L1438 最小翻转次数 ☆☆☆

题目描述

给定一个长度为 n 的二进制字符串 s（仅包含字符 0 和 1）和一个整数 k。

你可以执行一种操作：选择字符串中的任意一个长度为 k 的连续子串，并将该子串中的每一个字符进行翻转（即 0 变成 1，1 变成 0）。

你的目标是通过最少的操作次数，将整个字符串 s 变成全 0 字符串。如果无法实现，输出 -1。

输入输出格式

输入：第一行包含两个整数 n 和 k（$1 \le k \le n \le 10^5$），分别表示字符串长度和操作长度。第二行包含一个长度为 n 的字符串 s。

输出：输出一个整数，表示最少操作次数。如果无法实现，输出 -1。

输入示例	输出示例
5 3 10101	3

样例解释

初始字符串为 10101，目标是全 0，每次翻转长度 $k = 3$。

步骤	翻转区间	翻转后状态
1	[0, 2]	01001
2	[1, 3]	00111
3	[2, 4]	00000

算法分析

1. **贪心策略证明**

本题目具有以下两个关键性质：

- **顺序无关性：**操作的顺序不影响最终结果。先翻转区间 A 再翻转区间 B，和先翻转 B 再翻转 A，得到的结果是一样的。
- **操作的唯一性：**假设我们要处理当前最左边的一个位置 i，且下标 $0 \dots i-1$ 的部分已经全部变成 0 了。此时：
  （1） 如果 $s[i]$ 是 1，为了让它变成 0，我们必须翻转一个包含 i 的区间。虽然包含 i 的区间有很多（例如 $[i-1, i+k-2]$ 等），但只要区间的起点**小于** i，它就一定会覆盖到 i 左侧那些已经处理好变为 0 的位置。 **翻转这些位置会破坏我们之前的成果**（将 0 变回 1）。
  （2） **结论：**为了修复 $s[i]$ 且**绝对不破坏** i 左边已完成的全 0 前缀，唯一的选择就是翻转以 i 为起点的区间 $[i, i+k-1]$。

因此，每一个位置的操作决策是唯一确定的：遇到 1 必须翻转，且必须翻转以当前位置为开头的区间。

2. **算法选择**

（1） **解法一 - 暴力模拟 (TLE)：** 从左到右遍历，遇到 1 就翻转接下来的 k 个字符。单次翻转复杂度 $O(k)$，总复杂度 $O(nk)$。当 $n = 10^5$ 时会超时。

（2） **解法二 - 计数器优化 (AC)：** 我们不需要真的去改变数组里的每一个值，只需要记录当前位置受到了多少次翻转操作的影响。我们可以引入一个“计数器”和“过期标记”的概念，将复杂度降为 $O(n)$。

3. **实现思路 (计数器 + 标记)**

- **核心逻辑** 想象一个滑动窗口，所有针对当前位置 i 生效的翻转操作，都在这个窗口内。如果 $[i\ ,\ i+k-1]$ 区间需要翻转,我们的操作就是:

  （1） **计数器自加 (cur++)：**当前位置 i 的翻转次数 $+1$。

  （2） **区间结束标记：**在 $i+k-1$ 处打一个标记，表示翻转区间在此处结束。

- **遍历字符串：**当我们走到位置 i 时，首先检查有没有操作在这里失效：

```
cur += pre[i]; // 加上那些在当前位置结束影响的操作
```

- **判断是否需要翻转：**如果当前是‘1’ 且翻转次数是偶数，或当前是‘0’ 且翻转次数是奇数，则需要翻转。反之不需要翻转。

```
int ch = s[i] - '0';
if ((ch ^ cur) & 1) { 使用位运算简化判断逻辑
    cnt++;       // 记录操作次数
    cur++;       // 新增一个操作，当前位置的翻转次数+1
    if (i + m > n) return -1; // 长度不够，无法翻转，直接返回-1
    pre[i + m]--; // 并在 i+m 处打个标记：这个操作到时候要失效
}
```

拓展思考

- 如果题目要求将字符串变成全 1，代码需要做哪些微调？