Day 4 题解

Qingyu

Hailiang Foreign Language School

April 20, 2023

题意

有 n 件质量均匀分布的物品，可以进行不超过 2 次切割操作，将切割完成后的物品分为两个集合，最小化：

$$\varepsilon = \frac{|\sum_{i\in S} V_i - \sum_{i\in T} V_i|}{\sum_{i=1}^{n+k} V_i} + \frac{|\sum_{i\in S} m_i - \sum_{i\in T} m_i|}{\sum_{i=1}^{n+k} m_i}$$

题意

有 n 件质量均匀分布的物品，可以进行不超过 2 次切割操作，将切割完成后的物品分为两个集合，最小化：

$$\varepsilon = \frac{|\sum_{i\in S} V_i - \sum_{i\in T} V_i|}{\sum_{i=1}^{n+k} V_i} + \frac{|\sum_{i\in S} m_i - \sum_{i\in T} m_i|}{\sum_{i=1}^{n+k} m_i}$$

要求输出任意一组解。

题意

有 n 件质量均匀分布的物品，可以进行不超过 2 次切割操作，将切割完成后的物品分为两个集合，最小化：

$$\varepsilon = \frac{\left|\sum_{i\in S} V_i - \sum_{i\in T} V_i\right|}{\sum_{i=1}^{n+k} V_i} + \frac{\left|\sum_{i\in S} m_i - \sum_{i\in T} m_i\right|}{\sum_{i=1}^{n+k} m_i}$$

要求输出任意一组解。

$1 \le T \le 20, 1 \le n \le 10^5, 1 \le V_i, m_i \le 10^6$

Test $1 \sim 2(n \leq 3, V_i, m_i \leq 10)$

这 3 个物品里肯定有一个是没有被切割的，枚举是哪一个。

Test $1 \sim 2(n \leq 3, V_i, m_i \leq 10)$

这 3 个物品里肯定有一个是没有被切割的，枚举是哪一个。
剩余部分可以分类讨论解方程。

Test $1 \sim 2(n \le 3,\ V_i,\ m_i \le 10)$

这 3 个物品里肯定有一个是没有被切割的，枚举是哪一个。
剩余部分可以分类讨论解方程。
时间复杂度 $\mathcal{O}(\cdot)$，得分 10 分。

Test $3 \sim 6(n \leq 16$，性质 A)

不需要进行切割操作，每个元素只有属于 S 与属于 T 两种状态。

Test $3 \sim 6(n \leq 16$，性质 A)

不需要进行切割操作，每个元素只有属于 S 与属于 T 两种状态。

对所有可能的方法爆搜即可。

Test $3 \sim 6(n \leq 16$，性质 A)

不需要进行切割操作，每个元素只有属于 S 与属于 T 两种状态。

对所有可能的方法爆搜即可。

时间复杂度 $\mathcal{O}(n \cdot 2^n)$，得分 20 分。结合上述算法可以得到 30 分。

Lemma

实现了上述算法，为什么大样例的 ε 都是 0？

Lemma

实现了上述算法，为什么大样例的 ε 都是 0？
看起来你的操作非常强，不妨猜想：

Conjecture

对于任意初始局面，都存在策略使得答案为 0。

Test $3 \sim 9(n \leq 30$，性质 A)

根据我们的猜想，最终两个集合的 V_i 之和一定均为 $S_0 = \frac{1}{2}\sum V_i$，$m_i$ 之和均为 $m_0 = \frac{1}{2}\sum m_i$。

Test $3 \sim 9(n \leq 30$，性质 A)

根据我们的猜想，最终两个集合的 V_i 之和一定均为 $S_0 = \frac{1}{2}\sum V_i$，$m_i$ 之和均为 $m_0 = \frac{1}{2}\sum m_i$。

只需要保证 S 满足此限制即可。

Test $3 \sim 9(n \leq 30$，性质 A)

根据我们的猜想，最终两个集合的 V_i 之和一定均为 $S_0 = \frac{1}{2}\sum V_i$，$m_i$ 之和均为 $m_0 = \frac{1}{2}\sum m_i$。

只需要保证 S 满足此限制即可。

考虑我们将元素分成两部分，每部分 $\frac{n}{2}$ 进行 Meet in the middle。对于任意一部分，我们爆搜出所有 $2^{n/2}$ 种方案，然后对一侧任意一个使得 S 的 V 之和为 i，m 之和为 j 的方案，查询右侧是否存在一个 S 的和为 $S_0 - i$，m 之和为 $m_0 - j$ 的方案。

Test $3\sim 9(n\le 30$，性质 A)

根据我们的猜想，最终两个集合的 V_i 之和一定均为 $S_0=\frac{1}{2}\sum V_i$，m_i 之和均为 $m_0=\frac{1}{2}\sum m_i$。

只需要保证 S 满足此限制即可。

考虑我们将元素分成两部分，每部分 $\frac{n}{2}$ 进行 Meet in the middle。对于任意一部分，我们爆搜出所有 $2^{n/2}$ 种方案，然后对一侧任意一个使得 S 的 V 之和为 i，m 之和为 j 的方案，查询右侧是否存在一个 S 的和为 S_0-i，m 之和为 m_0-j 的方案。

i,j 的数量看起来很多（$\mathcal{O}(mV)$ 个），但是有解的数量在任意一侧都不超过 $\mathcal{O}(2^{n/2})$ 个。

Test $3 \sim 9(n \leq 30$，性质 A)

根据我们的猜想，最终两个集合的 V_i 之和一定均为 $S_0 = \frac{1}{2}\sum V_i$，$m_i$ 之和均为 $m_0 = \frac{1}{2}\sum m_i$。

只需要保证 S 满足此限制即可。

考虑我们将元素分成两部分，每部分 $\frac{n}{2}$ 进行 Meet in the middle。对于任意一部分，我们爆搜出所有 $2^{n/2}$ 种方案，然后对一侧任意一个使得 S 的 V 之和为 i，m 之和为 j 的方案，查询右侧是否存在一个 S 的和为 $S_0 - i$，m 之和为 $m_0 - j$ 的方案。

i, j 的数量看起来很多（$\mathcal{O}(mV)$ 个），但是有解的数量在任意一侧都不超过 $\mathcal{O}(2^{n/2})$ 个。

时间复杂度 $\mathcal{O}(2^{n/2})$，得分 35 分。

结合上述算法可以得到 45 分。

Solution

正解事实上比上述算法简洁的多。

Solution

正解事实上比上述算法简洁的多。

我们将一个物品视为一个底为体积 V_i，高度为密度 m_i/V_i 的矩形，并将所有矩形按照高度排序。

Solution

正解事实上比上述算法简洁的多。

我们将一个物品视为一个底为体积 V_i，高度为密度 m_i/V_i 的矩形，并将所有矩形按照高度排序。

我们将给出一个更强的结论，**不仅可以构造出一组解，而且还能保证 S 内的元素对应的矩形位于排序后连续的一段区间。**

Solution

正解事实上比上述算法简洁的多。

我们将一个物品视为一个底为体积 V_i，高度为密度 m_i/V_i 的矩形，并将所有矩形按照高度排序。

我们将给出一个更强的结论，**不仅可以构造出一组解，而且还能保证 S 内的元素对应的矩形位于排序后连续的一段区间**。

考虑我们的任务，其实就是找一段长为 $\sum V_i/2$ 的区间，使得这个区间上方图形的面积为 $\sum m_i/2$。

Solution

正解事实上比上述算法简洁的多。

我们将一个物品视为一个底为体积 V_i，高度为密度 m_i/V_i 的矩形，并将所有矩形按照高度排序。

我们将给出一个更强的结论，**不仅可以构造出一组解，而且还能保证 S 内的元素对应的矩形位于排序后连续的一段区间**。

考虑我们的任务，其实就是找一段长为 $\sum V_i/2$ 的区间，使得这个区间上方图形的面积为 $\sum m_i/2$。

不妨假设我们枚举这个区间的右端点 r，那么当 r 足够大时，必定存在一个左端点 l 满足 $[l,r]$ 上方的面积为 $\sum m_i/2$。

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

根据平均数的性质（$x_{\text{min}} \le \bar{x} \le x_{\text{max}}$），我们可以得知，对于区间 $[0, \sum V_i/2]$，它对应的面积一定小于等于 $\sum m_i/2$（因为我们将矩形按照高度排序后，$\sum m_i/2$ 对应的其实是两个矩形的面积的平均值，最小的矩形肯定不会大于平均数）。

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

根据平均数的性质（$x_{\text{min}} \leq \bar{x} \leq x_{\text{max}}$），我们可以得知，对于区间 $[0, \sum V_i/2]$，它对应的面积一定小于等于 $\sum m_i/2$（因为我们将矩形按照高度排序后，$\sum m_i/2$ 对应的其实是两个矩形的面积的平均值，最小的矩形肯定定不会大于平均数）。

所以对于第一个区间 $[0, r_0]$，必定有 $r_0 - 0 \geq \sum V_i/2$

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

根据平均数的性质（$x_{\text{min}} \le \bar{x} \le x_{\text{max}}$），我们可以得知，对于区间 $[0, \sum V_i/2]$，它对应的面积一定小于等于 $\sum m_i/2$（因为我们将矩形按照高度排序后，$\sum m_i/2$ 对应的其实是两个矩形的面积的平均值，最小的矩形肯定不会大于平均数）。

所以对于第一个区间 $[0, r_0]$，必定有 $r_0 - 0 \ge \sum V_i/2$

又因为在 r 增大时，面积增加的速度变快，所以 l 肯定会以更快的速度向右移动。因此 $r - l$ 肯定是在不断变小的。

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

根据平均数的性质（$x_{\min} \le \bar{x} \le x_{\max}$），我们可以得知，对于区间 $[0, \sum V_i/2]$，它对应的面积一定小于等于 $\sum m_i/2$（因为我们将矩形按照高度排序后，$\sum m_i/2$ 对应的其实是两个矩形的面积的平均值，最小的矩形肯定不会大于平均数）。

所以对于第一个区间 $[0, r_0]$，必定有 $r_0 - 0 \ge \sum V_i/2$

又因为在 r 增大时，面积增加的速度变快，所以 l 肯定会以更快的速度向右移动。因此 $r - l$ 肯定是在不断变小的。

最终区间长度一定会变得小于等于 $\sum V_i/2$，又因为我们的变化是连续不断的，因此过程中肯定会有一个合法的区间。

Solution

现在只需要保证区间的长度等于 $\sum V_i/2$。

根据平均数的性质（$x_{\text{min}} \le \bar{x} \le x_{\text{max}}$），我们可以得知，对于区间 $[0, \sum V_i/2]$，它对应的面积一定小于等于 $\sum m_i/2$（因为我们将矩形按照高度排序后，$\sum m_i/2$ 对应的其实是两个矩形的面积的平均值，最小的矩形肯定不会大于平均数）。

所以对于第一个区间 $[0, r_0]$，必定有 $r_0 - 0 \ge \sum V_i/2$

又因为在 r 增大时，面积增加的速度变快，所以 l 肯定会以更快的速度向右移动。因此 $r - l$ 肯定是在不断变小的。

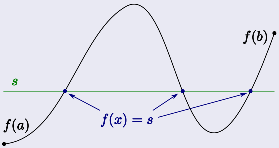
最终区间长度一定会变得小于等于 $\sum V_i/2$，又因为我们的变化是连续不断的，因此过程中肯定会有一个合法的区间。

最多只有首尾两个矩形是不完整的，把他们找出来即可。

Proof of the conjecture

Intermediate value theorem

设 f 在 $[a,b]$ 上连续， $s \in [f(a), f(b)]$，则必定存在 $\xi \in (a,b)$ 使得 $f(\xi) = s$。



。

题意

给定一棵 n 个点的树 T，其顶点标号为 $1 \sim n$，边集记为 E。第 i 个点有一个权值 a_i。设 $S \subseteq E$ 为边集 S 的子集，我们定义 $f(S)$ 的值如下：

- 考虑一张新的图 T'，T' 的点集与 T 相同，边集为 S。
- T' 中每个联通块的权值为该联通块所有点权值的最小值。
- $f(S)$ 的值即为 T' 所有联通块权值之和。

计算 $\sum\limits_{S \subseteq E} f(S)$。

算法一

暴力枚举每条边选不选。

算法一

暴力枚举每条边选不选。
时间复杂度 $O(n \cdot 2^n)$。

算法一

暴力枚举每条边选不选。
时间复杂度 $O(n \cdot 2^n)$。
期望通过测试点 1，得到 4 分。

算法二

考虑如下 dp：$f[x][v]$ 表示考虑以 x 为根的子树内选一个联通块，x 所在联通块的最小值为 v，这种情况下的得分之和。

算法二

考虑如下 dp：$f[x][v]$ 表示考虑以 x 为根的子树内选一个联通块，x 所在联通块的最小值为 v，这种情况下的得分之和。

转移考虑如果这个子树这条边不选，根据期望的线性性，它带一个 $2^c \cdot v$ 的系数贡献。如果选，那么 x 所在块的最小值需要进行更新。

算法二

考虑如下 dp：$f[x][v]$ 表示考虑以 x 为根的子树内选一个联通块，x 所在联通块的最小值为 v，这种情况下的得分之和。

转移考虑如果这个子树这条边不选，根据期望的线性性，它带一个 $2^c \cdot v$ 的系数贡献。如果选，那么 x 所在块的最小值需要进行更新。

暴力进行合并，合并次数 $O(n)$ 次，每次合并 $O(n^2)$，总的时间复杂度为 $O(n^3)$。

算法二

考虑如下 dp：$f[x][v]$ 表示考虑以 x 为根的子树内选一个联通块，x 所在联通块的最小值为 v，这种情况下的得分之和。

转移考虑如果这个子树这条边不选，根据期望的线性性，它带一个 $2^c \cdot v$ 的系数贡献。如果选，那么 x 所在块的最小值需要进行更新。

暴力进行合并，合并次数 $O(n)$ 次，每次合并 $O(n^2)$，总的时间复杂度为 $O(n^3)$。

期望通过测试点 $1 \sim 2$，得到 8 分。

算法三

事实上，一个 x 只有 $O(\mathrm{size}(x))$ 个位置有值。

算法三

事实上，一个 x 只有 $O(\mathrm{size}(x))$ 个位置有值。
所以我们可以只 for 那些有值的部分。

算法三

事实上，一个 x 只有 $O(\mathrm{size}(x))$ 个位置有值。
所以我们可以只 for 那些有值的部分。
根据树上背包的时间复杂度分析，复杂度为 $O(n^2)$。

算法三

事实上，一个 x 只有 $O(\text{size}(x))$ 个位置有值。
所以我们可以只 for 那些有值的部分。
根据树上背包的时间复杂度分析，复杂度为 $O(n^2)$。
期望通过测试点 $1 \sim 4$，得到 16 分。

算法四

性质 A、B、C 均有做法。由于和正解无关就不讲了。

算法四

性质 A、B、C 均有做法。由于和正解无关就不讲了。
性质 A：16 分。
性质 B：12 分。
性质 C：8 分。

算法四

性质 A、B、C 均有做法。由于和正解无关就不讲了。
性质 A：16 分。
性质 B：12 分。
性质 C：8 分。
总共 36 分。结合暴力共 52 分。

算法五

考虑优化算法三中的 dp。

算法五

考虑优化算法三中的 dp。

首先合并 x 与 y 的 dp 值不仅可以做到 $O(\mathrm{size}(x) \cdot \mathrm{size}(y))$，还可以做到 $O(\mathrm{size}(x) + \mathrm{size}(y))$。

算法五

考虑优化算法三中的 dp。

首先合并 x 与 y 的 dp 值不仅可以做到 $O(\text{size}(x)\cdot\text{size}(y))$，还可以做到 $O(\text{size}(x)+\text{size}(y))$。

由于我们转移到的位置是 $\min(v_1,v_2)$，因此我们枚举最小值的位置，剩下的贡献项均可以通过前缀和或后缀和解决。

算法五

考虑优化算法三中的 dp。

首先合并 x 与 y 的 dp 值不仅可以做到 $O(\text{size}(x)\cdot\text{size}(y))$，还可以做到 $O(\text{size}(x)+\text{size}(y))$。

由于我们转移到的位置是 $\min(v_1, v_2)$，因此我们枚举最小值的位置，剩下的贡献项均可以通过前缀和或后缀和解决。

但是这并不能改进复杂度，在图为一条链时复杂度会退化到 $O(n^2)$。

算法五

思考一下我们的转移在干什么！

算法五

思考一下我们的转移在干什么!
假设我们要将 $f[y][\cdot]$ 合并到 $f[x][\cdot]$。

算法五

思考一下我们的转移在干什么!
假设我们要将 $f[y][\cdot]$ 合并到 $f[x][\cdot]$。
每个 $f[y][\cdot]$ 的信息可以视作对 $f[x][\cdot]$ 做单点修改或区间乘。

算法五

思考一下我们的转移在干什么！
假设我们要将 $f[y][\cdot]$ 合并到 $f[x][\cdot]$。
每个 $f[y][\cdot]$ 的信息可以视作对 $f[x][\cdot]$ 做单点修改或区间乘。
我们考虑启发式合并，每次从较小的 $f[\cdot]$ 更新到较大的 $f[\cdot]$。

算法五

思考一下我们的转移在干什么！
假设我们要将 $f[y][\cdot]$ 合并到 $f[x][\cdot]$。
每个 $f[y][\cdot]$ 的信息可以视作对 $f[x][\cdot]$ 做单点修改或区间乘。
我们考虑启发式合并，每次从较小的 $f[\cdot]$ 更新到较大的 $f[\cdot]$。
使用动态开点线段树维护每个 dp 数组。

算法五

思考一下我们的转移在干什么！

假设我们要将 $f[y][\cdot]$ 合并到 $f[x][\cdot]$。

每个 $f[y][\cdot]$ 的信息可以视作对 $f[x][\cdot]$ 做单点修改或区间乘。

我们考虑启发式合并，每次从较小的 $f[\cdot]$ 更新到较大的 $f[\cdot]$。

使用动态开点线段树维护每个 dp 数组。

时间复杂度 $O(n\log n\log V)$。期望通过测试点 $1\sim 22$，得到 88 分。如果常数写的差一点，那可能就是 72 分。如果写的好一点，可能就直接通过了。

算法六

事实上，我们没有必要启发式合并，而是可以直接线段树合并。

算法六

事实上，我们没有必要启发式合并，而是可以直接线段树合并。

在线段树合并的时候，我们可以一并计算出转移所需的系数，不需要每次都从 y 对应的线段树上查询。

算法六

事实上，我们没有必要启发式合并，而是可以直接线段树合并。

在线段树合并的时候，我们可以一并计算出转移所需的系数，不需要每次都从 y 对应的线段树上查询。

时间复杂度优化至 $O(n\log n)$。

算法六

事实上，我们没有必要启发式合并，而是可以直接线段树合并。

在线段树合并的时候，我们可以一并计算出转移所需的系数，不需要每次都从 y 对应的线段树上查询。

时间复杂度优化至 $O(n\log n)$。

期望得分 100 分。

题意

有 n 个整数排成一圈，定义一次操作为：选择其中一个整数 a，将其变为 $-a$，并使圈上与其相邻的两个整数加上 a。

你希望进行若干次操作，使得最终所有的整数均非负。求出最小的操作次数。

算法一

暴力 BFS。

算法一

暴力 BFS。
期望得分：

算法一

暴力 BFS。
期望得分：10 分。

算法一

暴力 BFS。
期望得分：10 分。（为什么？）

算法二

如果是序列，不是环，怎么做？

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。
$A_{i-1}, A_i, A_{i+1} \Longrightarrow A_{i-1} + A_i, -A_i, A_{i+1} + A_i$

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。
$A_{i-1}, A_i, A_{i+1} \implies A_{i-1}+A_i, -A_i, A_{i+1}+A_i$
$S_{i-1}, S_i, S_{i+1} \implies S_i, S_{i-1}, S_{i+1}$

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。
$A_{i-1}, A_i, A_{i+1} \Longrightarrow A_{i-1} + A_i, -A_i, A_{i+1} + A_i$
$S_{i-1}, S_i, S_{i+1} \Longrightarrow S_i, S_{i-1}, S_{i+1}$
操作本质：交换前缀和相邻两项。

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。
$A_{i-1}, A_i, A_{i+1} \implies A_{i-1} + A_i, -A_i, A_{i+1} + A_i$
$S_{i-1}, S_i, S_{i+1} \implies S_i, S_{i-1}, S_{i+1}$
操作本质：交换前缀和相邻两项。
要求 $A_i \geq 0 \iff S_i \geq S_{i-1}$

算法二

如果是序列，不是环，怎么做？

记 S_i 为 A_i 的前缀和。

$A_{i-1}, A_i, A_{i+1} \Longrightarrow A_{i-1}+A_i, -A_i, A_{i+1}+A_i$

$S_{i-1}, S_i, S_{i+1} \Longrightarrow S_i, S_{i-1}, S_{i+1}$

操作本质：交换前缀和相邻两项。

要求 $A_i \geq 0 \Longleftrightarrow S_i \geq S_{i-1}$

补充 $S_0 = 0$。

算法二

如果是序列，不是环，怎么做？
记 S_i 为 A_i 的前缀和。
$A_{i-1}, A_i, A_{i+1} \Longrightarrow A_{i-1}+A_i, -A_i, A_{i+1}+A_i$
$S_{i-1}, S_i, S_{i+1} \Longrightarrow S_i, S_{i-1}, S_{i+1}$
操作本质：交换前缀和相邻两项。
要求 $A_i \geq 0 \Longleftrightarrow S_i \geq S_{i-1}$
补充 $S_0 = 0$。

相当于给定一个序列，可以交换相邻两个元素，要求最后逆序对数为 0。

算法二

如果是序列，不是环，怎么做？

记 S_i 为 A_i 的前缀和。

$A_{i-1}, A_i, A_{i+1} \Longrightarrow A_{i-1}+A_i, -A_i, A_{i+1}+A_i$

$S_{i-1}, S_i, S_{i+1} \Longrightarrow S_i, S_{i-1}, S_{i+1}$

操作本质：交换前缀和相邻两项。

要求 $A_i \geq 0 \Longleftrightarrow S_i \geq S_{i-1}$

补充 $S_0 = 0$。

相当于给定一个序列，可以交换相邻两个元素，要求最后逆序对数为 0。

求个逆序对就做完了。

算法二

但是原问题是环。

算法二

但是原问题是环。

考虑定义环上的广义前缀和。记 $A_0 = A_n$，定义 $S_i = S_{i-1} + A_{i \bmod n}$。

算法二

但是原问题是环。

考虑定义环上的广义前缀和。记 $A_0 = A_n$，定义 $S_i = S_{i-1} + A_{i \bmod n}$。

此时，对 A_i 进行一次操作后，对于每个 $k \in \mathbb{Z}$，S_{i-1+kn} 与 S_{i+kn} 被交换。

算法二

但是原问题是环。

考虑定义环上的广义前缀和。记 $A_0 = A_n$，定义 $S_i = S_{i-1} + A_{i \bmod n}$。

此时，对 A_i 进行一次操作后，对于每个 $k \in \mathbb{Z}$，S_{i-1+kn} 与 S_{i+kn} 被交换。

我们记 R_i 表示有多少个 $j > i$，使得 $S_i < S_j$。

算法二

但是原问题是环。

考虑定义环上的广义前缀和。记 $A_0 = A_n$，定义 $S_i = S_{i-1} + A_{i \bmod n}$。

此时，对 A_i 进行一次操作后，对于每个 $k \in \mathbb{Z}$，S_{i-1+kn} 与 S_{i+kn} 被交换。

我们记 R_i 表示有多少个 $j > i$，使得 $S_i < S_j$。

注意到若 $S_n \le 0$，那么原问题显然无解（除非 A_i 全为 0）。

算法二

但是原问题是环。

考虑定义环上的广义前缀和。记 $A_0 = A_n$，定义 $S_i = S_{i-1} + A_{i \bmod n}$。

此时，对 A_i 进行一次操作后，对于每个 $k \in \mathbb{Z}$，S_{i-1+kn} 与 S_{i+kn} 被交换。

我们记 R_i 表示有多少个 $j > i$，使得 $S_i < S_j$。

注意到若 $S_n \leq 0$，那么原问题显然无解（除非 A_i 全为 0）。

若不然，R_i 必定是有限的，因为 $S_i > S_{i+n}$。且容易发现 $R_i = R_{i+n}$。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。
当 $A_i > 0$ 时，操作后 $R'_{i-1} = R_i + 1, R'_i = R_{i-1}$。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。
当 $A_i > 0$ 时，操作后 $R'_{i-1} = R_i + 1, R'_i = R_{i-1}$。
即，$A_i < 0$ 时操作一次逆序对数减 1，$A_i > 0$ 时操作一次逆序对数加 1。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。
当 $A_i > 0$ 时，操作后 $R'_{i-1} = R_i + 1, R'_i = R_{i-1}$。
即，$A_i < 0$ 时操作一次逆序对数减 1，$A_i > 0$ 时操作一次逆序对数加 1。
故我们只会对负数进行操作，且进行操作只会更优。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。
当 $A_i > 0$ 时，操作后 $R'_{i-1} = R_i + 1, R'_i = R_{i-1}$。
即，$A_i < 0$ 时操作一次逆序对数减 1，$A_i > 0$ 时操作一次逆序对数加 1。

故我们只会对负数进行操作，且进行操作只会更优。
于是我们每轮暴力找到负数位置进行操作。

算法二

当 $A_i < 0$ 时，操作后 $R'_{i-1} = R_i - 1, R'_i = R_{i-1}$。

当 $A_i > 0$ 时，操作后 $R'_{i-1} = R_i + 1, R'_i = R_{i-1}$。

即，$A_i < 0$ 时操作一次逆序对数减 1，$A_i > 0$ 时操作一次逆序对数加 1。

故我们只会对负数进行操作，且进行操作只会更优。

于是我们每轮暴力找到负数位置进行操作。

期望通过测试点 $1 \sim 2$，得到 20 分。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。
考虑将所有的 j 按照同余分类统计。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。
考虑将所有的 j 按照同余分类统计。
若 $S_i > S_{i+k} (1 \le k < n)$，对答案的贡献为 $\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1$。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。
考虑将所有的 j 按照同余分类统计。
若 $S_i > S_{i+k} (1 \le k < n)$，对答案的贡献为 $\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1$。
即计算 $\sum_{1 \le i \le n, 1 \le k < n} [S_i > S_{i+k}] \left(\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1 \right)$

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。
考虑将所有的 j 按照同余分类统计。
若 $S_i > S_{i+k} (1 \le k < n)$，对答案的贡献为 $\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1$。
即计算 $\sum_{1 \le i \le n, 1 \le k < n} [S_i > S_{i+k}] \left(\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1 \right)$
直接暴力枚举 i, k，时间复杂度为 $O(n^2)$。

算法三

问题变为快速计算 $R_1, R_2, \cdots, R_n$。
注意到 $S_{i+n} = S_i + S_n$。
考虑将所有的 j 按照同余分类统计。
若 $S_i > S_{i+k}(1 \le k < n)$，对答案的贡献为 $\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1$。
即计算 $\sum_{1 \le i \le n, 1 \le k < n} [S_i > S_{i+k}] \left(\left\lfloor \frac{S_i - S_{i+k} - 1}{S_n} \right\rfloor + 1 \right)$
直接暴力枚举 i, k，时间复杂度为 $O(n^2)$。
期望通过测试点 $1 \sim 5$，得到 50 分。

算法四

测试点 6 满足 $S_n = 1$，答案即为

$$\sum_{1 \le i \le n, 1 \le k < n} [S_i > S_{i+k}] (S_i - S_{i+k})$$

算法四

测试点 6 满足 $S_n = 1$，答案即为

$$\sum_{1\le i\le n, 1\le k<n} [S_i > S_{i+k}] (S_i - S_{i+k})$$

二维偏序。

算法四

测试点 6 满足 $S_n = 1$，答案即为

$$\sum_{1\le i\le n, 1\le k<n} [S_i > S_{i+k}]\,(S_i - S_{i+k})$$

二维偏序。
时间复杂度 $O(n\log n)$。

算法四

测试点 6 满足 $S_n = 1$，答案即为

$$\sum_{1\le i\le n, 1\le k<n} [S_i > S_{i+k}]\,(S_i - S_{i+k})$$

二维偏序。
时间复杂度 $O(n\log n)$。
期望通过测试点 6，得到 10 分。结合算法三可以得到 60 分。

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i \ (0 \le Q_i < S_n)$。

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i \ (0 \le Q_i < S_n)$。
若 $S_i > S_j$，那么：

$$\left\lfloor \frac{S_i - S_j - 1}{S_n} \right\rfloor + 1 = \begin{cases} P_i - P_j & Q_i \le Q_j \\ P_i - P_j + 1 & Q_i > Q_j \end{cases}$$

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i \ (0 \le Q_i < S_n)$。
若 $S_i > S_j$，那么：

$$\left\lfloor \frac{S_i - S_j - 1}{S_n} \right\rfloor + 1 = \begin{cases} P_i - P_j & Q_i \le Q_j \\ P_i - P_j + 1 & Q_i > Q_j \end{cases}$$

变成了带权的三维数点。

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i\ (0 \le Q_i < S_n)$。
若 $S_i > S_j$，那么：

$$\left\lfloor \frac{S_i - S_j - 1}{S_n} \right\rfloor + 1 = \begin{cases} P_i - P_j & Q_i \le Q_j \\ P_i - P_j + 1 & Q_i > Q_j \end{cases}$$

变成了带权的三维数点。
由于 S_n 很小，因此 Q 那一维直接暴力数点即可。

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i \ (0 \le Q_i < S_n)$。
若 $S_i > S_j$，那么：

$$\left\lfloor \frac{S_i - S_j - 1}{S_n} \right\rfloor + 1 = \begin{cases} P_i - P_j & Q_i \le Q_j \\ P_i - P_j + 1 & Q_i > Q_j \end{cases}$$

变成了带权的三维数点。
由于 S_n 很小，因此 Q 那一维直接暴力数点即可。
$O(n \log n |S_n|)$

算法五

测试点 7 满足 $-10 \le S_n \le 10$。
记 $S_i = P_i S_n + Q_i \ (0 \le Q_i < S_n)$。
若 $S_i > S_j$，那么：

$$\left\lfloor \frac{S_i - S_j - 1}{S_n} \right\rfloor + 1 = \begin{cases} P_i - P_j & Q_i \le Q_j \\ P_i - P_j + 1 & Q_i > Q_j \end{cases}$$

变成了带权的三维数点。
由于 S_n 很小，因此 Q 那一维直接暴力数点即可。
$O(n \log n |S_n|)$
期望通过测试点 $6 \sim 7$，得到 20 分。结合算法三可以得到 70 分。

算法六

针对算法五，直接做三维数点。

算法六

针对算法五，直接做三维数点。
可以离线 CDQ 分治，时间复杂度 $O(n \log^2 n)$。

算法六

针对算法五，直接做三维数点。
可以离线 CDQ 分治，时间复杂度 $O(n\log^2 n)$。
期望通过测试点 $1\sim 8$，得到 80 分。

算法六

针对算法五，直接做三维数点。
可以离线 CDQ 分治，时间复杂度 $O(n\log^2 n)$。
期望通过测试点 $1\sim 8$，得到 80 分。
可能使劲卡卡常也能通过。

算法七

重新审视一下我们三个维度的限制：

- $i \le j < i + n$
- $S_i > S_j$
- $Q_i \le Q_j$

算法七

重新审视一下我们三个维度的限制：

- $i \le j < i+n$
- $S_i > S_j$
- $Q_i \le Q_j$

这个问题可以用动态的二维数点来解释：扫 i 时我们会删除一个点（i）、加入一个点（$i+n$），并要统计满足 $S_i > S_j$ 且 $Q_i \le Q_j$ 的点的点权之和。

算法七

重新审视一下我们三个维度的限制：

- $i \le j < i + n$
- $S_i > S_j$
- $Q_i \le Q_j$

这个问题可以用动态的二维数点来解释：扫 i 时我们会删除一个点（i）、加入一个点（$i+n$），并要统计满足 $S_i > S_j$ 且 $Q_i \le Q_j$ 的点的点权之和。

注意到我们删除的点 i 与加入的点 $i+n$ 相比，有 $S_{i+n} = S_i + S_n$ 且 $Q_{i+n} = Q_i$。即第二个维度被加了常数，第三个维度没变。

算法七

重新审视一下我们三个维度的限制：

- $i \le j < i + n$
- $S_i > S_j$
- $Q_i \le Q_j$

这个问题可以用动态的二维数点来解释：扫 i 时我们会删除一个点（i）、加入一个点（$i+n$），并要统计满足 $S_i > S_j$ 且 $Q_i \le Q_j$ 的点的点权之和。

注意到我们删除的点 i 与加入的点 $i+n$ 相比，有 $S_{i+n} = S_i + S_n$ 且 $Q_{i+n} = Q_i$。即第二个维度被加了常数，第三个维度没变。

而对应的点的权值有 $P_{i+n} = P_i + 1$，即点权加 1。

算法七

我们不需要进行插入与删除操作！

算法七

我们不需要进行插入与删除操作！
考虑如果去掉第一个限制，我们会多算什么。

算法七

我们不需要进行插入与删除操作!
考虑如果去掉第一个限制，我们会多算什么。
对于 $i < j$ 且 $s_i > s_j$ 的一对，会被我们多算 1。

算法七

我们不需要进行插入与删除操作！
考虑如果去掉第一个限制，我们会多算什么。
对于 $i<j$ 且 $s_i>s_j$ 的一对，会被我们多算 1。
为什么不需要考虑 Q_i 和 Q_j？因为无论是哪一种被多算的值都是 1。

算法七

我们不需要进行插入与删除操作！

考虑如果去掉第一个限制，我们会多算什么。

对于 $i < j$ 且 $s_i > s_j$ 的一对，会被我们多算 1。

为什么不需要考虑 Q_i 和 Q_j？因为无论是哪一种被多算的值都是 1。

可以将多算的部分减掉。

算法七

我们不需要进行插入与删除操作！
考虑如果去掉第一个限制，我们会多算什么。
对于 $i < j$ 且 $s_i > s_j$ 的一对，会被我们多算 1。
为什么不需要考虑 Q_i 和 Q_j？因为无论是哪一种被多算的值都是 1。
可以将多算的部分减掉。
这样可以转成两个二维偏序。

算法七

我们不需要进行插入与删除操作!

考虑如果去掉第一个限制，我们会多算什么。

对于 $i < j$ 且 $s_i > s_j$ 的一对，会被我们多算 1。

为什么不需要考虑 Q_i 和 Q_j？因为无论是哪一种被多算的值都是 1。

可以将多算的部分减掉。

这样可以转成两个二维偏序。

时间复杂度为 $O(n \log n)$。

算法七

我们不需要进行插入与删除操作!

考虑如果去掉第一个限制，我们会多算什么。

对于 $i < j$ 且 $s_i > s_j$ 的一对，会被我们多算 1。

为什么不需要考虑 Q_i 和 Q_j? 因为无论是哪一种被多算的值都是 1。

可以将多算的部分减掉。

这样可以转成两个二维偏序。

时间复杂度为 $O(n \log n)$。

期望得分 100 分。