《蒜头的奖杯》命题报告

安徽师范大学附属中学 朱震霆

1 概述

本文讨论2018年集训队互测中作者命制的一道试题。

本文会先给出题目信息，介绍本题的相关算法。然后将会从命题思路方面加以讨论，希望能够给予读者一些启发和思考。

2 题意简述

给定长度为 n 的六个序列 A, B, C, D, E, F ，求：

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} A_i B_j C_k D_{\gcd(i,j)} E_{\gcd(i,k)} F_{\gcd(j,k)}$$

答案对 2^{64} 取模。

3 数据范围

对于所有数据，$n \le 10^5$ ，输入序列中的数字不会超过 10^{18} 。

测试点编号 m	$n \le$	其他约定
1	100	-
2, 3, 4	2000	-
5, 6	10^5	$D_i = [i = 1], E_i = F_i = 1$
7	10^5	$A_i = B_i = C_i = D_i = 1, E_i = F_i = [i = 1]$
8	10^5	$D_i = 1, E_i = F_i = [i = 1]$
9, 10	10^5	$A_i = B_i = C_i = 1, D_i = E_i = F_i = [i = 1]$
11, 12, 13	$(m-3) \times 10^4$	$D_i = E_i = F_i = [i = 1]$
14 . . . 20	$(m-13) \times 10^4$	当 m 为奇数时，有 $D_i = E_i = F_i = i$
21 . . . 25	$5(m-5) \times 10^3$	当 m 为偶数时，有 $D_i = E_i = F_i = i$

4 算法介绍

4.1 算法一

按照题意枚举 i,j,k，暴力求和即可。算法一的时间复杂度为 $O(n^3\log n)$。如果预处理任意两数的 gcd，时间复杂度可以优化至 $O(n^3)$。该算法可以通过第 1 个数据点。

4.2 算法二

我们可以枚举 $\gcd(i,j)=d$，于是可得：

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}A_iB_jC_kD_{\gcd(i,j)}E_{\gcd(i,k)}F_{\gcd(j,k)}\\=&\sum_{d=1}^{n}D_d\sum_{id\le n}\sum_{jd\le n}\sum_{k=1}^{n}[\gcd(i,j)=1]A_{id}B_{jd}C_kE_{\gcd(id,k)}F_{\gcd(jd,k)}\end{aligned}$$

利用经典恒等式 $\sum_{d|n}\mu(d)=[n=1]$，并交换求和符号顺序，转而枚举 $T=dt$，可以得到：

$$\begin{aligned}&\sum_{d=1}^{n}D_d\sum_{id\le n}\sum_{jd\le n}\sum_{k=1}^{n}A_{id}B_{jd}C_kE_{\gcd(id,k)}F_{\gcd(jd,k)}\sum_{t|i,t|j}\mu(t)\\=&\sum_{T=1}^{n}\left(\sum_{d|T}D_d\mu\left(\frac{T}{d}\right)\right)\sum_{iT\le n}\sum_{jT\le n}\sum_{k=1}^{n}A_{iT}B_{jT}C_kE_{\gcd(iT,k)}F_{\gcd(jT,k)}\\=&\sum_{T=1}^{n}\left(\sum_{d|T}D_d\mu\left(\frac{T}{d}\right)\right)\sum_{k=1}^{n}C_k\sum_{iT\le n}\sum_{jT\le n}A_{iT}B_{jT}E_{\gcd(iT,k)}F_{\gcd(jT,k)}\\=&\sum_{T=1}^{n}\left(\sum_{d|T}D_d\mu\left(\frac{T}{d}\right)\right)\sum_{k=1}^{n}C_k\left(\sum_{iT\le n}A_{iT}E_{\gcd(iT,k)}\right)\left(\sum_{iT\le n}B_{iT}F_{\gcd(iT,k)}\right)\end{aligned}$$

于是只要按照上式，枚举 T,k,i 就可以计算原式了。

考虑计算时间复杂度，满足条件的 i,T 的对数显然为 $\sum_{i=1}^{n}\left\lfloor\frac{n}{i}\right\rfloor=n\log n+O(n)$。这样我们在枚举一层 k，预处理任意两数的 gcd 后，就可以得到一个时间复杂度为 $O(n^2\log n)$ 的做法了。该算法可以通过前 4 个数据点。

4.3 算法三

当 $D_i=[i=1],E_i=F_i=1$ 时，我们要求的式子为 $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}A_iB_jC_k[\gcd(i,j)=1]$。

容易发现只要最后乘上所有 C 的和就可以不用考虑 C 了。于是同样利用莫比乌斯反演，即求 $\sum_{d=1}^{n}\mu(d)\sum_{id\le n}A_{id}\sum_{jd\le n}B_{jd}$。只要 $O(n\log n)$ 计算即可。

该算法可以通过第 5 个和第 6 个数据点。结合算法二可以通过前 6 个数据点。

4.4 算法四

当 $A_i=B_i=C_i=D_i=1,E_i=F_i=[i=1]$ 时，实际上就是要求满足 $\gcd(i,k)=1,\gcd(j,k)=1$ 的三元组 (i,j,k) 个数，满足 $1\le i,j,k\le n$ 。

这个问题也很好解决，只要枚举 k，计算出和 k 互质的数的个数，我们就可以统计 (i,j) 的对数了。求出和 k 互质的数的个数仍然是一个基础的反演问题，即 $\sum_{i=1}^{n}[\gcd(k,i)=1]=\sum_{d|k}\mu(d)\left\lfloor\frac{n}{d}\right\rfloor$，于是这个问题也可以 $O(n\log n)$ 解决。

该算法可以通过第 7 个测试点。结合算法二和算法三，可以通过前 7 个测试点。

4.5 算法五

当 $D_i=1,E_i=F_i=[i=1]$ 时，我们计算的不再 $\gcd(i,k)=1,\gcd(j,k)=1$ 满足条件的三元组个数，但仍然可以用类似的方法解决。

同样枚举 k，我们计算出和 k 互质的数的 A,B 之和，我们就可以统计所有这样的三元组 (i,j,k) 的贡献了。这仍然是一个基础的反演问题，即 $\sum_{i=1}^{n}[\gcd(k,i)=1]A_k=\sum_{d|k}\mu(d)\sum_{id\le n}A_{id}$，于是这个问题同样也可以 $O(n\log n)$ 解决。

该算法可以通过第 7 个和第 8 个测试点。结合算法二和算法三，可以通过前 8 个测试点。

4.6 算法六

当 $A_i=B_i=C_i=1,D_i=E_i=F_i=[i=1]$ 时，实际上我们计算的就是 n 以内的两两互质的三元组个数[1]，即 $\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}[\gcd(i,j)=1][\gcd(i,k)=1][\gcd(j,k)=1]$。

因为有三条互质性质，我们可以先枚举 i，并拆掉其中一条 $\gcd(j,k)=1$，即：

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}[\gcd(i,j)=1][\gcd(i,k)=1][\gcd(j,k)=1]\\=&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{d|j,d|k}\mu(d)[\gcd(i,j)=1][\gcd(i,k)=1]\\=&\sum_{i=1}^{n}\sum_{d=1}^{n}\mu(d)\sum_{jd\le n}\sum_{kd\le n}[\gcd(i,jd)=1][\gcd(i,kd)=1]\end{aligned}$$

[1]该部分分即为SPOJ PCOPTRIP一题

化简到这里看起来并没有头绪，但我们同时能注意到另一条特殊性质，即 $[\gcd(a,bc)=1]=[\gcd(a,b)=1][\gcd(a,c)=1]$。于是我们可以利用这一性质继续推导：

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{d=1}^{n}\mu(d)\sum_{jd\le n}\sum_{kd\le n}[\gcd(i,jd)=1][\gcd(i,kd)=1]\\=&\sum_{i=1}^{n}\sum_{d=1}^{n}\mu(d)[\gcd(i,d)=1]\sum_{jd\le n}\sum_{kd\le n}[\gcd(i,j)=1][\gcd(i,k)=1]\\=&\sum_{i=1}^{n}\sum_{d=1}^{n}\mu(d)[\gcd(i,d)=1]\left(\sum_{jd\le n}[\gcd(i,j)=1]\right)^2\end{aligned}$$

令 $F_{i,m}=\sum_{j\le m}[\gcd(i,j)=1], G_{i,m}=\sum_{j\le m}[\gcd(i,j)=1]\mu(j)$，如果能求出所有的 $O(n\sqrt{n})$ 组 F 和 G，那么原问题就可以通过枚举 i，然后根号分段来解决了。

同时我们可以发现一个性质，所有我们要求的 $F_{i,j}$ 都有 i 是无平方因子数这一条件。因此求 F 是很简单的，令 p 为 i 的一个因子，那么有 $F_{i,j}=F_{\lfloor\frac{i}{p}\rfloor,j}-F_{\lfloor\frac{i}{p}\rfloor,\lfloor\frac{j}{p}\rfloor}$，实际上就是减去含有因子 p 的部分的贡献，这个转移是 $O(1)$ 的。

求 G 也类似处理，可以得到 $G_{i,j}=G_{\lfloor\frac{i}{p}\rfloor,j}+G_{\lfloor\frac{i}{p}\rfloor,\lfloor\frac{j}{p}\rfloor}$。这个转移也同样是 $O(1)$ 的，于是该算法的时间复杂度为 $O(n\sqrt{n})$。

该算法可以通过第 9 个和第 10 个测试点，结合算法二三五，可以通过前 10 个测试点。

4.7 算法七

为了方便后面的描述。我们定义作用在长度为 n 序列 x 上的变换 $g(x)$，满足 $g(x)$ 也是一个长度为 n 的序列，且 $g(x)_i=\sum_{ij\le n}x_{ij}$。

当只有 $D_i=E_i=F_i=[i=1]$ 时，之前的做法已经不再适用。因为发现 $[\gcd(a,bc)=1]=[\gcd(a,b)=1][\gcd(a,c)=1]$ 后，我们仍然无法将之前式子的右侧变得与 d 无关。

但我们仍然可以将结论 $[\gcd(a,bc)=1]=[\gcd(a,b)=1][\gcd(a,c)=1]$ 反过来利用，即：

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}A_iB_jC_k[(i,j)=1][(i,k)=1][(j,k)=1]\\=&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}A_iB_jC_k[(i,jk)=1][(j,k)=1]\end{aligned}$$

接下来枚举 i 和 jk 的 gcd，设其为 d，使用莫比乌斯反演可得：

$$\begin{aligned}&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}A_iB_jC_k[(i,jk)=1][(j,k)=1]\\=&\sum_{d=1}^{n}\mu(d)\sum_{i=1}^{\lfloor\frac{n}{d}\rfloor}A_{id}\sum_{j=1}^{n}\sum_{k=1}^{n}B_jC_k[d|jk][(j,k)=1]\\=&\sum_{d=1}^{n}\mu(d)g(A)_d\sum_{j=1}^{n}\sum_{k=1}^{n}B_jC_k[d|jk][(j,k)=1]\end{aligned}$$

容易发现右侧的式子只与 d 有关，我们可以令 $h_d=\sum_{j=1}^{n}\sum_{k=1}^{n}P_jP_k[d|jk][(j,k)=1]$，于是有：

$$\begin{aligned}h_d&=\sum_{j=1}^{n}\sum_{k=1}^{n}B_jC_k[d|jk][(j,k)=1]\\&=\sum_{j=1}^{n}\sum_{k=1}^{n}B_jC_k[(j,k)=1]\sum_{a|j}\sum_{b|k}[ab=d]\\&=\sum_{a=1}^{n}\sum_{b=1}^{n}[ab=d]\sum_{j=1}^{\lfloor\frac{n}{a}\rfloor}\sum_{k=1}^{\lfloor\frac{n}{b}\rfloor}B_{aj}C_{bk}[(aj,bk)=1]\end{aligned}$$

考虑我们要求的答案，可以考虑枚举 a 和 b，即有：

$$Ans=\sum_{ab\le n}\mu(ab)g(A)_{ab}\sum_{j=1}^{\lfloor\frac{n}{a}\rfloor}\sum_{k=1}^{\lfloor\frac{n}{b}\rfloor}B_{aj}C_{bk}[(aj,bk)=1]$$

由于 $ab\le n$，于是 $\min(a,b)\le\sqrt{n}$，那么我们可以枚举 a,b 中较小的那一个，对于一个 a，我们只要对所有 b 求出 $\sum_{j=1}^{\lfloor\frac{n}{a}\rfloor}\sum_{k=1}^{\lfloor\frac{n}{b}\rfloor}B_{aj}C_{bk}[(aj,bk)=1]$，就可以计算答案了。

事实上这是非常简单的，我们可以先求 $w_i=C_i\sum_{j=1}^{\lfloor\frac{n}{a}\rfloor}B_{aj}[(aj,i)=1]$，再求出 $g(w)$ 即可。

对于前者，只要令 $B'_t=B_t[a|t]$，那么 $w_i=C_i\sum_{j=1}^{n}B_j[(i,j)=1]$，只要莫比乌斯反演即可。时间复杂度为 $O(n\sqrt{n}\log n)$。结合上述所有做法，可以通过前 13 个测试点。

4.8 算法八

之前我们解决 $n\le10^5$ 的问题的方法都是针对数据的性质设计的，而这些数据性质中有一些保证了我们能够方便地进行莫比乌斯反演(如 $D_i=E_i=F_i=[i=1]$)。但容易发现，我们利用利用经典恒等式 $\sum_{d|n}\mu(d)=[n=1]$ 的过程，实际上就是对于一个序列 x，找到一个序列 y 满足 $x_i=\sum_{d|i}y_d$。令 $*$ 为狄利克雷卷积，上式即为 $x=y*1$。

于是有 $y=x*\mu$，这启发我们对于任意序列都可以类似进行反演，于是我们可以基于此对原式进行化简。定义作用在长度为 n 序列 x 上的变换 $f(x)$，满足 $f(x)$ 也是一个长度为

n 的序列，且 $x_i = \sum_{d|i} f(x)_d$。于是有：

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} A_i B_j C_k D_{\gcd(i,j)} E_{\gcd(i,k)} F_{\gcd(j,k)}$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} A_i B_j C_k \sum_{a|i,a|j}\sum_{b|i,b|k}\sum_{c|j,c|k} f(D)_a f(E)_b f(F)_c$$

于是我们可以考虑枚举 a, b, c，即：

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} A_i B_j C_k \sum_{a|i,a|j}\sum_{b|i,b|k}\sum_{c|j,c|k} f(D)_a f(E)_b f(F)_c$$
$$= \sum_{\mathrm{lcm}(a,b)\le n,\mathrm{lcm}(a,c)\le n,\mathrm{lcm}(b,c)\le n} g(A)_{\mathrm{lcm}(a,b)} g(B)_{\mathrm{lcm}(a,c)} g(C)_{\mathrm{lcm}(b,c)} f(D)_a f(E)_b f(F)_c$$

暴力枚举 a, b, c，仍然只能通过前 4 个测试点，考虑优化。我们可以建立一个图论模型，在 $\mathrm{lcm}(i, j) \le n$ 的 i, j 之间连上三种不同的边，权值分别为 $g(A)_{\mathrm{lcm}(i,j)}, g(B)_{\mathrm{lcm}(i,j)}, g(C)_{\mathrm{lcm}(i,j)}$，于是我们接下来所有要做的事情就是枚举图上所有的三元环，每个环都可能会对答案产生贡献。

有一个众所周知的结论：若一张图的边数为 E，则三元环的个数为 $O(E\sqrt{E})$ 的，且我们也可以在这样的时间复杂度内枚举出所有的三元环，于是我们只要暴力枚举这些三元环计算答案即可。

考虑计算时间复杂度：假设 i 和 j 的最大公约数为 g，且 $\mathrm{lcm}(i, j) \le n$，令 $p = i/g, q = j/g$，于是这样的三元组个数一定不超过 $pqg \le n$ 的三元组数目。而后者显然是 $O(n\log^2 n)$ 的，于是这个做法的时间复杂度为 $O(n\sqrt{n}\log^3 n)$，实现优秀的能够得到可观的分数。当 $D_i = E_i = F_i = [i = 1]$ 时，有 $f(D) = \mu$，于是有贡献的三元组会更少，基本可以通过这个部分分。

4.9 算法九

该算法也是命题之初我想到的算法，进行常数优化后，该算法只会在最后三个测试点上超时。下面我们先介绍一些复杂度上或是常数上的优化技巧，为接下来的算法做铺垫。

4.9.1 O(1) GCD

其实在我们之前的复杂度分析中，基本没有考虑欧几里得算法求最大公约数的复杂度，但实际上它是 $O(\log n)$ 的，并不能忽略。这里介绍一种经过预处理 $O(1)$ 求两个不超过 n 的数的最大公约数的方法。

假设我们现在要求 u 和 v 的最大公约数，我们可以将 u 拆分成 $u = a * b * c$ 的形式，其中 a, b, c 要么不超过 $\sqrt{n}$，要么是个素数。如果我们能够这样表示 u，那么就能通过 $O(n)$ 预处理不超过 $\sqrt{n}$ 的两个数的 gcd 来快速求 $\gcd(v, a)$ [2]，也就能够求得 $\gcd(u, v)$ 了。

如何证明一定存在这样的划分，并且任意顺序选取都满足条件呢？如果存在一个 $> \sqrt{n}$ 的 p 是显然的，否则我们可以得到四个两两乘积不小于 $\sqrt{n}$ 的数，这显然是不可能的，这样我们就得到了一个 O(1) GCD 的算法了。

但是在本题中 n 较小，我们可以将不超过 n 的数 u 拆分 $u = a * b$ 的形式，其中 a, b 要么不超过 $n^{2/3}$，要么是个素数。这样我们的预处理时间复杂度变为了 $O(n^{4/3})$，但查询的常数小了很多。

4.9.2 互素判定

如果只是要判定两个数是否互素，问题会简单很多，我们有一个常数比 O(1) GCD 小上许多的做法。在 $\sqrt{n}$ 以内，至多有 65 个素数，而超过 $\sqrt{n}$ 的素因子至多只有一个。我们可以单独处理同为偶数的情况，对于其他情况只要先判断超过 $\sqrt{n}$ 的因子是否相同，再用一个 64 位大整数存储小的素因子的存在情况，直接判断即可。

4.9.3 莫比乌斯变换

问题大致如下：给定长度为 n 的序列 x，求序列 y 满足 $y_i = \sum_{d|i} x_d$ 或 $y_i = \sum_{i|d} x_d$。

传统的解决该问题的做法是 $O(n \log n)$ 的(枚举约数或枚举倍数)，但实际上我们可以有更快的算法。观察我们要求的式子，实际上，若我们将一个数表示为一个向量 $\vec{a}$，其中 $\vec{a}_i$ 表示质因子 p_i 在这个数的因式分解中的次数，那么我们将 x 变为 y 的过程，就是做一次高维前缀和或是高维差分。

而高维前缀和或高维差分是可以枚举每一维单独处理的，而对于一个素因子 p，做一次高维前缀和或差分是 $O(\frac{n}{p})$ 的。

而素数的倒数和是 $O(\log \log n)$ 级别的，因此我们算法的时间复杂度为 $O(n \log \log n)$，实际测试中也比传统的做法的效率高上数倍[3]。

[2] 当 $a < \sqrt{n}$ 时先做一步 gcd 得到 $\gcd(a, v \bmod a)$，再利用预处理的值计算，否则可以直接通过整除关系求 gcd 值

[3] 感谢刘承奥同学给出该算法的优化方法

4.9.4 算法实现

我们仍然利用算法二的结果：

$$\sum_{d=1}^{n} f(D)_d \sum_{k=1}^{n} C_k \left(\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} A_{id} E_{\gcd(id,k)} \right) \left(\sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} B_{jd} F_{\gcd(jd,k)} \right)$$
$$= \sum_{d=1}^{n} f(D)_d \sum_{k=1}^{n} C_k \left(\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} A_{id} \sum_{p \mid id, p \mid k} f(E)_p \right) \left(\sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} B_{jd} \sum_{q \mid jd, q \mid k} f(F)_q \right)$$

如果继续推导，很难取得效果。容易发现瓶颈在于上式右侧部分的计算，考虑如何优化。

考虑一种新的思路，注意到 C 的贡献只与 p 和 q 有关。我们枚举 p 和 q，那么 p 和 q 的取值范围即为 d 的所有倍数的约数所组成的集合，设这个集合为 T，那么：

$$\sum_{d=1}^{n} f(D)_d \sum_{k=1}^{n} C_k \left(\sum_{i=1}^{\lfloor \frac{n}{d} \rfloor} A_{id} \sum_{p \mid id, p \mid k} f(E)_p \right) \left(\sum_{j=1}^{\lfloor \frac{n}{d} \rfloor} B_{jd} \sum_{q \mid jd, q \mid k} f(F)_q \right)$$
$$= \sum_{d=1}^{n} f(D)_d \sum_{k=1}^{n} C_k \sum_{p \in T} \sum_{q \in T} [\mathrm{lcm}(p,q) \mid k] s_p t_q$$
$$= \sum_{d=1}^{n} f(D)_d \sum_{p \in T} \sum_{q \in T} g(C)_{\mathrm{lcm}(p,q)} s_p t_q$$

其中 s, t 可以通过枚举所有倍数的约数的方法求得，考虑计算对所有倍数枚举约数的复杂度，一个数 d 会被枚举 $\sigma_0(d)$ 次，枚举的时间复杂度为 $\sigma_0(d)$，因此总的时间复杂度即为 $O(\sum_{i=1}^{n} \sigma_0^2(i)) = O(n \log^3 n)$[4]。

容易证明，$\sigma_0(ab) \le \sigma_0(a)\sigma_0(b)$，因此有所有 d 的倍数的约数和是不超过 $O(\frac{n}{d}\sigma_0(d) \log n)$ 的，由此我们也可以证得上面的复杂度。

同样，我们发现，所有集合 T 的大小之和就是 $\mathrm{lcm}(p,q) \le n$ 的 p, q 对数，这是 $O(n \log^2 n)$ 的。

我们可以分块，当 $|T| > S$ 时，我们可以暴力枚举 p 和 q，使用 $O(1)$ GCD 计算对答案的贡献，时间复杂度 $O(|T|^2)$，下面考虑当 $|T|$ 较大时怎么做，我们可以换一种方法计算：

$$\sum_{p \in T} \sum_{q \in T} g(C)_{\mathrm{lcm}(p,q)} s_p t_q$$
$$= \sum_{k=1}^{n} C_k \sum_{p \in T, p \mid k} s_p \sum_{q \in T, q \mid k} t_q$$

使用之前莫比乌斯变换的优化技巧，对于一个 $|T| \le S$，我们可以 $O(n \log \log n)$ 求得每个 k 所乘的值，因此也可以快速计算。

[4] http://oeis.org/A061502

接下来我们计算时间复杂度，即不超过 $O(S \times n\log^2 n + \frac{n\log^2 n}{S} \times n\log\log n)$，不妨取 $S = \sqrt{n\log\log n}$，总的时间复杂度不超过 $O(n\sqrt{n\log\log n}\log^2 n)$，常数也并不是很大[5]。

使用这个算法可以得到非常可观的分数。

4.10 算法十

在最终的解法中，我们采取一种完全不一样的思路，这在传统的数论题中是很少遇到的。

首先，我们同时枚举 i 和 k 的公约数 p，以及 j 和 k 的公约数 q，并再次枚举 p 和 q 的公约数 d，于是有：

$$\begin{aligned}
&\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} A_i B_j C_k D_{\gcd(i,j)} E_{\gcd(i,k)} F_{\gcd(j,k)} \\
=&\sum_{k=1}^{n} C_k \sum_{i=1}^{n}\sum_{j=1}^{n} A_i B_j D_{\gcd(i,j)} \sum_{p|i,p|k} f(E)_p \sum_{q|j,q|k} f(F)_q \\
=&\sum_{\mathrm{lcm}(p,q)\le n} f(E)_p f(F)_q g(C)_{\mathrm{lcm}(p,q)} \sum_{ip\le n, jq\le n} A_{ip} B_{jq} D_{\gcd(ip,jq)} \\
=&\sum_{d=1}^{n} \sum_{xy\le \lfloor \frac{n}{d} \rfloor, \gcd(x,y)=1} f(E)_{xd} f(F)_{yd} g(C)_{xyd} \sum_{ix\le \lfloor \frac{n}{d} \rfloor, jy\le \lfloor \frac{n}{d} \rfloor} A_{ixd} B_{jyd} D_{d\gcd(ix,jy)}
\end{aligned}$$

可以发现后面的部分的序列长度都变成了 $\lfloor \frac{n}{d} \rfloor$，于是我们可以枚举 d，令 $P_i = f(E)_{id}, Q_i = f(F)_{id}, R_i = g(C)_{id}, S_i = A_{id}, T_i = B_{id}, W_i = D_{id}$，于是问题变成求：

$$\sum_{xy\le n} P_x Q_y R_{xy} \sum_{ix\le n, jy\le n} S_{ix} T_{jy} W_{\gcd(ix,jy)} = \sum_{xy\le n} P_x Q_y R_{xy} \sum_{ix\le n, jy\le n} S_{ix} T_{jy} \sum_{d|ix, d|jy} f(W)_d$$

由于 $xy \le n$，因此 $\min(x, y) \le \sqrt{n}$，我们可以考虑对于一个 x 处理所有 y 的答案。

$$\begin{aligned}
&\sum_{ix\le n, jy\le n} S_{ix} T_{jy} \sum_{d|ix, d|jy} f(W)_d \\
=&\sum_{jy\le n} T_{jy} \sum_{d|jy} f(W)_d \sum_{ix\le n, d|ix} S_{ix} \\
=&\sum_{y|z} T_z \sum_{d|z} f(W)_d \sum_{d|r} S_r [x|r]
\end{aligned}$$

于是我们只需要做三次莫比乌斯变换就可以对所有 y 统计出答案了。

[5]怀疑有更紧的界，但未证出，也可能只是因为常数很小

这样做一次的时间复杂度为 $O(n\sqrt{n}\log\log n)$，同时常数因子也可以接受(没有 O(1) GCD等大常数操作)。

考虑总的时间复杂度，即为 $O(\sum_{d=1}^{n}\left(\frac{n}{d}\right)^{1.5}\log\log n)$。自然数倒数的 1.5 次方和收敛[6]，因此时间复杂度就是 $O(n\sqrt{n}\log\log n)$。在实现中，还有一些常数方面的优化。我们其实并没有必要进行第一次莫比乌斯变换，而是可以用枚举倍数的约数来代替，这样做是 $O(n\log^4 n)$ 的，常数很小，能起到一定的优化效果。

同样，我们没有必要进行最后一次莫比乌斯变换，而是用暴力枚举倍数的方法代替莫比乌斯变换。

为什么这不会使得复杂度退化呢？容易发现，由于 $y \geq x$，于是我们对于一个 x 来说，这样枚举的总复杂度为 $n\ln\left(\frac{n}{x^2}\right)+O(n)$，我们可以对该式进行一些推导，运用斯特林公式，得到：

$$\begin{aligned}
&\sum_{x\leq\sqrt{n}}\ln\left(\frac{n}{x^2}\right)\\
=&\ln(\prod_{x\leq\sqrt{n}}\frac{n}{x^2})\\
=&2\ln(\frac{\sqrt{n}^{\sqrt{n}}}{(\sqrt{n})!})+O(\sqrt{n})\\
=&2\ln(\exp(\sqrt{n}))+O(\sqrt{n})\\
=&O(\sqrt{n})
\end{aligned}$$

于是这一优化不会影响复杂度，反而会将这一部分原本在复杂度上贡献的 $\log\log$ 去掉，从而影响到算法的常数。当然，没有进行这样的常数优化也足以通过本题了。

5 数据生成方式

容易发现输入是否随机和本题并无太大关联，因此除了序列的特殊性质外所有数都是随机的。

6 命题思路

一开始命制此题的思路来自于 SPOJ PCOPTRIP 一题的加强版[7]，当时我在做到这一题的时候有了一个和题解建立图论模型的方法完全不同的想法[8]，并得到了比题解复杂度和效

[6]即黎曼 ζ 函数中，$\zeta(1.5)$ 的值，具体可以参考 https://en.wikipedia.org/wiki/Riemann_zeta_function

[7]该题满足本题的11至13测试点对应的数据性质

[8]本题的算法七

率更优的解法，这也启发我推广到更加广泛的问题，并尝试解决。原题的题解算法在反演时乘上的系数为莫比乌斯函数时有许多可以直接不用计算的贡献，在代码实现中获得了极大的优势，因此一开始的命题思路就是将原本输入的一个序列变为多个序列，并且使得最终的答案式中无用的贡献几乎没有，这样就能够使得原题的做法被卡满了复杂度，不能通过了。

起初我也被一些传统数论题目的经典套路所束缚，例如思维局限于反演时先枚举 gcd 再使用莫比乌斯反演的“套路”中，化简的过程非常繁琐。但实际上只要对原序列做一次狄利克雷卷积就不用进行反复的枚举了，这能够大大化简我们的运算过程。再比如一直尝试枚举其中一对 i, j 的公约数，但却始终得不到更加优秀的算法，但枚举两对却能得到出乎意料的结果。在推导的过程中，我曾经出现过很多错误，也推出过许多在本题中较为繁琐乃至不必要的做法：例如在算法九中当集合大小不大时采用分块进行等差数列区间加的方法，而不是暴力枚举来求得答案，但这种方法在其他题目中也可能存在价值。在命题之初，我便和陈江伦讨论，在推导过程中他给了我很多启发，也指出了我推导过程中的很多错误。在刘承奥同学验题时，他给出了一个 $O(n \log \log n)$ 进行莫比乌斯变换的想法，对问题的解决和复杂度的优化都有着很大的帮助。

7 总结

本题主要的思想有几个：一是对莫比乌斯反演的拓展、二是对莫比乌斯变换的优化、三是与传统数论题完全不同的同时枚举两组约数的想法、四是要想到算法十中后面复杂的式子是能够计算的。正解的过程并不多，代码量并不大，但思维难度确实不小，也具有很大的启发意义。

在本题中，正解和部分分的关联往往并没有那么大，但本题的部分分中许多做法和思想也值得我们借鉴。例如利用分块进行等差数列修改和求和来进行数论问题中值的维护，数论模型和图论模型的结合等，也能够起到对选手的锻炼作用。

我认为这是一道另辟蹊径的数论题。现在 OI 界的数论题普遍局限在数论函数求和上，并且往往都有解题的模式和套路。作者希望借本题抛砖引玉，引发大家对于类似的数论问题的思考。

如果有人能够想出本题更加优秀的做法，或是有了一些独特的理解想要交流，欢迎与我进行讨论。