复杂度与前缀和、差分、双指针

从 $O(n^2)$ 暴力超时到 $O(n)$ 预计算与单调扫描

HGNU-ACM 实验室算法组

承接第一篇第 8 章 STL 高效查找；配套四大专题 + 6 道精讲例题与 6 份源码

本章目录

时间空间复杂度与大 O 估算

一维与二维前缀和：$O(1)$ 回答区间累加

一维与二维差分：$O(1)$ 批量区间修改

双指针与滑动窗口：单调扫描 $O(n)$ 求解

前四节从第一篇的 $O(\log n)$ 查找出发，建立“防超时”意识并掌握前缀和、差分与双指针。

本章目录（续）

经典例题精讲（6 题 6 步逐题剖析）

常见错误、检查清单与练习安排

第五节逐题精讲 6 道配套例题（覆盖 6 份 C++17 源码）；第六节汇总常见错误、自测清单与练习。

生活引入：为什么“能过样例”交上去却超时了？

衔接上一章：从 $O(\log n)$ 查找来到算法效率

在第一篇第 8 章，我们学会用 `lower_bound` 和 `set` 把一次查找从 $O(n)$ 降到 $O(\log n)$。现在进入算法篇，先用记账本看一个新问题：

- 假设你记了 n 天的每日花销。老板问 q 次：“第 l 天到第 r 天一共花了多少？”
- 如果只有 5 天、问 3 次，你从第 l 天逐天加到第 r 天，一共只加十几次，瞬间就能答出来。
- 可如果账本有 $100{,}000$ 天，老板问 $100{,}000$ 次，每次都要你从头一页页加到尾，几天几夜都算不完！

本页要点

样例数据往往很小（$n = 5$），任何笨办法都能瞬间跑完；真正决定能否通过的，是最大数据范围下的总步数。

暴力做法运算次数计算：为什么会 TLE？

算一笔硬账：当 $n = 10^5,\ q = 10^5$ 时，两层循环一共加多少次？

每次询问都写一层 `for (i = l; i <= r; ++i) sum += a[i];`：

- 最坏情况下每次询问都要从 1 加到 10^5，做 10^5 次加法；
- 共 10^5 次询问，总基本操作次数 $= 10^5 \times 10^5 = 10^{10}$ 次！

- 评测机 1 秒大约只能执行 $10^7 \sim 10^8$ 次简单基本操作（加法、比较、赋值）。
- $10^{10} \div 10^8 = 100$ 秒！竞赛限时通常只有 1 秒，程序必定被判为 **TLE**（运行超时）。

本页要点

写代码前先算最坏情况的总运算次数：只要接近或超过 10^8，暴力写法就一定会超时。

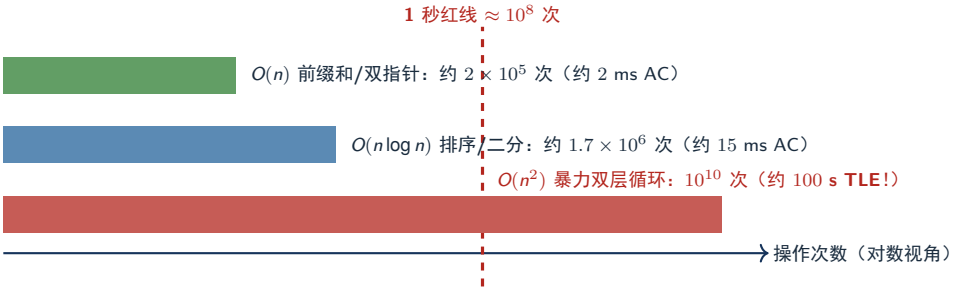
大白话解释新术语：时间复杂度、大 O 与空间限制

- 时间复杂度（渐进复杂度）：不纠结“跑了 3 毫秒还是 5 毫秒”，只看当数据规模 n 变大时，总操作次数增长的快慢趋势。
- 大 O 记号：只保留增长最快的那一项，并把前面的常数倍扔掉。例如总步数是 $3n^2+5n+10$，只记为 $O(n^2)$；$2n+6$ 只记为 $O(n)$。
- 为什么敢扔掉常数？当 $n=10^5$ 时，$n^2=10^{10}$，而 $5n=5\times10^5$，两者相差两万倍，决定生死的永远是“指数和幂次”。
- 空间复杂度与内存红线：程序运行时开数组占用的内存。常见限制 256 MB $\approx 2.6\times10^8$ 字节；一个 `int` 占 4 字节，最多开约 6×10^7 个；一个 `long long` 占 8 字节，最多开约 3×10^7 个。

本页要点

大 O 看增长趋势，考场估算看“最坏步数是否远小于 10^8”以及“数组总字节是否小于 256 MB”。

图解差距：10^8 红线下的三种量级（$n = 10^5$）



- 把 $O(n^2)$ 优化到 $O(n)$，相当于把“100 秒的龟速”变成“2 毫秒的闪电”，提速 $50{,}000$ 倍！

本页要点

考场看到 $n \geq 10^5$，心里立刻拉响警报：绝对不能写 $O(n^2)$，必须降到 $O(n \log n)$ 或 $O(n)$。

手算跟踪表：数清 4 种常见循环到底跑了多少步

设 $n = 4$，统计核心语句执行次数（含最后一次使循环条件为假的退出判断）：

循环形态	$n = 4$ 步数	退出判断	精准式	$n = 10^5$ 量级
单层 `for (1..n)`	4	$i = 5$ 假	n	$O(n) \approx 10^5$
双层全矩形 `1..n, 1..n`	$4 \times 4 = 16$	各多 1 次	n^2	$O(n^2) \approx 10^{10}$
双层上三角 `1..n, i..n`	$4 + 3 + 2 + 1 = 10$	内外判假	$n(n+1)/2$	$O(n^2) \approx 5 \times 10^9$
折半 `x=n; x>0; x/=2`	$4 \to 2 \to 1 \to 0$（3 步）	$x = 0$ 假	$\lfloor \log_2 n \rfloor + 1$	$O(\log n) \approx 17$

- 上三角虽然步数减半（$n^2/2$），但 $n = 10^5$ 时仍有 5×10^9 次，依然远超 10^8 红线！

本页要点

常数 $1/2$ 救不了平方复杂度：5×10^9 照样超时 50 秒，必须从算法上降维。

三段式估算原则：不靠数 for 层数猜复杂度

如何正确判断一段代码的复杂度？按以下三步检验：

步骤	判别方法与实例
1. 初始化（定核心）	找出最内层执行次数最多的语句，或耗时库函数（如 `sort`、串拷贝）。
2. 保持（算总量）	看全程总共执行多少次，不能只数几层括号： ▪ 一层也超时：循环里调 `sort`（$O(n^2 \log n)$）或 `s = s + 'a'`（$O(n^2)$）； ▪ 两层也线性：双指针外层走 n 步，内层 `while` 让 `l` 全程最多加 n 次，总计 $\leq 2n$ 步！
3. 终止（对红线）	去掉常数倍和低次项得大 O，代入题目最大 n 核对是否远小于 10^8。

本页要点

算复杂度数“核心操作全程总共执行了几次”，不数代码缩进了多少层。

模板代码：考场快速读入与输出规范

```cpp
int main() {
    ios::sync_with_stdio(false);        // 1. 关闭 C++ 与 C 流同步，读入提速近 10 倍
    cin.tie(nullptr);                   // 2. 解绑 cin 与 cout，避免读入前刷新输出
    int n; cin >> n;
    for (int i = 1; i <= n; ++i) {
        cout << i << '\n';              // 3. 大量输出一律用 '\n'，禁用 endl
    }
    return 0;
}
```

本页要点

输入输出达到 10^5 级别时，写上关闭流同步并把 `endl` 换成 `'\n'`，可防无辜 TLE。

上机验证：三组数据规模对比与反事实追问

本地测试：分别运行 $O(n^2)$ 暴力求和与 $O(n)$ 前缀和（单位：操作步数）

1. 最小测试（$n = 10,\ q = 10$）：暴力最多 100 步，前缀和约 20 步 $\rightarrow$ 均 < 1 ms。
2. 中等测试（$n = 3000,\ q = 3000$）：暴力约 9×10^6 步（勉强通过），前缀和 6000 步。
3. 满载测试（$n = 10^5,\ q = 10^5$）：暴力达 10^{10} 步（直接卡死 TLE），前缀和仅 2×10^5 步（< 50 ms）。

反事实追问：如果把内层循环改成只加偶数位置（`j += 2`），步数少了一半，能 AC 吗？

不能！总步数从 10^{10} 变成 5×10^9，依然是 $O(n^2)$，按每秒 10^8 次计算仍需 50 秒，一样 TLE！

本页要点

小数据不超时不代表算法对；上线前必须拿题目最大数据范围估算总步数。

新手死穴对照：复杂度与空间的三大翻车点

错误写法 / 做法	评测结果	原因与正确做法
循环里写 `cout << x << endl;`（$n = 2 \times 10^5$）	**TLE**	`endl` 每次强制刷新缓冲区极慢；改用 `'\n'` 并关同步。
循环里写 `s = s + ch;` 拼接字符	**TLE**	每次生成新串拷贝 $O(\|s\|)$，总耗时 $O(n^2)$；改用 `s += ch;`。
开二维数组 `int a[100005][100005]`	**MLE / RE**	需要 4×10^{10} 字节（约 40GB）直接爆内存；重新建模降维。

本页要点

卡在 TLE 先查循环总步数与 `endl`，卡在 MLE/RE 先算数组总字节数是否超过 256 MB。

生活引入：记账本怎么立刻查出第 2 到第 4 天花了多少？

具体小例子：连续 5 天每天支出 $a = [3, 1, 4, 1, 5]$ 元

如果你每天晚上除了记“今天花了多少钱”，还在旁边顺手记一笔“截至今天一共花了多少钱”：

- 第 1 天后累计 3 元；第 2 天后累计 4 元；第 3 天后累计 8 元；第 4 天后累计 9 元；第 5 天后累计 14 元。
- 现在老板突然问：“第 2 天到第 4 天一共花了多少？”
- 你根本不用再算 $1 + 4 + 1$！直接拿第 4 天的累计（9 元）减去第 1 天的累计（3 元）：$9 - 3 = 6$ 元！
- 哪怕有 100,000 天，问任意一段 $[l, r]$，都只需要做一次减法！

本页要点

提前用一遍加法算出“截至每一位的累计总和”，之后任何一段区间的和都只需一次减法。

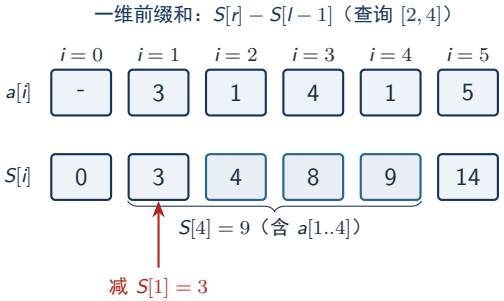
暴力为什么 TLE？用大白话讲清三个新术语

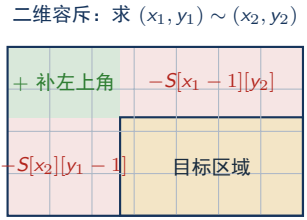
- 暴力做法估算：面对 $n = 2 \times 10^5$ 个数、$q = 2 \times 10^5$ 次区间求和，每次从 l 累加到 r，最坏总操作数高达 4×10^{10} 次 $\Longrightarrow$ 约 400 秒，必 **TLE**！
- 新术语 **1**——前缀和（$S[i]$）：数组从第 1 个元素到第 i 个元素的累计和，即 $S[i] = a[1] + a[2] + \cdots + a[i]$。
- 新术语 **2**——1-**based** 与垫脚石 $S[0] = 0$：把数据从下标 1 开始存，并规定 $S[0] = 0$。这样问 $[1, r]$ 时公式 $S[r] - S[0]$ 同样适用，不必特判左端点是不是 1。
- 新术语 **3**——二维前缀和与容斥原理：把一条线扩展成平面网格，$S[i][j]$ 记录左上角 $(1, 1)$ 到 (i, j) 矩形内所有格子之和；算任意子矩阵时“大矩形减左、减上，再把多减一次的左上角补回来”。

本页要点

预处理花 $O(n)$（二维 $O(nm)$）算一次前缀和，就能把每次询问从 $O(n)$ 压成 $O(1)$。

TikZ 图解：一维格子数组相减与二维矩形容斥





本页要点

一维砍掉 $l-1$ 之前；二维先取全角 $S[x_2][y_2]$，减去左侧与上方后，多减了一次的左上角必须加回来。

手算跟踪表：在 $a=[3,1,4,1,5]$ 上逐轮构造 $S[0\ldots5]$

初始令 $i=1$，$S[0]=0$，循环条件为 $i\le 5$：

判断前 i	$i\le5$？	读入 $a[i]$	更新 $S[i]=S[i-1]+a[i]$	含义说明
1	真	3	$S[1]=0+3=3$	前 1 项累计和
2	真	1	$S[2]=3+1=4$	前 2 项累计和
3	真	4	$S[3]=4+4=8$	前 3 项累计和
4	真	1	$S[4]=8+1=9$	前 4 项累计和
5	真	5	$S[5]=9+5=14$	前 5 项累计和
6	假（$6\le5$）	不读入	不更新（保留 $S[0\ldots5]$）	循环结束，预处理完成

本页要点

每一轮只做一次加法，循环结束共处理 5 个元素，第 6 次判断为假后退出。

正确性论证：为什么前缀和一定不会算错？

以循环 `for (int i = 1; i <= n; ++i) s[i] = s[i - 1] + a[i];` 为例，用不变量三段式验证：

检验阶段	必须成立的事实
1. 初始化	进入第 1 轮（$i = 1$）前，显式设置 `s[0] = 0`，恰好等于前 0 个数之和（空和）。
2. 保持	若进入第 i 轮前 $s[i-1] = a_1 + \cdots + a_{i-1}$ 成立，执行 `s[i]=s[i-1]+a[i]` 后 $s[i] = a_1 + \cdots + a_i$ 仍成立。
3. 终止	$i = n+1$ 退出时 $s[0 \ldots n]$ 全部成立，故任意 $[l, r]$ 有 $S[r] - S[l-1] = a_l + \cdots + a_r$。

本页要点

二维同理：$(1, 1) \sim (x_1 - 1, y_1 - 1)$ 被减了两次，末尾加回一次恰好不重不漏。

模板代码（1/2）：一维前缀和标准写法

```
int n, q; cin >> n >> q;
vector<long long> s(n + 1, 0);          // 1. 开 n+1 格且 s[0]=0; 和用 long long
for (int i = 1; i <= n; ++i) {          // 2. 从下标 1 开始存, 防 l=1 时越界
    long long x; cin >> x;              // 3. 边读边递推, 无需另存原数组 a
    s[i] = s[i - 1] + x;                // 4. 前 i 项和 = 前 i-1 项和 + 当前数
}
while (q--) {
    int l, r; cin >> l >> r;
    cout << s[r] - s[l - 1] << '\n';    // 5. O(1) 回答闭区间 [l, r]
}
```

本页要点

下标从 1 开始且 s[0]=0；前缀和数组必须用 long long。

模板代码（2/2）：二维前缀和标准写法

```
vector<vector<long long>> s(n + 1, vector<long long>(m + 1, 0)); // 第 0 行列全 0
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j) {
        long long x; cin >> x;
        s[i][j] = s[i - 1][j] + s[i][j - 1] - s[i - 1][j - 1] + x; // 递推
    }
while (q--) {
    int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
    long long ans = s[x2][y2] - s[x1 - 1][y2] - s[x2][y1 - 1] + s[x1 - 1][y1 - 1];
    cout << ans << '\n';                                          // O(1) 查矩形和
}
```

本页要点

递推“上加左减左上加自己”，查询“全角减上减左加左上”，第 0 行列是天然垫脚石。

上机验证：三组测试数据与反事实追问

一维前缀和上机验证（输入序列 [3, 1, 4, 1, 5]）

1. 普通区间：询问 $[2,4] \Longrightarrow S[4]-S[1]=9-3=6$（手工核对 $1+4+1=6$）。
2. 左边界 $l=1$：询问 $[1,5] \Longrightarrow S[5]-S[0]=14-0=14$（验证 `s[0]=0` 垫脚石）。
3. 单点区间：询问 $[3,3] \Longrightarrow S[3]-S[2]=8-4=4$（验证只保留第 3 项）。

反事实追问：如果查询时不小心把 s[r] - s[l - 1] 写成 s[r] - s[l] 会怎样？

会把左端点 $a[l]$ 本身也一起减掉！例如问 $[3,3]$ 会错算出 $S[3]-S[3]=0$，问 $[2,4]$ 会漏掉 $a[2]=1$ 只得到 5。

本页要点

每写完前缀和，先拿单点查询 $[3,3]$ 和贴边查询 $[1,n]$ 跑一遍，立刻能抓出 $l-1$ 写错。

新手死穴对照：一维与二维前缀和三大翻车点

错误写法	评测结果	原因与正确做法
`vector<int> s(n + 1)`	**WA**	$n = 2\times10^5$, $a_i = 10^9$ 时和达 2×10^{14}，`int` 溢出变负数；一律用 `long long`。
用 0-based 存数组并查 `s[l-1]`	**RE / WA**	当 $l = 0$ 时访问 `s[-1]` 越界崩溃；统一改成 1-based 存数且 `s[0]=0`。
题目要求“边单点修改边区间查询”还硬用前缀和	**TLE**	每次修改都要重算 $O(n)$ 个前缀和，总时间退化成 $O(nq)$；此类动态题需改用树状数组或线段树。

本页要点

前缀和只适用“先一次性读完（静态），再做大量区间求和”的场景！

生活引入：5 个房间批量发补贴，最后统一查账

具体小例子：有 5 个空房间，初始金额全为 0（$a = [0, 0, 0, 0, 0]$）

管理员收到两条批文，要求全部操作完成后统一看账本：

- 第 1 条：给第 2 号到第 4 号房，每间发 $+5$ 元；
- 第 2 条：给第 3 号到第 5 号房，每间扣 -2 元。
- 如果每次发钱都跑去把区间里每一间房的账本擦掉重写，遇到 200,000 间房、200,000 条批文就会累死！
- 聪明做法：只在“开始涨价”的第 l 号房记一笔 $+c$，在“结束后的下一间”第 $r+1$ 号房记一笔 $-c$；全部批文记完，从头到尾顺着加一遍就得出每间房最终金额！

本页要点

当修改很多、且只在所有修改结束后才查答案时，只需要在区间的两个端点记账。

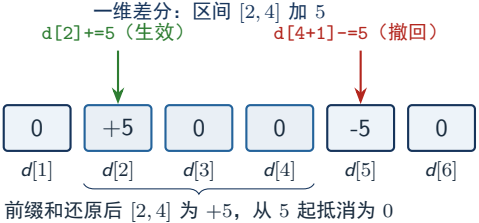
暴力为什么 TLE？用大白话讲清三个新术语

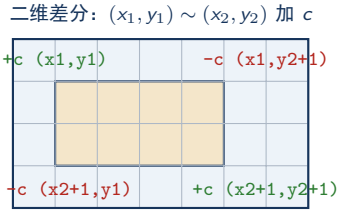
- 暴力做法估算：$n = 2 \times 10^5$ 间房、$m = 2 \times 10^5$ 次区间加，每次写一层 `for (i = l; i <= r; ++i) a[i] += c;`，最坏执行 $2 \times 10^5 \times 2 \times 10^5 = 4 \times 10^{10}$ 次加法 $\Longrightarrow$ 约 400 秒，必 **TLE**！
- 新术语 **1**——差分数组 d：记录相邻两项的增量差，定义 $d[i] = a[i] - a[i-1]$（其中 $a[0] = 0$）。对 d 求一次前缀和，正好还原出原数组 a。
- 新术语 **2**——差分抵消：在 $d[l]$ 加 c 会让后面所有位置都多 c，但在 $d[r+1]$ 减 c 后，从 $r+1$ 起一正一负恰好互相抵消成 0！
- 新术语 **3**——二维差分：把“一维两点打标”升级到平面矩形，只需修改 4 个角点，最后做一遍二维前缀和即可还原整个矩阵。

本页要点

差分和前缀和是一对互逆操作：一次区间加只需 $O(1)$ 改两处端点，最后用 $O(n)$ 前缀和统一还原。

TikZ 图解：一维差分两点抵消与二维差分四点标记





本页要点

记住“右端外一格”：一维必须开到 n+2 容纳 d[n+1]，二维必须开到（n+2）x（m+2）。

手算跟踪表：在 $n=5$ 上执行 $[2,4]+5$ 与 $[3,5]-2$ 并还原

初始令 $d[0\ldots6]=0$；先完成两次区间标记，再以 $i=1$，$cur=0$（条件 $i\le5$）还原：

步骤 / 判断前 i	$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	说明 / 还原后 cur
操作 $[2,4]+5$	0	+5	0	0	−5	0	改 d[2] 与 d[5]
操作 $[3,5]-2$	0	+5	−2	0	−5	+2	改 d[3] 与 d[6]
$i=1$（$1\le5$ 真）	**0**	+5	−2	0	−5	+2	$cur=0+0=$ **0**
$i=2$（$2\le5$ 真）	0	**+5**	−2	0	−5	+2	$cur=0+5=$ **5**
$i=3$（$3\le5$ 真）	0	+5	**-2**	0	−5	+2	$cur=5-2=$ **3**
$i=4$（$4\le5$ 真）	0	+5	−2	**0**	−5	+2	$cur=3+0=$ **3**
$i=5$（$5\le5$ 真）	0	+5	−2	0	**-5**	+2	$cur=3-5=$ **−2**
$i=6$（$6\le5$ 假）	-	-	-	-	-	+2	终止，输出 0 5 3 3 -2

本页要点

留意 d[6] = +2 虽不输出，却正是它保证区间在 5 号结束时没有越界崩溃！

正确性论证：差分抵消的三段式与不变量检验

对一次操作 (l, r, c)（`d[l]+=c; d[r+1]-=c;`），任意位置 $k \in [1, n]$ 的还原值 $a[k] = \sum_{j=1}^{k} d[j]$：

检验阶段 / 区间段	必须成立的贡献规律
1. 初始化	全 0 时 $d[1 \dots n+1] = 0$；若原数组非 0，先令 $d[i] = a[i] - a[i-1]$。
2. 保持（分段贡献）	每次 (l, r, c) 对前缀和 $\sum_{j=1}^{k} d[j]$ 的净增量分三段： ▪ $k < l$：既未加 $d[l]$ 也未减 $d[r+1]$，净增量 $= 0$； ▪ $l \le k \le r$：仅包含 $d[l]$，净增量 $= +c$； ▪ $k > r$：同时包含 $d[l]$ 与 $d[r+1]$，净增量 $= +c - c = 0$！
3. 终止	加法满足线性叠加，做完 m 次操作后顺次求一遍前缀和即得终值。

本页要点

差分能成立的数学本质是“前缀和线性性”：先叠加增量差，最后再统一求前缀和。

模板代码（1/2）：一维差分标准模板

```cpp
int n, m; cin >> n >> m;
vector<long long> diff(n + 2, 0);        // 1. 开 n+2 格，防 r=n 时 diff[r+1] 越界
for (int i = 0; i < m; ++i) {
    int l, r; long long c; cin >> l >> r >> c;
    diff[l] += c;                        // 2. 区间左端生效
    diff[r + 1] -= c;                    // 3. 区间右端后一位撤销
}
long long cur = 0;
for (int i = 1; i <= n; ++i) {
    cur += diff[i];                      // 4. 累加差分数组即还原最终结果
    cout << cur << (i == n ? '\n' : ' ');
}
```

本页要点

一维差分口诀：数组大小 n+2，左端 +c，右端后一位 r+1 处 -c。

模板代码（2/2）：二维差分标准模板

```
vector<vector<long long>> d(n + 2, vector<long long>(m + 2, 0)); // 开 (n+2)x(m+2)
while (q--) {
    int x1, y1, x2, y2; long long c; cin >> x1 >> y1 >> x2 >> y2 >> c;
    d[x1][y1] += c;                    // 左上角开始生效
    d[x2 + 1][y1] -= c;                // 向下出界撤回
    d[x1][y2 + 1] -= c;                // 向右出界撤回
    d[x2 + 1][y2 + 1] += c;            // 右下外斜角被撤两次，加回一次
}
for (int i = 1; i <= n; ++i)
    for (int j = 1; j <= m; ++j) {
        d[i][j] += d[i - 1][j] + d[i][j - 1] - d[i - 1][j - 1]; // 原地还原
        cout << d[i][j] << (j == m ? '\n' : ' ');
    }
```

本页要点

二维差分口诀：(x1,y1) 与 (x2+1,y2+1) 加 c，另外两个角减 c。

上机验证：三组输入输出与反事实追问

一维差分上机验证（$n = 5$，初始全 0）

1. 常规交叉操作：输入 $[2,4] + 5$ 与 $[3,5] - 2 \implies$ 输出 `0 5 3 3 -2`。
2. 单点修改：输入 $[1,1] + 7 \implies$ 输出 `7 0 0 0 0`（验证 `l=r=1` 正常工作）。
3. 贴边全区间：输入 $[1,5] + 3 \implies$ 输出 `3 3 3 3 3`（验证 `diff[6]` 不越界）。

反事实追问：如果忘了写 `diff[r+1] -= c`，或者误写成 `diff[r] -= c` 会怎样？

忘了减会让 $+c$ 一路波及到数组末尾 n；误写成 `diff[r] -= c` 会把右端点 r 自己也提前撤掉，导致区间变成左闭右开 $[l, r)$！

本页要点

测差分程序一定要造一组 $r = n$ 和一组 $l = r$ 的数据，立刻能查出下标偏移与越界。

新手死穴对照：一维与二维差分四大翻车点

错误写法	评测结果	原因与正确做法
数组只开 n+1 大小	**RE**	修改到 $r = n$ 时访问 diff[n+1] 越界崩溃；必须开到 n+2（二维 (n+2)x(m+2)）。
初始数组非 0 却直接令 d[i]=0 叠加	**WA**	忘记把原数组先转成差分（或最后加上原数组 a[i]）；应初始化 d[i]=a[i]-a[i-1]。
边修改边在中间询问区间状态	**TLE / WA**	每次询问都必须重扫一遍还原，总复杂度退化成 $O(mn)$；混合操作需改用树状数组。
二维差分还原漏减 d[i-1][j-1]	**WA**	左上角矩形被重复加了两次；二维前缀和还原公式必须有 - d[i-1][j-1]。

本页要点

差分适用于“先批量修改，全部改完再统一还原输出”的离线批处理问题！

生活引入：怎么最快找出“消费和 ≥ 6 元”的最短连续几天？

具体小例子：5 天的消费记录 $a = [2, 1, 3, 2, 4]$ 元，目标门槛 $K = 6$

想找连续的一段 $[l, r]$，使这段消费总和 ≥ 6 元，且天数 $r - l + 1$ 越少越好：

- 把数字写在纸条上，左手食指指着左端 l，右手食指指着右端 r，两只手都从最左边出发。
- 右手一步步往右移，把新的一天加进窗口；当前窗口总和还不够 6 时，继续扩右手。
- 一旦右手走到第 3 个数（$[2, 1, 3]$ 和刚好 $= 6$），先记下长度 3，然后左手试着向右收缩一格扔掉 2；接着右手继续右移，右手永远不往回退，左手也只往右不往左！

本页要点

左右两根手指同向滑动维护一个连续窗口，每只手最多从头走到尾一次。

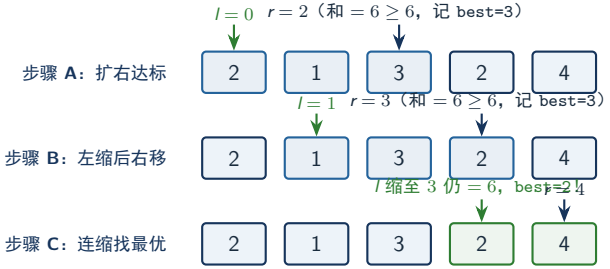
暴力为什么 TLE？用大白话讲清两个新术语

- 暴力做法估算：枚举所有起点 l 和终点 r（$1 \le l \le r \le n$），一共有 $n(n+1)/2$ 个连续区间。当 $n = 2 \times 10^5$ 时，区间数高达 2×10^{10} 个，即使配合前缀和也要 2×10^{10} 步 $\Longrightarrow$ 必 **TLE**！
- 新术语 **1**——单调性：因为所有数都非负（≥ 0），区间往右拉长，总和只会变大或不变；区间从左缩短，总和只会变小或不变。这种“只朝一个方向变化”的规律叫单调性。
- 新术语 **2**——双指针（尺取法）：利用单调性，用左指针 l 和右指针 r 像毛毛虫一样“右脚往前迈（扩大窗口），够格后左脚往前收（缩小窗口）”，两根指针都不走回头路。

本页要点

双指针不是盲目碰运气，它能成立的唯一前提是窗口指标具有“单调性”！

TikZ 图解：双指针 l, r 在真实格子数组上的滑动过程



本页要点

每次把 $a[r]$ 加进窗口，只要窗口总和 $\geq K$ 就循环更新答案并把 $a[l]$ 移出窗口、`l++`。

手算跟踪表：在 $a=[2,1,3,2,4]$, $K=6$ 上逐轮推演

初始 `left = 0`, `sum = 0`, `best = 6`（无穷大 $n+1$），条件 `r < 5`：

右端 r	前 sum	加 $a[r]$ 后判断 ≥ 6	收缩后状态	best
0（真）	0	$+2 \to 2 < 6$（不收缩）	$l=0$, $sum=2$	6
1（真）	2	$+1 \to 3 < 6$（不收缩）	$l=0$, $sum=3$	6
2（真）	3	$+3 \to 6 \geq 6$（记 3 扔 2）	$l=1$, $sum=4$	**3**
3（真）	4	$+2 \to 6 \geq 6$（记 3 扔 1）	$l=2$, $sum=5$	**3**
4（真）	5	$+4 \to 9 \geq 6$（缩 3 再缩 2）	$l=4$, $sum=4$	**2**
5（假）	4	$5 < 5$ 为假，终止	不更新	**2**

本页要点

$r=4$ 时 `while` 连缩两次（$l=2 \to 3 \to 4$），把长度 3 刷新成最优解 2。

正确性论证：为什么左指针 l 永远不用回头？

用三段式不变量证明算法不漏解且复杂度为 $O(n)$：

检验阶段	必须成立的事实
1. 初始化	$r = 0$ 开始前，$l = 0,\ sum = 0$，空窗口不含任何元素，和为 0。
2. 保持	当处理到 r 时，若某左端点 l_0 在更早的 $r' < r$ 时被扔掉，说明 $[l_0, r']$ 已达标；因元素非负，就算拉长到 r，长度 $r - l_0 + 1 > r' - l_0 + 1$，不可能更短，故 l_0 永不回头不会漏解。
3. 终止	$r = n$ 为假退出时遍历完所有右端。全程 r 加 n 次、l 最多加 n 次，总步数 $\leq 2n$，严格 $O(n)$！

本页要点

“两层循环仍只有 $O(n)$”的原因：外层 `r++` 走 n 步，内层 `l++` 全程合计最多也走 n 步。

模板代码：同向双指针最短窗口模板

```
int left = 0, best = n + 1;                 // 1. best 初值设为不可能达到的 n+1
long long sum = 0;                          // 2. 窗口和可能超过 int，用 long long
for (int right = 0; right < n; ++right) {
    sum += a[right];                        // 3. 右指针扩一格，把 a[right] 纳入窗口
    while (sum >= k) {                      // 4. 只要达标就记长度并缩左端（用 while!）
        best = min(best, right - left + 1);
        sum -= a[left++];                   // 5. 减掉 a[left] 后左指针右移一格
    }
}
cout << (best == n + 1 ? -1 : best) << '\n'; // 6. 从未更新过则无解输出 -1
```

本页要点

先减 a[left] 再移指针；收缩用 while 而非 if，才能一口气缩到最短。

上机验证：三组测试数据与反事实追问

三组测试数据

1. 常规：$a=[2,1,3,2,4]$，$K=6 \implies 2$（子段 $[3,4]$ 即 $2+4=6$）。
2. 无解：$a=[1,1,1]$，$K=9 \implies -1$（从未进 `while`）。
3. 单元素：$a=[10]$，$K=5 \implies 1$（$r=0$ 立即达标）。

反事实追问：若序列允许出现负数（如 $[5,-3,4,2]$），双指针还能用吗？

不能！负数会让“扩大和变大、缩小和变小”的单调性失效，必须改用“前缀和 + 单调队列”！

本页要点

用双指针前先确认：“区间元素是否非负，扩大/缩小是否满足单调性？”

新手死穴对照：双指针高频报错与定位

错误写法	评测结果	原因与正确做法
序列包含负数时直接套用双指针	**WA**	负数破坏单调性，左指针提前跳过或漏缩都会算错；含负数请用前缀和 + 单调队列。
把 `while (sum >= k)` 写成 `if (sum >= k)`	**WA**	每轮只缩一格，比如 $r = 4$ 需要连续缩两格才能拿到长度 2；必须用 `while`。
写成 `++left; sum -= a[left];`	**WA / RE**	减成了下一格元素，且 `left` 走到 n 时直接越界崩溃；务必先减 `a[left]` 再 `++left`。

本页要点

当题目需要求最长窗口（如“和 $\le K$ 的最长子段”）时，条件反过来：不合法时 `while` 缩左端，循环外记录答案。

例 1 | 区间和 [入门]：先读懂要求

大白话题意

给一个长 n 的整数序列 $a_1 \dots a_n$（$1 \le n, q \le 2 \times 10^5$，$|a_i| \le 10^9$）。随后有 q 次询问，每次给 $[l, r]$，输出 $a_l + \cdots + a_r$ 的总和。

1. 输入：第 1 行 n, q；第 2 行 $a_1 \dots a_n$；接下来 q 行每行 l, r。
2. 处理：构造 1-based 前缀和 $S[i] = S[i-1] + a_i$，每次计算 $S[r] - S[l-1]$。
3. 输出：q 行，每行一个整数（答案可达 $\pm 2 \times 10^{14}$）。

原题样例

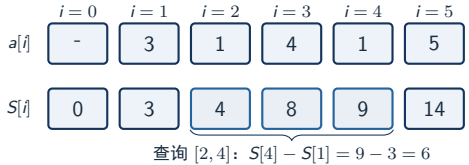
输入：`5 3 / 3 1 4 1 5 / 2 4 / 1 5 / 3 3` $\Longrightarrow$ 输出：`6 / 14 / 4`。

本页要点

先看数据范围：$n, q \le 2 \times 10^5$，$|a_i| \le 10^9$，必须用 $O(n+q)$ 算法与 `long long`。

例 1 | 怎么想到的：从暴力 TLE 到一维前缀和建模

- 暴力做法：每次从 l 循环加到 r，最坏操作数 $n \times q = 4 \times 10^{10}$ 次 $\Longrightarrow$ 必 **TLE**。
- 关键性质：全程无修改操作；任意连续段 $[l, r]$ 都等于“前 r 项总和减去前 $l-1$ 项总和”。



本页要点

“静态多次区间求和”是标准的一维前缀和信号，直接套用 1-based 前缀和模板。

例 1 | 样例手推表（$a=[3,1,4,1,5]$，$S=[0,3,4,8,9,14]$）

询问 $[l,r]$	右端 $S[r]$	减去 $S[l-1]$	计算结果	原序列手工验证
$[2,4]$	$S[4]=9$	$S[1]=3$	$9-3=\mathbf{6}$	$a_2+a_3+a_4=1+4+1=6$ 吻合
$[1,5]$	$S[5]=14$	$S[0]=0$	$14-0=\mathbf{14}$	$3+1+4+1+5=14$ 吻合
$[3,3]$	$S[3]=8$	$S[2]=4$	$8-4=\mathbf{4}$	单个元素 $a_3=4$ 吻合

本页要点

手推重点核对 $[1,5]$ 用到了 $S[0]=0$，以及单点 $[3,3]$ 的答案不等于 0。

例 1 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 选定容器类型	`vector<long long> s(n+1, 0)`	2×10^{14}，用 `int` 溢出吗？
2. 边读边递推	`s[i] = s[i-1] + x`	是否从 $i = 1$ 开始且 `s[0]=0`？
3. 回答每次询问	`s[r] - s[l-1]`	左端减的是 `l-1` 而不是 `l` 吗？

相对基础模板改了哪里？

- 后续无需访问原数组，故边读 `x` 边累加进 `s[i]`，省掉一半空间；
- 输入达 2×10^5 行，开头加上 `ios::sync_with_stdio(false); cin.tie(nullptr);`，换行用 `'\n'`。

本页要点

边读边算省空间，关同步加 `'\n'` 保 I/O 不超时。

例 1 | 核心代码（完整程序：code/range_sum.cpp）

```cpp
int n, q; cin >> n >> q;
vector<long long> s(n + 1, 0);      // 开 n+1 格且 s[0]=0 充当左边界垫脚石
for (int i = 1; i <= n; ++i) {
    long long x; cin >> x;
    s[i] = s[i - 1] + x;            // 前缀和递推，无需保存原数组 a
}
while (q--) {
    int l, r; cin >> l >> r;
    cout << s[r] - s[l - 1] << '\n'; // O(1) 回答区间 [l, r] 之和
}
```

本页要点

上机前在 main() 开头补上 ios::sync_with_stdio(false); cin.tie(nullptr);，逻辑与源码一致。

例 1 | 易错对照与三组自测边界

常见错误对照

- **WA（溢出）**：s 或 x 开 int 变负数 $\rightarrow$ 一律用 long long。
- **WA（差 1）**：误写 s[r] - s[l] 减掉 a_l $\rightarrow$ 左端减 l - 1；用 endl 易 TLE $\rightarrow$ 换 '\n'。

三组自测边界数据

1. 全负数：3 1 / -5 -2 -3 / 1 3 $\Longrightarrow$ -10。
2. 超 int 大数：2 1 / 1000000000 1000000000 / 1 2 $\Longrightarrow$ 2000000000。
3. 最小规模 $n = 1$：1 1 / 42 / 1 1 $\Longrightarrow$ 42（验证 s[1]-s[0]）。

本页要点

拿“全负数 + 超 int + $n = 1$”三组数据测通再提交。

例 2 | 两种区间求和（CF 433B）[基础]：先读懂要求

大白话题意

n 块石头价格为 $v_1 \dots v_n$（$1 \le n, m \le 10^5$，$1 \le v_i \le 10^9$）。有 m 次询问 $(type, l, r)$：$type = 1$ 求原顺序 $[l, r]$ 之和；$type = 2$ 求从小到大排序后 $[l, r]$ 之和。

1. 输入：第 1 行 n；第 2 行 $v_1 \dots v_n$；第 3 行 m；随后 m 行每行 $type, l, r$。
2. 输出：m 行，每行一个整数表示对应区间和。

原题样例

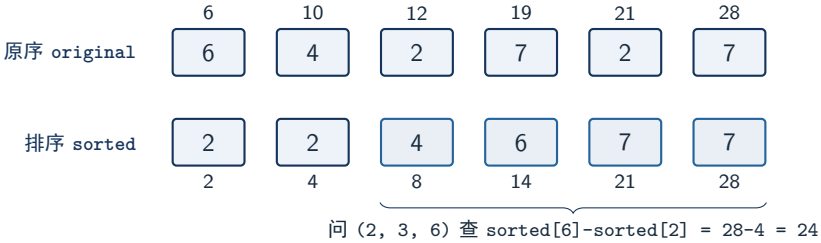
输入：`6 / 6 4 2 7 2 7 / 3 / 2 3 6 / 1 3 4 / 1 1 6` $\Longrightarrow$ 输出：`24 / 9 / 28`。

本页要点

看清题意：$type = 2$ 只是查询“排序后的静态版本”，并不真正修改原数组。

例 2 | 怎么想到的：双前缀和并行建模

- 暴力做法：每次 $type = 2$ 都排序再累加，耗时 $O(mn \log n) \approx 1.7 \times 10^{11} \implies$ 必 **TLE**。
- 核心洞察：排序结果固定不变！提前造两套前缀和：`original` 与 `sorted`。



本页要点

排序只需做一次 $O(n \log n)$，两套前缀和各自 $O(n)$ 预处理、$O(1)$ 查询。

例 2 | 样例手推表（原序 $[6,4,2,7,2,7]$ vs 排序 $[2,2,4,6,7,7]$）

两套前缀和：`original = [0, 6, 10, 12, 19, 21, 28]`，`sorted = [0, 2, 4, 8, 14, 21, 28]`。

$(type, l, r)$	选用数组	计算算式	答案	手工核对
2 3 6	`sorted`	`sorted[6]-sorted[2]`	$28-4=$ **24**	$4+6+7+7=24$
1 3 4	`original`	`original[4]-original[2]`	$19-10=$ **9**	$2+7=9$
1 1 6	`original`	`original[6]-original[0]`	$28-0=$ **28**	全部 6 块总和 28

本页要点

排序数组 `b` 时，0 号位存的是占位 0，绝对不能把 `b[0]` 也参与排序！

例 2 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 拷贝并排序	`b = a; sort(b.begin()+1, b.end());`	是否跳过 `b[0]` 从 +1 排？
2. 构双前缀和	`original[i]` 与 `sorted[i]`	两条前缀和都用 `long long` 吗？
3. 引用查询	`const auto &p = ...`	加引用 `&p` 防拷贝超时了吗？

相对基础模板改了哪里？

- 同时维护 `original` 与 `sorted` 两条前缀和；
- 用 `const auto &p = (type == 1 ? original : sorted);` 取引用，避免拷贝十万元素。

本页要点

1-based `vector` 排序铁律：`sort(b.begin() + 1, b.end())`。

例 2 | 核心代码（1/2）：拷贝排序与双前缀和（code/two_range_sums.cpp）

```
int n; cin >> n;
vector<long long> a(n + 1), original(n + 1), sorted(n + 1);
for (int i = 1; i <= n; ++i) cin >> a[i];
vector<long long> b = a;            // 拷贝一份专门排序，保持原数组 a 不动
sort(b.begin() + 1, b.end());       // 1-based 下标必须从 +1 开始排，跳过 b[0]
for (int i = 1; i <= n; ++i) {
    original[i] = original[i - 1] + a[i]; // 原顺序前缀和
    sorted[i] = sorted[i - 1] + b[i];     // 排序后前缀和
}
```

本页要点

original[0] 与 sorted[0] 由 vector 默认初始化为 0，无需额外赋值。

例 2 | 核心代码（2/2）：引用选择与 $O(1)$ 查询（code/two_range_sums.cpp）

```
    int m; cin >> m;
    while (m--) {
        int type, l, r;
        cin >> type >> l >> r;
        const auto &p = type == 1 ? original : sorted; // 常量引用，O(1) 选择数组
        cout << p[r] - p[l - 1] << '\n';
    }
    return 0;
}
```

本页要点

若漏掉 `&` 写成 `auto p = ...`，每次询问都会把 10^5 个元素整串拷贝，变成 $O(nm)$ TLE！

例 2 | 易错对照与三组自测边界

常见错误对照

- **WA**（b[0]=0 混入排序）：`sort(b.begin(), b.end())` 把 0 排进序列 $\rightarrow$ 用 `b.begin() + 1`。
- **TLE**（漏写 &）：`auto p = ...` 触发值拷贝 $\rightarrow$ 写成 `const auto &p`；`int` 溢出 $\rightarrow$ `long long`。

三组自测边界数据

1. 最小 $n = 1$：`1 / 7 / 2 / 1 1 1 / 2 1 1` $\Longrightarrow$ `7 / 7`。
2. 完全逆序：`4 / 4 3 2 1 / 2 / 1 1 2 / 2 1 2` $\Longrightarrow$ `7 / 3`（排完前两小 $1 + 2 = 3$）。
3. 全等大数：`3 / 1000000000 1000000000 1000000000 / 1 / 2 1 3` $\Longrightarrow$ `3000000000`。

本页要点

用“逆序序列测 $type = 2$”能立即检验排序起点和前缀和是否都正确。

例 3 | 和为 K 的倍数 [提高]：先读懂要求

大白话题意

给长 n 的正整数序列 $a_1 \dots a_n$（$1 \le n \le 2 \times 10^5$，$1 \le a_i \le 10^9$）与正整数 K（$1 \le K \le 10^5$）。统计有多少个非空连续子区间 $[l, r]$ 的和恰好是 K 的倍数。

1. 输入：第 1 行 n, K；第 2 行 n 个正整数 $a_1 \dots a_n$。
2. 处理：统计满足 $(a_l + \dots + a_r) \bmod K = 0$ 的区间 $[l, r]$ 总数。
3. 输出：一个整数（答案最多可达 $n(n+1)/2 \approx 2 \times 10^{10}$）。

原题样例

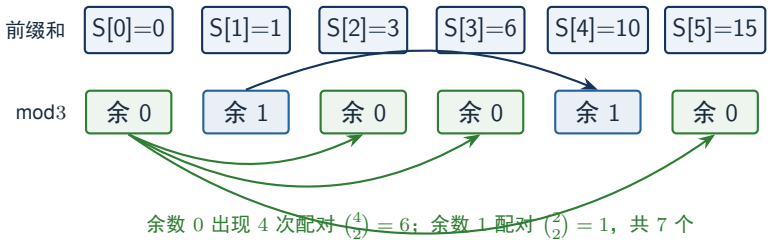
输入：`5 3 / 1 2 3 4 5` $\Longrightarrow$ 输出：`7`。

本页要点

答案数量可达 2×10^{10}，`ans` 必须开 `long long`！

例 3 | 怎么想到的：同余转化与余数桶图解

- 暴力做法：枚举 $[l, r]$ 即使有前缀和仍需 $O(n^2) \approx 2 \times 10^{10}$ 步 $\Longrightarrow$ 必 **TLE**。
- 同余前缀：$(S[r] - S[l-1]) \bmod K = 0 \iff S[r] \bmod K = S[l-1] \bmod K$；只需数前面有几个相同余数！



本页要点

用桶数组 `cnt[rem]` 实时统计同余前缀个数，将 $O(n^2)$ 降为 $O(n)$！

例 3 | 样例手推表（$a=[1,2,3,4,5]$, $K=3$）

初始化 `cnt = [1, 0, 0]`（`cnt[0]=1` 代表空前缀 $S[0]=0$），`s = 0`，`ans = 0`：

当前 i	$a[i]$	新余数 $s=(s+a[i]) \bmod 3$	查旧 `cnt[s]`	新 `ans`	新 `cnt[0,1,2]`
初始	-	0（空前缀 $S[0]$）	-	0	`[1, 0, 0]`
1	1	$(0+1) \bmod 3=1$	0	0	`[1, 1, 0]`
2	2	$(1+2) \bmod 3=0$	1	1	`[2, 1, 0]`
3	3	$(0+3) \bmod 3=0$	2	3	`[3, 1, 0]`
4	4	$(0+4) \bmod 3=1$	1	4	`[3, 2, 0]`
5	5	$(1+5) \bmod 3=0$	3	**7**	`[4, 2, 0]`

本页要点

顺序必须是“先查旧 `cnt[s]` 累加到 `ans`，再 `++cnt[s]`”，最终得 7。

例 3 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 初始化余数桶	`cnt[k]={0}; cnt[0]=1;`	计入空前缀 $S[0]=0$ 的 1 次了吗？
2. 滚动累加取模	`s = (s + x) % k`	每步取模防 s 累加溢出了吗？
3. 先查旧桶再入桶	`ans += cnt[s]; ++cnt[s];`	顺序没反吧？ans 开 long long 吗？

相对基础模板改了哪里？

- 不必存整张前缀和表 `s[0..n]`，改用单个滚动变量 `s` 边读边对 K 取模；
- 增设余数桶 `cnt[0..K-1]` 并预置 `cnt[0]=1`，一遍扫描直接累计合法区间对数。

本页要点

空前缀 `S[0]=0` 对应从第 1 个数起步的区间，漏掉 `cnt[0]=1` 必丢答案！

例 3 | 核心代码（完整程序：code/mod_k_subarrays.cpp）

```
int n, k; cin >> n >> k;
vector<long long> cnt(k, 0);
cnt[0] = 1;                          // 空前缀 S[0]=0 余数为 0，必须先计 1 次！
long long s = 0, ans = 0;            // ans 最高可达 2e10，必须用 long long
for (int i = 0; i < n; ++i) {
    long long x; cin >> x;
    s = (s + x) % k;                 // a[i]>0 直接取模（含负数写 ((s+x)%k+k)%k）
    ans += cnt[s];                   // 与前面每个同余前缀都能组成合法区间
    ++cnt[s];                        // 再把当前前缀余数登记入桶
}
cout << ans << '\n';
```

本页要点

含负数序列取模必须写成 `((s + x) % k + k) % k`，防止负下标越界崩溃！

例 3 | 易错对照与三组自测边界

常见错误对照

- **WA**（漏写 `cnt[0]=1`）：丢失所有以 a_1 开头的合法区间。
- **WA**（先 `++cnt[s]` 再累加）：把空区间算进去，答案多出 n；`ans` 用 `int` 易溢出。

三组自测边界数据

1. 最小单元素：`1 3 / 3` $\Longrightarrow$ `1`（检验 `cnt[0]=1`）。
2. 全不整除：`3 5 / 1 1 1` $\Longrightarrow$ `0`。
3. 全为 K 的倍数：`3 3 / 3 6 9` $\Longrightarrow$ `6`（$\binom{4}{2}=6$）。

本页要点

用 `1 3 / 3` 测一下，立刻能验证 `cnt[0]=1` 和配对顺序有没有写反。

例 4 | 区间加 [入门]：先读懂要求

大白话题意

长 n 初始全 0 的数组，执行 m 次操作（$1 \le n, m \le 2 \times 10^5$，$|c| \le 10^9$），每次给出 l, r, c，把 $[l, r]$ 内所有数加上 c，最后输出完整数组。

1. 输入：第 1 行 n, m；接下来 m 行每行三个整数 l, r, c。
2. 处理：用一维差分改两个端点，全部改完后做一遍前缀和还原。
3. 输出：一行 n 个整数，单空格分隔，行末换行。

原题样例

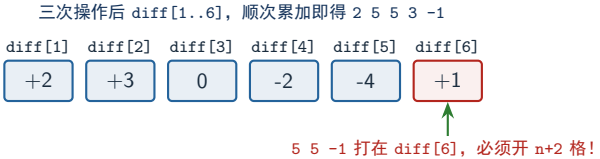
输入：`5 3 / 1 3 2 / 2 4 3 / 5 5 -1` $\Longrightarrow$ 输出：`2 5 5 3 -1`。

本页要点

“只改不查、最后输出”是一维差分最直接的应用场景。

例 4 | 怎么想到的：从暴力到差分端点打标图解

- 暴力做法：每次循环改 $a[l \dots r]$，总操作数达 $4 \times 10^{10} \implies$ 必 **TLE**。
- 差分建模：每条指令只修改 `diff[l] += c` 与 `diff[r+1] -= c` 两格。



本页要点

第 3 次操作 `5 5 -1` 的撤回点在 `diff[6]`，说明数组必须开到 `n+2`。

例 4 | 样例手推表（$n=5, m=3$，全部操作后还原）

操作 / 步	$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	标记说明 / 还原值
初始全 0	0	0	0	0	0	0	6 格全初始化为 0
1 3 2	+2	0	0	−2	0	0	改 d[1]+2，d[4]-2
2 4 3	+2	+3	0	−2	−3	0	改 d[2]+3，d[5]-3
5 5 -1	+2	+3	0	−2	−4	+1	改 d[5]-1，d[6]+1
$i=1..5$ 还原	**2**	**5**	**5**	**3**	**-1**	不输出	得 2 5 5 3 -1

本页要点

还原过程：$0+2=2 \to 2+3=5 \to 5+0=5 \to 5-2=3 \to 3-4=-1$，精准吻合！

例 4 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 开安全差分数组	`vector<long long> diff(n+2, 0)`	长度开到 `n+2` 且 `long long` 吗？
2. 处理 m 次修改	`diff[l] += c; diff[r+1] -= c;`	右端点后一位加了 `+1` 吗？
3. 前缀和还原输出	`cur += diff[i];`	行末是换行而非多余空格吗？

相对基础模板改了哪里？

- 初始全 0，用变量 `cur` 边累加边打印即可，省去结果数组；
- 输出控制第 n 项后换行 `'\n'`、其余项后输出空格（大数据建议补关同步）。

本页要点

行末不能多空格，用 `if (i == n) cout << '\n'; else cout << ' ';` 最稳妥。

例 4 | 核心代码（1/2）：读入与端点打标（code/range_add.cpp）

```cpp
int main() {
    int n, m;
    cin >> n >> m;
    vector<long long> diff(n + 2, 0);   // 开 n+2 格，防 r=n 时 diff[r+1] 越界 RE
    for (int i = 0; i < m; ++i) {
        int l, r;
        long long c;
        cin >> l >> r >> c;
        diff[l] += c;                   // 区间起点开始生效
        diff[r + 1] -= c;               // 区间终点之后一格撤销
    }
```

本页要点

提交 2×10^5 级别大数据时，可在 `cin >> n >> m;` 前加上关同步提速。

例 4 | 核心代码（2/2）：前缀和还原与输出（code/range_add.cpp）

```
    long long cur = 0;
    for (int i = 1; i <= n; ++i) {
        cur += diff[i];                    // 对差分数组求前缀和，即原数组当前项的值
        cout << cur;
        if (i == n) cout << '\n';          // 最后一项输出换行
        else cout << ' ';                  // 其余项用单个空格分隔
    }
    return 0;
}
```

本页要点

只用一个变量 cur 滚动累加，代码逻辑与源码 code/range_add.cpp 完全一致。

例 4 | 易错对照与三组自测边界

常见错误对照

- **RE**（开 n+1）：$r = n$ 时访问 `diff[n+1]` 崩溃 $\rightarrow$ 开 n+2。
- **WA**（`int` 溢出 / 漏 +1）：统一 `long long`；右端撤回写成 `diff[r+1] -= c`。

三组自测边界数据

1. 单元素贴边：`1 1 / 1 1 -5` $\Longrightarrow$ `-5`（验证 `diff[2]` 不越界）。
2. 正负抵消：`4 2 / 1 4 10 / 1 4 -10` $\Longrightarrow$ `0 0 0 0`。
3. 超大累加：`2 2 / 1 2 1000000000 / 1 2 1000000000` $\Longrightarrow$ `2000000000 2000000000`。

本页要点

用 `1 1 / 1 1 -5` 测 $n = 1$ 极限边界，不崩溃说明 n+2 生效。

例 5 | 子矩阵加 [基础]：先读懂要求

大白话题意

一个 n 行 m 列初始全 0 的矩阵（$1 \le n, m \le 1000$, $1 \le q \le 2 \times 10^5$, $|c| \le 10^9$）。依次执行 q 次操作 (x_1, y_1, x_2, y_2, c)，把对应子矩阵内所有元素加 c，最后输出整个矩阵。

1. 输入：第 1 行 n, m, q；接下来 q 行每行 x_1, y_1, x_2, y_2, c。
2. 处理：用二维差分在 4 个角点打标，最后原地二维前缀和还原。
3. 输出：n 行，每行 m 个整数，空格分隔。

原题样例

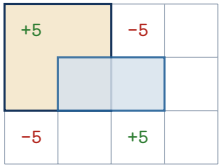
输入：`2 3 2 / 1 1 2 2 5 / 2 2 2 3 -2` $\Longrightarrow$ 输出：`5 5 0` 与 `5 3 -2`。

本页要点

$n, m \le 1000$ 但 $q \le 2 \times 10^5$，每次修改最多覆盖 10^6 格，绝对不能硬循环。

例 5 | 怎么想到的：二维四点标记建模图解

- 暴力做法：每次双层循环遍历子矩阵，运算量达 2×10^{11} 次 $\Longrightarrow$ 必 **TLE**。
- 二维差分：每次仅改 4 个角点（$O(1)$），最后做一次 $O(nm)$ 二维前缀和还原。



操作 1：(1,1)-(2,2)+5 边界落点

5	5	0	外
5	3	-2	外
外	外		

求前缀和后输出前 2×3 格

本页要点

当 $x_2 = 2, y_2 = 3$ 时会写入第 3 行第 4 列，所以矩阵必须开 $(n+2) \times (m+2)$。

例 5 | 样例手推表（2×3 矩阵先后加 $(1,1) \sim (2,2) + 5$ 与 $(2,2) \sim (2,3) - 2$）

行号 i	列 1	列 2	列 3	列 4（越界）	该行说明 / 还原过程
两次四点打标后的差分矩阵 $d[1\ldots3][1\ldots4]$					
1	$+5$	0	-5	0	$(1,1)+5,\ (1,3)-5$
2	0	-2	0	$+2$	$(2,2)-2,\ (2,4)+2$
3	-5	$+2$	$+5$	-2	$(3,1)-5, (3,2)+2, (3,3)+5, (3,4)-2$
原地前缀和还原后的输出 $1\ldots2$ 行、$1\ldots3$ 列					
1	**5**	**5**	**0**	-	$5,\ 5+0=5,\ 5-5=0$
2	**5**	**3**	**-2**	-	$5,\ 5+5-5-2=3,\ 0+3-5=-2$

本页要点

核对 $(2,2)$：$5+5-5+(-2)=3$，加减两次的左上角只扣了一遍！

例 5 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 开安全二维矩阵	`d(n+2, ... m+2 ...)`	行列都加 2 且 `long long` 吗？
2. 四点标记更新	`+c, -c, -c, +c`	四个角的坐标和符号配对了吗？
3. 原地二维前缀和	`d[i][j] += d[i-1][j] + d[i][j-1] - d[i-1][j-1]`	别漏掉减 `d[i-1][j-1]`！

相对基础模板改了哪里？

- 矩阵维度扩成 $(n+2)\times(m+2)$；
- 直接在 `d` 上原地累加还原最终矩阵，省去第二个 1000×1000 矩阵。

本页要点

用 `d` 原地累加时，行和列必须严格从小到大顺序遍历。

例 5 | 核心代码（1/2）：四点标记更新（code/matrix_diff.cpp）

```cpp
int n, m, q; cin >> n >> m >> q;
vector<vector<long long>> d(n + 2, vector<long long>(m + 2, 0));
while (q--) {
    int x1, y1, x2, y2; long long c;
    cin >> x1 >> y1 >> x2 >> y2 >> c;
    d[x1][y1] += c;                  // 左上角生效
    d[x2 + 1][y1] -= c;              // 下边界外撤回
    d[x1][y2 + 1] -= c;              // 右边界外撤回
    d[x2 + 1][y2 + 1] += c;          // 右下外斜角补偿加回
}
```

本页要点

加 c 的两角同奇偶（全不加 1 或全加 1），减 c 的两角只在一维上加 1。

例 5 | 核心代码（2/2）：原地前缀和还原（code/matrix_diff.cpp）

```
    for (int i = 1; i <= n; ++i) {
        for (int j = 1; j <= m; ++j) {
            d[i][j] += d[i - 1][j] + d[i][j - 1] - d[i - 1][j - 1]; // 还原前缀和
            cout << d[i][j];
            if (j == m) cout << '\n';   // 每行末尾换行
            else cout << ' ';           // 同行元素用空格隔开
        }
    }
    return 0;
}
```

本页要点

完整源码开头已包含 `ios::sync_with_stdio(false); cin.tie(nullptr);`，确保百万级输出不超时。

例 5 | 易错对照与三组自测边界

常见错误对照

- **RE**（开 `n+1` 乘 `m+1`）：操作贴边 $x_2 = n$ 或 $y_2 = m$ 立即越界崩溃。
- **WA**（四点符号或还原公式出错）：漏减 `d[i-1][j-1]` 会斜向重复累加。

三组自测边界数据

1. 最小 1×1：`1 1 1 / 1 1 1 1 -9` $\Longrightarrow$ `-9`。
2. 全矩阵覆盖：`2 2 1 / 1 1 2 2 4` $\Longrightarrow$ `4 4 / 4 4`。
3. 正负抵消：`2 2 2 / 1 1 2 2 5 / 1 1 2 2 -5` $\Longrightarrow$ 全 `0`。

本页要点

造一组覆盖到 (n, m) 右下角的样例，能同时验出越界和四点符号是否都对。

例 6 | 最短区间 [基础]：先读懂要求

大白话题意

给长 n 的正整数序列 $a_1 \dots a_n$（$1 \le n \le 2 \times 10^5$，$1 \le a_i \le 10^9$）和目标和 K（$1 \le K \le 10^{18}$）。求元素和至少为 K 的非空连续区间的最短长度；不存在则输出 `-1`。

1. 输入：第 1 行 n, K；第 2 行 n 个正整数 $a_1 \dots a_n$。
2. 处理：利用正数单调性，用同向双指针维护滑动窗口 $[left, right]$。
3. 输出：一个整数，表示最短长度或 `-1`。

原题样例

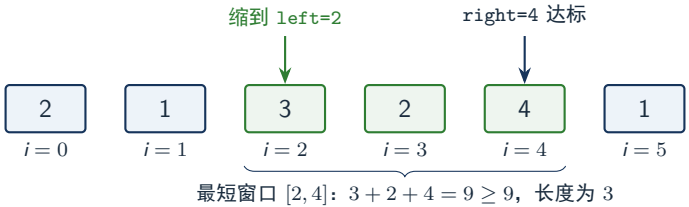
输入：`6 9 / 2 1 3 2 4 1` $\Longrightarrow$ 输出：`3`（区间 $[3,5]$ 即 $3+2+4=9$ 最短）。

本页要点

数据范围 $1 \le K \le 10^{18}$，所以 `k` 和 `sum` 都必须用 `long long`！

例 6 | 怎么想到的：正数单调性与滑窗建模图解

- 暴力做法：枚举所有 $[l, r]$ 共 2×10^{10} 个区间 $\Longrightarrow$ 必 **TLE**。
- 关键性质：$a_i \geq 1$ 全正！窗口右扩和变大、左缩和变小，被抛弃的左端永不回头。



本页要点

看到“正整数序列 + 区间和 $\geq K$ 最短长度”，第一反应就是同向双指针！

例 6 | 样例手推表（$a=[2,1,3,2,4,1]$，$K=9$，0-based）

初始 `left = 0`，`sum = 0`，`best = 7`（$n+1$），条件 `right < 6`：

右端 r	前 `sum`	加 $a[r]$ 后判断 ≥ 9	收缩后状态	`best`
0（真）	0	$+2 \to 2 < 9$	$l=0$，$sum=2$	7
1（真）	2	$+1 \to 3 < 9$	$l=0$，$sum=3$	7
2（真）	3	$+3 \to 6 < 9$	$l=0$，$sum=6$	7
3（真）	6	$+2 \to 8 < 9$	$l=0$，$sum=8$	7
4（真）	8	$+4 \to 12 \geq 9$（连缩 $l=0,1,2$）	$l=3$，$sum=6$	**3**
5（真）	6	$+1 \to 7 < 9$	$l=3$，$sum=7$	3
6（假）	7	条件 $6<6$ 为假，终止	不更新	**3**

本页要点

$r=4$ 时 `while` 依次记录长度 $5 \to 4 \to 3$，直到扔掉 $a[2]=3$ 才跌回 $6<9$ 停下。

例 6 | 核心步骤与相对模板的改动

按顺序计算	纸上得到	核对问题
1. 读入 n 与 K	`long long k; cin >> n >> k;`	$K \le 10^{18}$，用 `long long` 了吗？
2. 扩右缩左滑窗	`while (sum >= k)` 更新 `best`	收缩用 `while` 且先减 `a[left]` 吗？
3. 判断无解输出	`best == n + 1 ? -1 : best`	不够 K 时输出 `-1` 了吗？

相对基础模板改了哪里？

- 把目标阈值 `k` 声明为 `long long`；
- 源码 `shortest_window.cpp` 用 0-based 存数组，区间长度是 `right - left + 1`。

本页要点

“阈值 K 可达 10^{18}”是本题最隐蔽的陷阱，读题时就要圈出来用 `long long`。

例 6 | 核心代码（完整程序：code/shortest_window.cpp）

```cpp
vector<long long> a(n);
for (int i = 0; i < n; ++i) cin >> a[i];
int left = 0, best = n + 1;          // best 初值设为不可能达到的 n+1
long long sum = 0;                   // 窗口累加和同样用 long long
for (int right = 0; right < n; ++right) {
    sum += a[right];                 // 扩大窗口右端
    while (sum >= k) {               // 窗口和达标时尝试尽量缩小左端
        best = min(best, right - left + 1);
        sum -= a[left];              // 先减再移动 left，防止少减或多减
        ++left;
    }
}
if (best == n + 1) cout << -1 << '\n';
else cout << best << '\n';
```

本页要点

逻辑与源码 code/shortest_window.cpp 一致，两指针合计最多移动 $2n$ 步。

例 6 | 易错对照与三组自测边界

常见错误对照

- **WA**（k 用 int）：$K \le 10^{18}$ 读入即溢出成负数 $\rightarrow$ 用 long long。
- **WA**（while 写成 if / 忘判 -1）：无法连缩多格或无解时错输出 n+1。

三组自测边界数据

1. 全部相加仍无解：3 10 / 1 2 3 $\Longrightarrow$ -1。
2. 单元素立刻达标：2 5 / 10 1 $\Longrightarrow$ 1。
3. 超大 K 测试：2 2000000000000 / 1000000000000 1000000000000 $\Longrightarrow$ 2。

本页要点

用大于 2×10^9 的 K 本地自测一次，能立刻抓出忘记改 long long 的问题。

常见错误对照与评测定位指南

题目 / 场景	常见错误写法	评测结果与正确做法
例 1 区间和	开 `int` 存和；查询写 `s[r] - s[l]`	**WA**：改 `long long`；左端减 `s[l-1]`
例 2 两种区间求和	`sort(b.begin(), b.end())`	**WA**：1-based 应写 `b.begin()+1`
例 3 和为 K 的倍数	漏 `cnt[0]=1`；先加桶后计数	**WA**：空前缀占 1 次；先统计再入桶
例 4/5 差分系列	数组只开 `n+1` 大小	**RE**：统一开 `n+2`（二维 `(n+2)x(m+2)`）
例 6 最短区间	`k` 用 `int`；`while` 写成 `if`	**WA**：`k` 开 `long long`；必须用 `while`

- 定位口诀：**CE** 查头文件分号；**PE** 查多余空格；**TLE** 查 $< 10^8$ 与关同步；**RE** 查 `n+2`；**WA** 查 `long long` 与 $l-1$。

本页要点

看到 **WA** 先查 `long long` 和减 1，**RE** 查 `n+2`，**TLE** 算总操作数！

提交前检查清单（六必查）

1. 复杂度红线：代入题目最大 n, q, m 估算总操作步数，确认远低于 10^8（$n \geq 10^5$ 禁写 $O(n^2)$）？
2. 防溢出类型：前缀和、差分还原值、子区间总数、阈值 K 是否全用了 `long long`？
3. 前缀和边界：是否采用 1-based 下标且置 `S[0] = 0`？查询写的是 `S[r] - S[l - 1]` 吗？
4. 差分空间与时机：差分数组是否至少开到 `n + 2`（二维 `(n+2)x(m+2)`）？是否全改完才统一还原？
5. 双指针单调性：是否确认过元素非负（或条件具有单调性），收缩用了 `while` 且先减再移动？
6. 快读快写：输入输出达到 10^5 行时，是否写了 `ios::sync_with_stdio(false); cin.tie(nullptr);` 和 `'\n'`？

考场最后一步

先用 $n = 1$ 最小样例、全负数/全零样例、以及超 `int` 大数样例各跑一遍再提交！

本章全部练习与完成标准（主例题 6 题）

1. **[入门]** 区间和（题目/补充题目/前缀和/区间和 | 源码 `code/range_sum.cpp`）：熟练 1-based 前缀和。
2. **[基础]** 两种区间求和（CF 433B 课堂版 | 源码 `code/two_range_sums.cpp`）：掌握排序后前缀和与引用优化。
3. **[提高]** 和为 K 的倍数（题目/补充题目/前缀和/和为 K 的倍数 | 源码 `code/mod_k_subarrays.cpp`）：理解同余前缀与 `cnt[0]=1`。
4. **[入门]** 区间加（题目/补充题目/差分/区间加 | 源码 `code/range_add.cpp`）：一维差分打标与还原。
5. **[基础]** 子矩阵加（题目/补充题目/差分/子矩阵加 | 源码 `code/matrix_diff.cpp`）：二维差分四点标记与原地还原。
6. **[基础]** 最短区间（题目/补充题目/双指针/最短区间 | 源码 `code/shortest_window.cpp`）：非负序列同向双指针最短窗口。

完成标准

能独立估算任何题目的时间复杂度红线；不看讲义默写一维/二维前缀和、差分与双指针模板；独立 AC 全部 6 题。

同专题延伸练习题单（巩固提升）

以下延伸题均位于 题目/补充题目/ 对应目录，做完 6 道主例题后按顺序刷通：

题目名称	难度	核心思路提要
子矩阵求和	[基础]	构造 $(n+1)\times(m+1)$ 二维前缀和，用四项容斥 $O(1)$ 查矩形和。
推荐温度统计	[基础]	先一维差分还原覆盖次数 $\geq k$ 的合法点，再做前缀和统计。
搭建积木	[提高]	从相邻高度差 $d[i]=h[i]-h[i-1]$ 入手，累加所有正差值即得答案。
两数之和	[入门]	排序后用相向双指针逼近目标和，大了右移，小了左移。
和至少为 K 的连续区间	[基础]	正数滑窗：当 $[l,r]$ 达标时，从 r 延展到 n 的所有后缀全合法。
最长无重复子串	[基础]	用桶记录字符次数，右端重复时右移左指针直到窗口内无重复。

本页要点

先攻克主例题 6 题，再用“推荐温度统计”和“两数之和”检验迁移能力。

课后安排、下一章预告与本章一句话

- 本周复习：在空白纸上自己画出二维前缀和与二维差分四个角点坐标与正负号图，口述一遍为什么不越界。
- 代码实战：把 `code/` 下 6 份程序逐一手动复现，并造至少 3 组边界数据（$n = 1$、全负/全零、超 `int`）跑通。
- 下一章预告：第 2 章《排序与逆序对》——深入学习多关键字自定义比较器、严格弱序规范，以及用归并排序在 $O(n \log n)$ 统计逆序对！

本章一句话

先算红线防超时；前缀和用一次预处理换 $O(1)$ 查和，差分把区间改变成两点加，双指针靠单调性让两指针永不回头。

谢谢！