字符串哈希与 KMP 扩展

双哈希 $O(1)$ 判子串相等 $+$ 前缀函数 π 不回退线性匹配

HGNU-ACM 实验室算法组

配套：多项式滚动前缀哈希 $+$ KMP 前缀函数 π 与最小循环节 $+$ 4 道经典真题逐题精讲





本章目录

字符串哈希：把字符串变成大整数

KMP 前缀函数 π 与最小循环节

第 1 节学习字符串哈希：把字符串看成 B 进制大整数，用前缀作差 $O(1)$ 比较任意子串；第 2 节学习 KMP 的前缀函数 π、最小循环节以及扩展 KMP 简介。

本章目录（续）

例题精讲（一）：字符串哈希实战

例题精讲（二）：KMP 经典真题对比

常见错误、检查清单与练习安排

第 3、4 节逐题精讲《图书管理》《Oulipo》《剪花布条》《Power Strings》，第 5 节总结高频错误、六必查清单与练习安排。

先想一个生活场景：怎么一秒认出两本书名是不是同一本？

从图论编号到字符串指纹

上一篇《强连通分量与割点桥》里，我们把一张复杂的图缩成几个“编号”来比较；今天面对长达几万字的字符串，能不能也给每段文字算一个数字指纹？

- 假设图书馆有 3×10^4 本书，每本书名最长 200 个字符，每次查询都要问：“书架上有没有 `Effective Java`?”
- 如果每次把书名从第 1 个字母逐一比到最后一个字母，太慢了。
- 如果能把 `"abc"` 变成一个整数（比如 123），比较两段文字相不相等，就变成比较两个整数 `h1 == h2`——只需 1 次运算！

本页要点

字符串哈希（Hash）大白话就是：给每个字符串算一个“数字身份证号”，比串变成比整数。

为什么不能暴力逐字符比较？操作次数算给你看

- 问题：给你一个长度为 $n = 10^5$ 的字符串 S，接下来有 $q = 10^5$ 次询问：每次问子串 $S[l_1..r_1]$ 和 $S[l_2..r_2]$ 是否完全相同？
- 暴力做法：每次用 `for` 循环从头到尾逐个字符对比，单次最多对比 10^5 个字符。
- 总操作次数：

$$10^5 \text{ 次询问} \times 10^5 \text{ 字符} = 10^{10} \text{ 次运算}$$

- C++ 1 秒大约能做 10^8 次基本运算，10^{10} 次需要约 100 秒，必然 **TLE**（超时）！

写之前先回答三个问题

1. 怎么把字母变成数字？ 2. 怎么从整串快速截出子串的数字？ 3. 不同的串算成同一个数字怎么办？

本页要点

暴力比较单次 $O(n)$ 导致总时间 $O(qn)$ 超时；哈希的目标是预处理 $O(n)$，每次查询 $O(1)$。

核心思想：把字符串看成一个 B 进制的大整数

- 在十进制里，数字串 `"123"` 的值是 $1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 123$。
- 同理，把底数 10 换成质数 $B = 131$（或 13331），把字母映射成数字（如 `'a'`$=1$, `'b'`$=2$, `'c'`$=3$），就能算出 `"abc"` 的数值：

$$H(\texttt{"abc"}) = 1 \times 131^2 + 2 \times 131^1 + 3 \times 131^0 = 17161 + 262 + 3 = 17426$$

- 写代码时不用真的算幂，像十进制读数一样一位位往后推：新值 = 旧值 $\times B$ + 当前字符。
- 因为 131^{100000} 早就超过 `long long` 上限，每一步都要对一个大质数 M（如 $10^9 + 7$）取模。

本页要点

多项式滚动哈希就是把字符串当成 B 进制整数来求值，边乘 B 边对大质数 M 取模。

千万别踩的坑：为什么字符绝对不能映射成 0？

错误映射：'a' -> 0

"a" $= 0$

"aa" $= 0 \times 131 + 0 = 0$

"aaa" $= 0 \implies$ 全撞车！

正确映射：'a' -> 1（或 ASCII 码）

"a" $= 1$

"aa" $= 1 \times 131 + 1 = 132$

"aaa" $= 132 \times 131 + 1 = 17293$

- 如果把 'a' 算成 0，就相当于十进制里的前导零：0、00、000 数值完全一样！
- 实操最省心的做法：直接用字符的 ASCII 码（'a'=97, 'A'=65, 空格 =32），它们全都 ≥ 1，绝不会出现 0。

本页要点

映射规则必须保证所有有效字符对应的数值 ≥ 1（推荐直接用 unsigned char 的 ASCII 码）。

前缀哈希递推：手算 "abc" 的前缀表

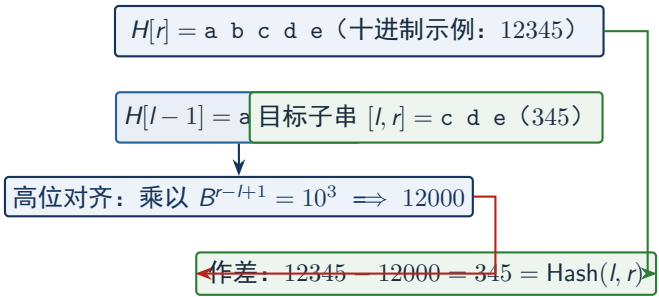
- 我们用 1-based 下标存前缀哈希：$H[0]=0$，$H[i]$ 表示前 i 个字符组成的整数（取模 M）。
- 递推公式：$H[i]=(H[i-1]\times B+\mathrm{val}(s[i])) \bmod M$。
- 顺便预处理幂次数组：$pw[0]=1$，$pw[i]=(pw[i-1]\times B) \bmod M$。

位置 i	字符	前缀串	$H[i]$ 的计算（以 $B=10$, a=1,b=2,c=3 演示）
0	空	""	$H[0]=0$
1	'a'	"a"	$H[1]=0\times 10+1=1$
2	'b'	"ab"	$H[2]=1\times 10+2=12$
3	'c'	"abc"	$H[3]=12\times 10+3=123$

本页要点

前缀哈希表 $H[i]$ 就像前缀和数组：从左到右 $O(n)$ 扫一遍，就能记住每个前缀的指纹。

图解前缀相减：怎么 $O(1)$ 切出任意子串 $[l, r]$？



本页要点

切出区间 $[l, r]$ 的公式：$\text{Hash}(l, r) = \big(H[r] - H[l-1] \times B^{r-l+1}\big) \bmod M$，把左前缀左移对齐后再减！

手算跟踪表：用 "abca"（$B = 10, M = 10^9 + 7$）追踪求值与查重

设 'a'=1，'b'=2，'c'=3，判断 $S[1..1]$ 与 $S[4..4]$、以及 $S[2..4]$ 的哈希值：

步骤	当前操作	$H[i]$ / 结果	$pw[i]$	说明
1	初始化 $i = 0$	$H[0] = 0$	$pw[0] = 1$	空前缀哈希为 0
2	读入 $s[1] =$ 'a'	$H[1] = 1$	$pw[1] = 10$	$0 \times 10 + 1 = 1$
3	读入 $s[2] =$ 'b'	$H[2] = 12$	$pw[2] = 100$	$1 \times 10 + 2 = 12$
4	读入 $s[3] =$ 'c'	$H[3] = 123$	$pw[3] = 1000$	$12 \times 10 + 3 = 123$
5	读入 $s[4] =$ 'a'	$H[4] = 1231$	$pw[4] = 10000$	终止预处理：$i = 4$ 到达串尾
6	查 $[2, 4]$（长 3）	231	用 $pw[3] = 1000$	$H[4] - H[1] \times 1000 = 1231 - 1000 = 231$
7	查 $[4, 4]$（长 1）	1	用 $pw[1] = 10$	$1231 - 123 \times 10 = 1 = \mathrm{Hash}(1, 1)$

本页要点

子串长度是 $len = r - l + 1$，所以乘的幂次永远是 $pw[r - l + 1]$，不要写成 $pw[r]$ 或 $pw[l]$。

为什么会减出负数？C++ 取模的“坑”与修正

- 由于 $H[r]$ 和 $H[l-1] \times pw[len]$ 都对 M 取过模，完全可能出现 $H[r] = 5$ 而 $H[l-1] \times pw[len] \bmod M = 9000$！
- 这时相减得到 -8995。在 C++ 里，`(-8995) % M` 的结果还是负数 `-8995`！
- 而同一段子串如果出现在串首，算出来的哈希是正数，两者一比就不相等了（**WA**）。

标准防负写法

```
long long ans = (h[r] - h[l - 1] * pw[r - l + 1] % M + M) % M;
```

本页要点

只要做减法取模，必须先 `% M` 把负数压进 $(-M, 0)$，再 `+ M`、再 `% M`！

为什么单模哈希会被卡？生日悖论与双哈希

- 什么是哈希碰撞？两个不同的字符串，对 $M = 10^9 + 7$ 取模后算出了同一个余数，程序就会误判它们相等！
- 生日悖论：一个班只要有 23 个人，就有超过 50% 的概率两人生日相同。同理，模数 $M \approx 10^9$ 时，只要对比超过 $\sqrt{M} \approx 3 \times 10^4$ 个不同子串，就极易撞车！
- 更糟的是，出题人还会专门构造数据卡掉常见的单模数（比如只用 $10^9 + 7$）。
- 解决方案——双模哈希：同时用两套不同的 (B_1, M_1) 和 (B_2, M_2) 算一对指纹 (h_1, h_2)，只有两个指纹都相等才算串相等！碰撞概率降到 10^{-18} 级别。

本页要点

比赛里除非题目卡内存，一律使用双模哈希（如 $B_1 = 131, M_1 = 10^9 + 7$ 与 $B_2 = 13331, M_2 = 10^9 + 9$）。

正确性论证：为什么前缀相减一定等于子串哈希？

步骤	区间哈希不变量论证（以未取模形式说明，取模同余性质一致）
初始化	$H[0]=0$，表示空串；对任意 $i=1$，$H[1]=s[1]$，满足前缀定义。
保持	若 $H[i-1]=\sum_{k=1}^{i-1} s[k]B^{i-1-k}$，乘以 B 再加 $s[i]$ 后，所有旧字符幂次集体 $+1$，正好得到 $H[i]=\sum_{k=1}^{i} s[k]B^{i-k}$。
终止	对 $[l,r]$，将 $H[l-1]$ 乘以 B^{r-l+1}，其每一项 $s[k]B^{(l-1-k)+(r-l+1)}=s[k]B^{r-k}$ 恰好与 $H[r]$ 的前 $l-1$ 项完全抵消，只剩 $\sum_{k=l}^{r} s[k]B^{r-k}$！

本页要点

乘以 B^{r-l+1} 的本质是把左前缀的每一位“高位对齐”到 $H[r]$ 的对应位置，然后完美相消。

模板代码：双模滚动前缀哈希（get_hash）

```cpp
const long long B1 = 131, M1 = 1000000007;       // 第一组底数与大质数模数
const long long B2 = 13331, M2 = 1000000009;     // 第二组底数与大质数模数
long long h1[N], h2[N], pw1[N], pw2[N];          // 1-based 前缀哈希与幂次表

void init_hash(const string &s) {
    int n = (int)s.size(); pw1[0] = pw2[0] = 1; // 0 次幂初值必须是 1
    for (int i = 1; i <= n; ++i) {
        unsigned char c = s[i - 1];              // 转无符号防负 ASCII 码
        pw1[i] = pw1[i - 1] * B1 % M1; pw2[i] = pw2[i - 1] * B2 % M2;
        h1[i] = (h1[i - 1] * B1 + c) % M1;       // 新值 = 旧值 * B + 当前字符
        h2[i] = (h2[i - 1] * B2 + c) % M2;
    }
}
```

本页要点

先在预处理循环里同时算好 pw[] 和 h[]，注意 pw[0] = 1 而 h[0] = 0。

模板代码（续）：$O(1)$ 查询区间 $[l, r]$ 的双哈希对

```
pair<long long, long long> get_hash(int l, int r) { // 查询 1-based 闭区间 [l, r]
    int len = r - l + 1;
    long long v1 = (h1[r] - h1[l - 1] * pw1[len] % M1 + M1) % M1;
    long long v2 = (h2[r] - h2[l - 1] * pw2[len] % M2 + M2) % M2;
    return {v1, v2};                                  // 返回一对整数指纹
}
```

为什么这样写？

- `unsigned char c = s[i - 1]`：即使遇到扩展 ASCII 字符也不会变成负数。
- `h1[l - 1] * pw1[len]` 最大约为 $10^9 \times 10^9 = 10^{18}$，在 `long long`（上限 9×10^{18}）内不会溢出。

本页要点

用 `pair<long long, long long>` 打包双哈希，可以直接放进 `set` 或用 `==` 比较。

上机验证与反事实追问

3 组上机测试数据（输入串 S 与两段 1-based 区间，输出是否相等）

1. 普通情况：$S =$ `"ababa"`，比 $[1,3]$（`"aba"`）与 $[3,5]$（`"aba"`）$\rightarrow$ 哈希值相同，输出 `YES`。
2. 最小边界：$S =$ `"a"`，比 $[1,1]$ 与 $[1,1] \rightarrow$ 输出 `YES`（测试 $l = 1$ 时读取 $H[0] = 0$）。
3. 特殊情况：$S =$ `"xyzabc"`，比 $[1,3]$ 与 $[4,6] \rightarrow$ 哈希值不同，输出 `NO`。

反事实追问

如果写 `get_hash` 时忘了乘 `pw1[len]`，直接写 `(h1[r] - h1[l - 1] + M1) % M1`，在测试 1 里会发生什么？

$\Longrightarrow$ $[1,3]$ 会算出 $H[3]$，而 $[3,5]$ 算出的却是 $H[5] - H[2]$，高位根本没对齐，结果变成 `NO`（**WA**）！

本页要点

写完哈希模板后，一定要测一次“不从下标 1 开始的相同子串”来确认 `pw[len]` 没写错。

字符串哈希高频易错对照与适用边界

错误写法	正确写法	触发的评测结果
只用单模 $M = 10^9 + 7$	用双模 (M_1, M_2) 或自然溢出加双模	撞车误判相等 $\rightarrow$ **WA**
`'a'` 映射为 0	映射为 ≥ 1（如直接取 ASCII）	`"a"` 与 `"aa"` 同为 $0 \rightarrow$ **WA**
`(h[r] - h[l-1]*pw) % M`	先 `% M` 再 `+ M` 再 `% M`	出现负余数判不等 $\rightarrow$ **WA**
`int` 存 $h \times pw$	全程使用 `long long`	10^{18} 乘法溢出成负数 $\rightarrow$ **WA**

什么时候该换相邻算法？

字符串哈希适合静态判重、子串相等、二分最长公共前缀；如果要找“模式串在长串里所有出现位置”或求“最小循环节”，用下一节的 **KMP** 更稳更快（完全零碰撞风险）！

本页要点

哈希快且好写但有极低碰撞概率；单串匹配与周期问题优先交给 KMP。

生活引入：长文章里找单词，匹配失败一定要从头退回吗？

- 假设你要在长串 "abababababc" 里找模式串 "ababc"。
- 第 1 次从头对齐比较：前 4 个字符 "abab" 全都对上了，直到第 5 个字符发现 'a' $\neq$ 'c'！
- 笨办法是把长串指针退回第 2 位重新比；但你刚看过的 "abab" 里，后两个字符 "ab" 明明和模式串开头的 "ab" 一模一样！
- 聪明的做法：长串指针根本不用倒退，只要把模式串向右滑两格，让前面的 "ab" 直接对齐，接着往后比第 3 个字符就行！

本页要点

KMP 的核心直觉：利用已经匹配过的字符信息，失配时只滑模式串，主串指针永不回头。

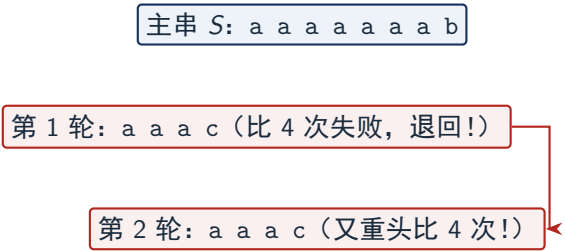
暴力匹配为何 TLE？最坏 $O(nm)$ 的构造卡法

- 设主串 S 长 $n = 10^6$，模式串 T 长 $m = 10^4$。
- 如果 $S =$ `"aaaaaaaa...ab"`（一百万个 `a`），$T =$ `"aaa...ac"`（一万个 `a` 末尾一个 `c`）：
- 每次都辛苦比完前面 9999 个 `'a'`，在最后一位 `'a'` $\neq$ `'c'` 失配；暴力做法却把 i 退回起点 $+1$，再比 9999 次！
- 总操作次数 $\approx 10^6 \times 10^4 = 10^{10}$ 次，妥妥 **TLE**！

本页要点

暴力回退之所以慢，是因为它完全忘记了刚才已经比对成功的大量前缀信息。

独占图解：暴力匹配的 $O(nm)$ 退回过程



本页要点

每轮都在最后一位失配，却把模式串只右移一格并从头重比，导致大量重复工作！

新术语大白话：真前缀、真后缀与前缀函数 pi[i]

- 真前缀：包含第一个字符、但不能是整个串自身的开头一段（如 "aba" 的真前缀有 "a", "ab"）。
- 真后缀：包含最后一个字符、但不能是整个串自身的结尾一段（如 "aba" 的真后缀有 "a", "ba"）。
- 前缀函数 pi[i]（很多教材也叫 next[i]）：把子串 $t[0..i]$ 单独拿出来，它最长的、既当真前缀又当真后缀的那一段的长度！
- 规定第 0 位没有真前后缀，所以永远有 $\mathrm{pi}[0] = 0$。

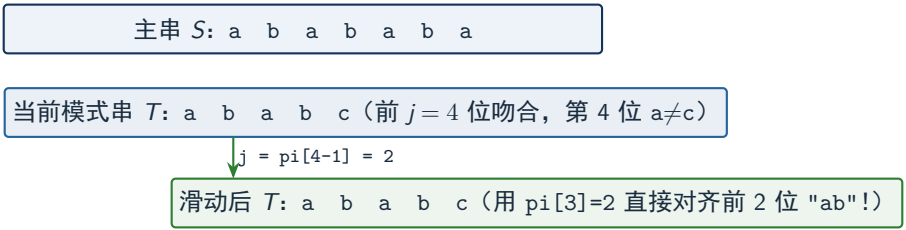
记住一句话

pi[i] 告诉你：如果在 $t[i+1]$ 处失配了，模式串最多能保留多少个开头字符继续接上！

本页要点

pi[i] 是 $t[0..i]$ 中最长的“头尾相等且不等于自身”的子串长度，范围 $0 \le \mathrm{pi}[i] \le i$。

独占图解：失配时为什么只要跳到 j = pi[j - 1]？



主串指针 i 停在原地不动，只把模式串长度 j 从 4 退回 2！

本页要点

因为刚刚匹配了 j 个字符，所以主串当前位置前面恰好有长度为 pi[j-1] 的后缀等于模式串前缀！

手算跟踪表（1/2）：求模式串 "ababa" 的 pi[] 数组

从 $i = 1$ 开始递推，令 $j = \text{pi}[i-1]$；若 $t[i] \neq t[j]$ 就让 $j = \text{pi}[j-1]$ 直到 $j = 0$ 或相等：

i	$t[0..i]$	初始 j	比较 $t[i]$ 与 $t[j]$	pi[i]	过程说明
0	"a"	—	—	0	固定起点 pi[0] = 0
1	"ab"	0	'b' $\neq$ 'a'	0	j 已为 0 无法再退，得 0
2	"aba"	0	'a' == 'a'	1	相等，j 加 1 得 1
3	"abab"	1	'b' == 'b'	2	从 $j = 1$ 接上，得 2
4	"ababa"	2	'a' == 'a'	3	终止：$i = 4$ 到达串尾（"aba" 长 3）

本页要点

求 pi[] 的过程其实就是“模式串自己和自己做 KMP 匹配”，时间复杂度也是线性 $O(m)$！

手算跟踪表（2/2）：用 "ababa" 在 "abababa" 中匹配（允许重叠）

已知 `pi = [0, 0, 1, 2, 3]`，$m = 5$，逐步跟踪主串指针 i 与已匹配长度 j：

i	$S[i]$	进循环前 j	匹配后 j	命中？	动作说明
0..3	`abab`	$0 \to 3$	4	否	连续匹配 4 个字符
4	`'a'`	4	5	第 **1** 次！	$j == 5$ 命中，允许重叠重置 `j = pi[4] = 3`
5	`'b'`	3	4	否	直接从 $j = 3$ 对比 $t[3] =$ `'b'`，吻合！
6	`'a'`	4	5	第 **2** 次！	$j == 5$ 命中，重置 `j = 3`；随后 $i = 7$ 终止循环

本页要点

主串每扫到一个字符，虽然里面有个 `while` 回退，但 j 最多加 n 次、也就最多退 n 次，总复杂度 $O(n + m)$！

正确性论证：为什么 j = pi[j-1] 绝对不会漏掉合法匹配？

步骤	KMP 匹配不变量论证
初始化	开始时 $i = 0, j = 0$，空前缀 $t[0..-1]$ 与主串当前位置前 0 个字符吻合。
保持	若 $t[0..j-1]$ 恰好等于 $S[i-j..i-1]$：假设中间存在某个更大的长度 $k > \mathrm{pi}[j-1]$ 也能对齐 S 的后缀，那么 $t[0..k-1]$ 必然也是 $t[0..j-1]$ 的真后缀，这与 $\mathrm{pi}[j-1]$ 是最长真前后缀矛盾！所以跳到 $\mathrm{pi}[j-1]$ 绝不会跳过任何合法起点。
终止	i 从 0 走到 $n-1$ 后结束；每次 j 达到 m 时即发现一个完整匹配，且立刻用 $j = \mathrm{pi}[m-1]$（或 0）保持不变量。

本页要点

势能分析：每次循环 j 最多增加 1，而每次 `while` 都会让 j 严格减小，所以总跳转次数不超过 n！

模板代码（1/2）：计算模式串的前缀函数 pi[]

```cpp
vector<int> build_pi(const string &t) {
    int m = (int)t.size();
    vector<int> pi(m, 0);                          // pi[0] 永远是 0
    for (int i = 1; i < m; ++i) {                  // 必须从 i = 1 开始！
        int j = pi[i - 1];                         // 继承前一位的最长真前后缀
        while (j > 0 && t[i] != t[j])              // 失配就退到次长真前缀
            j = pi[j - 1];
        if (t[i] == t[j]) ++j;                     // 能接上就把长度 +1
        pi[i] = j;
    }
    return pi;
}
```

本页要点

写 build_pi 牢记两点：循环从 i = 1 开始（否则 pi[-1] 越界），内部必须用 while 而非 if。

模板代码（2/2）：主串模式匹配（统计出现次数）

```cpp
int count_kmp(const string &s, const string &t, bool allow_overlap) {
    vector<int> pi = build_pi(t);
    int m = (int)t.size(), ans = 0, j = 0;
    for (char c : s) {                          // 主串指针从左到右不回退
        while (j > 0 && c != t[j]) j = pi[j - 1]; // 失配只回退模式串指针 j
        if (c == t[j]) ++j;
        if (j == m) {                           // 成功匹配整个模式串
            ++ans;
            j = (allow_overlap ? pi[j - 1] : 0); // 允许重叠退 pi[m-1]，否则清 0
        }
    }
    return ans;
}
```

本页要点

匹配成功 `j == m` 后必须重置 `j`，否则下一轮访问 `t[m]` 会数组越界触发 **RE** 或 **WA**！

上机验证与反事实追问

3 组上机测试数据（输入主串 S、模式串 T 与是否允许重叠，输出出现次数）

1. 普通情况：$S=$ `"ababa"`，$T=$ `"aba"`，允许重叠 $\to$ 输出 2；不允许重叠 $\to$ 输出 1。
2. 最小边界：$S=$ `"a"`，$T=$ `"a"`，任意模式 $\to$ 输出 1（测试 $m=1$ 时 `pi[0]=0` 正常工作）。
3. 特殊情况：$S=$ `"aaaaa"`，$T=$ `"aa"`，允许重叠 $\to$ 输出 4；不允许重叠 $\to$ 输出 2。

反事实追问

如果把 `while (j > 0 && t[i] != t[j]) j = pi[j - 1];` 误写成 `if`，会出什么错？

$\Longrightarrow$ 当回跳一次后仍然失配时，无法继续往更短的真前缀退（例如 `t = "aabaaa"` 在最后一位失配需连跳两次），导致 `pi[]` 偏小或匹配漏掉（**WA**）！

本页要点

交题前造一组全相同字符 `"aaaaa"` 配 `"aa"`，一眼就能测出重叠重置写没写对。

KMP 高频易错对照

错误写法	正确写法	触发的评测结果
`for(int i=0; i<m; ++i)` 算 `pi`	从 `i = 1` 开始，留出 `pi[0] = 0`	访问 `pi[-1]` 越界 → **RE**
`j == m` 后忘了重设 `j`	允许重叠设 `pi[m-1]`，不重叠设 0	下轮读 `t[m]` 越界或漏计数 → **RE/WA**
把 `while` 写成 `if`	必须用 `while` 连续回退	连续失配时没退到位 → **WA**
混淆 0-based 与 1-based 公式	0-based 用 `pi[j-1]`，1-based 用 `pi[j]`	偏移一位导致死循环或错位 → **TLE/WA**

本页要点

本套讲义统一使用 0-based `string` 与 `pi[]`：回退永远写 `j = pi[j - 1]`，最不易混。

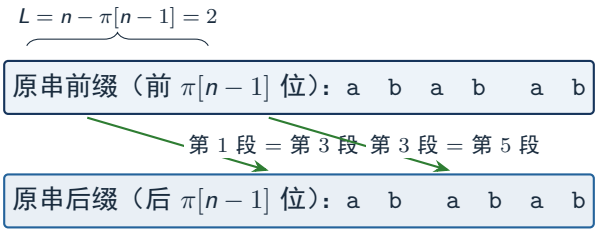
KMP 重要扩展（一）：最小循环节结论 $L = n - \pi[n-1]$

- 很多题目不直接考模式匹配，而是问：一个长为 n 的字符串，最少能看成由哪个短串重复拼接而成？
- 例如 `"ababab"`（$n = 6$），它的最小循环节是 `"ab"`（长 2），一共重复了 3 次。
- 黄金结论：令 $L = n - \pi[n-1]$。
  1. L 一定是串的最小周期（把串向右错位 L 格，重叠部分完全一致）；
  2. 如果 $n \bmod L == 0$，那么长度 L 的前缀就是最小完整循环节，最大重复次数为 n/L；
  3. 如果 $n \bmod L \neq 0$，说明末尾留了半截，完整重复次数只有 1（整个串自己）。

本页要点

记牢公式：最小周期 $L = n - \pi[n-1]$；只有当 n 能被 L 整除时，整个串才是循环节的整数倍拼接！

图解为什么 $n - \pi[n-1]$ 就是最小周期？



- 因为前 $\pi[n-1]$ 个字符等于后 $\pi[n-1]$ 个字符，相当于把串整体向右平移了 $L = n - \pi[n-1]$ 格后仍然处处相等！
- 所以每隔 L 个字符就必然一模一样；而 $\pi[n-1]$ 取最大，保证了 L 最小。

本页要点

首尾错位重叠长度越大（$\pi[n-1]$ 越大），错开的步长 $L = n - \pi[n-1]$ 就越小！

循环节手算对比："ababab" vs "ababa"

字符串	长度 n	$\pi[n-1]$	$L = n - \pi[n-1]$	$n \bmod L$	结论
"ababab"	6	4（"abab"）	$6-4=2$	0（整除）	最小循环节 "ab"，重复 3 次
"ababa"	5	3（"aba"）	$5-3=2$	1（不整除）	最小周期是 2，但不是整数次循环，答案 1
"aaaa"	4	3（"aaa"）	$4-3=1$	0（整除）	最小循环节 "a"，重复 4 次
"abcd"	4	0（无）	$4-0=4$	0（整除）	循环节只有自己，重复 1 次

本页要点

做《Power Strings》这类题，千万不能直接输出 n/L，必须先判断 `n % L == 0`！

扩展 KMP（Z 函数）简介：和 pi[] 的分工

- 什么是 **Z** 函数？对字符串 S，定义 $z[i]$ 为 S 与它的后缀 $S[i..n-1]$ 的最长公共前缀（**LCP**）长度（规定 $z[0]=0$）。
- 例如 $S=$ "aabcaab"：从下标 4 开始的后缀 "aab" 与整个串的开头 "aab" 前 3 位相同，所以 $z[4]=3$。
- 怎么算？维护一个最靠右的匹配区间 $[l,r]$（Z-box），像 Manacher 一样复用之前算过的 $z[i-l]$，总时间也是线性 $O(n)$。
- 什么时候用谁？
  - 求单串匹配、循环节、Border（前后缀相等）$\rightarrow$ 用 **KMP** 的 pi[]，代码更短更好背；
  - 求每个后缀与主串开头能匹配多长 $\rightarrow$ 用 **Z** 函数（也可把 $T+$ '#' $+S$ 拼起来跑 Z 函数实现模式匹配）。

本页要点

pi[] 看的是“以 i 结尾的前后缀”，Z 函数看的是“以 i 开头的后缀与整个串开头的公共前缀”。

例 1 | 图书管理（一本通 1456）[入门]：先读懂要求

大白话题意

图书馆一开始是空的，接下来有 $n \le 30000$ 条指令：

- `add s`：把书名为 s 的书放进图书馆；
- `find s`：查询书名 s 在不在图书馆里，在就输出 `yes`，不在输出 `no`。

要素	本题具体要求
输入	第 1 行操作数 n；接下来 n 行每行一条指令，书名 s 可能含空格（长 ≤ 200），区分大小写。
处理	维护已有书名的集合，响应每次 `find` 查询。
输出	样例 `add Inside C#` → `find Effective Java`（输出 `no`）→ `add Effective Java` → `find Effective Java`（输出 `yes`）。

本页要点

读题抓两个关键细节：操作次数最多 30000；书名里带有空格（如 `Inside C#`）。

例 1 | 图书管理：怎么想到的？

- 如果直接开 set<string>，每次插入和查找都要逐个字符比较，而且把长 200 的字符串全部存进红黑树很占空间。
- 换成双模字符串哈希：把每本整本书的书名压成一对整数 (h1, h2)，放进 set<pair<long long, long long>>！
- 比较一对整数只需要两次 ==，既省内存又快。



本页要点

这里不需要求子串，所以不用开 h[] 和 pw[] 数组，直接对整本书名从左到右滚动算出最终 (h1, h2) 即可！

例 1 | 图书管理：样例手推跟踪表

模拟原题样例的 4 条指令（用简写表示每本书算出的双哈希指纹）：

步骤	读入整行 line	切出 title	集合 seen 变化	输出
1	add Inside C#	"Inside C#"	插入指纹 H(Inside C#)	无
2	find Effective Java	"Effective Java"	集合里没有该指纹	no
3	add Effective Java	"Effective Java"	插入指纹 H(Effective Java)	无
4	find Effective Java	"Effective Java"	集合里查到该指纹	yes

本页要点

注意切书名时用的是 `line.substr(pos + 1)`，空格后面的所有字符（包括 Effective 和 Java 中间的空格）都会原样保留。

例 1 | 图书管理：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读 n 并吞掉换行	光标停在下一行开头	cin >> n 后有没有写 getline？
2. 读一行并删 ˚	干净的指令 + 书名字符串	Windows 数据行尾带 ˚ 清理了吗？
3. 找首个空格切片	指令字母 + 完整书名	用 cin >> title 会把空格后截断吗？
4. 算 (h1, h2) 查 set	插入或输出 yes/no	有没有用 unsigned char 防负值？

相对基础模板改了哪里？

基础模板预处理整串前缀 h[1..n] 查区间；本题每次只算整本书名的哈希，直接用滚动变量 h1，h2 累加即可，无需开数组。

本页要点

用 line.find(' ') 定位第一个空格，就能把指令和带空格的书名一刀切开。

例 1 | 图书管理：核心代码（1/2）读入预处理与双哈希

```cpp
int n; if (!(cin >> n)) return 0;
string line; getline(cin, line);                    // 极其关键：吃掉 n 后面的换行符！
const long long B1 = 131, M1 = 1000000007;
const long long B2 = 13331, M2 = 1000000009;
set<pair<long long, long long>> seen;
auto calc_hash = [&](const string &s) {             // 计算整本书名的双哈希指纹
    long long h1 = 0, h2 = 0;
    for (unsigned char c : s) {
        h1 = (h1 * B1 + c) % M1; h2 = (h2 * B2 + c) % M2;
    }
    return pair<long long, long long>{h1, h2};
};
```

本页要点

与 code/str_hash_books.cpp 第 7–18 行一致：getline(cin, line) 先吞掉第一行残留的回车符。

例 1 | 图书管理：核心代码（2/2）指令解析与查重输出

```
while (n-- && getline(cin, line)) {
    if (!line.empty() && line.back() == '\r')
        line.pop_back();                          // 兼容 Windows CRLF 行尾的 '\r'
    size_t pos = line.find(' ');                  // 找到指令和书名之间的第一个空格
    string title = line.substr(pos + 1);
    auto hv = calc_hash(title);
    if (line[0] == 'a') seen.insert(hv);          // add 指令：存入指纹集合
    else cout << (seen.count(hv) ? "yes" : "no") << '\n';
}
```

本页要点

与 code/str_hash_books.cpp 第 19–26 行一致：区分大小写是天然满足的，因为 'A'(65) 与 'a'(97) ASCII 码不同。

例 1 | 图书管理：易错点与 3 组自测边界

常见错误对照

- `cin >> n` 后直接进 `while (getline)`：第一次 `getline` 会读到 `n` 后面的空换行，导致最后一条指令漏读（**WA**）！
- 用 `cin >> op >> title` 读指令：遇到 `Inside C#` 只读进 `Inside`，把 `C#` 留给下一轮（**WA**）！
- 没删 `\r`：Linux 评测机读入 Windows 换行测试点时，书名末尾会多出隐藏字符 `\r`。

3 组自测边界

1. 最小：$n = 1$，`find a` $\rightarrow$ `no`；　2. 大小写：`add Book` 后 `find book` $\rightarrow$ `no`；　3. 多空格：`add a b` 与 `find a b` 应输出 `no`。

本页要点

只要题目出现“带空格的字符串”，立刻想到：`cin >> n` 后补一行 `getline` 吞回车 + 删 `\r`！

例 2 | Oulipo（一本通 1455）[基础]：先读懂要求

大白话题意

给你 T 组数据，每组给出一个短的模式串 s_1（长 $\le 10^4$）和一个长的主串 s_2（长 $\le 10^6$），都是大写字母。问 s_1 在 s_2 中一共出现了多少次（允许位置重叠）？

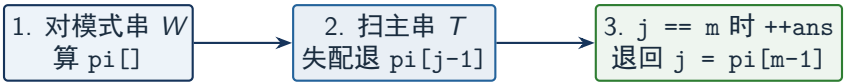
要素	本题具体要求
输入	第一行 T；每组两行分别是 s_1（模式串）和 s_2（主串）。
处理	统计 s_1 在 s_2 中允许重叠的匹配次数。
输出	样例：BAPC 在 BAPC 中出现 1 次；AZA 在 AZAAZAAZA 中出现 3 次；VEEDI 在 AVERDXIVYERDLAN 中出现 0 次。

本页要点

注意输入顺序：先读模式串 W（s_1），再读主串 T（s_2），千万别把主串和模式串传反了！

例 2 | Oulipo：怎么想到的？

- $|s_1| \le 10^4$，$|s_2| \le 10^6$，暴力枚举起点每次比对 10^{10} 次必 **TLE**。
- 标准的单模式串计数问题，直接套用 **KMP** 前缀函数 `pi[]`，时间复杂度 $O(|s_1| + |s_2|)$。
- 特别注意：题面例子 $s_1 =$ `"ABA"`，$s_2 =$ `"ABAABA"` 答案为 2；而对 $s_1 =$ `"ABA"`，$s_2 =$ `"ABABA"`，两次出现共用了中间的 `'A'`，说明允许重叠！



本页要点

允许重叠时，每匹配完一次，就把已经对齐的完整模式串当成“当前最大匹配”，回退到 `j = pi[m - 1]` 继续接后面的字符！

例 2 | Oulipo：样例手推跟踪表

以样例第 2 组 $W=$ `"AZA"`（$m=3$），$T=$ `"AZAAZAAZA"` 为例，先算 `pi = [0, 0, 1]`：

主串下标 i	$T[i]$	进循环前 j	匹配后 j	命中计数	说明
$0,1$	`AZ`	$0\to1$	2	0	连续匹配前两位 `"AZ"`
2	`'A'`	2	3	1	$j==3$ 命中！重置 `j = pi[2] = 1`
3	`'A'`	1	1	1	`'A'` $\neq W[1]$(`'Z'`)，退到 $j=0$ 再匹配 $W[0]$ 得 1
$4,5$	`ZA`	$1\to2$	3	2	第 2 次命中！重置 `j = 1`
$6..8$	`AZA`	$1\to2$	3	3	先退到 0 再配 $W[0..2]$，第 3 次命中后结束

本页要点

看下标 3：即使 `j` 重置为 1 后发现下一位不配 `Z`，`while` 循环会自动退回 0 再从 `A` 重新对齐，绝不会出错！

例 2 | Oulipo：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 W 和 T	短串 W，长串 T	有没有把 W 和 T 传反？
2. 算 W 的 `pi[0..m-1]`	长度为 m 的回跳表	循环是否从 $i = 1$ 开始？
3. 扫 T，动态维护 `j`	每次对齐的最大前缀长度	失配是否用了 `while (j > 0 ...)`？
4. `j == m` 时累加并回退	`++ans, j = pi[m-1]`	是否误清成 `j = 0`？

相对基础模板改了哪里？

直接把 `allow_overlap` 参数设为 `true`，外层加一个 `while (T--)` 读入多组测试数据即可。

本页要点

多组数据每次调用 `count_kmp` 都会重新申请局部变量 `pi` 与 `j = 0`，天然避免脏数据残留。

例 2 | Oulipo：核心代码（1/2）计算前缀函数

```cpp
int count_kmp(const string &s, const string &t, bool allow_overlap) {
    int m = (int)t.size();
    vector<int> pi(m, 0);                          // pi[i] 表示 t[0..i] 的最长真前后缀长度
    for (int i = 1; i < m; ++i) {
        int j = pi[i - 1];
        while (j > 0 && t[i] != t[j]) j = pi[j - 1];
        if (t[i] == t[j]) ++j;
        pi[i] = j;
    }
```

本页要点

与 code/kmp_oulipo.cpp 第 5–13 行完全一致：注意参数顺序是（主串 s，模式串 t，是否允许重叠）。

例 2 | Oulipo：核心代码（2/2）主串匹配与多组读入

```
    int ans = 0, j = 0;
    for (char c : s) {
        while (j > 0 && c != t[j]) j = pi[j - 1];
        if (c == t[j]) ++j;
        if (j == m) { ++ans; j = (allow_overlap ? pi[j - 1] : 0); } // 重叠退 pi[m
    -1]
    }
    return ans; }
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); int T; cin >> T;
    while (T--) { string w, t; cin >> w >> t; cout << count_kmp(t, w, true) << '\n'
    ; }
}
```

本页要点

与 code/kmp_oulipo.cpp 一致：传入 count_kmp(t, w, true)，先主串 t 后模式串 w！

例 2 | Oulipo：易错点与 3 组自测边界

常见错误对照

- 把 `w` 和 `t` 顺序传反：题目先输模式串 s_1 再输主串 s_2，传反会全输出 0（**WA**）。
- 忘记加 `ios::sync_with_stdio(false)`：主串总长可达 10^6 且有多组数据，不用快读可能 **TLE**。
- 匹配后忘记重置 `j`：下一轮读 `t[m]` 越界导致 **RE**。

3 组自测边界

1. *W* 比 *T* 长：`"ABCD"` in `"AB"` $\to 0$；　2. 重叠：`"ABA"` in `"ABABA"` $\to 2$；　3. 全相同：`"AA"` in `"AAAAA"` $\to 4$。

本页要点

记住：Oulipo 这道题的输入顺序是“先短串后长串”，写 `count_kmp(t, w, true)` 时多看一眼参数名！

例 3 | 剪花布条（一本通 1465）[基础]：先读懂要求

大白话题意

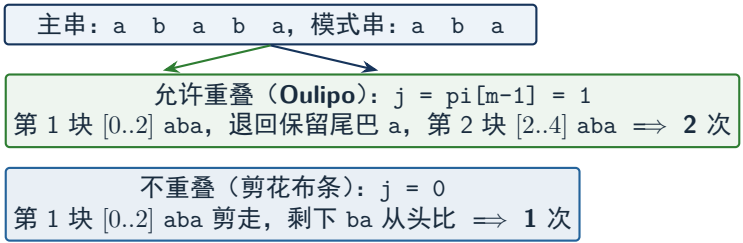
有一块大花布条（主串 S）和一块小饰条（模式串 T），长度都不超过 1000。每次从大花布条里剪下一块小饰条，剪掉的部分不能再用来拼下一块（即不允许重叠）。问最多能剪出几块？

要素	本题具体要求
输入	多组数据，每行由空格隔开的两个字符串 S 和 T；读到单独一个 `"#"` 时结束。
处理	统计 T 在 S 中互不重叠的最大出现次数。
输出	样例：`abcde a3` $\to 0$；`aaaaaa aa` $\to 3$。

本页要点

和《Oulipo》唯一的核心差异：剪下来就拿走了，所以不能重叠！

例 3 | 剪花布条：独占对比图（允许重叠 vs 剪断不重叠）



本页要点

为什么不重叠直接设 `j = 0` 就行？因为贪心从左到右能剪就剪，剪完后前面的字符全被拿走了，下一段必须从 `j = 0` 全新开始！

例 3 | 剪花布条：样例手推跟踪表

跟踪样例第 2 组：$S =$ "aaaaaa"，$T =$ "aa"（$m = 2$，不重叠模式 `j = 0`）：

i	$S[i]$	进循环前 j	匹配后 j	剪出块数	说明
0	'a'	0	1	0	匹配第 1 个 'a'
1	'a'	1	2	1	$j == 2$ 剪出第 1 块！直接清零 `j = 0`
2	'a'	0	1	1	重新从 $j = 0$ 开始，匹配第 1 个 'a'
3	'a'	1	2	2	剪出第 2 块，清零 `j = 0`
4	'a'	0	1	2	匹配第 1 个 'a'
5	'a'	1	2	3	剪出第 3 块，清零 `j = 0`，终止输出 3

本页要点

如果这里错写成 `j = pi[1] = 1`，第 2 个字符开始每次都会命中，算出 5 块（**WA**）！

例 3 | 剪花布条：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 S	若 $S ==$ "#" 退出	注意必须是整个 S 等于 "#"
2. 再读入 T	本组模式串 T	不要在读 T 之前就判结束
3. 调用 `count_kmp(S, T, false)`	匹配后 `j = 0`	确认传入了 `false` 不重叠
4. 输出次数	每组一行整数	多组数据换行别漏

相对《Oulipo》改了哪里？

1. 读入条件改成 `while (cin >> s && s != "#") { cin >> t; ... };`
2. 调用 `count_kmp(s, t, false)`，主串先、模式串后，且关闭重叠！

本页要点

只需一个布尔开关 `allow_overlap`，同一份 KMP 模板就能同时解决重叠与不重叠两类题。

例 3 | 剪花布条：核心代码（调用与终止判断）

```cpp
// 直接复用前面写好的 count_kmp(s, t, false)
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    string s, t;
    while (cin >> s && s != "#") {              // 只有整个字符串等于 "#" 才结束
        cin >> t;                               // 再读小饰条 t
        cout << count_kmp(s, t, false) << '\n'; // 不重叠：匹配成功后 j 直接归 0
    }
    return 0;
}
```

本页要点

注意原题提醒：“若某字符串开头有 #，不意味着读入结束”，写 `s != "#"` 比写 `s[0] != '#'` 更准确！

例 3 | 剪花布条：易错点与 3 组自测边界

常见错误对照

- 写 `if (s[0] == '#') break;`：如果某组花布条刚好叫 `"#abc"`，会提前错误退出（**WA**）！
- 匹配成功还是写了 `j = pi[j - 1]`：把剪掉的布条又拿回来凑下一块（**WA**）。

3 组自测边界

1. 含 # 开头：`#a#a #a` $\rightarrow$ 应输出 2，不能直接退出；
2. 不可重叠：`ababa aba` $\rightarrow$ 应输出 1；
3. 无匹配：`abc def` $\rightarrow$ 应输出 0，随后输入 # 正常结束。

本页要点

把《Oulipo》和《剪花布条》放一起练，彻底分清 `j = pi[m-1]` 与 `j = 0` 的适用场景。

例 4 | Power Strings（一本通 1457）[提高]：先读懂要求

大白话题意

给定若干行字符串 S（长度 $\le 10^6$），问每个字符串最多能由多少个相同的子串重复连接而成？例如 `"ababab"` 由 3 个 `"ab"` 拼成，输出 3。输入单个句号 `"."` 结束。

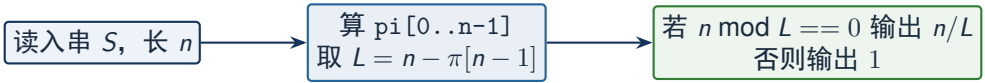
要素	本题具体要求
输入	多行字符串，以单独一行 `"."` 作为结束标志。
处理	对每个长度为 n 的串，求最大正整数 k，使得 $S = A^k$（A 重复 k 次）。
输出	样例：`abcd` $\to 1$；`aaaa` $\to 4$；`ababab` $\to 3$。

本页要点

因为 $n \le 10^6$，如果枚举每个长度 L 再暴力验证，虽然因子不多但仍不如直接用 KMP 一步到位 $O(n)$ 稳妥。

例 4 | Power Strings：怎么想到的？

- 要让重复次数 $k = n/L$ 最大，等价于找最短的完整循环节长度 L。
- 由第 2 节学的 KMP 结论：最小周期候选就是 $L = n - \pi[n-1]$！
- 若 $n \bmod L == 0$，则 S 一定恰好由 n/L 个长为 L 的串拼成；否则无法拼成整数倍，只能输出 1。



本页要点

整道题甚至不需要主串匹配那半段代码，只要跑一遍 `build_pi(s)` 看最后一位 `pi[n-1]` 就够了！

例 4 | Power Strings：样例手推跟踪表

逐一验证原题 3 组样例：

输入串 S	长度 n	$\pi[n-1]$	$L = n - \pi[n-1]$	$n \bmod L == 0$?	输出 k
`"abcd"`	4	0	$4-0=4$	是（$4 \bmod 4 = 0$）	$4/4=1$
`"aaaa"`	4	3	$4-3=1$	是（$4 \bmod 1 = 0$）	$4/1=4$
`"ababab"`	6	4	$6-4=2$	是（$6 \bmod 2 = 0$）	$6/2=3$
（额外）`"ababa"`	5	3	$5-3=2$	否（$5 \bmod 2 = 1$）	1

本页要点

为什么 `n % L != 0` 时答案一定是 1？因为如果存在更短的完整循环节 d 整除 n，那 d 也必然是周期，就会推出 $L \le d$，而 L 是最小周期不可能再拆出更小循环节！

例 4 | Power Strings：核心步骤表与模板改动

按顺序计算	纸上得到	核对问题
1. 读入 s	若 $s ==$ "." 结束	读入总长可达 10^6，开了快读吗？
2. 计算 `pi = build_pi(s)`	得到 `pi[n - 1]`	是否确保了 $n \geq 1$？
3. 计算 `L = n - pi[n - 1]`	最小周期长度 L	是减 `pi[n-1]` 还是 `pi[n]`？
4. 判整除输出	`n % L == 0 ? n / L : 1`	有没有忘记判整除？

相对基础模板改了哪里？

只保留 `build_pi(s)`，读取最后一个元素 `pi.back()` 后用一行三目表达式出答案。

本页要点

记住：在 0-based 下标下，最后一个位置的下标是 `n - 1`，所以用 `n - pi[n - 1]`。

例 4 | Power Strings：核心代码

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr); string s;
    while (cin >> s && s != ".") {                  // 读到单独一个句号 "." 结束
        int n = (int)s.size(); vector<int> pi(n, 0);
        for (int i = 1; i < n; ++i) {
            int j = pi[i - 1];
            while (j > 0 && s[i] != s[j]) j = pi[j - 1];
            if (s[i] == s[j]) ++j; pi[i] = j;
        }
        int L = n - pi[n - 1];                      // 最小周期长度
        cout << (n % L == 0 ? n / L : 1) << '\n';
    }
}
```

本页要点

核心代码只有 13 行，时间复杂度 $O(\sum |S|)$，跑 10^6 长度不到 20 毫秒！

例 4 | Power Strings：易错点与 3 组自测边界

常见错误对照

- 直接输出 `n / L` 不判整除：遇到 `"ababa"` 会算出 $5/2 = 2$（**WA**）！
- 开静态数组 `pi[100005]` 没开够：本题串长最大 10^6，开小会触发 **RE**，用 `vector<int> pi(n, 0)` 最安全。

3 组自测边界

1. 最小单字符：`"a"` $\to n = 1, \pi[0] = 0, L = 1 \implies 1$；
2. 不整除半周期：`"abaababaab"`（长 10，$\pi[9] = 5 \implies L = 5 \implies 2$）vs `"abaababa"`（长 8，$\pi[7] = 3 \implies L = 5 \implies 1$）；
3. 全同串：`"zzzzz"` $\to 5$。

本页要点

只要看到题目问“最小循环节”“最多重复几次”，条件反射写出 `L = n - pi[n - 1]`！

字符串哈希与 KMP 高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
单模哈希遭卡碰撞	只用 1 个模数 10^9+7	数据量大极易冲突 **WA**，改用双模哈希 `pair<h1, h2>`
字符映射成 0	令 `'a' -> 0`	`"a"` 与 `"aa"` 哈希同为 0，改为 ASCII 码或 ≥ 1
子串哈希减出负数	算 `(h[r] - h[l-1]*pw) % M`	C++ 负数取模仍为负，写成 `((...) % M + M) % M`
`cin` 后接 `getline`	读完 *n* 直接 `getline`	先写一句 `getline` 吞掉换行，并清理末尾 `'\r'`
匹配成功忘重置 `j`	`j == m` 后放任越界或错归零	允许重叠重置 `j = pi[m-1]`，不重叠重置 `j = 0`
循环节忘记判整除	直接输出 `n / (n - pi[n-1])`	必须先判 `n % L == 0`，否则输出 1

本页要点

“双模防碰撞，减法加模再取模，`j == m` 必重置，读整行先吞回车”。

评测结果定位指南（CE / WA / RE / TLE）

评测结果	本章高频排查顺序
CE	检查 `pair<long long, long long>` 括号及 C++17 头文件 `<bits/stdc++.h>` 是否拼写正确。
WA	1. 哈希是否只用单模或减法漏 `+ M`；2. `getline` 前是否漏吞回车或没删 `\r`；3. 重叠 (`pi[m-1]`) 与不重叠 (0) 是否用反；4. Oulipo 主串模式串是否传反。
RE	1. `build_pi` 是否从 `i=0` 开始访问了 `pi[-1]`；2. `j == m` 后忘记重置 `j` 导致下轮读 `t[m]` 越界；3. 10^6 长度串的静态数组开小。
TLE	1. 是否用了暴力 $O(nm)$ 对比没用 KMP/哈希；2. 10^6 读入是否忘了 `ios::sync_with_stdio(false)`；3. KMP 回跳是否误写成死循环。

本页要点

遇到 **WA** 先测“带空格/\r”和“全同字符 aaaaa”；遇到 **RE** 先查 `j == m` 和数组大小。

提交前检查清单（六必查）

1. 字符串哈希是否使用了双模数（如 10^9+7 与 10^9+9），且避免把有效字符映射成 0？
2. 求子串区间哈希相减时，有没有先乘 `pw[r - l + 1]` 再用 `(val % M + M) % M` 转为非负？
3. 混合使用 `cin >> n` 与 `getline` 时，中间是否吃掉了行末换行且清理了 `'\r'`？
4. KMP 求 `pi[]` 是否从 `i = 1` 开始（留出 `pi[0] = 0`），且回退用的是 `while`？
5. 主串匹配成功（`j == m`）时，有没有按“允许重叠 `pi[m-1]` / 不重叠 `0`”正确重置 `j`？
6. 利用 KMP 求最小循环节时，有没有先判断 `n % (n - pi[n-1]) == 0` 再输出 `n / L`？

最后一步

用全相同字符 `"aaaaa"` 配 `"aa"`，验证允许重叠得 4、不重叠得 2 后再提交！

本页要点

六条全部打勾再交题，能挡住 95% 的低级失误。

本章全部练习与完成标准

题目（一本通题号）	难度	核心思路提要
图书管理（1456）	[入门]	吞换行后整行读入，双模哈希压成 (h1, h2) 查重
收集雪花（1464）	[基础]	双指针维护无重复滑动窗口，哈希表或 set 判重
Oulipo（1455）	[基础]	KMP 统计出现次数，允许重叠置 j = pi[m-1]
剪花布条（1465）	[基础]	KMP 不重叠计数置 j = 0，整串 "#" 才结束
Power Strings（1457）	[提高]	最小周期 $L = n - \pi[n-1]$，整除则 n/L 否则 1

完成标准

能默写双哈希与 KMP pi[]；分清重叠与不重叠重置；独立 AC 上述 5 题。

本页要点

先把 code/ 下两份代码亲手敲一遍并造极端样例跑通，再去 OJ 刷剩余练习。

课后安排与下一章预告

- 复习：合上讲义，手算 "ababab" 的 π 数组，并用 $n-\pi[n-1]$ 验证最小循环节是 "ab"。
- 练习：独立默写 code/str_hash_books.cpp 与 code/kmp_oulipo.cpp，并把 "#abc" 测入剪花布条。
- 下一章预告：第 2 章《Trie 树与 AC 自动机》——KMP 一次只能找 **1** 个模式串，如果要在长文章里同时查找一万个敏感词怎么办？把 KMP 的 pi[] 搬到字典树上，变成多模式一次扫描的 AC 自动机！

本章一句话

哈希把子串压成一对整数实现 $O(1)$ 查重，KMP 借 π 数组让失配指针知错能改绝不走回头路。

谢谢！