) \le F(C)$。

具体地，需要维护 cnt_k 表示当前有多少个前缀 A 满足 $F(A) = k$，利用哈希判定 $(AB)^i$ 是否是 S 的前缀，对每个后缀预处理 $F(C)$ 等等。可以做到 $O(|S| \log |S| + 26|S|)$。

接下来会发现不枚举 i 也是可以的。

NOIP2020 字符串匹配

注意到 AB 是 S 的前缀，因此第一个想法是枚举 AB 的长度和 i，然后计算 AB 有多少个前缀 A 满足 $F(A) \leq F(C)$。

具体地，需要维护 cnt_k 表示当前有多少个前缀 A 满足 $F(A) = k$，利用哈希判定 $(AB)^i$ 是否是 S 的前缀，对每个后缀预处理 $F(C)$ 等等。可以做到 $O(|S| \log |S| + 26|S|)$。

接下来会发现不枚举 i 也是可以的。

$F(C)$ 的定义为出现奇数次的字符的数量，因此 $F(C)$ 只和 i 的奇偶性有关，当 i 为奇数时都和 $i = 1$ 相同，当 i 为偶数时 $F(C)$ 固定为 $F(S)$。

NOIP2020 字符串匹配

即使这样，也需要知道 i 最大能达到多少，用哈希不能做到 $O(1)$，但是 Z 函数就可以。

NOIP2020 字符串匹配

即使这样，也需要知道 i 最大能达到多少，用哈希不能做到 $O(1)$，但是 Z 函数就可以。

至此，我们考虑 $i=1$ 的情况，把所有前缀 A 的 $F(A)$ 插入树状数组，可以做到 $O(|S|\log 26)$。而 i 为偶数的情况更简单，容易做到 $O(|S|)$。

NOIP2020 字符串匹配

即使这样，也需要知道 i 最大能达到多少，用哈希不能做到 $O(1)$，但是 Z 函数就可以。

至此，我们考虑 $i=1$ 的情况，把所有前缀 A 的 $F(A)$ 插入树状数组，可以做到 $O(|S|\log 26)$。而 i 为偶数的情况更简单，容易做到 $O(|S|)$。

进一步优化需要用到一个事实，当 AB 的长度 $+1$ 时，$F(C)$ 只会 ± 1。把问题抽象出来就是：

- 维护一个序列 cnt，支持单点加和查询前缀和，每次查询的位置只会 ± 1。

NOIP2020 字符串匹配

即使这样，也需要知道 i 最大能达到多少，用哈希不能做到 $O(1)$，但是 Z 函数就可以。

至此，我们考虑 $i=1$ 的情况，把所有前缀 A 的 $F(A)$ 插入树状数组，可以做到 $O(|S|\log 26)$。而 i 为偶数的情况更简单，容易做到 $O(|S|)$。

进一步优化需要用到一个事实，当 AB 的长度 $+1$ 时，$F(C)$ 只会 ± 1。把问题抽象出来就是：

- 维护一个序列 cnt，支持单点加和查询前缀和，每次查询的位置只会 ± 1。

维护 $sum=\sum_{i=0}^{F(C)} cnt_i$，单点加和与 $F(C)\pm 1$ 时都能 $O(1)$ 修改 sum。最终做到了 $O(|S|)$。

CF1073G Yet Another LCP Problem

给定长度为 n 的字符串 s 和 q 个查询，每个查询由两个整数集合 $a = \{a_1, a_2, \ldots, a_k\}$ 和 $b = \{b_1, b_2, \ldots, b_l\}$ 组成。计算每个查询的结果 $\sum_{i=1}^{k} \sum_{j=1}^{l} \mathrm{LCP}(s[a_i, n], s[b_j, n])$。

$$n, q \leq 2 \times 10^5, \sum |a|, \sum |b| \leq 2 \times 10^5$$

题目链接

CF1073G Yet Another LCP Problem

先建立 SA 和 height 数组，使得我们可以 $O(1)$ 查询 LCP。

把集合 a 和 b 放在一起得到可重集 S，并把 S 按照 SA 的 rank 排序。定义 $h_i = \text{LCP}(S_i, S_{i-1})$，则 S 中两个字符串的 LCP 为 h 上一段区间的最小值。

CF1073G Yet Another LCP Problem

先建立 SA 和 height 数组，使得我们可以 $O(1)$ 查询 LCP。

把集合 a 和 b 放在一起得到可重集 S，并把 S 按照 SA 的 rank 排序。定义 $h_i = \text{LCP}(S_i, S_{i-1})$，则 S 中两个字符串的 LCP 为 h 上一段区间的最小值。

问题转化为：序列上有 a 和 b 两种点，定义两点间的贡献为两点所构成区间的最小值，求所有 a 点和所有 b 点之间的贡献总和。

CF1073G Yet Another LCP Problem

先建立 SA 和 height 数组，使得我们可以 $O(1)$ 查询 LCP。

把集合 a 和 b 放在一起得到可重集 S，并把 S 按照 SA 的 rank 排序。定义 $h_i = \text{LCP}(S_i, S_{i-1})$，则 S 中两个字符串的 LCP 为 h 上一段区间的最小值。

问题转化为：序列上有 a 和 b 两种点，定义两点间的贡献为两点所构成区间的最小值，求所有 a 点和所有 b 点之间的贡献总和。

钦定最小值的位置为 i，则左端点限制于某个区间 $[L_i, i]$，右端点限制于某个区间 $[i, R_i]$，其中 L_i 和 R_i 可以通过单调栈求出。

CF914F Substrings in a String

给定一个字符串 s，处理 q 次查询，每次查询有以下两种形式之一：

- 1 i c —将字符串的第 i 个字符改为 c
- 2 l r y —考虑从位置 l 到 r 的子串，输出 y 在其中作为子串出现的次数

$|s| \leq 10^5$，$q \leq 10^5$，第二种查询的所有 y 长度之和不超过 10^5

提示：

题目链接

CF914F Substrings in a String

给定一个字符串 s，处理 q 次查询，每次查询有以下两种形式之一：

- 1 i c —将字符串的第 i 个字符改为 c
- 2 l r y —考虑从位置 l 到 r 的子串，输出 y 在其中作为子串出现的次数

$|s| \le 10^5$，$q \le 10^5$，第二种查询的所有 y 长度之和不超过 10^5

提示：复杂度 $O(\frac{n^2}{w})$。

题目链接

CF914F Substrings in a String

对于每次询问，可以使用 bitset 求出 y 在 s 中的出现位置集合（左端点）。

CF914F Substrings in a String

对于每次询问，可以使用 bitset 求出 y 在 s 中的出现位置集合（左端点）。

具体地，设 pos_c 表示字符 c 在 s 中的出现位置集合，那么 y 在 s 中的出现位置集合为：

$$\text{AND}_{i=1}^{|y|} pos_{y_i} >> (i-1)$$

该结果在 $[l, r-|y|+1]$ 中 1 的数量就是答案。

CF914F Substrings in a String

对于每次询问，可以使用 bitset 求出 y 在 s 中的出现位置集合（左端点）。

具体地，设 pos_c 表示字符 c 在 s 中的出现位置集合，那么 y 在 s 中的出现位置集合为：

$$\text{AND}_{i=1}^{|y|} pos_{y_i} >> (i-1)$$

该结果在 $[l, r-|y|+1]$ 中 1 的数量就是答案。

思考：怎么提取 bitset 在一个区间中 1 的数量。

CF914F Substrings in a String

对于每次询问，可以使用 bitset 求出 y 在 s 中的出现位置集合（左端点）。

具体地，设 pos_c 表示字符 c 在 s 中的出现位置集合，那么 y 在 s 中的出现位置集合为：

$$\text{AND}_{i=1}^{|y|} pos_{y_i} >> (i-1)$$

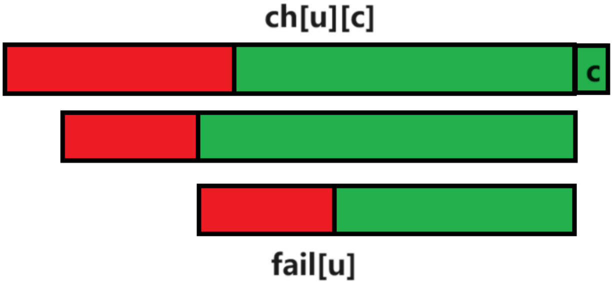
该结果在 $[l, r-|y|+1]$ 中 1 的数量就是答案。

思考：怎么提取 bitset 在一个区间中 1 的数量。

单次询问复杂度为 $O(|y| \cdot \frac{n}{w})$，总复杂度为 $O(\frac{n^2}{w})$。

SAM 简单应用

在 SAM 中，有两个重要的数组：$ch_{u,c}$ 表示等价类 u 中的字符串向后添加字符 c 后所在的等价类；$fail_u$ 表示等价类 u 中最短子串删去首字符后所在的等价类。



如果 i 向 $fail_i$ 连边，会得到一棵树，我们称之为 parent tree。

SAM 简单应用

给定一个长度为 n 的字符串，求它的本质不同子串个数。

$n \leq 10^6$

SAM 简单应用

给定一个长度为 n 的字符串，求它的本质不同子串个数。

$n \leq 10^6$

用 fail 做：$\sum_u len(u) - len(fail_u)$。

SAM 简单应用

给定一个长度为 n 的字符串，求它的本质不同子串个数。

$n \le 10^6$

用 fail 做：$\sum_u len(u) - len(fail_u)$。

用 ch 做：这相当于从起点开始走 ch 的路径数，直接 DP 即可。

SAM 简单应用

给定一个长度为 n 的字符串，求其本质不同子串中的第 k 小子串。

$n \le 10^6$

SAM 简单应用

给定一个长度为 n 的字符串，求其本质不同子串中的第 k 小子串。

$n \le 10^6$

相当于求字典序第 k 小的路径。

SAM 简单应用

给定一个长度为 n 的字符串，求其本质不同子串中的第 k 小子串。

$n \leq 10^6$

相当于求字典序第 k 小的路径。

先用 DP 求出路径数。对于节点 u，它的路径按照字典序划分为：

SAM 简单应用

给定一个长度为 n 的字符串，求其本质不同子串中的第 k 小子串。

$n \leq 10^6$

相当于求字典序第 k 小的路径。

先用 DP 求出路径数。对于节点 u，它的路径按照字典序划分为：

- 在 u 停止的路径。
- 下一步走到 $ch_{u,a}$ 的路径。
- 下一步走到 $ch_{u,b}$ 的路径。
- ...
- 下一步走到 $ch_{u,z}$ 的路径。

SAM 简单应用

给定一个长度为 n 的字符串，求其本质不同子串中的第 k 小子串。

$n \leq 10^6$

相当于求字典序第 k 小的路径。

先用 DP 求出路径数。对于节点 u，它的路径按照字典序划分为：

- 在 u 停止的路径。
- 下一步走到 $ch_{u,a}$ 的路径。
- 下一步走到 $ch_{u,b}$ 的路径。
- …
- 下一步走到 $ch_{u,z}$ 的路径。

通过每个节点的 dp 可以确定第 k 小子串在哪一类，然后在这一类里继续找。

SAM 简单应用

给定一个长度为 n 的字符串，对于每个位置 i，求出最小的 l，使得存在一个长度为 l 的子串 $S[j, j+l-1]$ 满足：

- $i \in [j, j+l-1]$
- $S[j, j+l-1]$ 在 S 中只出现一次。

$n \leq 5 \times 10^5$

SAM 简单应用

给定一个长度为 n 的字符串，对于每个位置 i，求出最小的 l，使得存在一个长度为 l 的子串 $S[j, j+l-1]$ 满足：

- $i \in [j, j+l-1]$
- $S[j, j+l-1]$ 在 S 中只出现一次。

$n \le 5 \times 10^5$

只出现一次的子串即为所有 $|Endpos(u)| = 1$ 的等价类对应的子串。

SAM 简单应用

给定一个长度为 n 的字符串，对于每个位置 i，求出最小的 l，使得存在一个长度为 l 的子串 $S[j, j+l-1]$ 满足：

- $i \in [j, j+l-1]$
- $S[j, j+l-1]$ 在 S 中只出现一次。

$n \leq 5 \times 10^5$

只出现一次的子串即为所有 $|Endpos(u)| = 1$ 的等价类对应的子串。

对于每个等价类，考察它对答案的贡献：对 $[x - len(fail_u), x]$ 贡献一个定值，对 $[x - len(u), x - len(fail_u) - 1]$ 贡献一个公差为 -1 的等差数列。

SAM 简单应用

给定一个长度为 n 的字符串，对于每个位置 i，求出最小的 l，使得存在一个长度为 l 的子串 $S[j, j+l-1]$ 满足：

- $i \in [j, j+l-1]$
- $S[j, j+l-1]$ 在 S 中只出现一次。

$n \le 5 \times 10^5$

只出现一次的子串即为所有 $|Endpos(u)| = 1$ 的等价类对应的子串。

对于每个等价类，考察它对答案的贡献：对 $[x - len(fail_u), x]$ 贡献一个定值，对 $[x - len(u), x - len(fail_u) - 1]$ 贡献一个公差为 -1 的等差数列。

两部分分别使用排序和并查集即可。

SAM 简单应用

给定一个长度为 n 的字符串，对于每个位置 i，求出最小的 l，使得存在一个长度为 l 的子串 $S[j, j+l-1]$ 满足：

- $i \in [j, j+l-1]$
- $S[j, j+l-1]$ 在 S 中只出现一次。

$n \leq 5 \times 10^5$

只出现一次的子串即为所有 $|Endpos(u)| = 1$ 的等价类对应的子串。

对于每个等价类，考察它对答案的贡献：对 $[x - len(fail_u), x]$ 贡献一个定值，对 $[x - len(u), x - len(fail_u) - 1]$ 贡献一个公差为 -1 的等差数列。

两部分分别使用排序和并查集即可。

将只出现一次改为恰好出现 k 次的方法也是一样的。

SAM 简单应用

给定一个长度为 n 的字符串 S，在 SAM 上定位 $S[l,r]$，要求 $\log n$。

SAM 简单应用

给定一个长度为 n 的字符串 S，在 SAM 上定位 $S[l,r]$，要求 $\log n$。

$S[l,r]$ 是 $S[1,r]$ 的 fail 树上的祖先。定位 $S[l,r]$ 相当于找到点 u 满足 $len_{fail_u} < r-l+1 \leq len_u$ 。

因为一个点到根的路径一定满足 len 递减，因此树上倍增即可。

SAM 简单应用

给定一个长度为 n 的字符串， q 次查询 $S[l,r]$ 在 $S[x,y]$ 中的出现次数。

可以离线/强制在线

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

SAM 简单应用

给定一个长度为 n 的字符串，q 次查询 $S[l,r]$ 在 $S[x,y]$ 中的出现次数。

可以离线/强制在线

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

首先定位 $S[l,r]$，它在原串中的出现位置集合（右端点）即为所在等价类的 Endpos。

问题转化为计算 Endpos 中有多少元素在 $[x+(r-l), y]$ 中，线段树合并 Endpos 即可。

SAM 简单应用

给定一个长度为 n 的字符串，q 次查询 $S[l,r]$ 在 $S[x,y]$ 中的出现次数。

可以离线/强制在线

$n, q \le 5 \times 10^5$

首先定位 $S[l,r]$，它在原串中的出现位置集合（右端点）即为所在等价类的 Endpos。

问题转化为计算 Endpos 中有多少元素在 $[x+(r-l), y]$ 中，线段树合并 Endpos 即可。

如果不强制在线，可以把 Endpos 看做子树，然后转二维数点，用树状数组做。

SAM 简单应用

给定两个字符串 S 和 T，对于 T 的每一个前缀 $T[1,i]$，求其最长的后缀 $T[j,i]$，使其是 S 的子串。

$n \leq 10^6$

SAM 简单应用

给定两个字符串 S 和 T，对于 T 的每一个前缀 $T[1,i]$，求其最长的后缀 $T[j,i]$，使其是 S 的子串。

$n \le 10^6$

对 S 建立 SAM，将 T 在 SAM 上匹配。

如果已经求出 i 对应的 $T[j,i]$，考虑求 $i+1$ 的答案，显然新的 j 不会小于之前的。

SAM 简单应用

给定两个字符串 S 和 T，对于 T 的每一个前缀 $T[1,i]$，求其最长的后缀 $T[j,i]$，使其是 S 的子串。

$n \le 10^6$

对 S 建立 SAM，将 T 在 SAM 上匹配。

如果已经求出 i 对应的 $T[j,i]$，考虑求 $i+1$ 的答案，显然新的 j 不会小于之前的。

设下一个字符为 c，如果 $T[j,i]$ 所在等价类存在 $ch_{u,c}$，那么 $T[j,i+1]$ 就合法。

SAM 简单应用

给定两个字符串 S 和 T，对于 T 的每一个前缀 $T[1,i]$，求其最长的后缀 $T[j,i]$，使其是 S 的子串。

$n \le 10^6$

对 S 建立 SAM，将 T 在 SAM 上匹配。

如果已经求出 i 对应的 $T[j,i]$，考虑求 $i+1$ 的答案，显然新的 j 不会小于之前的。

设下一个字符为 c，如果 $T[j,i]$ 所在等价类存在 $ch_{u,c}$，那么 $T[j,i+1]$ 就合法。

否则增大 j 直到合法，相当于一直跳 $fail$，直到 $ch_{u,c}$ 存在。复杂度分析和双指针相同，都是 $O(n)$。

CF232D Fence

给定一个长度为 n 的序列 $h_{1..n}$。有 q 次查询，每次查询给定区间 $[l, r]$，计算有多少个与该区间匹配的区间。

两个区间 $[l_1, r_1], [l_2, r_2]$ 匹配的条件是：

- 区间不重叠；
- 区间长度相同；
- 对于每个 i，满足 $h_{l_1+i} + h_{l_2+i} = h_{l_1} + h_{l_2}$。

$n, q \leq 10^5, 1 \leq h_i \leq 10^9$

尝试运用 SAM。

题目链接

CF232D Fence

分析第三个条件说明 $h_{l_1+i} + h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

CF232D Fence

分析第三个条件说明 $h_{l_1+i} + h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

定义 $\Delta h_i = h_{i+1} - h_i, \Delta h'_i = -\Delta h_i$，那么第三个条件可以改写为：$\Delta h[l_1, r_1 - 1] = \Delta h'[l_2, r_2 - 1]$。

CF232D Fence

分析第三个条件说明 $h_{l_1+i}+h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

定义 $\Delta h_i = h_{i+1} - h_i, \Delta h'_i = -\Delta h_i$，那么第三个条件可以改写为：$\Delta h[l_1, r_1 - 1] = \Delta h'[l_2, r_2 - 1]$。

对 Δh 和 $\Delta h'$ 拼接后建立 SAM。对于一次查询 $[l, r]$，先用倍增定位 $\Delta h[l, r-1]$ 所在的 SAM 结点，那么 $\Delta h'[l_2, r_2 - 1]$ 的出现位置集合也就知道了。

CF232D Fence

分析第三个条件说明 $h_{l_1+i}+h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

定义 $\Delta h_i = h_{i+1}-h_i, \Delta h'_i = -\Delta h_i$，那么第三个条件可以改写为：$\Delta h[l_1, r_1-1] = \Delta h'[l_2, r_2-1]$。

对 Δh 和 $\Delta h'$ 拼接后建立 SAM。对于一次查询 $[l, r]$，先用倍增定位 $\Delta h[l, r-1]$ 所在的 SAM 结点，那么 $\Delta h'[l_2, r_2-1]$ 的出现位置集合也就知道了。

Endpos 线段树合并当然可以做，但是有更简单的方法。

CF232D Fence

分析第三个条件说明 $h_{l_1+i}+h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

定义 $\Delta h_i=h_{i+1}-h_i,\Delta h'_i=-\Delta h_i$，那么第三个条件可以改写为：$\Delta h[l_1,r_1-1]=\Delta h'[l_2,r_2-1]$。

对 Δh 和 $\Delta h'$ 拼接后建立 SAM。对于一次查询 $[l,r]$，先用倍增定位 $\Delta h[l,r-1]$ 所在的 SAM 结点，那么 $\Delta h'[l_2,r_2-1]$ 的出现位置集合也就知道了。

Endpos 线段树合并当然可以做，但是有更简单的方法。

我们需要计算有多少 i 满足以下两个条件：

- $i\in endpos(s)$，即 $s[1,i]$ 代表的结点在 u 子树中（parent tree 上）。
- $[i-(r-l)+1,i+1]$ 和 $[l,r]$ 不重叠，即 $i\notin[l-1,2r-l-1]$。

CF232D Fence

分析第三个条件说明 $h_{l_1+i}+h_{l_2+i}$ 是一个定值，这实际上表示差分互为相反数。

定义 $\Delta h_i = h_{i+1}-h_i, \Delta h'_i = -\Delta h_i$，那么第三个条件可以改写为：$\Delta h[l_1, r_1-1] = \Delta h'[l_2, r_2-1]$。

对 Δh 和 $\Delta h'$ 拼接后建立 SAM。对于一次查询 $[l, r]$，先用倍增定位 $\Delta h[l, r-1]$ 所在的 SAM 结点，那么 $\Delta h'[l_2, r_2-1]$ 的出现位置集合也就知道了。

Endpos 线段树合并当然可以做，但是有更简单的方法。

我们需要计算有多少 i 满足以下两个条件：

- $i \in endpos(s)$，即 $s[1, i]$ 代表的结点在 u 子树中（parent tree 上）。
- $[i-(r-l)+1, i+1]$ 和 $[l, r]$ 不重叠，即 $i \notin [l-1, 2r-l-1]$。

相当于二维数点，离线后树状数组处理即可，复杂度 $O(n\log n)$。

P6292 区间本质不同子串个数

给定一个长度为 n 字符串 S，每次查询给定区间 $[l, r]$，计算该子串中有多少个本质不同的非空子串。

$n \leq 10^5, m \leq 2 \times 10^5$

提示：区间数颜色数量是怎么做的?

题目链接

P6292 区间本质不同子串个数

先回顾区间数颜色的方法：采用扫描线的思路，逐步移动右端点 r，设 $last_c$ 表示颜色 c 在 $[1,r]$ 中最后一次出现的位置，当颜色 c 在 r 出现时，就会在 $last_c$ 处 -1，r 处 $+1$。

P6292 区间本质不同子串个数

先回顾区间数颜色的方法：采用扫描线的思路，逐步移动右端点 r，设 $last_c$ 表示颜色 c 在 $[1, r]$ 中最后一次出现的位置，当颜色 c 在 r 出现时，就会在 $last_c$ 处 -1，r 处 $+1$。

类比这个想法，对每个本质不同的子串 T，设 $last_T$ 表示 T 在 $S[1, r]$ 中最后一次出现的位置（右端点），那么当查询的左端点取到 $[1, last_T - |T| + 1]$ 这个区间内时，T 会对答案产生 1 的贡献。

P6292 区间本质不同子串个数

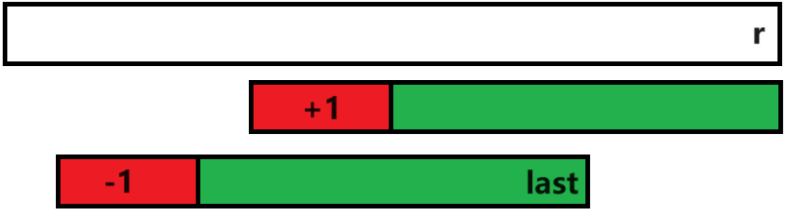
先回顾区间数颜色的方法：采用扫描线的思路，逐步移动右端点 r，设 $last_c$ 表示颜色 c 在 $[1,r]$ 中最后一次出现的位置，当颜色 c 在 r 出现时，就会在 $last_c$ 处 -1，r 处 $+1$。

类比这个想法，对每个本质不同的子串 T，设 $last_T$ 表示 T 在 $S[1,r]$ 中最后一次出现的位置（右端点），那么当查询的左端点取到 $[1, last_T - |T| + 1]$ 这个区间内时，T 会对答案产生 1 的贡献。

本质不同的子串总数是 $O(n^2)$，因此我们把思路转换到 SAM 的 $O(n)$ 个等价类上，因为每个等价类中的字符串总是具有相同的 $last_T$。

P6292 区间本质不同子串个数

初步做法：当右端点从 $r-1$ 移动到 r 时，会新出现 r 个子串，它们在 parent tree 上对应了 $S[1,r]$ 所在等价类到根的路径，枚举这条路径上的等价类，在 $last_T$ 处撤销贡献，r 处新增贡献，最后将整条路径的 $last_T$ 赋值为 r。



P6292 区间本质不同子串个数

注意到“将整条路径的 $last_T$ 赋值为 r”这个操作和 LCT 关系密切，使用 LCT 完成赋值操作的话，LCT 上的每条实链将具有相同的 $last_T$。

P6292 区间本质不同子串个数

注意到“将整条路径的 $last_T$ 赋值为 r”这个操作和 LCT 关系密切，使用 LCT 完成赋值操作的话，LCT 上的每条实链将具有相同的 $last_T$。

因此，我们可以在 access 的过程中完成撤销贡献和新增贡献的操作。

P6292 区间本质不同子串个数

注意到“将整条路径的 $last_T$ 赋值为 r”这个操作和 LCT 关系密切，使用 LCT 完成赋值操作的话，LCT 上的每条实链将具有相同的 $last_T$。

因此，我们可以在 access 的过程中完成撤销贡献和新增贡献的操作。

最终，问题转化为了 $O(n \log n)$ 次区间加，$O(m)$ 次区间求和，使用树状数组即可。

CF1276F Asterisk Substrings（选讲）

给定一个长度为 n 的字符串 s，将 s 中的每个字符分别替换为星号'*'，形成字符串集合 $\{s, t_1, \ldots, t_n\}$。其中，t_i 表示将 s 的第 i 个字符替换为'*' 后的字符串。

计算在这个字符串集合中出现的所有本质不同子字符串的数量（包括空字符串）。注意'*' 不是通配符。

$n \le 10^5$

题目链接

谢谢大家！