字符串速通

xay5421

2025 年 2 月 9 日

目录

目录

定义 1.1

(哈希函数). 哈希函数是一个将字符串映射到整数的函数。

最常用的哈希函数是:

$$f(S) = \sum_{i=1}^{n} S_i \times B^{n-i} \bmod P$$

若两个字符串 S 和 T 满足 $f(S) \neq f(T)$,则 $S \neq T$,否则我们认为 $S = T$。
若 $S \neq T$ 且 $f(S) = f(T)$,我们称为哈希碰撞,这需要尽量避免。

Hash is all you need

定义 1.1

(哈希函数). 哈希函数是一个将字符串映射到整数的函数。

最常用的哈希函数是:

$$f(S) = \sum_{i=1}^{n} S_i \times B^{n-i} \bmod P$$

若两个字符串 S 和 T 满足 $f(S) \neq f(T)$,则 $S \neq T$,否则我们认为 $S = T$。
若 $S \neq T$ 且 $f(S) = f(T)$,我们称为哈希碰撞,这需要尽量避免。
为了降低哈希碰撞概率,可以取多个不同的 B 和 P,一旦有一组 (B, P) 哈希值不同,就认为两个字符串不同,称为多模数哈希。

Hash is all you need

定义 1.1

(哈希函数). 哈希函数是一个将字符串映射到整数的函数。

最常用的哈希函数是:

$$f(S) = \sum_{i=1}^{n} S_i \times B^{n-i} \bmod P$$

若两个字符串 S 和 T 满足 $f(S) \neq f(T)$，则 $S \neq T$，否则我们认为 $S = T$。
若 $S \neq T$ 且 $f(S) = f(T)$，我们称为哈希碰撞，这需要尽量避免。
为了降低哈希碰撞概率，可以取多个不同的 B 和 P，一旦有一组 (B, P) 哈希值不同，就认为两个字符串不同，称为多模数哈希。
也有不取模，用 unsigned long long 自然溢出的哈希（即 $P = 2^{64}$），优点是 P 足够大可以当双（模数）哈希用，并且快，缺点是 P 被知道了，容易被叉。

$O(n)$ 预处理，$O(1)$ 查询区间哈希值。

```
const int P = 998244853, B = 31;
int ghs(int l,int r) {
    return (h[r] - 1ll * h[l - 1] * pw[r - l + 1] % P + P) % P;
}
void init(int n) {
    for (int i = 1; i <= n; ++i) {
        h[i] = (1ll * h[i - 1] * B + s[i] - 'a' + 1) % P;
        pw[i] = 1ll * pw[i - 1] * B % P;
    }
}
```

单点修改维护哈希值

例题 1.1

给定一个字符串 S，支持以下操作：

- 修改 S 的第 i 个字符为 c。
- 查询 S 的子串 $S[l;r]$ 的哈希值。

单点修改维护哈希值

例题 1.1

给定一个字符串 S，支持以下操作：

- 修改 S 的第 i 个字符为 c。
- 查询 S 的子串 $S[l;r]$ 的哈希值。

题解

哈希值支持合并，因此可以用线段树维护每个区间的哈希值。
哈希值可减，也可以用树状数组维护。

后缀排序

例题 1.2

给定一个字符串 S，求出所有后缀的字典序排序。
如 $S=$ “abaa”，则后缀排序为：

$$\text{“a”} < \text{“aa”} < \text{“abaa”} < \text{“baa”}$$

因此输出为 $4,3,1,2$。
$|S| \leq 10^5$。

后缀排序

例题 1.2

给定一个字符串 S，求出所有后缀的字典序排序。
如 $S =$ “*abaa*”，则后缀排序为：

$$\text{“}a\text{''} < \text{“}aa\text{''} < \text{“}abaa\text{''} < \text{“}baa\text{''}$$

因此输出为 $4, 3, 1, 2$。
$|S| \leq 10^5$。

题解

字符串哈希可以判断两个字符串是否相等，通过二分两个串公共前缀长度可以支持比较，通过比较就能实现排序。(比较次数较多，建议使用双哈希)
总时间复杂度 $O(n \log^2 n)$。
实际上后缀排序还有复杂度更小的方法，比如后缀数组。(但都没哈希好写)

```cpp
iota(id+1,id+n+1,1);
sort(id+1,id+n+1,[&](int x,int y){
    int l=1,r=n-max(x,y)+1,ret=0;
    while(l<=r){
        int mid=(l+r)>>1;
        if(ghs(x,x+mid-1)==ghs(y,y+mid-1))l=mid+1,ret=mid;
        else r=mid-1;
    }
    return s[x+ret]<s[y+ret];
});
```

二分哈希也可以用于求两个串的最长公共前缀，单次查询复杂度 $O(\log n)$。

BZOJ 2795

定义 1.2

(循环节). 对于字符串 S，如果存在字符串 T 和正整数 k，满足 $S = T^k$，则称 T 是 S 的循环节。

例题 1.3

多次询问一个字符串的一段子串的最小循环节长度。
$n \leq 5 \times 10^5, m \leq 2 \times 10^6$。

BZOJ 2795

引理 1.1

S 的循环节长度一定是 $|S|$ 的约数。

BZOJ 2795

引理 1.1

S 的循环节长度一定是 $|S|$ 的约数。

引理 1.2

如果 d 是合法循环节长度，那么 kd（其中 kd 也是 $|S|$ 的约数）也一定是合法的循环节长度。

BZOJ 2795

引理 1.1

S 的循环节长度一定是 $|S|$ 的约数。

引理 1.2

如果 d 是合法循环节长度，那么 kd（其中 kd 也是 $|S|$ 的约数）也一定是合法的循环节长度。

引理 1.3

如果 d_1, d_2 是合法循环节长度，那么 $\gcd(d_1, d_2)$ 也一定是合法循环节长度。

BZOJ 2795

引理 1.1

S 的循环节长度一定是 $|S|$ 的约数。

引理 1.2

如果 d 是合法循环节长度，那么 kd（其中 kd 也是 $|S|$ 的约数）也一定是合法的循环节长度。

引理 1.3

如果 d_1, d_2 是合法循环节长度，那么 $\gcd(d_1, d_2)$ 也一定是合法循环节长度。

证明

即只要在范围内，有 $S[i] = S[i + d_1]$ 和 $S[i] = S[i + d_2]$。如果无范围的限制，根据裴蜀定理，有 $S[i] = S[i + \gcd(d_1, d_2)]$。有范围限制，只需要 $d_1 + d_2 \leq |S|$ 即可 (不妨设 $\gcd(d_1, d_2) = k_1 d_1 - k_2 d_2$，$(k_1, k_2 > 0)$，那任意时刻加 d_1 和减 d_2 必有合法的)。因此如果 d_1, d_2 都不等于 $|S|$，说明 $\leq \frac{|S|}{2}$，是满足的，有等于也显然是满足的。

BZOJ 2795

题解

枚举循环节长度 d，判断是否存在长度为 d 的循环节（用哈希判断子串 $s[l;r-d]$ 和 $s[l+d;r]$ 是否相等），这得到了一个 $O(d(n))$ 的做法。

BZOJ 2795

题解

枚举循环节长度 d，判断是否存在长度为 d 的循环节（用哈希判断子串 $s[l;r-d]$ 和 $s[l+d;r]$ 是否相等），这得到了一个 $O(d(n))$ 的做法。
如何更快？根据引理，我们可以枚举 $r-l+1$ 的所有质因子，来求出这个质因子在最短循环节中每个质因子的指数次数。具体来说，对于现有循环节，判断除以质因子后是否还是循环节，如果是则除，不是则不变。

BZOJ 2795

题解

枚举循环节长度 d，判断是否存在长度为 d 的循环节（用哈希判断子串 $s[l;r-d]$ 和 $s[l+d;r]$ 是否相等)，这得到了一个 $O(d(n))$ 的做法。
如何更快？根据引理，我们可以枚举 $r-l+1$ 的所有质因子，来求出这个质因子在最短循环节中每个质因子的指数次数。具体来说，对于现有循环节，判断除以质因子后是否还是循环节，如果是则除，不是则不变。
单次询问复杂度至多是质因子个数，总复杂度 $O(m\log n)$。

???

例题 1.4

模 m 意义下的 *01* 背包, 询问每个容量能否装到。$n, m \le 3 \times 10^5$

???

例题 1.4

模 m 意义下的 *01* 背包, 询问每个容量能否装到。$n, m \le 3 \times 10^5$

题解

由于背包大小较小，背包状态切换次数至多 m 次，因此如果有办法快速找到一个改变的位置，问题就解决了。

???

例题 1.4

模 m 意义下的 *01* 背包, 询问每个容量能否装到。$n, m \leq 3 \times 10^5$

题解

由于背包大小较小，背包状态切换次数至多 m 次，因此如果有办法快速找到一个改变的位置，问题就解决了。
形式化地说，如果 old_i 上轮是 1，但 $new_{(i+x) \bmod m}$ 是 0，那么就把 $new_{(i+x) \bmod m}$ 赋值为 1。

???

例题 1.4

模 m 意义下的 *01* 背包, 询问每个容量能否装到。$n, m \le 3 \times 10^5$

题解

由于背包大小较小，背包状态切换次数至多 m 次，因此如果有办法快速找到一个改变的位置，问题就解决了。
形式化地说，如果 old_i 上轮是 1，但 $new_{(i+x) \bmod m}$ 是 0，那么就把 $new_{(i+x) \bmod m}$ 赋值为 1。
我们用二分哈希找到 old 和 $new + x$ 不同的第一个位置，如果 old_i 是 1，那么修改，否则不动。

???

例题 1.4

模 m 意义下的 *01* 背包, 询问每个容量能否装到。$n, m \leq 3 \times 10^5$

题解

由于背包大小较小，背包状态切换次数至多 m 次，因此如果有办法快速找到一个改变的位置，问题就解决了。
形式化地说，如果 old_i 上轮是 1，但 $new_{(i+x) \bmod m}$ 是 0，那么就把 $new_{(i+x) \bmod m}$ 赋值为 1。
我们用二分哈希找到 old 和 $new + x$ 不同的第一个位置，如果 old_i 是 1，那么修改，否则不动。
注意到过程中遇到 *0-1* 的数量和 *1-0* 的数量是相等的，因此每不动一次对应修改一次，复杂度依然正确。

例题 1.4

模 m 意义下的 *01* 背包，询问每个容量能否装到。$n, m \le 3 \times 10^5$

题解

由于背包大小较小，背包状态切换次数至多 m 次，因此如果有办法快速找到一个改变的位置，问题就解决了。

形式化地说，如果 old_i 上轮是 1，但 $new_{(i+x) \bmod m}$ 是 0，那么就把 $new_{(i+x) \bmod m}$ 赋值为 1。

我们用二分哈希找到 old 和 $new + x$ 不同的第一个位置，如果 old_i 是 1，那么修改，否则不动。

注意到过程中遇到 *0-1* 的数量和 *1-0* 的数量是相等的，因此每不动一次对应修改一次，复杂度依然正确。

动态维护哈希值需要树状数组，再加上二分的 $\log n$，总复杂度 $O((n+m)\log^2 n)$

目录

KMP

KMP 算法是一种用于在一个字符串中查找一个子串的算法，其核心思想是利用已经匹配的信息来减少不必要的匹配。

KMP

KMP 算法是一种用于在一个字符串中查找一个子串的算法，其核心思想是利用已经匹配的信息来减少不必要的匹配。

定义 2.1

(border). 对于字符串 s，如果存在一个非空的严格前缀和严格后缀相等，那么这个前缀就是 s 的 *border*。

KMP

KMP 算法是一种用于在一个字符串中查找一个子串的算法，其核心思想是利用已经匹配的信息来减少不必要的匹配。

定义 2.1

(border). 对于字符串 s，如果存在一个非空的严格前缀和严格后缀相等，那么这个前缀就是 s 的 *border*。

定义 2.2

($next$ 数组). 定义 $next_i$ 为 $s[1;i]$ 的最长 *border* 长度

KMP

KMP 算法是一种用于在一个字符串中查找一个子串的算法，其核心思想是利用已经匹配的信息来减少不必要的匹配。

定义 2.1

(border). 对于字符串 s，如果存在一个非空的严格前缀和严格后缀相等，那么这个前缀就是 s 的 *border*。

定义 2.2

($next$ 数组). 定义 $next_i$ 为 $s[1;i]$ 的最长 *border* 长度

KMP

引理 2.1

border 的 *border* 还是 *border*。

KMP

引理 2.1

border 的 *border* 还是 *border*。

引理 2.2

任意 *border* 是最长 *border* 或者最长 *border* 的 *border*。

KMP

引理 2.1

border 的 *border* 还是 *border*。

引理 2.2

任意 *border* 是最长 *border* 或者最长 *border* 的 *border*。

定理 2.1

不断跳最长 *border* 可以枚举出所有的 *border*。

引理 2.1

border 的 *border* 还是 *border*。

引理 2.2

任意 *border* 是最长 *border* 或者最长 *border* 的 *border*。

定理 2.1

不断跳最长 *border* 可以枚举出所有的 *border*。

增量构造 $next$ 数组的算法：顺序枚举 i，设要求 $next_i$，注意到 $next_i - 1$ 必然是 $s[1; i-1]$ 的 border（除非 $next_i = 0$），那么枚举所有 $i-1$ 的 border（不停跳最长 border），找到最长的且后一个字符是 s_i 的 border，其长度加一可得到 $next_i$。

现在知道了 s 的 $next$ 数组，要求 t 的中有多少个 s。
我们顺序枚举 i，维护最长的 j_i，满足 $t[i-j_i+1;i]=s[1;j_i]$。

引理 2.3

设 $s[1;j_i]$ 能匹配 $t[i-j_i+1;i]$，则比 j_i 短的能匹配 $t[1;i]$ 的等长后缀的有且仅有 $s[1;j_i]$ 的 *border*。

现在知道了 s 的 $next$ 数组，要求 t 的中有多少个 s。
我们顺序枚举 i，维护最长的 j_i，满足 $t[i-j_i+1;i]=s[1;j_i]$。

引理 2.3

设 $s[1;j_i]$ 能匹配 $t[i-j_i+1;i]$，则比 j_i 短的能匹配 $t[1;i]$ 的等长后缀的有且仅有 $s[1;j_i]$ 的 *border*。

证明

易证：

- *border* 合法的。
- 合法的必是 *border*。

而 $s[1;j_i]$ 能匹配 $t[i-j_i+1;i]$，必有 $s[1;j_i-1]$ 能匹配 $t[i-j_i+1;i-1]$，则 j_i-1 是 $s[1;j_{i-1}]$ 或它的 border。
因此匹配过程为：令 $j_i=j_{i-1}$，不停 $j_i=next_{j_i}$ 直到 $s_{j_i+1}=t_i$，然后 j_i 自增 1 得到最终的 j_i。

```cpp
void get_next() {
    next[1] = 0;
    for (int i = 2, j = 0; i <= n; ++i) {
        while (j && s[i] != s[j + 1]) j = next[j];
        if (s[i] == s[j + 1]) ++j;
        next[i] = j;
    }
}
void kmp() {
    for (int i = 1, j = 0; i <= m; ++i) {
        while (j && t[i] != s[j + 1]) j = next[j];
        if (t[i] == s[j + 1]) ++j;
        if (j == n) { // find
            j = next[j];
        }
    }
}
```

显然时间复杂度 $O(n+m)$ (不过是均摊的，一个 i 对应 j 跳的次数可以是 O(n) 的)。

BZOJ 1729

例题 2.1

给定两个串 S 和 T，问 S 中有多少子串与 T "匹配"，这里的"匹配"是指离散化后相等。
$|S|,|T| \leq 10^5, |\Sigma| \leq 26$。

BZOJ 1729

例题 2.1

给定两个串 S 和 T，问 S 中有多少子串与 T "匹配"，这里的"匹配"是指离散化后相等。

$|S|, |T| \leq 10^5, |\Sigma| \leq 26$。

Bonus: $|S|, |T|, |\Sigma| \leq 10^7$（即要求线性）。

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

注意到 *border* 的 *border* 还是 *border*，一个 *border* 不是最长 *border* 就是最长 *border* 的 *border*，诸如这些性质并没有消失，因此还是可以使用 *KMP* 算法解决。

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

注意到 *border* 的 *border* 还是 *border*，一个 *border* 不是最长 *border* 就是最长 *border* 的 *border*，诸如这些性质并没有消失，因此还是可以使用 *KMP* 算法解决。

题解 (Bonus)

要做到线性，可以将相等条件描述为对于 i，找到最紧 x 和 y 满足 $S_{i-x} \leq S_i \leq S_{i-y}$，并记录相等或者不相等。

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

注意到 *border* 的 *border* 还是 *border*，一个 *border* 不是最长 *border* 就是最长 *border* 的 *border*，诸如这些性质并没有消失，因此还是可以使用 *KMP* 算法解决。

题解 (Bonus)

要做到线性，可以将相等条件描述为对于 i，找到最紧 x 和 y 满足 $S_{i-x} \le S_i \le S_{i-y}$，并记录相等或者不相等。
比较的时候要求是 $T_{j-x} \le T_j \le T_{j-y}$，并且取等状态相同，这样比较是 $O(1)$ 的。

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

注意到 *border* 的 *border* 还是 *border*，一个 *border* 不是最长 *border* 就是最长 *border* 的 *border*，诸如这些性质并没有消失，因此还是可以使用 *KMP* 算法解决。

题解 (Bonus)

要做到线性，可以将相等条件描述为对于 i，找到最紧 x 和 y 满足 $S_{i-x} \le S_i \le S_{i-y}$，并记录相等或者不相等。

比较的时候要求是 $T_{j-x} \le T_j \le T_{j-y}$，并且取等状态相同，这样比较是 $O(1)$ 的。

举个例子，$S = [10,8,5,7,5,4]$，那么第 4 个字符 7 就记录 $x = 1$ 和 $y = 2$，即 $S[4-1] \le S[4] \le S[4-2]$，即 7 需要在 5 和 8 之间；第 5 个字符 5 就会记录 $x = y = 2$，且取到等号。

BZOJ 1729

题解

在 *KMP* 的基础上，修改字母相等条件为：

- 当前字符串中和当前字符相同的个数相等
- 当前字符串中比当前字符小的个数相等

注意到 *border* 的 *border* 还是 *border*，一个 *border* 不是最长 *border* 就是最长 *border* 的 *border*，诸如这些性质并没有消失，因此还是可以使用 *KMP* 算法解决。

题解 (Bonus)

要做到线性，可以将相等条件描述为对于 i，找到最紧 x 和 y 满足 $S_{i-x} \le S_i \le S_{i-y}$，并记录相等或者不相等。
比较的时候要求是 $T_{j-x} \le T_j \le T_{j-y}$，并且取等状态相同，这样比较是 $O(1)$ 的。
举个例子，$S = [10,8,5,7,5,4]$，那么第 4 个字符 7 就记录 $x = 1$ 和 $y = 2$，即 $S[4-1] \le S[4] \le S[4-2]$，即 7 需要在 5 和 8 之间；第 5 个字符 5 就会记录 $x = y = 2$，且取到等号。
找 x 和 y 可以用链表倒序删除的方式做到线性。

LG P5287 [HNOI2019] JOJO

例题 2.2

有 n 个操作和一个初始为空的串 S，每个操作是两种之一：

- *1 x c:* 在 S 的末尾加入 x 个字符 c。保证此时 S 为空或者末尾字符和 c 不同。
- *2 x:* 将 S 还原到第 x 次操作之后的状态。

每个操作结束后查询 S 的 $next$ 数组元素之和。
保证 $n \leq 10^5$ 且 *1* 操作中的 $x \leq 10^4$。

可持久化 KMP

如果保证 1 操作的 x 是 1，就是可持久化 KMP。
KMP 复杂度是均摊的，如果要支持可持久化，时间复杂度就不能是均摊的了。

可持久化 KMP

如果保证 1 操作的 x 是 1，就是可持久化 KMP。
KMP 复杂度是均摊的，如果要支持可持久化，时间复杂度就不能是均摊的了。
有两种方法可以支持可持久化 KMP：

- 用数组（复杂度字符集）或可持久化线段树（复杂度 $\log$ 字符集）维护每个字符跳 $next$ 最终会跳到哪里。

可持久化 KMP

如果保证 1 操作的 x 是 1，就是可持久化 KMP。
KMP 复杂度是均摊的，如果要支持可持久化，时间复杂度就不能是均摊的了。
有两种方法可以支持可持久化 KMP：

- 用数组（复杂度字符集）或可持久化线段树（复杂度 $\log$ 字符集）维护每个字符跳 $next$ 最终会跳到哪里。
- 注意到在失配时，如果 $i \le 2next_i$，说明有一个长度为 $i - next_i$ 的循环节，可以直接跳到循环节的开头位置 $i \bmod (i - next_i)$。这种方法每次跳长度至少减少一半，因此单次操作复杂度不超过 $O(\log n)$。

LG P5287 [HNOI2019] JOJO

题解

将 (x, c) 的 *pair* 看成字符，可以看作是对这样的串进行 *KMP*（一个特殊情况是当前字符 c 与第一个字符 c 相同，且 x 不小于第一个字符 x，这样是可以匹配的)。

LG P5287 [HNOI2019] JOJO

题解

将 (x, c) 的 *pair* 看成字符，可以看作是对这样的串进行 *KMP*（一个特殊情况是当前字符 c 与第一个字符 c 相同，且 x 不小于第一个字符 x，这样是可以匹配的）。
算最后放入的 x_i 个字符的 $next$ 之和，那么就是不停跳 $next$，看 c_{j+1} 是否等于 c_i 并且算 x_{j+1} 对答案的贡献（贡献是一个区间和的形式，对答案有贡献的 x_{j+1} 单增，且贡献区间是上一个和当前这个形成的区间）。

LG P5287 [HNOI2019] JOJO

题解

将 (x, c) 的 *pair* 看成字符，可以看作是对这样的串进行 *KMP*（一个特殊情况是当前字符 c 与第一个字符 c 相同，且 x 不小于第一个字符 x，这样是可以匹配的）。
算最后放入的 x_i 个字符的 $next$ 之和，那么就是不停跳 $next$，看 c_{j+1} 是否等于 c_i 并且算 x_{j+1} 对答案的贡献（贡献是一个区间和的形式，对答案有贡献的 x_{j+1} 单增，且贡献区间是上一个和当前这个形成的区间）。
用第二种可持久化 *KMP* 优化，每次跳的复杂度 $O(\log n)$，总复杂度 $O(n \log n)$。

定义 2.3

(Z 函数). 定义 $Z[i]$ 为 $s[i;n]$ 和 $s[1;n-i+1]$ 的最长公共前缀长度。

定义 2.3

(Z 函数). 定义 $Z[i]$ 为 $s[i;n]$ 和 $s[1;n-i+1]$ 的最长公共前缀长度。

引理 2.4

设 $i+Z[i]-1 \geq j$，那么 $Z[j] \geq \min(i+Z[i]-j, Z[j-i+1])$。

exKMP

定义 2.3

(Z 函数). 定义 $Z[i]$ 为 $s[i;n]$ 和 $s[1;n-i+1]$ 的最长公共前缀长度。

引理 2.4

设 $i+Z[i]-1 \geq j$，那么 $Z[j] \geq \min(i+Z[i]-j, Z[j-i+1])$。

引理 2.5

如果 $Z[j-i+1] < i+Z[i]-j$，那么 $Z[j] = Z[j-i+1]$。

```
void get_z() {
    z[1] = n;
    for (int i = 2, l = 0, r = 0; i <= n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l + 1]);
        while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}
```

```
void get_z() {
    z[1] = n;
    for (int i = 2, l = 0, r = 0; i <= n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l + 1]);
        while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}
```

复杂度可能有问题的地方在于 while 循环，但是由于每次 while 循环至少会使 r 增加 1，因此总复杂度为 $O(n)$。

```cpp
void get_z() {
    z[1] = n;
    for (int i = 2, l = 0, r = 0; i <= n; ++i) {
        if (i <= r) z[i] = min(r - i + 1, z[i - l + 1]);
        while (i + z[i] <= n && s[i + z[i]] == s[1 + z[i]]) ++z[i];
        if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
    }
}
```

复杂度可能有问题的地方在于 while 循环，但是由于每次 while 循环至少会使 r 增加 1，因此总复杂度为 $O(n)$。

通过线性求 Z 函数，可以得到一个线性求一个串 S 和另一个串 T 的所有后缀的 LCP（最长公共前缀）的算法：将 T 拼接在 S 后，求 Z 函数。(二分哈希可以做到 $O(n \log n)$)

Trie 树

定义 2.4

(Trie 树). *Trie* 树是一种树形结构，用于存储字符串集合，其每个节点代表一个字符串的前缀。可以看作是 $|\Sigma|$ 叉线段树，通常是动态开点的，支持线段树合并、可持久化等线段树操作。

Trie 树

定义 2.4

(Trie 树). *Trie* 树是一种树形结构，用于存储字符串集合，其每个节点代表一个字符串的前缀。可以看作是 $|\Sigma|$ 叉线段树，通常是动态开点的，支持线段树合并、可持久化等线段树操作。

```
void ins(char *s) {
    int u = 0;
    for (int i = 1; s[i]; ++i) {
        int c = s[i] - 'a';
        if (!ch[u][c]) ch[u][c] = ++tot;
        u = ch[u][c];
    }
}
```

AC 自动机

定义 2.5

(AC 自动机). *AC* 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机

定义 2.5

(AC 自动机). *AC* 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机的状态可以看作是一个 Trie 树。

AC 自动机

定义 2.5

(AC 自动机). *AC* 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机的状态可以看作是一个 Trie 树。
如何理解？

AC 自动机

定义 2.5

(AC 自动机). *AC* 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机的状态可以看作是一个 Trie 树。
如何理解?
$n=1$ 时就是 KMP，状态是模式串的每个前缀，表示匹配到了这个前缀，这确实是 $n=1$ 时的 Trie 树。

AC 自动机

定义 2.5

(AC 自动机). AC 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机的状态可以看作是一个 Trie 树。
如何理解?
$n = 1$ 时就是 KMP，状态是模式串的每个前缀，表示匹配到了这个前缀，这确实是 $n = 1$ 时的 Trie 树。
要维护出每个串匹配到的位置，只需要知道最长的匹配即可推出其它所有串的匹配，因此，状态是 Trie 树节点。

AC 自动机

定义 2.5

(AC 自动机). *AC* 自动机是一个多模式串匹配算法，每个状态蕴含了 n 个模式串在当前串上的的匹配长度。

AC 自动机的状态可以看作是一个 Trie 树。
如何理解？
$n = 1$ 时就是 KMP，状态是模式串的每个前缀，表示匹配到了这个前缀，这确实是 $n = 1$ 时的 Trie 树。
要维护出每个串匹配到的位置，只需要知道最长的匹配即可推出其它所有串的匹配，因此，状态是 Trie 树节点。

定义 2.6

($fail$ 指针). 定义 $fail[u]$ 表示状态 u 失配后的状态（*Trie* 树上一个点表示一个串，会跳到这个串的最长在 *Trie* 树上存在的后缀）。

```cpp
void build() {
    queue<int> q;
    for (int i = 0; i < 26; ++i) {
        if (ch[0][i]) q.push(ch[0][i]);
    }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = 0; i < 26; ++i) {
            if (ch[u][i]) {
                fail[ch[u][i]] = ch[fail[u]][i];
                q.push(ch[u][i]);
            } else {
                ch[u][i] = ch[fail[u]][i];
            }
        }
    }
}
```

这段代码的复杂度是和字符集相关的，如果字符集很大应该怎么做呢?

```cpp
void build() {
    queue<int> q;
    for (int i = 0; i < 26; ++i) {
        if (ch[0][i]) q.push(ch[0][i]);
    }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = 0; i < 26; ++i) {
            if (ch[u][i]) {
                fail[ch[u][i]] = ch[fail[u]][i];
                q.push(ch[u][i]);
            } else {
                ch[u][i] = ch[fail[u]][i];
            }
        }
    }
}
```

这段代码的复杂度是和字符集相关的，如果字符集很大应该怎么做呢？
其实这段代码将空孩子指向了一直跳 $fail$ 指针，直到跳到的点有字符 c 的出边的出边对应点。因此，可以不维护空孩子，代价是每次走一步需要跳若干次 $fail$，均摊复杂度是 $O(1)$ 的，如果需要严格复杂度的场景会出问题。

```
void build() {
    queue<int> q;
    for (int i = 0; i < 26; ++i) {
        if (ch[0][i]) q.push(ch[0][i]);
    }
    while (!q.empty()) {
        int u = q.front(); q.pop();
        for (int i = 0; i < 26; ++i) {
            if (ch[u][i]) {
                fail[ch[u][i]] = ch[fail[u]][i];
                q.push(ch[u][i]);
            } else {
                ch[u][i] = ch[fail[u]][i];
            }
        }
    }
}
```

这段代码的复杂度是和字符集相关的，如果字符集很大应该怎么做呢?
其实这段代码将空孩子指向了一直跳 $fail$ 指针，直到跳到的点有字符 c 的出边的出边对应点。因此，可以不维护空孩子，代价是每次走一步需要跳若干次 $fail$，均摊复杂度是 $O(1)$ 的，如果需要严格复杂度的场景会出问题。
观察代码，发现 ch_u 和 ch_{fail_u} 只有在 Trie 树上 u 的出边这些位置不同，因此可用可持久化线段树维护 ch_u。

CF 590E

例题 2.3

给定 n 个仅包含 a,b 的字符串，保证它们两两不同。
你需要去掉尽可能少的字符串，使得剩下的字符串中不存在某一个串是另一个串的子串。
$n \le 750, \sum_{i=1}^{n} |s_i| \le 10^7$。

CF 590E

题解

包含关系形成一个偏序关系，如果能快速求出字符串之间的偏序关系，那么就是要求最长反链，而最长反链等于最小链覆盖，传递闭包后等于最小不交链覆盖，等于 n 减去拆点后二分图最大匹配。

CF 590E

题解

包含关系形成一个偏序关系，如果能快速求出字符串之间的偏序关系，那么就是要求最长反链，而最长反链等于最小链覆盖，传递闭包后等于最小不交链覆盖，等于 n 减去拆点后二分图最大匹配。
因此我们只需要维护出包含关系，S 是 T 的子串说明 S 是 T 某个前缀的后缀，那么 S 表示的串在 $fail$ 指针形成的树（简称 $fail$ 树）上是 T 某个前缀的祖先。

CF 590E

题解

包含关系形成一个偏序关系，如果能快速求出字符串之间的偏序关系，那么就是要求最长反链，而最长反链等于最小链覆盖，传递闭包后等于最小不交链覆盖，等于 n 减去拆点后二分图最大匹配。

因此我们只需要维护出包含关系，S 是 T 的子串说明 S 是 T 某个前缀的后缀，那么 S 表示的串在 $fail$ 指针形成的树（简称 $fail$ 树）上是 T 某个前缀的祖先。

我们枚举 T 的一个前缀，找到一个最长的 S' 满足 S' 是这个前缀的后缀，将 S' 和 T 连边。易证，S 肯定能直接或间接与 T 连边（因为 S' 是最长的，如果不等于 S，那么 S' 肯定是 S 的子串，而我们就是要连子串关系形成的边）。

border 理论（问题引入）

定义 2.7

(周期). 对于字符串 S，若整数 $0 < p < |S|$ 满足 $S_i = S_{i+p}(i \le |S| - p)$，则称 p 是 S 的一个周期。周期和循环节的区别是周期的最后一段不需要是完整的。

border 理论（问题引入）

定义 2.7

(周期). 对于字符串 S，若整数 $0 < p < |S|$ 满足 $S_i = S_{i+p}(i \le |S| - p)$，则称 p 是 S 的一个周期。周期和循环节的区别是周期的最后一段不需要是完整的。

定义 2.8

(最小周期). 用 $\mathrm{minper}(S)$ 表示 S 最小的周期。

border 理论（问题引入）

定义 2.7

(周期). 对于字符串 S，若整数 $0 < p < |S|$ 满足 $S_i = S_{i+p}(i \le |S| - p)$，则称 p 是 S 的一个周期。周期和循环节的区别是周期的最后一段不需要是完整的。

定义 2.8

(最小周期). 用 $\operatorname{minper}(S)$ 表示 S 最小的周期。

例题 2.4

给定一个字符串 $S(|S| \le 10^5)$，$Q(Q \le 10^5)$ 次询问给出 l, r，求出 $S[l; r]$ 的所有周期。（周期可能有 $r - l + 1$ 个，我们希望用一个更简洁的形式表示）

border 理论

定理 2.2

p 是 S 的周期当且仅当 $|S| - p$ 是 S 的 *border*。

因此区间周期询问等价于区间 border 询问。

border 理论

定理 2.2

p 是 S 的周期当且仅当 $|S| - p$ 是 S 的 *border*。

因此区间周期询问等价于区间 border 询问。

引理 2.6

(弱周期引理). 若 p, q 均为 S 的周期，且 $p + q \leq |S|$，则 $\gcd(p, q)$ 也是 S 的周期。

border 理论

定理 2.2

p 是 S 的周期当且仅当 $|S| - p$ 是 S 的 *border*。

因此区间周期询问等价于区间 border 询问。

引理 2.6

(弱周期引理). 若 p, q 均为 S 的周期，且 $p + q \leq |S|$，则 $\gcd(p, q)$ 也是 S 的周期。

引理 2.7

设 $p = \mathrm{minper}(S)$，则 S 的所有不超过 $n - p$ 的周期一定形如 $p, 2p, \ldots, kp$。

border 理论

定理 2.2

p 是 S 的周期当且仅当 $|S| - p$ 是 S 的 *border*。

因此区间周期询问等价于区间 border 询问。

引理 2.6

(弱周期引理). 若 p, q 均为 S 的周期，且 $p + q \le |S|$，则 $\gcd(p, q)$ 也是 S 的周期。

引理 2.7

设 $p = \text{minper}(S)$，则 S 的所有不超过 $n - p$ 的周期一定形如 $p, 2p, \ldots, kp$。

证明

假设有周期 $q \le n - p$ 不是 p 的倍数，则 $\gcd(p, q) < p$ 也是周期，矛盾。

border 理论

推论 2.1

S 的所有 $\geq \frac{|S|}{2}$ 的 *border* 构成等差数列。

border 理论

推论 2.1

S 的所有 $\geq \frac{|S|}{2}$ 的 *border* 构成等差数列。

证明

设 p 是最小周期，如果超过一半，则没有 $\geq \frac{|S|}{2}$ 的 *border*；如果没超过一半，则 $\leq \frac{|S|}{2}$ 的周期都是 p 的倍数，意味着长度 $\geq \frac{|S|}{2}$ 的 *border* 形成等差数列。

border 理论

推论 2.1

S 的所有 $\geq \frac{|S|}{2}$ 的 *border* 构成等差数列。

证明

设 p 是最小周期，如果超过一半，则没有 $\geq \frac{|S|}{2}$ 的 *border*；如果没超过一半，则 $\leq \frac{|S|}{2}$ 的周期都是 p 的倍数，意味着长度 $\geq \frac{|S|}{2}$ 的 *border* 形成等差数列。

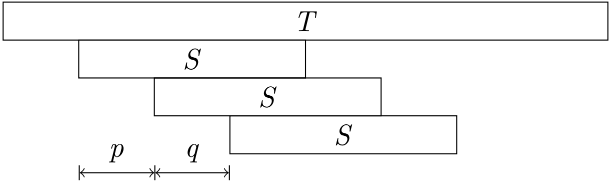
推论 2.2

若 S 是 T 的 *border* 且满足 $|T| \leq 2|S|$，则 S 在 T 中出现的位置构成等差数列。并且如果出现至少 *3* 次，则公差恰好就是 $\operatorname{minper}(S)$。

border 理论

证明

如图，任取相邻 3 次出现，p, q 均为 S 的周期，那么 $p, q \le |S| - \mathrm{minper}(S)$，因此 p, q 都是 $\mathrm{minper}(S)$ 的倍数。唯一的可能就是 $p = q = \mathrm{minper}(S)$，否则这不可能是三次相邻的出现。



border 理论

定义 2.9

(pre, suf). 对于一个字符串 S，用 $pre(S,k)$ 表示 S 长度为 k 的前缀，用 $suf(S,k)$ 表示 S 长度为 k 的后缀。

border 理论

定义 2.9

(pre, suf). 对于一个字符串 S，用 $pre(S, k)$ 表示 S 长度为 k 的前缀，用 $suf(S, k)$ 表示 S 长度为 k 的后缀。

引理 2.8

对于两个串 S, T，$|S| = |T| = n$，$PS(S, T) = \{k : pre(S, k) = suf(T, k)\}$ 中 $\geq \frac{n}{2}$ 的元素构成等差数列。

border 理论

定义 2.9

(pre, suf). 对于一个字符串 S，用 $pre(S,k)$ 表示 S 长度为 k 的前缀，用 $suf(S,k)$ 表示 S 长度为 k 的后缀。

引理 2.8

对于两个串 S, T，$|S| = |T| = n$，$PS(S,T) = \{k : pre(S,k) = suf(T,k)\}$ 中 $\geq \frac{n}{2}$ 的元素构成等差数列。

证明

取出 $PS(S,T)$ 中的最大值 p，那么 $PS(S,T)$ 恰好就是 $pre(S,p)$ 的所有 *border*。而 $pre(S,p)$ 所有 $\geq \frac{p}{2}$ 的 *border* 构成等差数列。

推论 2.3

对于一个字符串 S 和 $k \le \frac{|S|}{2}$，S 所有长度在 $[k, 2k)$ 中的 *border* 构成等差数列。

border 理论

推论 2.3

对于一个字符串 S 和 $k \leq \frac{|S|}{2}$，S 所有长度在 $[k, 2k)$ 中的 *border* 构成等差数列。

证明

S 长度在 $[k, 2k)$ 中的 *border* 就是 $PS(pre(S, 2k), suf(S, 2k))$ 中 $\geq k$ 的元素，由引理可得构成等差数列。

我们可以对于每个 i 求出在 $[2^{i-1}, 2^i)$ 中的所有 border，这样只用 $\log|S|$ 个等差数列就能表示 S 所有的 border。

区间 border

引理 2.9

对于 $2^{i-1} \leq k < 2^i$，k 是 border 当且仅当 $suf(pre(S,k),2^{i-1}) = suf(S,2^{i-1})$ 且 $pre(suf(S,k),2^{i-1}) = pre(S,2^{i-1})$。

那么我们只需要找到 $suf(S,2^{i-1})$ 在 $pre(S,2^i)$ 中所有出现位置，以及 $pre(S,2^{i-1})$ 在 $suf(S,2^i)$ 中所有出现位置，用等差数列表示，然后偏移，然后等差数列求交即可得到所有合法的 k。

区间 border

引理 2.9

对于 $2^{i-1} \le k < 2^i$，k 是 *border* 当且仅当 $suf(pre(S,k),2^{i-1}) = suf(S,2^{i-1})$ 且 $pre(suf(S,k),2^{i-1}) = pre(S,2^{i-1})$。

那么我们只需要找到 $suf(S,2^{i-1})$ 在 $pre(S,2^i)$ 中所有出现位置，以及 $pre(S,2^{i-1})$ 在 $suf(S,2^i)$ 中所有出现位置，用等差数列表示，然后偏移，然后等差数列求交即可得到所有合法的 k。

定义 2.10

(基本子串字典). $\log|S|$ 个数组，第 t 个数组 N_t 存所有长度为 2^t 的排名，即 $N_t(i)$ 表示 $S[i; i+2^t-1]$ 的排名。这些 N 数组就称为 S 的基本子串字典。

N_t 可以通过之后要讲的后缀数组的倍增算法求出，复杂度 $O(n\log n)$。

区间 border

现在我们想求出 $T = suf(S, 2^{i-1})$ 在 $S[1; 2^i]$ 中出现位置的等差数列，只需要找到第一次、第二次和最后一次出现位置。

区间 border

现在我们想求出 $T = suf(S, 2^{i-1})$ 在 $S[1; 2^i]$ 中出现位置的等差数列，只需要找到第一次、第二次和最后一次出现位置。

对于每个 x 维护 $N_t(i) = x$ 的所有 i 构成的有序表，只需要在 T 所在的有序表中二分查找即可，复杂度 $O(\log n)$。

区间 border

现在我们想求出 $T = suf(S, 2^{i-1})$ 在 $S[1; 2^i]$ 中出现位置的等差数列，只需要找到第一次、第二次和最后一次出现位置。
对于每个 x 维护 $N_t(i) = x$ 的所有 i 构成的有序表，只需要在 T 所在的有序表中二分查找即可，复杂度 $O(\log n)$。
然后我们需要对等差数列求交，用三元组 (s, d, k) 表示起点为 s，公差为 d，长度为 k 的等差数列。对两个等差数列 (s_1, d_1, k_1) 和 (s_2, d_2, k_2) 求交，用一般方法，需要求出 $s_1 + d_1x = s_2 + d_2y$ 的整数解，复杂度 $O(\log \max(d_1, d_2))$。

区间 border

现在我们想求出 $T = suf(S, 2^{i-1})$ 在 $S[1;2^i]$ 中出现位置的等差数列，只需要找到第一次、第二次和最后一次出现位置。
对于每个 x 维护 $N_t(i) = x$ 的所有 i 构成的有序表，只需要在 T 所在的有序表中二分查找即可，复杂度 $O(\log n)$。
然后我们需要对等差数列求交，用三元组 (s, d, k) 表示起点为 s，公差为 d，长度为 k 的等差数列。对两个等差数列 (s_1, d_1, k_1) 和 (s_2, d_2, k_2) 求交，用一般方法，需要求出 $s_1 + d_1x = s_2 + d_2y$ 的整数解，复杂度 $O(\log \max(d_1, d_2))$。
这样我们得到了一个最基本的单次复杂度 $O(\log^2 n)$ 的做法。

区间 border

定理 2.3

如果两个等差数列的长度都 ≥ 3，那么他们的公差相等。

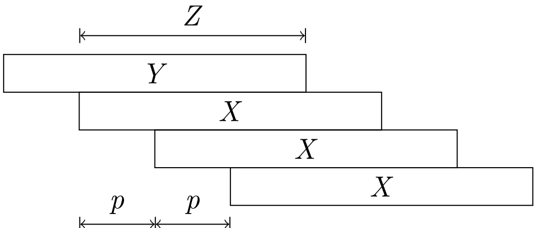
区间 border

定理 2.3

如果两个等差数列的长度都 ≥ 3，那么他们的公差相等。

证明

如图，设 $X = suf(S, 2^{i-1}), Y = pre(S, 2^{i-1})$，取出 X 出现（条件告诉我们 X 在 $S[1; 2^i]$ 出现 ≥ 3 次）的前 3 次，则公差 $p = \text{minper}(X)$，记 Z 为第一个 X 与 Y 的重叠部分（由于长度限制，必与每个 X 都重叠）。那么 $\text{minper}(Z)$ 也是 p，故 $\text{minper}(Y) \geq p$。这说明 $\text{minper}(Y) \geq \text{minper}(X)$，同理，不等号反转也正确，因此 $\text{minper}(X) = \text{minper}(Y)$。



区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。
之前的做法是每个 x 维护 $N_t(i) = x$ 的有序表，在有序表二分查找。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。
之前的做法是每个 x 维护 $N_t(i) = x$ 的有序表，在有序表二分查找。
事实上，我们如果将这个有序表中的元素的值域按照 2^t 大小分块，如果没有就空着，如果有就压缩成等差数列（如果块大小长了，就不一定能表示成等差数列了），复杂度也是正确的（因为对于一个 t，$S[i; i+2^t-1]$ 只会使 $x = N_t(i)$ 对应的有序表大小增加 1，与 i 所在块关系不大）。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。
之前的做法是每个 x 维护 $N_t(i) = x$ 的有序表，在有序表二分查找。
事实上，我们如果将这个有序表中的元素的值域按照 2^t 大小分块，如果没有就空着，如果有就压缩成等差数列（如果块大小长了，就不一定能表示成等差数列了），复杂度也是正确的（因为对于一个 t，$S[i; i + 2^t - 1]$ 只会使 $x = N_t(i)$ 对应的有序表大小增加 1，与 i 所在块关系不大）。
查询我们只需要查两个位置（位置是一些参数的 tuple，需要哈希）的等差数列（因为查询的区间长度是 2^t，最多用两个 2^t 区间就能覆盖）。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。
之前的做法是每个 x 维护 $N_t(i) = x$ 的有序表，在有序表二分查找。
事实上，我们如果将这个有序表中的元素的值域按照 2^t 大小分块，如果没有就空着，如果有就压缩成等差数列（如果块大小长了，就不一定能表示成等差数列了），复杂度也是正确的（因为对于一个 t，$S[i; i + 2^t - 1]$ 只会使 $x = N_t(i)$ 对应的有序表大小增加 1，与 i 所在块关系不大)。
查询我们只需要查两个位置（位置是一些参数的 tuple，需要哈希）的等差数列（因为查询的区间长度是 2^t，最多用两个 2^t 区间就能覆盖)。
为了方便理解，给出一个查询需要的参数：t，一个长度为 2^t 的串的位置 i (可以得到 $x = N_t(i)$)，还有属于第几个块（按照大小 2^t 分)。

区间 border

这样长度大于等于 3 的情况都能 $O(1)$ 合并，否则暴力枚举短的等差数列的元素，判断是否在另一个中，于是等差数列求交就做到 $O(1)$ 了。
复杂度的瓶颈在于求出 $suf(S, 2^{i-1})$ 在 $pre(S, 2^i)$ 中的位置。
之前的做法是每个 x 维护 $N_t(i) = x$ 的有序表，在有序表二分查找。
事实上，我们如果将这个有序表中的元素的值域按照 2^t 大小分块，如果没有就空着，如果有就压缩成等差数列（如果块大小长了，就不一定能表示成等差数列了），复杂度也是正确的（因为对于一个 t，$S[i; i + 2^t - 1]$ 只会使 $x = N_t(i)$ 对应的有序表大小增加 1，与 i 所在块关系不大）。
查询我们只需要查两个位置（位置是一些参数的 tuple，需要哈希）的等差数列（因为查询的区间长度是 2^t，最多用两个 2^t 区间就能覆盖）。
为了方便理解，给出一个查询需要的参数：t，一个长度为 2^t 的串的位置 i（可以得到 $x = N_t(i)$），还有属于第几个块（按照大小 2^t 分）。
综上，对于一个 t，就能做到 $O(1)$ 求出 border 的等差数列，故区间 border 询问可以做到单次 $O(\log n)$ 的复杂度。

目录

后缀数组（引入）

后缀数组的经典应用就是后缀排序，复杂度 $O(n\log n)$（倍增后缀数组）或者 $O(n)$（DC3，SA-IS）。
还记得后缀排序吗？（之前给了一个哈希做法，复杂度是 $O(n\log^2 n)$）。

例题 3.1

给定一个字符串 S，求出所有后缀的字典序排序。
如 $S=$ “*abaa*”，则后缀排序为：

$$\text{“}a\text{”} < \text{“}aa\text{”} < \text{“}abaa\text{”} < \text{“}baa\text{”}$$

因此输出为 $4,3,1,2$。
$|S|\le 10^5$。

后缀数组（引入）

例题 3.2

有字符串 A, B, C, D，$|A| = |C|$，如何比较 AB 和 CD 的大小关系?

后缀数组（引入）

例题 3.2

有字符串 A, B, C, D，$|A| = |C|$，如何比较 AB 和 CD 的大小关系?

题解

先比较 A 和 C。

- 相等：比较 B 和 D，返回 B 和 D 的大小关系。
- 不相等：返回大小关系。

(A 和 C 的长度需要相等，B 和 D 长度可以看成相等的，不等时往末尾补 $'\backslash 0'$)
如果知道 A, B, C, D 的排名，其实就是 (A, B) 和 (C, D) 的比较。

倍增后缀数组

回顾一下基本子串字典的定义。

定义 3.1

(基本子串字典). $\log|S|$ 个数组，第 t 个数组 N_t 存所有长度为 2^t 的排名，即 $N_t(i)$ 表示 $S[i;i+2^t-1]$ 的排名。这些 N 数组就称为 S 的基本子串字典。

倍增后缀数组

回顾一下基本子串字典的定义。

定义 3.1

(基本子串字典). $\log|S|$ 个数组，第 t 个数组 N_t 存所有长度为 2^t 的排名，即 $N_t(i)$ 表示 $S[i; i+2^t-1]$ 的排名。这些 N 数组就称为 S 的基本子串字典。

我们想象 S 末尾有足够多个 $'\backslash 0'$，如果我们知道 $N_{\lceil\log|S|\rceil}(1\ldots|S|)$ 就能知道每个后缀的排名，这正是我们需要的。
因此问题转化为：

例题 3.3

求出 S 的基本子串字典。
$O(n\log^2 n)$?
$O(n\log n)$?

倍增后缀数组

题解

N_0 平凡，是单个字符的排名，事实上，N_t 只要大小关系对即可，具体的值不重要，因此 N_0 可以直接设为 S 的字符的 *ASCII* 码。
从 $t=1$ 开始，假设我们已经求出了 N_{t-1}，我们要求出 N_t。
比较 $N_t(i)$ 和 $N_t(j)$ 等价于
比较 $(N_{t-1}(i), N_{t-1}(i+2^{t-1}))$ 和 $(N_{t-1}(j), N_{t-1}(j+2^{t-1}))$ 的大小关系。
用基于比较的排序复杂度 $O(n\log n)$，而 N_{t-1} 的值域是 $O(n)$，我们可以用基数排序，复杂度 $O(n)$。
因此后缀排序总复杂度 $O(n\log n)$。
将边界条件处理一下，就是倍增后缀数组了。

倍增后缀数组

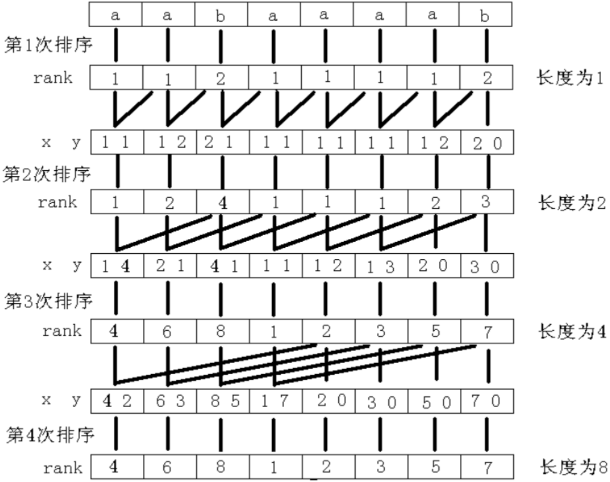


图 1: 倍增后缀排序示意图（来源：oi-wiki）

倍增后缀数组

定义 3.2

(rk_i). 定义 rk_i 表示后缀 i 的排名。*(*其实就是在过程中以滚动数组的形式维护 $N_t(i)$*)*

定义 3.3

(sa_i). 排名为 i 的后缀。（在第 t 轮时，只考虑后缀的前 2^t 个字符）

两个相同的串的 rk 相同，但是 sa 不同（顺序任意）。
经过 $\log|S|$ 轮后，rk 和 sa 都会是一个排列，并且满足 $sa_{rk_i} = rk_{sa_i} = i$。

```cpp
char s[N];
int n, m, p, rk[N * 2], oldrk[N], sa[N * 2], id[N], cnt[N];

n = strlen(s + 1);
m = 128;

for (int i = 1; i <= n; i++) cnt[rk[i] = s[i]]++;
for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
for (int i = n; i >= 1; i--) sa[cnt[rk[i]]--] = i;

for (int w = 1;; w <<= 1, m = p) {  // m = p 即为值域优化
int cur = 0;
for (int i = n - w + 1; i <= n; i++) id[++cur] = i;
for (int i = 1; i <= n; i++)
    if (sa[i] > w) id[++cur] = sa[i] - w;

memset(cnt, 0, sizeof(cnt));
for (int i = 1; i <= n; i++) cnt[rk[i]]++;
for (int i = 1; i <= m; i++) cnt[i] += cnt[i - 1];
for (int i = n; i >= 1; i--) sa[cnt[rk[id[i]]]--] = id[i];

p = 0;
memcpy(oldrk, rk, sizeof(oldrk));
for (int i = 1; i <= n; i++) {
    if (oldrk[sa[i]] == oldrk[sa[i - 1]] && oldrk[sa[i] + w] == oldrk[sa[i - 1] + w])
        rk[sa[i]] = p;
    else
        rk[sa[i]] = ++p;
}

if (p == n) break;  // p = n 时无需再排序
}
```

Height 数组

后缀数组还有一个重要数组。

定义 3.4

(height 数组). $height_i = \mathrm{LCP}(sa_i, sa_{i-1})$，即第 i 名的后缀和它前一名的后缀的最长公共前缀。$height[1] = 0$。

定理 3.1

$height$ 数组满足 $height_{rk_i} \geq height_{rk_{i-1}} - 1$。

证明

根据定义，后缀 $i-1$ 和它排名前一名的后缀的最长公共前缀是 $height_{rk_{i-1}}$。
将这个公共前缀开头删去一个字符得到的串是后缀 i 和某个串的最长公共前缀，因此 $height_{rk_i}$ 至少是 $height_{rk_{i-1}} - 1$。

利用上述定理，可以得到一个 $O(n)$ 求 $height$ 数组的算法。

```
void get_height() {
    for (i = 1, k = 0; i <= n; ++i) {
        if (rk[i] == 0) continue;
        if (k) --k;
        while (s[i + k] == s[sa[rk[i] - 1] + k]) ++k;
        height[rk[i]] = k;
    }
}
```

复杂度比较显然，因为每次 k 至多减少 1，最终 k 不超 n，这意味着 k 增加的次数最多 $2n$ 次。

Height 数组的应用

例题 3.4

求两个子串的最长公共前缀。

Height 数组的应用

例题 3.4

求两个子串的最长公共前缀。

题解

$l := \mathrm{LCP}(sa_i, sa_j) = \min_{k \in (i,j]} height_k$。
而区间最小值问题可以用 *ST* 表做到 $O(n \log n)$ 预处理之后 $O(1)$ 查询。
证明：首先 l 必然是公共前缀，并且如果不是最长公共前缀，说明 $sa_{i \dots j}$ 的前 $l+1$ 个字符都相等，因此 $\forall k \in (i,j], height_k \geq l+1$，与 l 的定义矛盾。

Height 数组的应用

例题 3.4

求两个子串的最长公共前缀。

题解

$l := \mathrm{LCP}(sa_i, sa_j) = \min_{k\in(i,j]} height_k$。
而区间最小值问题可以用 *ST* 表做到 $O(n\log n)$ 预处理之后 $O(1)$ 查询。
证明：首先 l 必然是公共前缀，并且如果不是最长公共前缀，说明 $sa_{i...j}$ 的前 $l+1$ 个字符都相等，因此 $\forall k\in(i,j], height_k \geq l+1$，与 l 的定义矛盾。

例题 3.5

比较两个子串的大小关系。

Height 数组的应用

例题 3.4

求两个子串的最长公共前缀。

题解

$l := \mathrm{LCP}(sa_i, sa_j) = \min_{k\in(i,j]} height_k$。
而区间最小值问题可以用 *ST* 表做到 $O(n\log n)$ 预处理之后 $O(1)$ 查询。
证明：首先 l 必然是公共前缀，并且如果不是最长公共前缀，说明 $sa_{i\dots j}$ 的前 $l+1$ 个字符都相等，因此 $\forall k\in(i,j], height_k \geq l+1$，与 l 的定义矛盾。

例题 3.5

比较两个子串的大小关系。

题解

知道了 LCP 可以直接比较，时间复杂度 $O(1)$。
(也有哈希做法 $O(\log n)$)

Height 数组的应用

例题 3.6

求出本质不同子串数目。

Height 数组的应用

例题 3.6

求出本质不同子串数目。

题解

子串就是后缀的前缀，所以可以枚举每个后缀，计算前缀总数，再减掉重复。
后缀的前缀总数就是子串个数，为 $\frac{n(n+1)}{2}$。
重复的子串就是 $height$ 数组的和。(考虑 sa_i 的前缀在 $sa_{1\ldots i-1}$ 的前缀中出现过的其实就是 sa_i 与 sa_{i-1} 的 LCP 的前缀，那就是 $height_i$ 个)
因此答案为：

$$\frac{n(n+1)}{2} - \sum_{i=2}^{n} height_i$$

最小表示法

例题 3.7

给定字符串 S，寻找最小的循环移位。(即每次将 S 的末尾放到开头，重复 n 次，求过程中得到的字典序最小的 S)
$|S| \le 10^5$。

最小表示法

例题 3.7

给定字符串 S，寻找最小的循环移位。(即每次将 S 的末尾放到开头，重复 n 次，求过程中得到的字典序最小的 S)
$|S| \leq 10^5$。

题解

将字符串 S 复制一份变为 SS 就转化为后缀排序问题。

多次在字符串中查找子串

例题 3.8

多组询问，每次在线的在主串 T 中寻找模式串 S。
保证 $T, \sum |S| \leq 10^5$

多次在字符串中查找子串

例题 3.8

多组询问，每次在线的在主串 T 中寻找模式串 S。
保证 $T, \sum |S| \le 10^5$

题解

先将 T 后缀排序，在后缀数组上二分。比较 S 和当前后缀的时间复杂度为 $O(|S|)$，因此找子串的时间复杂度为 $O(|S| \log |T|)$。

多次在字符串中查找子串

例题 3.8

多组询问，每次在线的在主串 T 中寻找模式串 S。
保证 $T, \sum |S| \leq 10^5$

题解

先将 T 后缀排序，在后缀数组上二分。比较 S 和当前后缀的时间复杂度为 $O(|S|)$，因此找子串的时间复杂度为 $O(|S| \log |T|)$。
如果需要出现的次数，由于每次出现在后缀数组上都是相邻的，因此可以通过再次二分得到另一个边界。这个区间代表的后缀都有 S 作为前缀。

最长公共子串

例题 3.9

给定 k 个字符串 S_i，求出它们的最长公共子串。
$n, k, \sum |S_i| \le 10^5$。

最长公共子串

例题 3.9

给定 k 个字符串 S_i，求出它们的最长公共子串。
$n, k, \sum |S_i| \le 10^5$。

题解

将所有字符串拼接起来，字符串之间用不同的特殊字符隔开，得到新串 S。将 S 后缀排序，求出 $height$ 数组。

最长公共子串

例题 3.9

给定 k 个字符串 S_i，求出它们的最长公共子串。
$n, k, \sum |S_i| \le 10^5$。

题解

将所有字符串拼接起来，字符串之间用不同的特殊字符隔开，得到新串 S。将 S 后缀排序，求出 $height$ 数组。
问题转化为：在 $height$ 数组上找出一段，使得这一段包含来自给定的每个字符串的至少一个后缀，并且要最大化 $height$ 数组的最小值。

最长公共子串

例题 3.9

给定 k 个字符串 S_i，求出它们的最长公共子串。
$n, k, \sum |S_i| \leq 10^5$。

题解

将所有字符串拼接起来，字符串之间用不同的特殊字符隔开，得到新串 S。将 S 后缀排序，求出 $height$ 数组。
问题转化为：在 $height$ 数组上找出一段，使得这一段包含来自给定的每个字符串的至少一个后缀，并且要最大化 $height$ 数组的最小值。
这个问题使用双指针和单调队列维护可做到 $O(n)$。

LG P2852 [USACO06DEC] Milk Patterns G

例题 3.10

给定一个串 S，求至少出现 k 次的最长子串。
$|S|, k \le 2 \times 10^4$。

LG P2852 [USACO06DEC] Milk Patterns G

例题 3.10

给定一个串 S，求至少出现 k 次的最长子串。
$|S|, k \leq 2 \times 10^4$。

题解

出现至少 k 次意味着后缀排序后有至少连续 k 个后缀以这个子串作为公共前缀。
所以求出每相邻 $k-1$ 个 $height$ 的最小值，这些最小值的最大值就是答案。
可以用单调队列 $O(n)$ 解决，用其它方式也足以通过。

不重叠的重复子串

例题 3.11

给定字符串 S，求一个最长的字符串 T 使得 T 在 S 中不重叠的出现了至少两次。$|S| \leq 10^5$。

不重叠的重复子串

例题 3.11

给定字符串 S，求一个最长的字符串 T 使得 T 在 S 中不重叠的出现了至少两次。$|S| \leq 10^5$。

题解

二分答案 k，根据 $height$ 数组将 S 的所有后缀分组，同一组中的 LCP 至少为 k。

不重叠的重复子串

例题 3.11

给定字符串 S，求一个最长的字符串 T 使得 T 在 S 中不重叠的出现了至少两次。$|S| \leq 10^5$。

题解

二分答案 k，根据 $height$ 数组将 S 的所有后缀分组，同一组中的 LCP 至少为 k。只需检查是否存在一组中位置相差至少为 k 即可，复杂度 $O(n \log n)$。

不重叠的重复子串

例题 3.11

给定字符串 S，求一个最长的字符串 T 使得 T 在 S 中不重叠的出现了至少两次。
$|S| \le 10^5$。

题解

二分答案 k，根据 $height$ 数组将 S 的所有后缀分组，同一组中的 LCP 至少为 k。
只需检查是否存在一组中位置相差至少为 k 即可，复杂度 $O(n\log n)$。
还能更快，发现 k 从 n 减小到 1 的过程中后缀的分组是逐渐合并的，可用并查集模拟这个过程做到 $O(n\alpha)$，或者把合并两个分组改成新建一个节点，左右孩子是对应的两个点，最后遍历一下树，做到 $O(n)$。

不重叠的重复子串

例题 3.11

给定字符串 S，求一个最长的字符串 T 使得 T 在 S 中不重叠的出现了至少两次。$|S| \le 10^5$。

题解

二分答案 k，根据 $height$ 数组将 S 的所有后缀分组，同一组中的 LCP 至少为 k。只需检查是否存在一组中位置相差至少为 k 即可，复杂度 $O(n \log n)$。
还能更快，发现 k 从 n 减小到 1 的过程中后缀的分组是逐渐合并的，可用并查集模拟这个过程做到 $O(n\alpha)$，或者把合并两个分组改成新建一个节点，左右孩子是对应的两个点，最后遍历一下树，做到 $O(n)$。
后者几乎就是从后缀数组建立后缀树的过程。(后缀树之后会讲)

后缀自动机

定义 3.5

(后缀自动机). 后缀自动机是一个能识别一个字符串的所有后缀的自动机。

后缀自动机

定义 3.5

(后缀自动机). 后缀自动机是一个能识别一个字符串的所有后缀的自动机。

后缀自动机可以做到复杂度 $O(n)$ 增量构造。

后缀自动机

定义 3.5

(后缀自动机). 后缀自动机是一个能识别一个字符串的所有后缀的自动机。

后缀自动机可以做到复杂度 $O(n)$ 增量构造。

定义 3.6

(endpos 集合/ right 集合). 对于一个子串 t，它在原串中的结束位置集合记为 $\mathrm{endpos}(t)$。

后缀自动机

定义 3.5

(后缀自动机). 后缀自动机是一个能识别一个字符串的所有后缀的自动机。

后缀自动机可以做到复杂度 $O(n)$ 增量构造。

定义 3.6

(endpos 集合/ right 集合). 对于一个子串 t，它在原串中的结束位置集合记为 $\text{endpos}(t)$。
例如：$s = abcbc, t = bc$，那么 $\text{endpos}(t) = \{3, 5\}$。(字符串下标从 1 开始)

后缀自动机

定义 3.5

(后缀自动机). 后缀自动机是一个能识别一个字符串的所有后缀的自动机。

后缀自动机可以做到复杂度 $O(n)$ 增量构造。

定义 3.6

(endpos 集合/ right 集合). 对于一个子串 t，它在原串中的结束位置集合记为 $\mathrm{endpos}(t)$。
例如：$s = abcbc, t = bc$，那么 $\mathrm{endpos}(t) = \{3, 5\}$。(字符串下标从 1 开始)

定义 3.7

(等价类). 所有 endpos 相同的子串是一个等价类，在后缀自动机上表现为一个点 u。

后缀自动机

引理 3.1

非空子串 u, w（假设 $|u| \leq |w|$）的 endpos 相同，当且仅当 u 在 s 中的每次出现，都是以 w 的后缀的形式存在的。

后缀自动机

引理 3.1

非空子串 u, w（假设 $|u| \le |w|$）的 endpos 相同，当且仅当 u 在 s 中的每次出现，都是以 w 的后缀的形式存在的。

引理 3.2

两个非空子串 u, w（假设 $|u| \le |w|$）。要么 $\operatorname{endpos}(u) \cap \operatorname{endpos}(w) = \varnothing$，要么 $\operatorname{endpos}(u) \subseteq \operatorname{endpos}(v)$，取决于 u 是否为 w 的一个后缀。

后缀自动机

引理 3.1

非空子串 u, w（假设 $|u| \le |w|$）的 endpos 相同，当且仅当 u 在 s 中的每次出现，都是以 w 的后缀的形式存在的。

引理 3.2

两个非空子串 u, w（假设 $|u| \le |w|$）。要么 $\mathrm{endpos}(u) \cap \mathrm{endpos}(w) = \varnothing$，要么 $\mathrm{endpos}(u) \subseteq \mathrm{endpos}(v)$，取决于 u 是否为 w 的一个后缀。

引理 3.3

一个 endpos 集合的等价类，将类中的所有子串按照长度排序，较短的是较长的串的后缀，并且等价类中的子串长度恰好覆盖一个区间。

后缀自动机（parent 树）

定义 3.8

(parent 树). 将 endpos 按照子集关系可建立的一棵树，称为 parent 树。树的一个节点对应一个等价类。用 fa_u 表示 u 的父亲。

后缀自动机（parent 树）

定义 3.8

(parent 树). 将 endpos 按照子集关系可建立的一棵树，称为 parent 树。树的一个节点对应一个等价类。用 fa_u 表示 u 的父亲。

定义 3.9

(len 数组). len_u 表示 u 节点等价类最长的串的长度。

后缀自动机（parent 树）

定义 3.8

(parent 树). 将 endpos 按照子集关系可建立的一棵树，称为 parent 树。树的一个节点对应一个等价类。用 fa_u 表示 u 的父亲。

定义 3.9

(len 数组). len_u 表示 u 节点等价类最长的串的长度。

引理 3.4

点 u 表示的串的长度区间是 $(len_{fa_u}, len_u]$。

后缀自动机（parent 树）

定理 3.2

等价类个数（parent 树节点数）是 $O(n)$。

后缀自动机（parent 树）

定理 3.2

等价类个数（parent 树节点数）是 $O(n)$。

证明

parent 树的叶节点不交，至多 n 个叶节点，而非叶节点儿子数至少为 2，故节点数不超过 $2n-1$。

后缀自动机（parent 树）

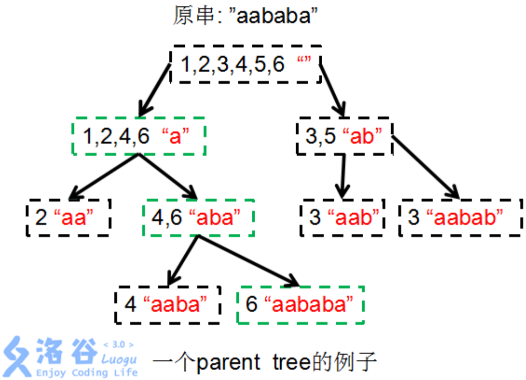


图 2: parent 示意图

parent 树是前缀树，也是反串的后缀树。(这间接给出了后缀树的定义)

后缀自动机（parent 树）

parent 树性质：

- 从下往上是 endpos 合并的过程，从上往下是 endpos 分裂的过程。

后缀自动机（parent 树）

parent 树性质：

- 从下往上是 endpos 合并的过程，从上往下是 endpos 分裂的过程。
- 从 parent 树上从上往下取出一条链，是不断在串的开头加入字符的过程。

后缀自动机（parent 树）

parent 树性质：

- 从下往上是 endpos 合并的过程，从上往下是 endpos 分裂的过程。
- 从 parent 树上从上往下取出一条链，是不断在串的开头加入字符的过程。
- 儿子的长度大于父亲的长度，父亲是儿子的后缀。

后缀自动机（parent 树）

parent 树性质：

- 从下往上是 endpos 合并的过程，从上往下是 endpos 分裂的过程。
- 从 parent 树上从上往下取出一条链，是不断在串的开头加入字符的过程。
- 儿子的长度大于父亲的长度，父亲是儿子的后缀。
- 两个串的 LCS 是对应等价类 LCA 的 len（如果是祖先-后代关系需要特判）

后缀自动机（自动机）

等价类作为点，$ch_{u,c}$ 表示点 u 末尾加入 c 是哪个点。显然一个等价类末尾同时加上一个字符还是等价类。

后缀自动机（自动机）

自动机的性质：

- ch 的结构构成了一个 DAG。

后缀自动机（自动机）

自动机的性质：

- ch 的结构构成了一个 DAG。
- 任意 S 的子串都能被识别，非 S 的子串都不能识别。(如果没有后面的条件，是后缀预言机（Factor Oracle)，可以做到节点数为 n)

后缀自动机（自动机）

自动机的性质：

- ch 的结构构成了一个 DAG。
- 任意 S 的子串都能被识别，非 S 的子串都不能识别。(如果没有后面的条件，是后缀预言机（Factor Oracle)，可以做到节点数为 n)
- DAG 上的一条路径对应一个本质不同子串。

parent 树和自动机的关系

- parent 树的节点和后缀自动机的节点都是等价类（通过 endpos 得到）
- parent 树往孩子走是向前加字符，走自动机是向后加字符。

后缀自动机的构造

背板子，注意数组大小开两倍。
这个写法时间、空间复杂度都是 $O(n|\Sigma|)$。
得到的 fa_u 就是 parent 树上 u 节点的父亲，$ch_{u,c}$ 就是 u 在自动机上字符为 c 的出边。

```
struct SAM{
    int fa[N*2],len[N*2],tot,lst,ch[N*2][26];
    SAM(){tot=lst=1;}
    void extend(int c){
        int p=lst,np=lst=++tot; len[np]=len[p]+1;
        for(;p&&!ch[p][c];p=fa[p])ch[p][c]=np;
        if(!p)fa[np]=1;
        else{
            int q=ch[p][c];
            if(len[p]+1==len[q])fa[np]=q;
            else{
                int nq=++tot; len[nq]=len[p]+1;
                memcpy(ch[nq],ch[q],sizeof(ch[nq]));
                fa[nq]=fa[q],fa[q]=fa[np]=nq;
                for(;p&&ch[p][c]==q;p=fa[p])ch[p][c]=nq;
            }
        }
    }
}sam;
```

子串定位

例题 3.12

给定 S，Q 次询问给定 l, r，询问 $S[l;r]$ 在后缀自动机上节点。
$|S|, Q \le 10^5$。

子串定位

例题 3.12

给定 S，Q 次询问给定 l, r，询问 $S[l; r]$ 在后缀自动机上节点。
$|S|, Q \leq 10^5$。

题解

在构造后缀自动机的过程中，可以得到 $S[1; i]$ 在后缀自动机上的节点（每次加入之后的 lst 变量）。

子串定位

例题 3.12

给定 S，Q 次询问给定 l, r，询问 $S[l;r]$ 在后缀自动机上节点。
$|S|, Q \leq 10^5$。

题解

在构造后缀自动机的过程中，可以得到 $S[1;i]$ 在后缀自动机上的节点（每次加入之后的 lst 变量）。
在 parent 树上往父亲跳等价于删除开头字符，因此跳到第一个 $len_u \leq r - l + 1$ 的点 u，就是 $S[l;r]$ 在后缀自动机上对应的点。

子串定位

例题 3.12

给定 S，Q 次询问给定 l, r，询问 $S[l; r]$ 在后缀自动机上节点。
$|S|, Q \leq 10^5$。

题解

在构造后缀自动机的过程中，可以得到 $S[1; i]$ 在后缀自动机上的节点（每次加入之后的 lst 变量)。
在 parent 树上往父亲跳等价于删除开头字符，因此跳到第一个 $len_u \leq r - l + 1$ 的点 u，就是 $S[l; r]$ 在后缀自动机上对应的点。
这个过程可以使用倍增优化做到 $O(\log n)$。

子串定位

例题 3.13

给定 S，Q 次询问给定 T，询问 T 是否是 S 的子串，如果是，还要找到对应节点。$|S|, Q, \sum |T| \leq 10^5$。

子串定位

例题 3.13

给定 S，Q 次询问给定 T，询问 T 是否是 S 的子串，如果是，还要找到对应节点。$|S|, Q, \sum |T| \leq 10^5$。

题解

自动机做法：枚举 T 的字符，不停走自动机上字符，走到的点就是最终节点。
parent 树做法：倒着枚举 T 的字符，并维护当前的长度，如果长度不是最大长度，那么只有一个字符是能加在开头的，如果是最大长度，看有没有孩子是往开头加这个字符得到的。

本质不同子串

例题 3.14

求 S 的本质不同子串数目。

本质不同子串

例题 3.14

求 S 的本质不同子串数目。

题解

后缀数组。

本质不同子串

例题 3.14

求 S 的本质不同子串数目。

题解

后缀数组。

题解

用 parent 树。不同等价类表示的串两两不同（因为 endpos 不同），而一个等价类内的串个数是 $len_u - len_{fa_u}$ 个。因此答案就是 $\sum_u len_u - len_{fa_u}$。

本质不同子串

例题 3.14

求 S 的本质不同子串数目。

题解

后缀数组。

题解

用 parent 树。不同等价类表示的串两两不同（因为 endpos 不同），而一个等价类内的串个数是 $len_u - len_{fa_u}$ 个。因此答案就是 $\sum_u len_u - len_{fa_u}$。

题解

用自动机。*DAG* 上一条路径对应一个本质不同子串，转化为 *DAG* 路径数问题，直接 *DP*：$f_u = \sum_c f_{ch_{u,c}} + 1$，答案是 f_{root}。

子串出现次数

例题 3.15

多次询问，每次询问给定字符串 T，求 T 在 S 中出现次数。(这里给定字符串 T 的方式可以是输入，也可以是给定 l, r 表示 $S[l;r]$ 这个子串，只需要能够定位即可)

子串出现次数

例题 3.15

多次询问，每次询问给定字符串 T，求 T 在 S 中出现次数。(这里给定字符串 T 的方式可以是输入，也可以是给定 l, r 表示 $S[l;r]$ 这个子串，只需要能够定位即可)

题解

一个子串在 S 出现次数就是对应等价类 endpos 集合大小。
而 endpos 有包含或不交的性质，因此在构造后缀自动机的过程，每次加入后将 lst 节点的大小增加 1，然后求 parent 树子树大小之和即为 endpos 集合大小。

子串出现次数

例题 3.16

多次询问，每次给定字符串 T 和 l, r，求 T 在 $S[l; r]$ 中出现次数。

子串出现次数

例题 3.16

多次询问，每次给定字符串 T 和 l, r，求 T 在 $S[l; r]$ 中出现次数。

题解

其实就是 endpos 在某个区间的元素个数。可以使用线段树合并维护 endpos 集合，复杂度 $O(n \log n)$。
也有题目会用到 *set* 启发式合并（$O(n \log^2 n)$），或者 *bitset*（$O(\frac{n^2}{\omega})$）维护 endpos。

最长公共子串

例题 3.17

给定字符串 S, T，求 S, T 的最长公共子串。$|S|, |T| \leq 10^5$。

最长公共子串

例题 3.17

给定字符串 S, T，求 S, T 的最长公共子串。$|S|, |T| \leq 10^5$。

题解

对 S 构造后缀自动机，对 T 的每个前缀找到出现在 S 中的最长后缀即可解决此题。

最长公共子串

例题 3.17

给定字符串 S, T，求 S, T 的最长公共子串。$|S|, |T| \le 10^5$。

题解

对 S 构造后缀自动机，对 T 的每个前缀找到出现在 S 中的最长后缀即可解决此题。具体的，维护当前匹配的节点 p 和长度 l，对于 T 当前枚举的字符 c，看是否存在 $ch_{p,c}$。

- 如果存在，那么匹配，l 增加。

最长公共子串

例题 3.17

给定字符串 S, T，求 S, T 的最长公共子串。$|S|, |T| \le 10^5$。

题解

对 S 构造后缀自动机，对 T 的每个前缀找到出现在 S 中的最长后缀即可解决此题。具体的，维护当前匹配的节点 p 和长度 l，对于 T 当前枚举的字符 c，看是否存在 $ch_{p,c}$。

- 如果存在，那么匹配，l 增加。
- *while* (不存在)，跳 parent 树，$p = fa_p, l = len_p$。

最长公共子串

例题 3.17

给定字符串 S,T，求 S,T 的最长公共子串。$|S|,|T| \le 10^5$。

题解

对 S 构造后缀自动机，对 T 的每个前缀找到出现在 S 中的最长后缀即可解决此题。具体的，维护当前匹配的节点 p 和长度 l，对于 T 当前枚举的字符 c，看是否存在 $ch_{p,c}$。

- 如果存在，那么匹配，l 增加。
- *while* (不存在)，跳 parent 树，$p = fa_p, l = len_p$。

这是后缀自动机上 *two-pointer* 的经典套路。

多个串的最长公共子串

例题 3.18

给定 n 个串 $S_1 \ldots, S_n$，求它们的最长公共子串。$n \leq 10, \sum_{i=1}^{n} |S_i| \leq 10^5$。

多个串的最长公共子串

例题 3.18

给定 n 个串 $S_1 \ldots, S_n$，求它们的最长公共子串。$n \le 10, \sum_{i=1}^{n} |S_i| \le 10^5$。

题解

第一个做法是对第一个串建立后缀自动机，其它串在上面跑，维护自动机上每个点最长匹配长度的最小值。
现在要维护点 u 的最长匹配长度的最小值，除了在经过 u 的时候更新，还需要在更新子树的时候更新，因为一旦匹配到子树内的点说明存在和 u 匹配长度为 len_u 的子串。

多个串的最长公共子串

例题 3.18

给定 n 个串 $S_1 \ldots, S_n$，求它们的最长公共子串。$n \le 10, \sum_{i=1}^{n} |S_i| \le 10^5$。

题解

第二个做法是将 S_i 拼接起来，构造后缀自动机。
每个节点 u 维护一个大小为 n 的数组 val_u，$val_{u,i}$ 记录 u 表示的串中最长的完全在 S_i 中的长度。
val_u 是可以从孩子的 val 更新维护的。
那么就是找到一个节点 u，最大化这个节点的 $val_{u,i}$ 的最小值。
这个做法通过线段树合并应该能做 n 更大的情况。

广义后缀自动机

对于一些多个串的问题，除了将字符串拼接构造后缀自动机，还有一种叫做广义后缀自动机的数据结构。
笔者没有见过必须要用到广义后缀自动机的题目，一般来说拼接字符串够用了。(除了洛谷模板题，要输出广义后缀自动机节点数)

广义后缀自动机

先讲一个网传的错误写法：每次插入一个串之后，重新将 lst 设为 1。这个做法的问题是会产生空节点（就是无法从根走到的点），不过有些问题有空节点也没有问题。事实上，这个写法只需要改一下后缀自动机的插入部分就对了。(这涉及到后缀自动机的结构部分，背板子就行了)

广义后缀自动机

再讲一个正确的离线写法：先建出 n 个串的 $trie$，再在 $trie$ 上 bfs（如果用 dfs 的话可能也会有空节点的问题），每次加入新字符就将它的父亲对应节点作为 lst，其它和正常后缀自动机相同。

bfs 复杂度是 $O(\text{trie 树节点数})$。

这个做法缺点是必须离线。

NOI2018 你的名字

至此，你已经可以使用后缀自动机秒天秒地了。
此时，你穿越到了 NOI2018 的赛场上，看到了这道题：

NOI2018 你的名字

至此，你已经可以使用后缀自动机秒天秒地了。
此时，你穿越到了 NOI2018 的赛场上，看到了这道题：

例题 3.19

给定一个字符串 S 和 Q 次询问，每次询问给出一个字符串 T 和一个区间 l, r，询问 T 有多少本质不同的子串，满足这个子串没有在 $S[l;r]$ 出现。
$|S| \le 5 \times 10^5, Q \le 10^5, \sum |T| \le 10^6$。

NOI2018 你的名字

例题 3.19

给定一个字符串 S 和 Q 次询问，每次询问给出一个字符串 T 和一个区间 l, r，询问 T 有多少本质不同的子串，满足这个子串没有在 $S[l; r]$ 出现。
$|S| \leq 5 \times 10^5, Q \leq 10^5, \sum |T| \leq 10^6$。

题解

Hint: 对 T 构建后缀自动机。

NOI2018 你的名字

例题 3.19

给定一个字符串 S 和 Q 次询问，每次询问给出一个字符串 T 和一个区间 l, r，询问 T 有多少本质不同的子串，满足这个子串没有在 $S[l;r]$ 出现。
$|S| \leq 5 \times 10^5, Q \leq 10^5, \sum |T| \leq 10^6$。

题解

对 T 构建后缀自动机，那么 T 的后缀自动机节点 p，不在 S 中出现的其实是长度大于某个值的串，设为 $[val_p, len_p]$，我们需要求出 val_p。
设 lim_i 表示字符串 $T[1;i]$ 能在 S 匹配到的最长后缀（即 $T[i-lim_i+1;i]$ 是 S 的子串且 lim_i 最大）。

NOI2018 你的名字

例题 3.19

给定一个字符串 S 和 Q 次询问，每次询问给出一个字符串 T 和一个区间 l, r，询问 T 有多少本质不同的子串，满足这个子串没有在 $S[l;r]$ 出现。
$|S| \le 5 \times 10^5, Q \le 10^5, \sum |T| \le 10^6$。

题解

对 T 构建后缀自动机，那么 T 的后缀自动机节点 p，不在 S 中出现的其实是长度大于某个值的串，设为 $[val_p, len_p]$，我们需要求出 val_p。
设 lim_i 表示字符串 $T[1;i]$ 能在 S 匹配到的最长后缀（即 $T[i - lim_i + 1; i]$ 是 S 的子串且 lim_i 最大)。
要维护 val_p，其实只需要知道 lim_i。
有 $val_p = \max(len_{fa_p}, lim_{pos_p}) + 1$，其中 pos_p 是 p 的任意一个 endpos 中的元素。

NOI2018 你的名字

例题 3.19

给定一个字符串 S 和 Q 次询问，每次询问给出一个字符串 T 和一个区间 l, r，询问 T 有多少本质不同的子串，满足这个子串没有在 $S[l;r]$ 出现。
$|S| \leq 5 \times 10^5, Q \leq 10^5, \sum |T| \leq 10^6$。

题解

对 T 构建后缀自动机，那么 T 的后缀自动机节点 p，不在 S 中出现的其实是长度大于某个值的串，设为 $[val_p, len_p]$，我们需要求出 val_p。
设 lim_i 表示字符串 $T[1;i]$ 能在 S 匹配到的最长后缀（即 $T[i - lim_i + 1; i]$ 是 S 的子串且 lim_i 最大)。
我们枚举 T 的字符，在 S 的后缀自动机上走，维护当前在 S 后缀自动机上节点和匹配长度，判断是否在 $S[l;r]$ 出现，等价于判断 endpos 中有无在某个区间中的元素。注意有个细节是删除开头字符是减少长度，并判断是否要跳父亲，而不是直接跳父亲。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l, r]$，求 $S[l; r]$ 本质不同子串个数。
$|S|, Q \leq 2 \times 10^5$。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l,r]$，求 $S[l;r]$ 本质不同子串个数。
$|S|, Q \le 2 \times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l, r]$，求 $S[l; r]$ 本质不同子串个数。
$|S|, Q \le 2 \times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1; r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S， Q 次询问给定区间 $[l, r]$，求 $S[l; r]$ 本质不同子串个数。
$|S|, Q \leq 2 \times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1; r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。
用 *LCT* 的 *access* 操作的均摊分析可得总变化量是 $O(n \log n)$ 级别的。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l,r]$，求 $S[l;r]$ 本质不同子串个数。
$|S|,Q\leq 2\times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1;r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。
用 *LCT* 的 *access* 操作的均摊分析可得总变化量是 $O(n\log n)$ 级别的。
证明：将树重链剖分，此时树边有两个属性重边和轻边，实边和虚边，设势能 p 为重虚边个数，每次 *access* 操作会访问所有到根的虚边。至多走 $\log_2 n$ 条轻虚边，每条轻虚边变成轻实边时可能会产生一条重虚边，因此 p 至多增加 $O(\log n)$；而每次访问一条重虚边，就会花费 $O(1)$ 的代价使得 p 减去 1，并且不会有新的重虚边产生。因此复杂度是均摊 $O(n\log n)$ 的。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l,r]$，求 $S[l;r]$ 本质不同子串个数。
$|S|,Q \le 2\times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1;r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。
用 *LCT* 的 *access* 操作的均摊分析可得总变化量是 $O(n\log n)$ 级别的。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S，Q 次询问给定区间 $[l, r]$，求 $S[l; r]$ 本质不同子串个数。
$|S|, Q \le 2 \times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1; r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。
用 *LCT* 的 *access* 操作的均摊分析可得总变化量是 $O(n \log n)$ 级别的。
现在就是将一段直上直下的链的 endpos 改成 r 了，直上直下的链对应一个长度区间，而我们需要维护的是左端点位置，这条链对应串左端点构成了一个区间，因此最终问题就转化为区间加区间求和问题。

LG P6292 区间本质不同子串个数

例题 3.20

给定字符串 S， Q 次询问给定区间 $[l,r]$，求 $S[l;r]$ 本质不同子串个数。
$|S|, Q \leq 2 \times 10^5$。

题解

对于一个串，我们在最后一次出现的左端点算贡献，那么扫描线 r 后就变成了求 l 到 r 的贡献和了。
于是扫描线 r，将 $S[1;r]$ 对应后缀自动机节点到根路径上的 endpos 全更新成 r，其实就是 *LCT* 的 *access* 操作。
用 *LCT* 的 *access* 操作的均摊分析可得总变化量是 $O(n \log n)$ 级别的。
现在就是将一段直上直下的链的 endpos 改成 r 了，直上直下的链对应一个长度区间，而我们需要维护的是左端点位置，这条链对应串左端点构成了一个区间，因此最终问题就转化为区间加区间求和问题。
有 $O(n \log n)$ 次操作，因此复杂度是 $O(n \log^2 n)$ 的。

DAG 链剖分

在字符串定位的问题中，除了关心最终走到哪，我们有时还关心在自动机的 DAG 上的整条路径。
路径长度是 $O(n)$ 的，因此当询问较多时，我们不可能求出路径上的每个点。
但是，我们可以参考树链剖分的想法：将一条路径划分成 $\log$ 条重链上的区间。对于每条重链（允许很长），在单独作区间处理，从而应对此类完整路径的查询/操作等问题。

DAG 链剖分

对于每个 DAG 上的节点 v，令 f_v 表示以 v 为起点的路径数，g_v 表示以 v 为终点的路径数。
则 $u \to v$ 是重边，当且仅当 $2f_v > f_u, 2g_u > g_v$。
Q: 为什么又要 f 又要 g 的限制，直接每个点向最大的 f 连边不行吗?
A: 不行，这样连边会出现两个不同点重儿子指向同一个点。

DAG 链剖分

对于每个 DAG 上的节点 v，令 f_v 表示以 v 为起点的路径数，g_v 表示以 v 为终点的路径数。
则 $u \to v$ 是重边，当且仅当 $2f_v > f_u, 2g_u > g_v$。

引理 3.5

一条路径上至多 $O(\log n)$ 条轻边。

DAG 链剖分

对于每个 DAG 上的节点 v，令 f_v 表示以 v 为起点的路径数，g_v 表示以 v 为终点的路径数。
则 $u \to v$ 是重边，当且仅当 $2f_v > f_u, 2g_u > g_v$。

引理 3.5

一条路径上至多 $O(\log n)$ 条轻边。

证明

显然，f_v, g_v 不超过 n^2。走一条轻边，f 至少除以 2 或 g 至少乘以 2，因此至多走过 $O(\log n)$ 条轻边。

DAG 链剖分（子串定位）

使用 DAG 链剖分也能实现子串定位，设需要定位的串为 S。从起点出发，沿着路径走，考虑当前点所在重链，那么就是要求 S 与当前重链的 LCP。
求出 LCP 长度，可以直接跳到重链上的对应位置，然后选择相应的出边（应当是轻边）走过去，到达下一条重链。
重复这个过程，直到走完 S。

DAG 链剖分（子串定位）

使用 DAG 链剖分也能实现子串定位，设需要定位的串为 S。从起点出发，沿着路径走，考虑当前点所在重链，那么就是要求 S 与当前重链的 LCP。
求出 LCP 长度，可以直接跳到重链上的对应位置，然后选择相应的出边（应当是轻边）走过去，到达下一条重链。
重复这个过程，直到走完 S。
注：

- 我们要支持查询 LCP（一条重链的某个子串与 S 的某个子串），因此将 S 和所有重链拼接成一个大串，用后缀数组/后缀自动机处理 LCP。
- 这部分复杂度预处理 $O(n \log n)$，查询复杂度 $O(1)$。(也可以哈希，查询复杂度多 $\log$)
- 由于至多经过 $O(\log n)$ 条轻边，单次查询复杂度 $O(\log n)$。

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \le 10^5$。

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \le 10^5$。

题解

令 $S = S_1 +' *' + S_2 +' *' + \cdots +' *' + S_n$，对 S 构建后缀自动机，对 *DAG* 链剖分。
其中 $'*'$ 是一个特殊字符。

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \le 10^5$。

题解

令 $S = S_1 +' *' + S_2 +' *' + \cdots +' *' + S_n$，对 S 构建后缀自动机，对 *DAG* 链剖分。
其中 $'*'$ 是一个特殊字符。
令每个点出边按字典序排序，则 k 小子串为从起点开始 *DFS* 到达的第 k 个节点 (限制路径不经过'*' 边)。

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \le 10^5$。

题解

令 $S = S_1 +' *' + S_2 +' *' + \cdots +' *' + S_n$，对 S 构建后缀自动机，对 *DAG* 链剖分。
其中 $'*'$ 是一个特殊字符。
令每个点出边按字典序排序，则 k 小子串为从起点开始 *DFS* 到达的第 k 个节点 (限制路径不经过'*' 边)。
对于询问，我们只需要支持查询：当前重链往后要走多远?

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \leq 10^5$。

题解

令 $S = S_1 +' *' + S_2 +' *' + \cdots +' *' + S_n$，对 S 构建后缀自动机，对 *DAG* 链剖分。
其中 $'*'$ 是一个特殊字符。
令每个点出边按字典序排序，则 k 小子串为从起点开始 *DFS* 到达的第 k 个节点 (限制路径不经过'*' 边)。
对于询问，我们只需要支持查询：当前重链往后要走多远?
预处理每个点往后 *DFS* 到达的节点数量，我们可以在重链上二分得到分叉点。

多串 k 小子串

例题 3.21

给定 n 个串 $S_1, \ldots, S_n$。
Q 次询问，每次询问输入 k，求出本质不同子串中字典序第 k 小的子串在哪个串中的哪个位置（多个位置任意输出）。
$Q, \sum_{i=1}^{n} |S_i| \leq 10^5$。

题解

令 $S = S_1 +' *' + S_2 +' *' + \cdots +' *' + S_n$，对 S 构建后缀自动机，对 *DAG* 链剖分。
其中 $'*'$ 是一个特殊字符。
令每个点出边按字典序排序，则 k 小子串为从起点开始 *DFS* 到达的第 k 个节点 (限制路径不经过'*' 边)。
对于询问，我们只需要支持查询：当前重链往后要走多远?
预处理每个点往后 *DFS* 到达的节点数量，我们可以在重链上二分得到分叉点。
单次询问复杂度 $O(\log^2 n)$。

UOJ752 Border 的第五种求法

例题 3.22

给定字符串 S，和一个长度为 $n = |S|$ 的数组 f。
定义 occ_T 表示 T 在 S 中的出现次数，定义一个字符串 T 的价值为 f_{occ_T}。
Q 次询问，每次询问给定 l, r，询问 $S[l;r]$ 的所有 *border* 的价值之和。(注意这里价值是定义在 S 上的，而不是定义在 $S[l;r]$ 上的)
$|S|, Q \leq 5 \times 10^5, 1 \leq f_i \leq 10^9$。

UOJ752 Border 的第五种求法

例题 3.22

给定字符串 S，和一个长度为 $n=|S|$ 的数组 f。
定义 occ_T 表示 T 在 S 中的出现次数，定义一个字符串 T 的价值为 f_{occ_T}。
Q 次询问，每次询问给定 l,r，询问 $S[l;r]$ 的所有 *border* 的价值之和。(注意这里价值是定义在 S 上的，而不是定义在 $S[l;r]$ 上的)
$|S|,Q\le 5\times 10^5, 1\le f_i\le 10^9$。

题解

找到所有 $1\le i\le r-l+1$ 满足 $S[l;l+i-1]$ 是 $S[1;r]$ 的后缀，这些 i 对应了 $S[l;r]$ 的 *border*。如果我们找到了 $S[l;l], S[l;l+1], \dots S[l;r]$ 在 *DAG* 上的链，答案即为这些点中作为 parent 树上 $S[1;r]$ 的祖先的 occ 之和。

UOJ752 Border 的第五种求法

例题 3.22

给定字符串 S，和一个长度为 $n = |S|$ 的数组 f。
定义 occ_T 表示 T 在 S 中的出现次数，定义一个字符串 T 的价值为 f_{occ_T}。
Q 次询问，每次询问给定 l, r，询问 $S[l; r]$ 的所有 *border* 的价值之和。(注意这里价值是定义在 S 上的，而不是定义在 $S[l; r]$ 上的)
$|S|, Q \leq 5 \times 10^5, 1 \leq f_i \leq 10^9$。

题解

找到所有 $1 \leq i \leq r - l + 1$ 满足 $S[l; l + i - 1]$ 是 $S[1; r]$ 的后缀，这些 i 对应了 $S[l; r]$ 的 *border*。如果我们找到了 $S[l; l], S[l; l + 1], \ldots S[l; r]$ 在 *DAG* 上的链，答案即为这些点中作为 parent 树上 $S[1; r]$ 的祖先的 occ 之和。
occ 只与后缀自动机的节点有关，因此可将 f_{occ} 放到节点上，将 *DAG* 上的链转化为区间，问题转化为只考虑某个点到根路径上的点，某个区间的权值和（区间和等于前缀和的差）。离线后枚举前缀，就是单点修改到根路径求和问题，可转为区间求和问题。或者 *dfs* 枚举到根路径，就是单点修改区间求和问题。

UOJ697 广为人知题

例题 3.23

给定一个长度为 n 的字符串 S，和 m 个模式串，第 i 个为 $S[tl_i;tr_i]$（即 S 的一个子串）。
Q 次询问，每次给定 $[ql_i, qr_i]$，求所有模式串在 $S[ql_i;qr_i]$ 中的出现次数之和。
$n \le 4 \times 10^5, m \le 10^6, Q \le 10^5$

UOJ697 广为人知题

例题 3.23

给定一个长度为 n 的字符串 S，和 m 个模式串，第 i 个为 $S[tl_i;tr_i]$（即 S 的一个子串）。
Q 次询问，每次给定 $[ql_i, qr_i]$，求所有模式串在 $S[ql_i;qr_i]$ 中的出现次数之和。
$n \le 4 \times 10^5, m \le 10^6, Q \le 10^5$

题解

一个前缀的后缀对应一个子串。

UOJ697 广为人知题

例题 3.23

给定一个长度为 n 的字符串 S，和 m 个模式串，第 i 个为 $S[tl_i;tr_i]$（即 S 的一个子串)。
Q 次询问，每次给定 $[ql_i, qr_i]$，求所有模式串在 $S[ql_i;qr_i]$ 中的出现次数之和。
$n \le 4 \times 10^5, m \le 10^6, Q \le 10^5$

题解

一个前缀的后缀对应一个子串。
如果知道了 $S[ql_i;qr_i]$ 在 *DAG* 链上经过的所有点（这是枚举前缀)，$S[ql_i;j]$ 对答案的贡献是 parent 树上到根路径（这是枚举前缀的后缀）模式串个数减去点对应的等价类中比 $j - ql_i + 1$ 长的模式串个数。

UOJ697 广为人知题

例题 3.23

给定一个长度为 n 的字符串 S，和 m 个模式串，第 i 个为 $S[tl_i;tr_i]$（即 S 的一个子串）。
Q 次询问，每次给定 $[ql_i, qr_i]$，求所有模式串在 $S[ql_i;qr_i]$ 中的出现次数之和。
$n \le 4\times 10^5, m \le 10^6, Q \le 10^5$

题解

一个前缀的后缀对应一个子串。
如果知道了 $S[ql_i;qr_i]$ 在 *DAG* 链上经过的所有点（这是枚举前缀），$S[ql_i;j]$ 对答案的贡献是 parent 树上到根路径（这是枚举前缀的后缀）模式串个数减去点对应的等价类中比 $j-ql_i+1$ 长的模式串个数。
第一类贡献只需要在 *DAG* 链上，以点到根路径模式串个数为权值求前缀和即可，第二类贡献是一个斜 *45* 度四边形数点问题，将坐标变换后就是普通的二维数点问题。

UOJ697 广为人知题

例题 3.23

给定一个长度为 n 的字符串 S，和 m 个模式串，第 i 个为 $S[tl_i; tr_i]$（即 S 的一个子串）。
Q 次询问，每次给定 $[ql_i, qr_i]$，求所有模式串在 $S[ql_i; qr_i]$ 中的出现次数之和。
$n \le 4 \times 10^5, m \le 10^6, Q \le 10^5$

题解

一个前缀的后缀对应一个子串。
如果知道了 $S[ql_i; qr_i]$ 在 *DAG* 链上经过的所有点（这是枚举前缀），$S[ql_i; j]$ 对答案的贡献是 parent 树上到根路径（这是枚举前缀的后缀）模式串个数减去点对应的等价类中比 $j - ql_i + 1$ 长的模式串个数。
第一类贡献只需要在 *DAG* 链上，以点到根路径模式串个数为权值求前缀和即可，第二类贡献是一个斜 *45* 度四边形数点问题，将坐标变换后就是普通的二维数点问题。
时间复杂度 $O((n + m) \log n + Q \log^2 n)$。

基本子串结构

后缀自动机的终极形态是基本子串结构。
设要求的是字符串 S 的基本子串结构。

基本子串结构

后缀自动机的终极形态是基本子串结构。
设要求的是字符串 S 的基本子串结构。

例题 3.24

给定字符串 S，Q 次询问，询问正串 parent 树和反串 parent 树中指定两个节点代表的字符串的交集（集合大小可能较大，需要用某种方式表示出这个集合）。
$|S|, Q \leq 10^5$。

基本子串结构

后缀自动机的终极形态是基本子串结构。
设要求的是字符串 S 的基本子串结构。

例题 3.24

给定字符串 S，Q 次询问，询问正串 parent 树和反串 parent 树中指定两个节点代表的字符串的交集（集合大小可能较大，需要用某种方式表示出这个集合）。
$|S|, Q \leq 10^5$。

定义 3.10

(occ). 定义 $occ(T)$ 表示 T 在 S 中出现次数。

基本子串结构

定义 3.11

(扩展串). 对于一个子串 T，定义其扩展串 $\mathrm{ext}(T)$ 表示 S 最长的子串 T'，满足 T 是 T' 的子串并且 $\mathrm{occ}(T) = \mathrm{occ}(T')$。

基本子串结构

定义 3.11

(扩展串). 对于一个子串 T，定义其扩展串 $\mathrm{ext}(T)$ 表示 S 最长的子串 T'，满足 T 是 T' 的子串并且 $\mathrm{occ}(T) = \mathrm{occ}(T')$。

引理 3.6

对于 S 的子串 T_1, T_2，若 T_1 包含 T_2，则 $\mathrm{occ}(T_1) \leq \mathrm{occ}(T_2)$。

基本子串结构

定义 3.11

(扩展串). 对于一个子串 T，定义其扩展串 $\mathrm{ext}(T)$ 表示 S 最长的子串 T'，满足 T 是 T' 的子串并且 $\mathrm{occ}(T) = \mathrm{occ}(T')$。

引理 3.6

对于 S 的子串 T_1, T_2，若 T_1 包含 T_2，则 $\mathrm{occ}(T_1) \leq \mathrm{occ}(T_2)$。

引理 3.7

对于 S 的子串 T，$\mathrm{ext}(T)$ 存在且唯一。

基本子串结构

定义 3.11

(扩展串). 对于一个子串 T，定义其扩展串 $\mathrm{ext}(T)$ 表示 S 最长的子串 T'，满足 T 是 T' 的子串并且 $\mathrm{occ}(T) = \mathrm{occ}(T')$。

引理 3.6

对于 S 的子串 T_1, T_2，若 T_1 包含 T_2，则 $\mathrm{occ}(T_1) \leq \mathrm{occ}(T_2)$。

引理 3.7

对于 S 的子串 T，$\mathrm{ext}(T)$ 存在且唯一。

证明

存在性显然。唯一性使用反证法，假设存在 $T_1 \neq T_2$，且它们都是 $\mathrm{ext}(T)$，那么设 $T_1 = ATA', T_2 = BTB'$（这是因为 T 是它们的子串），令 C 为 A 和 B 中长度长的串，C' 为 A' 和 B' 中长度长的串，那么 CTC' 是 $\mathrm{ext}(T)$，且长度更长。

基本子串结构

推论 3.1

设 $\operatorname{ext}(S[l;r]) = S[l';r'], [l,r] \subseteq [l',r']$，那么 $\forall l'' \in [l',l], r'' \in [r',r]$，都有 $\operatorname{ext}(S[l'';r'']) = \operatorname{ext}(S[l;r])$。

基本子串结构

推论 3.1

设 $\operatorname{ext}(S[l;r]) = S[l';r'], [l,r] \subseteq [l',r']$，那么 $\forall l'' \in [l',l], r'' \in [r',r]$，都有 $\operatorname{ext}(S[l'';r'']) = \operatorname{ext}(S[l;r])$。

推论 3.2

对于子串 T，有 $\operatorname{ext}(\operatorname{ext}(T)) = \operatorname{ext}(T)$。

基本子串结构

推论 3.1

设 $\mathrm{ext}(S[l;r]) = S[l';r'], [l,r] \subseteq [l',r']$，那么 $\forall l'' \in [l',l], r'' \in [r',r]$，都有 $\mathrm{ext}(S[l'';r'']) = \mathrm{ext}(S[l;r])$。

推论 3.2

对于子串 T，有 $\mathrm{ext}(\mathrm{ext}(T)) = \mathrm{ext}(T)$。

定义 3.12

(等价关系, 等价类). 根据 ext 可以定义等价关系：两个子串 x, y 等价，当且仅当 $\mathrm{ext}(x) = \mathrm{ext}(y)$。根据等价关系可以将 S 的子串自然的划分成若干等价类。

基本子串结构

推论 3.1

设 $\mathrm{ext}(S[l;r]) = S[l';r'], [l,r] \subseteq [l',r']$，那么 $\forall l'' \in [l',l], r'' \in [r',r]$，都有 $\mathrm{ext}(S[l'';r'']) = \mathrm{ext}(S[l;r])$。

推论 3.2

对于子串 T，有 $\mathrm{ext}(\mathrm{ext}(T)) = \mathrm{ext}(T)$。

定义 3.12

(等价关系, 等价类). 根据 ext 可以定义等价关系：两个子串 x, y 等价，当且仅当 $\mathrm{ext}(x) = \mathrm{ext}(y)$。根据等价关系可以将 S 的子串自然的划分成若干等价类。

注：在基本子串结构中提到的等价类通常是 ext 等价类，要和后缀自动机节点的 endpos 等价类区分。

基本子串结构

定义 3.13

(代表元). 对于子串 T，若 $\mathrm{ext}(T) = T$，则称 T 为 T 所在等价类的代表元。若代表元 T 所属等价类为 g，则用 $\mathrm{rep}(g) = T$ 代表 g 的代表元。

基本子串结构

定义 3.13

(代表元). 对于子串 T，若 $\mathrm{ext}(T) = T$，则称 T 为 T 所在等价类的代表元。若代表元 T 所属等价类为 g，则用 $\mathrm{rep}(g) = T$ 代表 g 的代表元。

引理 3.8

对每个等价类 g，代表元 $\mathrm{rep}(g)$ 存在且唯一。

基本子串结构

定义 3.13

(代表元). 对于子串 T，若 $\mathrm{ext}(T) = T$，则称 T 为 T 所在等价类的代表元。若代表元 T 所属等价类为 g，则用 $\mathrm{rep}(g) = T$ 代表 g 的代表元。

引理 3.8

对每个等价类 g，代表元 $\mathrm{rep}(g)$ 存在且唯一。

引理 3.9

对于子串 T_1, T_2，若 $\mathrm{occ}(T_1) = \mathrm{occ}(T_2)$ 且 T_1 包含 T_2，则 T_1 与 T_2 等价。

基本子串结构

引入平面直角坐标系上的点来代表子串。

基本子串结构

引入平面直角坐标系上的点来代表子串。

定义 3.14

$(\mathrm{posl}, \mathrm{posr})$. 设 T 在 S 第一次出现位置为 $S[\mathrm{posl}(T); \mathrm{posr}(T)]$。

基本子串结构

引入平面直角坐标系上的点来代表子串。

定义 3.14

($\mathrm{posl}, \mathrm{posr}$*)*. 设 T 在 S 第一次出现位置为 $S[\mathrm{posl}(T); \mathrm{posr}(T)]$。

引理 3.10

建立 l 为横轴，r 为纵轴的平面直角坐标系，子串 T 位于 $(\mathrm{posl}(T), \mathrm{posr}(T))$，则等价类内的子串在平面上对应的点构成了上侧和左侧对齐，下侧和右侧呈阶梯形的阶梯状点阵。

基本子串结构

引入平面直角坐标系上的点来代表子串。

定义 3.14

$(\mathrm{posl}, \mathrm{posr})$. 设 T 在 S 第一次出现位置为 $S[\mathrm{posl}(T); \mathrm{posr}(T)]$。

引理 3.10

建立 l 为横轴，r 为纵轴的平面直角坐标系，子串 T 位于 $(\mathrm{posl}(T), \mathrm{posr}(T))$，则等价类内的子串在平面上对应的点构成了上侧和左侧对齐，下侧和右侧呈阶梯形的阶梯状点阵。

证明

对于任意等价类 g 的代表元与等价类 g 中的任意点形成的矩形中的点都在等价类中，因此等价类中的点形成阶梯形。

基本子串结构

引入平面直角坐标系上的点来代表子串。

定义 3.14

$(\mathrm{posl}, \mathrm{posr})$. 设 T 在 S 第一次出现位置为 $S[\mathrm{posl}(T); \mathrm{posr}(T)]$。

引理 3.10

建立 l 为横轴，r 为纵轴的平面直角坐标系，子串 T 位于 $(\mathrm{posl}(T), \mathrm{posr}(T))$，则等价类内的子串在平面上对应的点构成了上侧和左侧对齐，下侧和右侧呈阶梯形的阶梯状点阵。

推论 3.3

所有 $1 \le l \le r \le |S|$ 的点被等价类划分成了若干个互不相交的阶梯形。其中等价类 g 对应的阶梯形出现了 $\mathrm{occ}(\mathrm{rep}(g))$ 次。

基本子串结构

设 $S=$ aababcd，那么可以划分成如下等价类：

g_1 ={aa,aab,aaba,aababc,aababcd}∪
{aba,abab,ababc,ababcd}∪
{ba,bab,babc,babcd}∪
{abc,abcd} ∪ {bc,bcd} ∪ {c,cd} ∪ {d}

g_2 ={b,ab}

g_3 ={a}

两个点在同一个等价类说明两个子串的 ext 相等，并且两点的 endpos 集合只差一个系数。

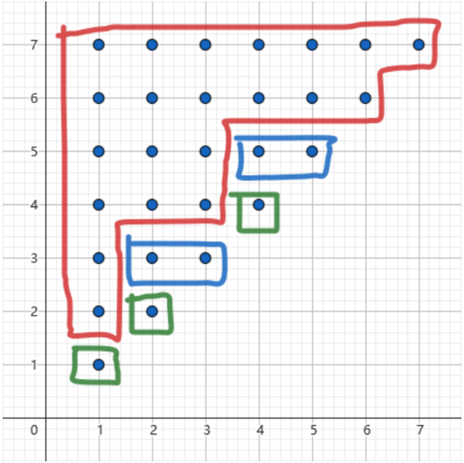


Fig 3.2 字符串 aababcd 的等价类形态

基本子串结构

定理 3.3

一个等价类的一行对应一个 endpos 等价类，从而对应一个正串 parent 树上节点。
一个等价类的一列对应一个 startpos 等价类，从而对应一个反串 parent 树上节点。

基本子串结构

定理 3.3

一个等价类的一行对应一个 endpos 等价类，从而对应一个正串 parent 树上节点。
一个等价类的一列对应一个 startpos 等价类，从而对应一个反串 parent 树上节点。

证明

endpos/startpos 相同的只可能在同一个等价类内，而等价类内一行/列的 endpos/startpos 相同。

基本子串结构

定义 3.15

(周长). 一个等价类 g 的周长 $\mathrm{per}(g)$ 为其阶梯形点阵的行数与列数之和。

基本子串结构

定义 3.15

(周长). 一个等价类 g 的周长 $\operatorname{per}(g)$ 为其阶梯形点阵的行数与列数之和。

定理 3.4

$\sum_g \operatorname{per}(g) = O(n)$。

基本子串结构

定义 3.15

(周长). 一个等价类 g 的周长 $\mathrm{per}(g)$ 为其阶梯形点阵的行数与列数之和。

定理 3.4

$\sum_g \mathrm{per}(g) = O(n)$。

证明

行数可以对应到正串后缀自动机节点数，列数可以对应到反串后缀自动机节点数，而后缀自动机点数 $O(n)$，故周长和为 $O(n)$。

基本子串结构

定义 3.15

(周长). 一个等价类 g 的周长 $\operatorname{per}(g)$ 为其阶梯形点阵的行数与列数之和。

定理 3.4

$\sum_g \operatorname{per}(g) = O(n)$。

证明

行数可以对应到正串后缀自动机节点数，列数可以对应到反串后缀自动机节点数，而后缀自动机点数 $O(n)$，故周长和为 $O(n)$。

我们可以将 parent 树上的边连到等价类上，这个结构就是基本子串结构。

基本子串结构

定义 3.15

(周长). 一个等价类 g 的周长 $\mathrm{per}(g)$ 为其阶梯形点阵的行数与列数之和。

定理 3.4

$\sum_g \mathrm{per}(g) = O(n)$。

证明

行数可以对应到正串后缀自动机节点数，列数可以对应到反串后缀自动机节点数，而后缀自动机点数 $O(n)$，故周长和为 $O(n)$。

我们可以将 parent 树上的边连到等价类上，这个结构就是基本子串结构。
按照连边方式，基本子串结构还有另一种含义：基本子串结构是以本质不同子串为节点的 DAG，其中 T 连向 $T+c$ 和 $c+T$。（T 为子串，c 为字符）

基本子串结构

后缀自动机的自动机部分也能放在基本子串结构中。
一个节点走自动机的边：

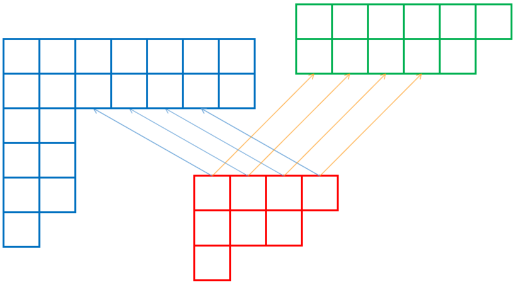
- 如果这个节点不在上边界，那么肯定只有唯一的字符能走。
- 如果这个节点在上边界，会走到其它块的一段连续且对齐的下边界。

基本子串结构

后缀自动机的自动机部分也能放在基本子串结构中。
一个节点走自动机的边：

- 如果这个节点不在上边界，那么肯定只有唯一的字符能走。
- 如果这个节点在上边界，会走到其它块的一段连续且对齐的下边界。

第二种情况自动机的边是和 parent 树边一一对应的。



基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

步骤 1 的复杂度线性（因为构建后缀自动机能做到线性)。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

步骤 2 如果使用倍增定位复杂度 $O(n \log n)$。
在自动机上 dfs，只走 $len_v = len_u + 1$ 的边，在 T_1 上维护节点编号。支持开头结尾加入删除的后缀自动机上点定位单次是 $O(1)$ 的，因此步骤 2 复杂度 $O(n)$。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

步骤 3 复杂度线性。

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

步骤 4 按照拓扑序放就是排序的结果，复杂度线性。(或者按照后缀自动机构建时加入的顺序)

基本子串结构（构建）

1. 先对 S 建出正反串后缀自动机。
2. 找出所有代表元，枚举正串 parent 树的节点，将最长的串在反串 parent 树上定位，判断是否是等价类内最长的串。(顺便得到了代表元在正反串 parent 树上对应的点)
3. 给后缀自动机上节点找到对应等价类编号，如果不是代表元所在等价类，那么只有唯一的出边，按照拓扑序可求出等价类编号。
4. 将等价类的节点的行按照 r 排序，列按照 l 排序。

因此，构建基本子串结构复杂度是线性的。

基本子串结构

例题 3.25

给定字符串 S，Q 次询问，询问正串 parent 树和反串 parent 树中指定两个节点代表的字符串的交集（集合大小可能较大，需要用某种方式表示出这个集合）。
$|S|, Q \leq 10^5$。

基本子串结构

例题 3.25

给定字符串 S，Q 次询问，询问正串 parent 树和反串 parent 树中指定两个节点代表的字符串的交集（集合大小可能较大，需要用某种方式表示出这个集合）。
$|S|, Q \leq 10^5$。

题解

知道了正反 parent 树上的节点，可以找到在等价类对应的行列。
如果不在同一个等价类，显然交集为空，否则交集为行列的交点。

基本子串结构（与 border）

之前我们介绍了用基本子串字典处理区间 border 的方法，DAG 链剖分也给了一种处理区间 border 的方法，而基本子串结构给了另一种看待区间 border 问题的视角。

基本子串结构（与 border）

之前我们介绍了用基本子串字典处理区间 border 的方法，DAG 链剖分也给了一种处理区间 border 的方法，而基本子串结构给了另一种看待区间 border 问题的视角。子串 $S[l;r]$ 的 border 即同时为前缀和后缀的串的集合，相当于 $S[l;r]$ 在正串和反串 parent 树到根路径的交。

基本子串结构（与 border）

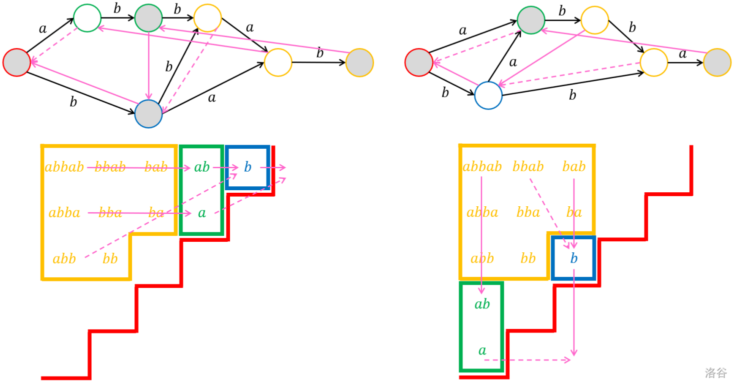
之前我们介绍了用基本子串字典处理区间 border 的方法，DAG 链剖分也给了一种处理区间 border 的方法，而基本子串结构给了另一种看待区间 border 问题的视角。子串 $S[l;r]$ 的 border 即同时为前缀和后缀的串的集合，相当于 $S[l;r]$ 在正串和反串 parent 树到根路径的交。
对两棵 parent 树重链剖分，以正串 parent 树为例，由于同一个等价类在直角坐标系出现多次，我们需要给每个等价类选择一个出现位置，使得所有重儿子都是直接的转移（即转移是同一行）。

基本子串结构（与 border）

之前我们介绍了用基本子串字典处理区间 border 的方法，DAG 链剖分也给了一种处理区间 border 的方法，而基本子串结构给了另一种看待区间 border 问题的视角。
子串 $S[l;r]$ 的 border 即同时为前缀和后缀的串的集合，相当于 $S[l;r]$ 在正串和反串 parent 树到根路径的交。
对两棵 parent 树重链剖分，以正串 parent 树为例，由于同一个等价类在直角坐标系出现多次，我们需要给每个等价类选择一个出现位置，使得所有重儿子都是直接的转移（即转移是同一行）。
这个选位置的算法是存在的，对于 parent 树的一条重链，叶子的出现次数是 1，因此在直角坐标系的位置固定，然后对每个祖先，其出现位置为重儿子的右侧。(每个叶子所在行不同，可以将重链标号为行数)。

基本子串结构（与 border）

下图展示了 abbab 的正反串 parent 树的重链剖分和对应的基本子串结构中等价类的位置，可以看到，重边都是横平竖直的。



基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

证明

因为标号就是行号/列号。

基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

再看区间 border 问题，问题转化为求两棵树上的两个到根路径的交，先将每个路径拆成 $O(\log|S|)$ 重链，只有 $O(\log|S|)$ 对重链可能有交（因为至少要求长度相等）。

基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

再看区间 border 问题，问题转化为求两棵树上的两个到根路径的交，先将每个路径拆成 $O(\log|S|)$ 重链，只有 $O(\log|S|)$ 对重链可能有交（因为至少要求长度相等)。一个正串节点，对应基本子串结构一个等价类的一行，根据引理，这个些串在反串 parent 树上的重链标号是连续的。(即一个正串节点对应一个反串重链标号的区间)

基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

再看区间 border 问题，问题转化为求两棵树上的两个到根路径的交，先将每个路径拆成 $O(\log|S|)$ 重链，只有 $O(\log|S|)$ 对重链可能有交（因为至少要求长度相等）。一个正串节点，对应基本子串结构一个等价类的一行，根据引理，这个些串在反串 parent 树上的重链标号是连续的。(即一个正串节点对应一个反串重链标号的区间) 如果建立一个 x 轴为反串重链标号，y 轴为长度的平面直角坐标系，那么一个正串等价类对应了一条斜率为 1 线段。(我们要考虑的其实不是一整条重链，而是重链的一个前缀，这个通过增加对长度限制即可解决，仍是对整条重链的查询问题)

基本子串结构（与 border）

引理 3.11

同一个等价类中，正（反）串 parent 树的点所在重链标号连续。

再看区间 border 问题，问题转化为求两棵树上的两个到根路径的交，先将每个路径拆成 $O(\log|S|)$ 重链，只有 $O(\log|S|)$ 对重链可能有交（因为至少要求长度相等）。
一个正串节点，对应基本子串结构一个等价类的一行，根据引理，这个些串在反串 parent 树上的重链标号是连续的。(即一个正串节点对应一个反串重链标号的区间)
如果建立一个 x 轴为反串重链标号，y 轴为长度的平面直角坐标系，那么一个正串等价类对应了一条斜率为 1 线段。(我们要考虑的其实不是一整条重链，而是重链的一个前缀，这个通过增加对长度限制即可解决，仍是对整条重链的查询问题)
以区间 border 计数为例，对于每个正串重链，可转化为：给定平面中若干个斜率为 1 的斜线段（这若干个斜线段对应了一条正串重链上所有节点)，若干次询问一条平行于 y 轴的线段（对应了一条反串重链）与多少个斜线段有交，这可以简单的转化为二维数点问题。

从而我们用基本子串结构也解决了区间 border 问题，复杂度相比基本子串字典多一个 $\log$，因为二维数点一个 $\log$，重链有 $\log$ 条。复杂度两个 $\log$。

LG P8351 [SDOI/SXOI2022] 子串统计

例题 3.26

给定长度为 n 的字符串 S，初始 $T_0 = S$，每次可以删除 T_i 开头或结尾删除的字符得到一个新字符串 T_{i+1}，经过 $n-1$ 操作，会得到一个只有一个字符的串 T_{n-1}，根据每次删除的选择，共有 2^{n-1} 种可能的操作序列。(如果有一次删除开头和结尾得到相同的串，我们仍然将其看作不同的序列)
对于一个串 T，定义 $\operatorname{occ}(T)$ 表示 T 在 S 的出现次数。
对于一个操作序列，它的贡献是：

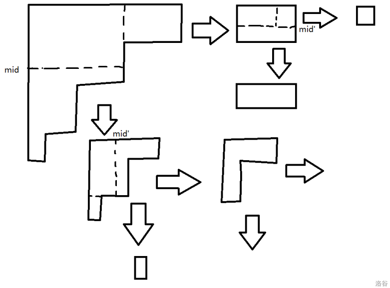
$$\prod_{i=1}^{n-1} \operatorname{occ}(T_i)$$

求所有操作序列的贡献和，对 *998244353* 取模。

LG P8351 [SDOI/SXOI2022] 子串统计

题解

每种方案可以看作是在基本子串结构上从左上角走（只能向下、向右走）到 $x=y$ 上任意一个点路径，我们需要对所有路径经过的权值的乘积求和，权值是 occ，因此一个等价类内的权值相同。
我们单独考虑一个等价类，如果路径经过这个等价类，一定是连接这个等价类的右下轮廓的某个点和左上边界某个点，这部分的贡献可以使用一个分治 *NTT* 求出（如右图，维护到阶梯左上边界的权值和，每次选择长的边界折半转移，拆成矩形和两个阶梯继续递归，最后矩形的转移可直接 *NTT*）
算一个等价类的复杂度是 $O(\mathrm{per(g)}\log^2\mathrm{per(g)})$，总复杂度 $O(|S|\log^2|S|)$。



UOJ577 打击复读

例题 3.27

给定字符串 S，每个位置还给出权值 wl_i, wr_i，定义一个子串 t 的左权值 $vl(t)$ 为其在原串出现的所有位置的左端点的 wl_i 之和，右权值 $vr(t)$ 为其在原串出现的所有位置的右端点的 wr_i 之和。
Q 询问，每次修改某个 wl_i，求：

$$\sum_{i=1}^{n}\sum_{j=i}^{n} vl(S[i;j]) \times vr(S[i;j])$$

$|S| \le 5 \times 10^5$。(可以到线性)

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。
一个串的左权值为所有 $i \in \mathrm{startpos}$ 的 wl_i 之和，右权值为所有 $i \in \mathrm{endpos}$ 的 wr_i 之和。可在正反串 parent 树节点求出。

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。
一个串的左权值为所有 $i \in \text{startpos}$ 的 wl_i 之和，右权值为所有 $i \in \text{endpos}$ 的 wr_i 之和。可在正反串 parent 树节点求出。
因此基本子串结构的一个块内一行（对应正串 parent 树）有一个右权值 vr，一列 (对应反串 parent 树）有一个左权值 vl，一个点的贡献就是 $vl \times vr \times$ 出现次数。

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。
一个串的左权值为所有 $i \in \mathrm{startpos}$ 的 wl_i 之和，右权值为所有 $i \in \mathrm{endpos}$ 的 wr_i 之和。可在正反串 parent 树节点求出。
因此基本子串结构的一个块内一行（对应正串 parent 树）有一个右权值 vr，一列(对应反串 parent 树）有一个左权值 vl，一个点的贡献就是 $vl \times vr \times$ 出现次数。
我们想要直到的 wl_i 的系数可以通过每个块的每列的 vl 的系数求得。因此我们只需要求每个块的 vl 的系数。

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。
一个串的左权值为所有 $i \in \text{startpos}$ 的 wl_i 之和，右权值为所有 $i \in \text{endpos}$ 的 wr_i 之和。可在正反串 parent 树节点求出。
因此基本子串结构的一个块内一行（对应正串 parent 树）有一个右权值 vr，一列 (对应反串 parent 树）有一个左权值 vl，一个点的贡献就是 $vl \times vr \times$ 出现次数。
我们想要直到的 wl_i 的系数可以通过每个块的每列的 vl 的系数求得。因此我们只需要求每个块的 vl 的系数。
块内一列的 vl 的系数就是每个块这列上每个点的 $vr \times$ 出现次数 ，块内出现次数是固定的，因此只需要求块内 vr 的前缀和。

UOJ577 打击复读

题解

答案关于所有 wl_i 是线性函数，于是尝试求出每个 wl_i 的系数。
一个串的左权值为所有 $i \in \mathrm{startpos}$ 的 wl_i 之和，右权值为所有 $i \in \mathrm{endpos}$ 的 wr_i 之和。可在正反串 parent 树节点求出。
因此基本子串结构的一个块内一行（对应正串 parent 树）有一个右权值 vr，一列 (对应反串 parent 树）有一个左权值 vl，一个点的贡献就是 $vl \times vr \times$ 出现次数。
我们想要直到的 wl_i 的系数可以通过每个块的每列的 vl 的系数求得。因此我们只需要求每个块的 vl 的系数。
块内一列的 vl 的系数就是每个块这列上每个点的 $vr \times$ 出现次数 ，块内出现次数是固定的，因此只需要求块内 vr 的前缀和。
复杂度线性!