Shift Game 题解

znzryb

April 24, 2026

Contents

1 题目信息

- **题目名称**：Shift Game
- **来源**：CodeChef Starters 235
- **链接**：codechef.com/problems/SHFTGAME

2 题意解读

给定两个长度为 N 的 01 串 A、B。从分数 0 开始，每一次迭代：

1. 把 $A_1 \cdot B_1$ 累加到分数上（A_1、B_1 表示当前两串的首字符）；
2. 若 $A_1 = B_1$，把 A 替换为 left_shift(A)（首字符删掉并追加到末尾）；
3. 否则把 B 替换为 left_shift(B)。

有 Q 次独立询问，每次给出 K，问恰好执行 K 次迭代后的分数。

数据范围：$1 \le T \le 10^4$，$1 \le N, Q \le 10^6$（所有用例的 $\sum N$、$\sum Q$ 各自不超过 10^6），$1 \le K \le 10^{12}$。时限 3 秒。

关键观察一下约束：K 高达 10^{12}，逐步模拟不可行；必须挖出结构做 $O(\log K)$ 级别的单次询问。

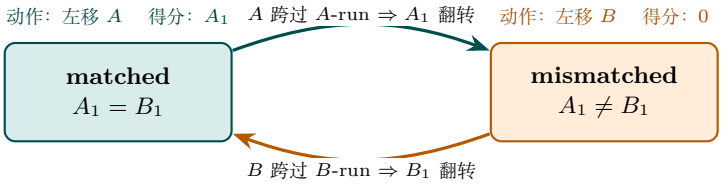
3 性质分析

3.1 两态状态机：matched / mismatched

已知：每步分数增量 $= A_1 \cdot B_1 \in \{0, 1\}$；规则"$A_1 = B_1$ 左移 A，否则左移 B"——"谁前进"完全绑定在 A_1, B_1 的匹配关系上。

要求：把"何时得分 / 何时移 A / 何时移 B / 何时翻转"合成一张状态机骨架——后面 phase 划分、周期分析、零头分类，全部从这张骨架长出。

如何实现：按 A_1 与 B_1 是否相等，过程天然切成两态——**matched** ($A_1 = B_1$) 与 **mismatched** ($A_1 \neq B_1$)——两态在 run 边界处交替切换：



图注：系统的两态状态机。左态 matched、右态 mismatched；每个状态内标注动作与本步得分，箭头上标切换条件（run 边界）。得分只在 matched 且 $A_1 = 1$ 时发生。

把上面图信息表化以便查表：

状态	条件	动作	本步得分	切换到另一态
matched	$A_1 = B_1$	左移 A	$A_1 \in \{0, 1\}$	A 跨过 A-run（A_1 翻转）
mismatched	$A_1 \neq B_1$	左移 B	0	B 跨过 B-run（B_1 翻转）

关键结论：$A_1 \cdot B_1 = 1 \iff A_1 = B_1 = 1$，所以 **得分 = matched 状态下且 $A_1 = 1$ 的步数之和**；matched 段内每步都在左移 A，mismatched 段内每步都在左移 B、零得分。这就把"算分数"变成了"统计 matched 段内 A 指针的带权前进步数"。

记号约定（下文一直沿用）：$c_A := A_1$ 初始数值、$c_B := B_1$ 初始数值，都是 $\{0, 1\}$ 中的**常量**（代码里 `A[0]`、`B[0]` 就是它俩——因为实现是指针模拟、从不真修改字符串）。

3.2 Phase：一个 matched 段 + 一个 mismatched 段

已知：由 §3.1，时间轴被 run 边界切成交替的 matched / mismatched 段；matched 段长 = len(当前A-run)，mismatched 段长 = len(当前B-run)。

要求：找一个**长度确定、可加**的分解单位，使总步数、累计分数、两串指针推进量都能按这个单位逐份累加，并且**每份的内部结构都一致**——这是后面二分 + 零头 $O(1)$ 闭式的根基。

如何实现：把**相邻的一个 matched 段 + 一个 mismatched 段**合成一个 **phase**（起始匹配态决定谁先谁后）。每个 phase 恰好吃掉 A 的 1 个 run 和 B 的 1 个 run，步数 = len(A-run) + len(B-run)，**与起始匹配态无关**。后文的循环变量 P 就是”已走完的 phase 数”。

量	每个 phase 内取值
A 跨过的 A-run 数	恰好 1 个
B 跨过的 B-run 数	恰好 1 个
A 指针累计移动	= len(当前A-run)
B 指针累计移动	= len(当前B-run)
本 phase 步数	= len(A-run) + len(B-run)
phase 首尾的匹配关系	保持不变（A_1, B_1 各翻 1 次，相对关系还原）
内部子结构（谁先谁后）	由 $c_A \overset{?}{=} c_B$ 决定，**对所有 phase 一致**

关键不变量（后面零头分类直接用）：**每个 phase 的内部子结构都和第 1 个一样，只需判一次 $c_A \overset{?}{=} c_B$ 即可**。原因：每过一个 phase，A_1 和 B_1 各翻 1 次，相对关系还原；phase 起点的 $A_1 \overset{?}{=} B_1$ 就等价于初始常量 $c_A \overset{?}{=} c_B$。

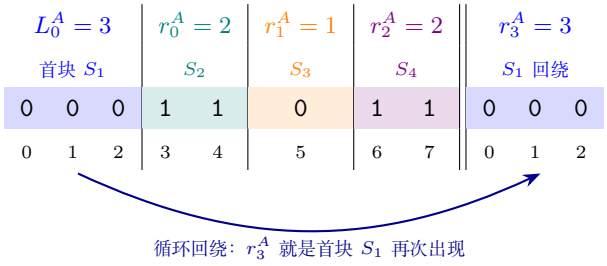
附记：这里的 phase 定义与 AC 代码中的循环变量 P 严格对齐（即 $f_A(P), f_B(P)$ 分别给出走完 P 个 phase 后 A, B 的累计偏移）。

3.3 跳过首块后的周期性

已知：由 §3.2，每个 phase 吃 1 个 A-run + 1 个 B-run；初始 A 的首块长度记为 $\boldsymbol{L_0^A}$——与 A_1 同字符的极长前缀长度，它是”残缺”的（前面 A 已被部分消耗）。

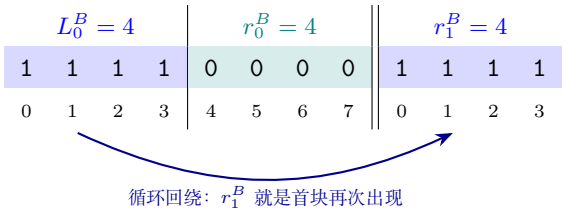
要求：把 A 的所有 run（首块 L_0^A 加上之后完整的 run 序列）编成 $O(1)$ 可查结构（数组 + 前缀和），支持”第 P 个 phase 对应哪段 A-run”、”前 P 个 phase 的 A 指针累计偏移 / 累计得分”等询问。

如何实现：跳过首块后，从位置 $L_0^A \bmod N$ 开始把 A 当**循环串**扫一圈，得到完整 run 长度序列 $\boldsymbol{r_0^A, r_1^A, \ldots, r_{k_A-1}^A}$（共 $\boldsymbol{k_A}$ 段）；**段长总和 $= N$，之后按段数 A 为周期重复**。以 $A =$ 00011011（$N = 8$）为例，A 自然分成四个极长同色段 S_1, S_2, S_3, S_4（与第二个串 B 无关，只是 A 内部的 run），其中 $S_1 =$ 000 是首块、$L_0^A = 3$；从位置 L_0^A 开始循环读一圈依次得到 S_2, S_3, S_4, S_1——最后一段回绕到首块。示意图中串复制了一份，双竖线右侧是”绕回首块”的部分：



图注：以 $A =$ 00011011、$N = 8$ 为例。首块 $S_1 =$ 000 单独拎出，长度记 $L_0^A = 3$（蓝）；跨过首块后循环读一圈得到 $r^A = (2, 1, 2, 3)$，依次对应 A 的第 2、3、4 段 S_2, S_3, S_4（绿、橙、紫）。**双竖线右侧的 $r_3^A = 3$ 与首块 S_1 同色（蓝）**，因为它们是物理上的同一段 run——循环读到末尾绕回 pos 0–2，重新进入首块。全部段长之和 $\sum_i r_i^A = 2+1+2+3 = 8 = N$，段数 $k_A = 4$。S_i 只是 A 内部的 run 编号，**和串名 B 无关**。

B 侧同理——以 $B =$ 11110000（$c_B = 1$，$L_0^B = 4$）为例，跨过首块后的循环 run 序列只剩两段：



图注：$B = 11110000$、$N = 8$。首块 1111 长度 $L_0^B = 4$（蓝）；跨过首块后循环读一圈得到 $r^B = (4, 4)$——r_0^B 对应 pos 4–7 的 0000（绿），r_1^B 与首块同色（蓝）因为它就是 pos 0–3 的 1111 回绕再现。段数 $k_B = 2$。

一般来说段数 $A \neq$ 段数 B（本例段数 $A = 4$，段数 $B = 2$）——所以后面 $f_A(P)$ 用自己的段数 A 和 pref_a、$f_B(P)$ 用自己的段数 B 和 pref_b，两套预处理彼此独立；同一个 P 在 A 侧和 B 侧走过的圈数、零头段号可以完全不同。

P 与段的对应关系：**phase 1 吃掉的 A-run 是残缺首块 L_0^A**（初始状态 A 恰好从首块内部起步）；**phase $2, 3, \ldots, k_A + 1$ 依次吃掉 $r_0^A, r_1^A, \ldots, r_{k_A-1}^A$**，之后按段数 A 为周期绕圈。B 侧同构。这就是下节 $f_A(P)$ 里" 首块 + 整圈 + 零头" 的由来——" 首块" 项 L_0^A 对应 phase 1 那份残缺。

段长前缀和（预处理一次、后面反复查）：

数组	含义	定义
pref_a[r]	A 侧前 r 段完整 run 长度之和（不含首块）	$\sum_{i=0}^{r-1} r_i^A$
pref_b[r]	B 侧前 r 段完整 run 长度之和（不含首块）	$\sum_{i=0}^{r-1} r_i^B$

边界：pref_a[0] = pref_b[0] = 0；整圈正好等于串长 pref_a[k_A] = pref_b[k_B] = N。预处理 $O(N)$ 一次扫完，查询 $O(1)$。

3.4 每段 A 的分数率

已知：c_A 已于 §3.1 定义为 A_1 初始数值；§3.1 的 matched 段内每步贡献 A_1 分；§3.2 的每个 phase 吃 1 个 A-run，phase 间 A_1 翻转一次。

要求：给 A 的每一段 run（首块 L_0^A 与之后的 $r_0^A, r_1^A, \ldots, r_{k_A-1}^A$）定义一个**分数率 rate$_i$**，使得" 该段落入某 phase 的 matched 子段时，A 指针每前进一步贡献 rate$_i$ 分"——如此按段累加 $r_i^A \cdot \text{rate}_i$ 就是累计得分。

如何实现：r_0^A 是**跳过首块后**的第一段，此时 A 首字符已翻转为 $1 - c_A$；此后每跨一段再翻一次。rate$_i$ 直接取当段 A_1 值：

所在段	A_1 值	分数率 rate
首块 L_0^A	c_A	c_A
r_0^A（跳首块后第 1 段）	$1 - c_A$	$1 - c_A$
r_1^A	c_A	c_A
r_2^A	$1 - c_A$	$1 - c_A$
$\vdots$	$\vdots$	$\vdots$
r_i^A	偶 i: $1 - c_A$；奇 i: c_A	同 A_1 值

至此" 算分数" 被分解为：首块部分 $L_0^A \cdot c_A$ + 按段累加 $r_i^A \cdot \text{rate}_i$。一整圈（$k_A$ 段）A 的总得分记为 $W_A = \sum_{i=0}^{k_A-1} r_i^A \cdot \text{rate}_i$，后面按周期整段跳时反复用到。

得分前缀和（和 §3.3 的 pref_a/pref_b 一起，构成 $O(1)$ 查询所需的全部预处理）：

$$\text{pref\_score\_A}[r] := \sum_{i=0}^{r-1} r_i^A \cdot \text{rate}_i$$

边界：pref_score_A[0] = 0，pref_score_A[k_A] = W_A。

4 思路推导

4.1 暴力的瓶颈

已知：题目要对 K 步独立询问即时得到分数；$K \leq 10^{12}$，单组测试 Q 次询问，$\sum N, \sum Q \leq 10^6$。

要求：在 3 秒内回答所有询问。

如何实现：按题意逐步模拟，每步 $O(1)$ 更新两串首字符指针。单次询问 $O(K)$——**K 可至 10^{12}，一次就炸**。即使预处理到某上界的前缀和，也无法覆盖所有可能的 K。暴力不行，必须把" 每次一步" 换成**" 每次跳一段"**。`brute.cpp` 保留这份模拟，仅用于小数据对拍。

4.2 按 phase 一次跳一整个 run 对

已知：由 §3.2，每个 phase 吃 A 的 1 个 run + B 的 1 个 run；跨过首块 L_0^A 后 A 的 run 长按 $r_0^A, \ldots, r_{k_A-1}^A$ 周期循环，一圈 k_A 段走 N 步、贡献 W_A 分（B 侧对称）。

要求：对任意 P，$O(1)$ 算出走完 P 个 phase 后 A 指针偏移 $\boldsymbol{f_A(P)}$、B 指针偏移 $\boldsymbol{f_B(P)}$、累计分数 $\boldsymbol{g_A(P)}$。

如何实现：套 **" 首块 + 整圈 + 余数" 三段式**。下面先把 P 个 phase 在 A、B 两侧各自拆解成" 首块 + 若干整圈 + 零头段"，再对三个函数逐个推导。

Step 1：把 P 个 phase 拆成" 首块 + 周期部分"

由 §3.2，phase 1 是特殊的——它吃掉的是 A 的残缺首块 L_0^A（同时吃掉 B 的残缺首块 L_0^B）；从 phase 2 起才进入完整 run 序列，A 侧按 $r_0^A, r_1^A, \ldots$ 以段数 A 为周期循环，B 侧按 $r_0^B, r_1^B, \ldots$ 以段数 B 为周期循环。

所以走完 P 个 phase（$P \geq 1$）等价于：" 先走 phase 1 吃掉首块，再把 $P-1$ 个 phase 在 run 序列里跑完"。把这 $P-1$ 个 phase 按 A 侧周期拆分：

$$P-1 \;=\; \boldsymbol{q_A} \cdot k_A + \boldsymbol{r_A}, \qquad q_A = \lfloor (P-1)/k_A \rfloor, \;\; r_A = (P-1) \bmod k_A$$

即 A 侧看到"q_A 整圈 + 零头 r_A 段"。B 侧结构对称，只是周期换成 k_B：

$$P-1 \;=\; \boldsymbol{q_B} \cdot k_B + \boldsymbol{r_B}, \qquad q_B = \lfloor (P-1)/k_B \rfloor, \;\; r_B = (P-1) \bmod k_B$$

两侧用各自的周期分别拆分——这正是 §3.3 强调" 段数 A 与段数 B 一般不相等、两套预处理彼此独立" 的用武之地。同一个 P 可能 A 看来绕了 10 圈零 3 段、B 看来只绕了 5 圈零 7 段；下文凡出现 q, r 都要分清是 (q_A, r_A) 还是 (q_B, r_B)。

Step 2：推导 A 指针偏移量 $f_A(P)$

按 Step 1 的拆分，A 指针在 P 个 phase 里的累计推进量是三项之和：

- **首块项**：phase 1 吃首块，A 指针推进 L_0^A 步。
- **整圈项**：每圈吃完 $r_0^A + r_1^A + \cdots + r_{k_A-1}^A$ 段；由 §3.3 知一整圈段长总和恰为 N（段长覆盖整串一次），故 q_A 整圈贡献 $q_A \cdot N$ 步。
- **零头项**：剩下 r_A 段贡献 $r_0^A + r_1^A + \cdots + r_{r_A-1}^A$，由 §3.3 定义直接查表得 pref_a$[r_A]$。

三项相加：

$$f_A(P) \;=\; L_0^A \;+\; q_A \cdot N \;+\; \text{pref\_a}[r_A]$$

$f_B(P)$ 完全同构，下标全换到 B 侧：$L_0^B + q_B \cdot N + \text{pref\_b}[r_B]$。

Step 3：推导累计得分 $g_A(P)$

分数同样按" 首块 + 整圈 + 零头" 三段累加，只是每一项乘的不是段长而是" 段长 × 分数率"：

- **首块项**：由 §3.4，phase 1 的 A-run 全程 $A_1 = c_A$，首块分数率就是 c_A，贡献 $L_0^A \cdot c_A$ 分（$c_A = 0$ 时此项为 0，$c_A = 1$ 时整块计分）。
- **整圈项**：§3.4 已将一整圈 k_A 段的得分聚合为 $W_A = \sum_{i=0}^{k_A-1} r_i^A \cdot \text{rate}_i$，$q_A$ 整圈贡献 $q_A \cdot W_A$。
- **零头项**：零头 r_A 段按 §3.4 的得分前缀和直接查表 pref_score_A$[r_A]$。

三项相加：

$$g_A(P) \;=\; L_0^A \cdot c_A \;+\; q_A \cdot W_A \;+\; \text{pref\_score\_A}[r_A]$$

Step 4：边界 $P = 0$ 单独特判

$P = 0$ 表示" 一步都没走"，三个函数应返回 0。**不能直接代入通用公式**——代入后首块项 L_0^A（或 $L_0^A \cdot c_A$）会被白白算进去，并且 $P - 1 = -1$ 做下取整和取模在 C++ 里行为还依赖符号约定，徒增隐患。代码里统一加一行 `if (P == 0) return 0;` 前置挡掉。

汇总：三个 $O(1)$ 函数同构，只差" 走的是段长还是得分"、" 用 A 侧还是 B 侧的周期"：

函数	含义（走完 P 个 phase 后）	公式
$f_A(P)$	A 被左移的总次数	$L_0^A + q_A \cdot N + \text{pref\_a}[r_A]$
$f_B(P)$	B 被左移的总次数	$L_0^B + q_B \cdot N + \text{pref\_b}[r_B]$
$g_A(P)$	累计得分	$L_0^A \cdot c_A + q_A \cdot W_A + \text{pref\_score\_A}[r_A]$

记号：$(q_A, r_A) = (\lfloor (P-1)/k_A \rfloor,\ (P-1) \bmod k_A)$，$(q_B, r_B) = (\lfloor (P-1)/k_B \rfloor,\ (P-1) \bmod k_B)$；$P = 0$ 时三者都为 0。pref_a, pref_b 已在 §3.3 定义、pref_score_A 和 W_A 在 §3.4。

落到代码就是三个 $O(1)$ 函数，结构和上表一一对应：

```
long long f_A(long long P) {                    // A 指针偏移
    if (P == 0) return 0;
    long long q = (P - 1) / k_A, r = (P - 1) % k_A;
    return L0_A + q * N + pref_a[r];            // 首块 + q 整圈 · N + r 段零头
}

long long f_B(long long P) {                    // B 指针偏移（完全对称）
    if (P == 0) return 0;
    long long q = (P - 1) / k_B, r = (P - 1) % k_B;
    return L0_B + q * N + pref_b[r];
}

long long g_A(long long P) {                    // 累计分数
    if (P == 0) return 0;
    long long q = (P - 1) / k_A, r = (P - 1) % k_A;
    long long V0 = L0_A * cA;                   // 首块全程 A[0]=cA，贡献 V0
    return V0 + q * W_A + pref_score_A[r];      // 首块 + q 整圈 · W_A + r 段分数零头
}
```

4.3 二分找最大 P

已知：每一步要么推进 A 要么推进 B，因此走完 P 个 phase 的总迭代数恰为 $f_A(P) + f_B(P)$，且关于 P 单调递增；f_A, f_B 各 $O(1)$ 可求。

要求：给定询问 K，找最大的 $\boldsymbol{P^*}$ 使 $f_A(P^*) + f_B(P^*) \le K$——此时答案主体就是 $g_A(P^*)$。

如何实现：单调函数上求最大满足点，直接在 $[0, K]$ 上二分 P。每次判定 $O(1)$（一次 $f_A + f_B$），**单次询问 $O(\log K)$**。

4.4 零头 R 的贡献

已知：二分给出 P^* 后，$\boldsymbol{R} := K - (f_A(P^*) + f_B(P^*))$ 是落在第 P^*+1 个 phase 内、还没跨出去的零头步数。该 phase 的 A-run 长 $\text{len}_{\boldsymbol{A}}$、$B$-run 长 $\text{len}_{\boldsymbol{B}}$、$A$ 分数率 $\textbf{rate}_{\boldsymbol{A}}$ 可直接从 r^A, r^B 取出：$P^* = 0$ 取首块 (L_0^A, L_0^B, c_A)；$P^* \ge 1$ 取 $r^A_{(P^*-1) \bmod k_A}$ 等。

要求：把 R 步的分数贡献加到 $g_A(P^*)$ 上。

如何实现：由 §3.2 的关键不变量，**所有 phase 的内部子结构都一样**——只看一次 $c_A \overset{?}{=} c_B$ 即可查表定案：

c_A vs c_B	phase 内部结构	R 步累计得分
$c_A = c_B$	先 matched (A-run) 再 mismatched (B-run)	$\min(R,\ \text{len}_A) \cdot \text{rate}_A$
$c_A \ne c_B$	先 mismatched (B-run) 再 matched (A-run)	$\max(R - \text{len}_B,\ 0) \cdot \text{rate}_A$

口诀：$\boldsymbol{c_A = c_B}$ 时 A 在前，R 步先花在 matched 段上（超出部分是 B 在动、不得分）；$\boldsymbol{c_A \ne c_B}$ 时 B 在前，R 步先花在 mismatched 段（不得分），多出来的 $R - \text{len}_B$ 步才落到 matched 段上。代码里 `A[0] == B[0]` 就是在判这个（`A[0], B[0]` 实际上就是 c_A, c_B）。

4.5 uniform 特判

已知：上面所有公式都依赖"A 或 B 的 run 序列周期交替翻转"这一结构。**当 A 或 B 是全同字符时翻转结构退化，奇偶分数率公式直接出事。**

要求：对退化的三类情形（都 uniform、只 A uniform、只 B uniform）直接给闭式答案，绕开通解。

如何实现：按下表分类闭式：

情形	条件（uniform 下首字符即全串）	答案
都 uniform	$c_A = c_B = 1$	K
	其他	0
只 A uniform	$c_A = 0$	0
	$c_A = 1,\ c_B = 1$	K
	$c_A = 1,\ c_B = 0$	$\max(0,\ K - L_0^B)$
只 B uniform	$c_A = 0$ 或 $c_B = 0$	0
	$c_A = 1,\ c_B = 1$	$\min(K,\ L_0^A)$

两行注解：

- 只 A uniform 且 $c_A = 1, c_B = 0$：要先熬过 B 的 L_0^B 个 0，此后 B_1 翻成 1、B 又是 uniform 的 $0\ldots01\ldots1$ 轮换，但这时 A 恒为 1，每步都是 $1 \cdot 1 = 1$ 得分，故答案 $\max(0, K - L_0^B)$；
- 只 B uniform 且 $c_A = c_B = 1$：A 首块内每步都得 1 分，A 首块走完后 A_1 翻成 0、再也不匹配，故答案 $\min(K, L_0^A)$。

4.6 复杂度

已知：预处理要扫 A, B 并建前缀和；每次询问一次二分 + $O(1)$ 零头；$\sum N, \sum Q \le 10^6$，$K \le 10^{12}$。
要求：估算总复杂度，确认能 3 s 内跑完。
如何实现：预处理 $O(N)$，单次询问 $O(\log K)$。合计

$$O(N + Q \log K) \approx 10^6 + 10^6 \cdot 40 \approx 4 \times 10^7$$

常数很小，**稳过 3 s 时限**。

5 AC 代码

```cpp
// ============================================================================
// Shift Game (CodeChef starters-235) 正解
//
// 题意回顾：两个长度 N 的 01 串 A、B，每步分数加 A[0] · B[0]；
//   若 A[0]==B[0] 则左移 A，否则左移 B。独立询问：K 步后分数是多少。
//
// 核心思路：
//   分数只有在 A[0]==B[0]==1 时增加。把 A、B 各自按极长同色段（run）分块，
//   首块（与 A[0]、B[0] 同字符的极长前缀）长度分别记 L0_A、L0_B；之后循环
//   一圈得到 runs_A、runs_B（长度 k_A、k_B）。关键观察：在"A 的第 i 段"里，
//   A[0] 保持不变；A[0]==B[0] 时整段在左移 A，A[0]!=B[0] 时先左移 B 到翻段
//   再进入下一种模式。因此"走完 P 段"所用的总步数 = L0_A + L0_B + Σ 前 P
//   段 A、B 各自段长和；分数贡献也能按段累加。答案 = 二分最大的 P 使
//   f_A(P)+f_B(P) <= K，再处理剩余不足一整段的零头。
//
// 复杂度：O(N + Q · log K)。ΣN、ΣQ <= 1e6，K <= 1e12。3s 时限稳过。空间 O(N)。
// ============================================================================

#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

void solve() {
    long long N, Q;
    if (!(cin >> N >> Q)) return;
    string A, B;
    cin >> A >> B;
```

```
    vector<long long> K_queries(Q);
    for (int i = 0; i < Q; i++) cin >> K_queries[i];

    // ------------------------------------------------------------
    // 一、判定 A、B 是否为单一字符组成 (uniform 分支单独处理)
    // 全同字符时 runs 退化、没有周期性交替，走下面的通用公式会出问题
    // ------------------------------------------------------------
    bool is_uniform_A = true, is_uniform_B = true;
    for (int i = 1; i < N; i++) {
        if (A[i] != A[0]) is_uniform_A = false;
        if (B[i] != B[0]) is_uniform_B = false;
    }

    // ---- case 1: A、B 都 uniform ----
    // 两串都是全 0 或全 1。只有 A[0]==B[0]=='1' 时每步都得 1，其余全 0。
    if (is_uniform_A && is_uniform_B) {
        for (int q = 0; q < Q; q++) {
            long long K = K_queries[q];
            long long ans = (A[0] == B[0] && A[0] == '1') ? K : 0;
            cout << ans << (q == Q - 1 ? "" : " ");
        }
        cout << "\n";
        return;
    }

    // ---- case 2: 只 A uniform ----
    // A 全是同一个字符：若 A 全 0 永远不得分；若 A 全 1，则分数 = K 步里 B[0]==1 的步数。
    // 当 B[0]=='0' 时要先熬过 B 的 LO_B 个 0，熬完 B 左移后就变成全 1，从此每步得分。
    if (is_uniform_A && !is_uniform_B) {
        long long LO_B = 0;
        while (LO_B < N && B[LO_B] == '0') LO_B++;
        for (int q = 0; q < Q; q++) {
            long long K = K_queries[q];
            long long ans = 0;
            if (A[0] == '0') ans = 0;
            else if (B[0] == '1') ans = K;
            else ans = max(0LL, K - LO_B);
            cout << ans << (q == Q - 1 ? "" : " ");
        }
        cout << "\n";
        return;
    }

    // ---- case 3: 只 B uniform ----
    // 对称于 case 2：B 全是同一个字符。B 全 0 或 A[0]=='0' 永远不得分；
    // 否则 B 全 1 且 A 首块为 1，得分 = min(K, LO_A)（A 首块结束后就再也不匹配）。
    if (!is_uniform_A && is_uniform_B) {
        long long LO_A = 0;
        while (LO_A < N && A[LO_A] == '1') LO_A++;
        for (int q = 0; q < Q; q++) {
            long long K = K_queries[q];
            long long ans = 0;
            if (B[0] == '0' || A[0] == '0') ans = 0;
            else ans = min(K, LO_A);
            cout << ans << (q == Q - 1 ? "" : " ");
        }
        cout << "\n";
        return;
    }

    // ------------------------------------------------------------
    // 二、通用情形 (A、B 都不 uniform)：预处理 runs + 前缀和
    // ------------------------------------------------------------
```

```cpp
// 首块长度 LO_A、LO_B：与各自 s[0] 同字符的极长前缀长度
long long LO_A = 0;
while (LO_A < N && A[LO_A] == A[0]) LO_A++;
long long LO_B = 0;
while (LO_B < N && B[LO_B] == B[0]) LO_B++;

// 从 A 的第 LO_A 位（首块末尾的下一位）开始，把 A 当循环串扫一圈，
// 得到跳过残缺首块后的完整周期块长度序列 runs_A
vector<long long> runs_A;
int p_A = LO_A % N;
int curr_len = 0;
char curr_char = A[p_A];
for (int i = 0; i < N; i++) {
    char c = A[(p_A + i) % N];
    if (c == curr_char) {
        curr_len++;
    } else {
        runs_A.push_back(curr_len);
        curr_char = c;
        curr_len = 1;
    }
}
runs_A.push_back(curr_len);                 // 最后一段收尾

long long k_A = runs_A.size();               // A 的周期段数
vector<long long> pref_a(k_A + 1, 0);        // runs_A 的前缀和：pref_a[i] = 前 i 段长度和
for (int i = 0; i < k_A; i++) pref_a[i + 1] = pref_a[i] + runs_A[i];

// pref_score_A[i]：前 i 段 A 带来的分数累计。
// runs_A[0] 是跳过首块后的第一段，此时 A 首字符已翻转为 1-cA；之后奇偶段交替。
// i 偶 → 该段 A 首字符 = 1-cA，i 奇 → cA。只有首字符为 1 才产生得分，因此 rate 取这个值。
vector<long long> pref_score_A(k_A + 1, 0);
long long cA = A[0] - '0';
for (int i = 0; i < k_A; i++) {
    long long rate = (i % 2 == 0) ? (1 - cA) : cA;
    pref_score_A[i + 1] = pref_score_A[i] + runs_A[i] * rate;
}
long long W_A = pref_score_A[k_A];           // 整圈 A（k_A 段）的总得分，用于按圈跳跃

// 同理处理 B：得到 runs_B、pref_b。B 不需要分数前缀和，只需段长前缀和用来计算指针偏移。
vector<long long> runs_B;
int p_B = LO_B % N;
curr_len = 0;
curr_char = B[p_B];
for (int i = 0; i < N; i++) {
    char c = B[(p_B + i) % N];
    if (c == curr_char) {
        curr_len++;
    } else {
        runs_B.push_back(curr_len);
        curr_char = c;
        curr_len = 1;
    }
}
runs_B.push_back(curr_len);

long long k_B = runs_B.size();
vector<long long> pref_b(k_B + 1, 0);
for (int i = 0; i < k_B; i++) pref_b[i + 1] = pref_b[i] + runs_B[i];

// ------------------------------------------------------------
// 三、三个辅助闭包：给定"走完 P 段"，O(1) 算出 A 指针偏移、B 指针偏移、累计得分
```

```
    // ------------------------------------------------------------

    // f_A(P): 走完 P 段后 A 被左移的总次数 = 首块 LO_A + q 整圈 · N + r 段零头
    auto f_A = [&](long long P) -> long long {
        if (P == 0) return 0LL;
        long long q = (P - 1) / k_A, r = (P - 1) % k_A;
        return LO_A + q * N + pref_a[r];
    };

    auto f_B = [&](long long P) -> long long {
        if (P == 0) return 0LL;
        long long q = (P - 1) / k_B, r = (P - 1) % k_B;
        return LO_B + q * N + pref_b[r];
    };

    // g_A(P): 走完 P 段累计分数。首块 LO_A 全程 A[0]=cA, 贡献 V0 = LO_A · cA;
    // 整圈一共贡献 W_A; 零头 r 段贡献 pref_score_A[r]。
    auto g_A = [&](long long P) -> long long {
        if (P == 0) return 0LL;
        long long q = (P - 1) / k_A, r = (P - 1) % k_A;
        long long V0 = LO_A * cA;
        return V0 + q * W_A + pref_score_A[r];
    };

    // ------------------------------------------------------------
    // 四、对每个询问二分找 P, 再处理零头
    // ------------------------------------------------------------
    for (int q = 0; q < Q; q++) {
        long long K = K_queries[q];
        long long low = 0, high = K, best_P = 0;

        // 二分最大的 P 使 f_A(P)+f_B(P) <= K (两者之和关于 P 单调递增)
        while (low <= high) {
            long long mid = low + (high - low) / 2;
            if (f_A(mid) + f_B(mid) <= K) {
                best_P = mid;
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        long long P = best_P;
        long long ans = g_A(P);
        long long R = K - (f_A(P) + f_B(P));     // 零头步数, 落在第 P+1 段内

        // 取出当前正在走的"第 P+1 段"对应的段长与分数率
        long long len_A, rate_A, len_B;
        if (P == 0) {
            // P==0 说明还没跨过首块, 此时正处于首块内
            len_A = LO_A;
            rate_A = cA;
            len_B = LO_B;
        } else {
            long long idx_A = (P - 1) % k_A;
            len_A = runs_A[idx_A];
            rate_A = (idx_A % 2 == 0) ? (1 - cA) : cA;

            long long idx_B = (P - 1) % k_B;
            len_B = runs_B[idx_B];
        }

        // 零头贡献分两种情况:
```

```
        //   A[0]==B[0]: 当前段 A、B 首字符相同 → 整段都在左移 A，R 步全算 A 的分数率
        //   A[0]!=B[0]: 当前段先左移 B 耗掉 len_B 步，剩下的才算 A 的分数
        if (A[0] == B[0]) {
            if (R <= len_A) {
                ans += R * rate_A;
            } else {
                ans += len_A * rate_A;            // 超出的那部分对应 B 指针移动，无得分
            }
        } else {
            if (R > len_B) {
                ans += (R - len_B) * rate_A;
            }
        }

        cout << ans << (q == Q - 1 ? "" : " ");
    }
    cout << "\n";
}

int main() {
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    int T;
    if (cin >> T) {
        while (T--) solve();
    }
    return 0;
}
```

6 模型分析

这题的抽象模型是”**双指针在两个循环串上按规则推进，大 K 询问**”：

- 每一步只有一个指针前进，且前进哪个由两串当前首字符的关系决定；
- 进程在**阶段**层面天然周期化（每一阶段对应其中一个循环串的一个极长段）；
- 大 K 的询问 → 先算整圈贡献（除法），再算零头（取模），最外层用二分锁定”整段数”。

这是”**循环结构 + 前缀和 + 二分**”的经典组合：只要能把”时间”切成可周期化的阶段，任意大 K 的询问都能压到 $O(\log K)$。

7 扩展知识点

- **Run-Length Encoding（游程编码）**：把 01 串压成”极长同色段长度序列”。在字符串、位运算题中非常常用，能把”按字符处理”换成”按段处理”，常数时间级别的收益。
- **循环串上的前缀和**：当”逻辑前缀”越过原串末尾时，可用整圈长度 N + 取模余数 = 原串前缀的方式 $O(1)$ 查询。本题 f_A、f_B、g_A 都是这套模板。
- **二分答案在单调函数上**：一旦某个量关于参数单调，查询”满足条件的最大/最小参数”就可以二分；只要求值是 $O(1)$ 或 $O(\log)$，整体就是 $O(\log)$ 级别。
- **零头处理**：大 K 查询通过”整圈跳跃 + 余量处理”求解的模板：$q = K/T$（整圈数），$r = K \bmod T$（零头），先用 q 乘以一圈贡献，再把 r 放进一个段内精细分类讨论。

8 常见思维考点

- **”时间 → 阶段”的降维**：当步数 K 过大无法逐步模拟，要想办法把时间切成可周期化的单位。本题里这个单位就是”A 首字符保持不变的一段”，在别的题里可能是”一个状态循环”、”一个 cycle”、”一段 cooldown”等。识别出这个单位是本题的关键跨步。

- **观察得分只在 $A_1 = B_1 = 1$ 时发生**：这个朴素结论把”算分数”直接转化为”统计带分数率的段内 A 指针移动步数”，大大简化建模。很多位运算/状态机题也是靠这种”得分条件极简”简化的。
- **特判 uniform 边界**：通解的公式会在退化情形（段数 $A = 0$ 或段数 $B = 0$）下出现除零、空数组访问。提前**把所有退化分支抽出去单独闭式处理**是写稳这类分块题的必备动作。
- **”A 和 B 异步但耦合”的双指针**：每步只推进一个指针，但推进哪个依赖两者的关系。这类题型的破解思路几乎都是”找到相对同步点”——本题就是每段末尾 A 指针翻入下一段，作为自然同步点。

9 练习题推荐

- Luogu P1965　NOIP 2013 转圈游戏（**入门暖身**，通过 29734 人）——K 次循环位移后落到哪，配合快速幂，是**”时间 → 周期”**最朴素的那类。
- Luogu P1962　斐波那契数列（**入门暖身**，通过 33013 人）——求 $F(K)$，$K \le 10^{18}$。线性递推 + 矩阵快速幂，和本题一样的**”大 K 无法模拟，必须挖结构”**套路，只是这边的结构是矩阵幂、本题是 run-length 周期。
- CF 778A　String Game（**入门暖身**，rating 1663，通过 12420 人）——在字符串上二分最大操作次数 k 使条件保持成立，和本题**”二分最大 P”**同构。
- Yukicoder 1245　ANDOR ゲ🄵ム（进阶，通过 140 人）——给定数列 A 和操作序列 S（每位 0/1 决定 AND/OR），Q 次询问每次给不同初始值 t，问走完 N 次操作后 $\sum |X_{\text{new}} - X_{\text{old}}|$。和本题**”固定数据 + Q 次独立询问 + 按规则累积得分”**的骨架同构，解法也同样是”预处理一次 + 单查询 $O(\log)$”。
- QOJ 7522　Sequence Shift（进阶，通过 126 人）——两个长度相等的数组、Q 次操作维护得分 $\max_i(A_i + B_i)$。把本题的”双串 + 得分 + Q 询问”壳子换成可修改版本，是**”双序列耦合 + 在线维护”**的进阶练习。
- Gym 103076C　Cellular Automaton（进阶，通过 59 人）——01 串按规则迭代 K 次求终态，K 很大。**”01 串 + 大 K 演化”**的最直白形态，不找周期就过不去。
- CodeChef CRAZYROW　Crazy Row（硬核挑战，rating 2988，通过 33 人）——一个自相似的 01 串在给定区间上数 1 的个数。**”周期 / 自相似 + 区间计数”**，和本题”首块 + 整圈 + 余数”三段式分解同构。