浅谈一类最小公倍数的求和问题及其拓展

成都外国语学校 张隽恺

摘 要

本文介绍了一类最小公倍数的求和问题及问题相对一般的情况，以及这类问题上一个通过合并无用状态减少状态数的一个动态规划做法。

1 前言

在信息学竞赛中，与最大公约数相关的问题出现频率较高。但与之相对的最小公倍数问题因为较为复杂而极少出现。本文将简单描述一类与最小公倍数相关问题及这类问题的拓展。

在本文第二节中，首先介绍了一个最小公倍数的求和问题以及几个朴素做法，然后描述了一个通过结合前几个做法的思路，并使用一些性质减少状态数的动态规划做法，接着分析了该算法的复杂度以及一些拓展。

在第三节中简要分析了这类问题拓展到更一般情况下的结果以及上一个算法此时的表现。

2 一类最小公倍数的求和问题

例题 1 给定正整数 n，求

$$\sum_{S\subset\{1,2,\ldots,n\}} LCM(S)$$

$n \le 2000$，答案对质数取模。

这里 $LCM(S)$ 表示集合 S 中的所有元素的最小公倍数。特别的，定义 $LCM(\emptyset) = 1$。

2.1 算法一

使用朴素的动态规划算法，设 $dp_{i,j}$ 表示前 i 个数组成的集合的子集中，子集内所有元素的最小公倍数为 j 的子集数量。

在从 dp_i 转移到 dp_{i+1} 时，枚举 $i+1$ 是否在子集中即可。

设 S_n 为 $\{1,2,...,n\}$ 的子集的最小公倍数的种类数，则这个算法的时间复杂度为 $O(S_n * n * \log n)$ 或 $O(S_n * n)$。可以在 1 秒内通过 $n = 50$ 的数据。

2.2 算法二

2.2.1 定义

为了更好地描述接下来的算法，在这里先进行一些定义。

设 $1,2,...,n$ 中的所有质数从小到大排序后为 $p_1, p_2, ..., p_k$。定义正整数 a 能被 $p_1, p_2, ..., p_k$ 分解，当且仅当将 a 质因数分解后，得到的所有质因数都属于 $\{p_1, p_2, ..., p_k\}$。对于能被 $p_1, p_2, ..., p_k$ 分解的 a，a 进行质因数分解的结果为 $a = p_1^{q_1} * p_2^{q_2} * ... * p_k^{q_k}$，定义 a 的分解式为一个 k 维向量 $F_a = (q_1, q_2, ..., q_k)$。

对于两个 k 维向量 $F = (q_1, q_2, ..., q_k), F' = (q_1', q_2', ..., q_k')$，定义 $F \le F'$ 当且仅当 $\forall i \in \{1,2,...,k\}, q_i \le q_i'$.

对于 $F = (q_1, q_2, ..., q_k), F' = (q_1', q_2', ..., q_k')$，定义它们的 max 为

$$\max(F, F') = (\max(q_1, q_1'), \max(q_2, q_2'), ..., \max(q_k, q_k'))$$

显然这样定义的 max 满足交换律。使用类似的方式定义 min：

$$\min(F, F') = (\min(q_1, q_1'), \min(q_2, q_2'), ..., \min(q_k, q_k'))$$

定义 $\max(F_1, F_2, ..., F_l) = \max(\max(F_1, \max(F_2, ..., F_l))$。

可以得到如下结论：

Lemma 2.1. 如果 a, b 都能被 $p_1, p_2, ..., p_k$ 分解，则 $LCM(a,b)$ 能被 $p_1, p_2, ..., p_k$ 分解，且 $F_{LCM(a,b)} = \max(F_a, F_b)$。

证明. 设 $F_a = (a_1, a_2, ..., a_k), F_b = (b_1, b_2, ..., b_k)$。则 $a = \prod_{i=1}^{k} p_i^{a_i}, b = \prod_{i=1}^{k} p_i^{b_i}$。因此 $LCM(a,b) = \prod_{i=1}^{k} p_i^{\max(a_i, b_i)}$。因此 $LCM(a,b)$ 能被 $p_1, p_2, ..., p_k$ 分解，且 $F_{LCM(a,b)} = (\max(a_1, b_1), ..., \max(a_k, b_k)) = \max(F_a, F_b)$。

□

扩展到多个数的情况，可以得到：

Lemma 2.2. 如果 $a_1, a_2, ..., a_l$ 都能被 $p_1, p_2, ..., p_k$ 分解，则 $LCM(\{a_1, a_2, ..., a_l\})$ 能被 $p_1, p_2, ..., p_k$ 分解，且 $F_{LCM(\{a_1, a_2, ..., a_l\})} = \max(F_{a_1}, F_{a_2}, ..., F_{a_l})$。

证明. 使用与 Lemma 2.1 相同的方法即可证明。

□

2.2.2 算法

显然 $1,2,...,n$ 都能被 $p_1,p_2,...,p_k$ 分解。根据 Lemma 2.2，所有子集的最小公倍数也能被 $p_1,p_2,...,p_k$ 分解，且满足子集中所有数的最小公倍数等于 a 的子集个数等于满足子集中所有数的分解式的 max 等于 F_a 的子集数量。

考虑如下引理：

Lemma 2.3. 对于分解式 F_a,F_b,F_c，$\max(F_a,F_b)\le F_c$ 当且仅当 $F_a\le F_c,F_b\le F_c$。

证明. 设 $F_a=(a_1,a_2,...,a_k),F_b=(b_1,b_2,...,b_k),F_c=(c_1,c_2,...,c_k)$，因为 $F_a\le F_c,F_b\le F_c$，则 $\forall i\in\{1,2,...,k\},a_i\le c_i,b_i\le c_i$，因此 $\max(a_i,b_i)\le c_i$，反之同理。因此引理成立。

□

Lemma 2.4. 设 $G=(\lfloor\log_{p_1}n\rfloor,\lfloor\log_{p_2}n\rfloor,...,\lfloor\log_{p_k}n\rfloor)$，则 $\forall i\in\{1,2,...,n\},F_i\le G$。

证明. 若引理不成立，则 $\exists a\in\{1,2,...,n\}$ 满足 $F_a\not\le G$。设 $F_a=(a_1,a_2,...,a_k)$，则 $\exists r\in\{1,2,...,k\},a_r>\lfloor\log_{p_r}n\rfloor$。此时 $a=\prod_{i=1}^k p_i^{a_i}\ge p_r^{a_r}\ge p_r^{\lfloor\log_{p_r}n\rfloor+1}>n$，与 $a\in\{1,2,...,n\}$ 矛盾。因此引理成立。

□

Lemma 2.5. 对于所有满足 s 能被 $p_1,p_2,...,p_k$ 分解，且 $F_s\le(\lfloor\log_{p_1}n\rfloor,\lfloor\log_{p_2}n\rfloor,...,\lfloor\log_{p_k}n\rfloor)$ 的正整数 s，存在一个集合 $S\subset\{1,2,...,n\}$ 满足 $LCM(S)=s$。

证明. 设 $F_s=(q_1,q_2,...,q_k)$，则 $\forall i\in\{1,2,...,k\},q_i\le\lfloor\log_{p_i}n\rfloor,p_i^{q_i}\le n$。构造序列 $t_{1,...,k},\forall i\in\{1,2,...,k\},t_i=p_i^{q_i}$，则 $LCM(t_1,t_2,...,t_k)=\prod_{i=1}^k p_i^{q_i}$，序列中元素构成的集合满足条件。

□

满足 $F\le(\lfloor\log_{p_1}n\rfloor,\lfloor\log_{p_2}n\rfloor,...,\lfloor\log_{p_k}n\rfloor)$ 且 F 每一维上的值均为非负整数的 F 共有 $\prod_{i=1}^k(\lfloor\log_{p_i}n\rfloor+1)$ 个。根据 Lemma 2.5，对于所有满足这个条件的 F，都存在一个 $S\subset\{1,2,...,n\}$ 满足 $LCM(S)$ 的分解式等于它。根据 Lemma 2.3 和 Lemma 2.4，$\forall S\subset\{1,2,...,n\},F_{LCM(S)}\le(\lfloor\log_{p_1}n\rfloor,\lfloor\log_{p_2}n\rfloor,...,\lfloor\log_{p_k}n\rfloor)$。因此最小公倍数的种类数 S_n 等于 $\prod_{i=1}^k(\lfloor\log_{p_i}n\rfloor+1)$。

对于每一个满足这一条件的 F，设 h_F 为满足子集中所有数的最小公倍数的分解式 F_{LCM} 满足 $F_{LCM}\le F$ 的子集数量。设 ct_F 为 $\{1,2,...,n\}$ 中满足 $F_i\le F$ 的数 i 的数量。根据 Lemma 2.3，满足条件的子集中的所有数 a 都满足 $F_a\le F$，且若子集中所有数 a 都满足 $F_a\le F$，则这个集合满足条件。因此 $h_F=2^{ct_F}$。因为 $ct_F=\sum_{i=1}^n[F_i\le F]$，求出 ct 相当于求高维前缀和，使用 [1] 中的做法可以以 $O(S_n*k)$ 的复杂度求出 ct。

设 $H_{F'}$ 为子集中所有数的最小公倍数的分解式 $F_{LCM}=F'$ 的子集数量。通过 H 求 h 为高维前缀和的过程，因此通过高维差分可以通过 h 求 H。

这个做法的复杂度为 $O(S_n*k)$，效率与上一个算法接近。

2.3 算法三

$1,...,n$ 中的所有数质因数分解后最多有一个大于 $\sqrt{n}$ 的质因子。在动态规划时，只记录状态中所有不超过 $\sqrt{n}$ 的质因子。将所有数按其大于 $\sqrt{n}$ 的质因子分类，对每一类进行动态规划。这部分也可以使用算法二中的高维前缀和进行优化。

这个算法的具体细节与之后的算法无关，因此在这里不再展开。

2.4 算法四

2.4.1 算法

考虑算法一中的动态规划算法，设 $g_{i,F}$ 表示前 i 个数的集合的子集中，子集的最小公倍数的分解式为 F 的方案数。

定义 $g_{i,F}$ 对答案的贡献 $v_{i,F}$ 为：(设 $F=(q_1,q_2,...,q_k)$)

$$v_{i,F}=\sum_{S\subset\{i+1,i+2,...,n\}} LCM(\prod_{j=1}^{k} p_j^{q_j}, LCM(S))$$

则有如下引理：

Lemma 2.6. 设问题的答案为 ans，则

$$\forall i\in\{1,2,...,n\}, \sum_F g_{i,F}*v_{i,F}=ans$$

证明.

$$\begin{aligned}
ans &= \sum_{S\subset\{1,2,...,n\}} LCM(S)\\
&= \sum_{S_1\subset\{1,2,...,i\}} \sum_{S_2\subset\{i+1,i+2,...,n\}} LCM(LCM(S_1),LCM(S_2))\\
&= \sum_{F=(q_1,q_2,...,q_k)} g_{i,F} \sum_{S_2\subset\{i+1,i+2,...,n\}} LCM(\prod_{j=1}^{k} p_j^{q_j}, LCM(S_2))\\
&= \sum_F g_{i,F}*v_{i,F}
\end{aligned}$$

□

Lemma 2.7. 对于 $g_{i,F}$，设 $LCM(i+1,i+2,...,n)=s, F=(q_1,q_2,...,q_k), F_s=(q_1',q_2',...,q_k')$。若存在 $a\in\{1,2,...,k\}$ 满足 $q_a>q_a'$，设 $F'=(q_1,q_2,...,q_{a-1},q_a-1,q_{a+1},q_{a+2},...,q_k)$，则 $v_{i,F}=p_a*v_{i,F'}$。

证明. 考虑集合 $S\subset\{i+1,i+2,...,n\}$，设 $LCM(S)=s', F_{s'}=(s_1,s_2,...,s_k)$。由 2.3 有 $F_{s'}\le F_s$。因此 $s_a\le q_a'\le q_a-1$。

$$LCM(\prod_{j=1}^{k} p_j^{q_j}, s') = (\prod_{j=1,j\neq a}^{k} p_j^{\max(q_j,s_j)}) * p_a^{\max(q_a,s_a)} = (\prod_{j=1,j\neq a}^{k} p_j^{\max(q_j,s_j)}) * p_a^{q_a}$$

$$LCM((\prod_{j=1,j\neq a}^{k} p_j^{q_j}) * p_a^{q_a-1}, s') = (\prod_{j=1,j\neq a}^{k} p_j^{\max(q_j,s_j)}) * p_a^{\max(q_a-1,s_a)} = (\prod_{j=1,j\neq a}^{k} p_j^{\max(q_j,s_j)}) * p_a^{q_a-1}$$

所以 $LCM(\prod_{j=1}^{k} p_j^{q_j}, s') = p_a * LCM((\prod_{j=1,j\neq a}^{k} p_j^{q_j}) * p_a^{q_a-1}, s'))$。对所有 S 求和即可得到 $v_{i,F} = p_a * v_{i,F'}$。

□

对于一个满足 Lemma 2.7 中条件的 F，设 g_i' 为：

1. $g_{i,F}' = 0, g_{i,F'}' = g_{i,F'} + g_{i,F} * p_a$

2. 对于所有满足 $F'' \neq F, F'' \neq F'$ 的 $F'', g_{i,F''}' = g_{i,F''}$

根据 Lemma 2.7 可得 $\sum_F g_{i,F} * v_{i,F} = \sum_F g_{i,F}' * v_{i,F}$。根据 Lemma 2.6，将 g_i 替换为 g_i' 不会改变答案。定义将 g_i 变为 g_i' 的过程为一次操作。对于任意的 g_i，易证在有限次操作后可以使所有非零的 $g_{i,F}$ 都满足 $F \leq F_{LCM(i+1,i+2,...,n)}$。

由 Lemma 2.3，$\{1, 2, ..., i\}$ 的任意子集 S 满足 $F_S \leq F_{LCM(1,2,...,i)}$。对于每一个 i，在求出 g_i 后进行若干次操作，直到所有满足 $g_{i,F}$ 非零的 F 都满足 $F \leq F_{LCM(i+1,i+2,...,n)}$。这时满足 $g_{i,F}$ 非零的 F 满足 $F \leq F_{LCM(1,2,...,i)}, F \leq F_{LCM(i+1,i+2,...,n)}$。在从 g_i 转移到 g_{i+1} 的过程中，满足 $g_{i+1,F}$ 非零的 F 一定满足 $F \leq F_{LCM(1,2,...,i+1)}, F \leq F_{LCM(i+1,i+2,...,n)}$。记向量 $L_i = \min(F_{LCM(1,2,...,i)}, F_{LCM(i+1,i+2,...,n)}), L_i' = \min(F_{LCM(1,2,...,i)}, F_{LCM(i,i+1,...,n)})$。则可以得到这样一个算法：

维护若干状态 F 以及这些状态对应的 g_F，依次考虑 $i = 1, 2, ..., n$，对于每个 i 依次进行以下三步操作：

1. 将当前维护的 F 的范围变为所有满足 $F \leq L_i'$ 且 F 的每一维都是非负整数的 F。显然 $L_{i-1} \leq L_i'$，因此这一步不会改变 g。

2. 计算在子集中加入 i 的转移，进行 g 上的转移。

3. 通过进行若干次操作，使所有非零的 g_F 都满足 $F \leq F_{LCM(i+1,i+2,...,n)}$，将维护的 F 的范围变为所有满足 $F \leq L_{i+1}$ 且 F 的每一维都是非负整数的 F。

根据 Lemma 2.7，在进行 i 的操作时，每一个操作后 $\sum_F g_F * v_{i,F}$ 不变。因此在进行上面的操作过程中，这个值始终等于答案。因为 $L_{n+1} = (0, 0, ..., 0)$，进行所有操作后只有一个状态 $F = (0, 0, ..., 0)$，显然 $v_{n+1,F} = 1$，因此这个状态对应的 g 即为问题的答案。

使用算法二中的方式优化转移。设 $f_F = \sum_{F' \leq F} g_{F'}$，即 f 为 g 进行高维前缀和后的结果。在这个算法中，任意时刻都存在一个 k 维向量 L_i'' (等于 L_i 或 L_i')，满足当前所有满足 g_F 非零的 F 都满足 $F \leq L_i''$。因此只需要维护满足 $F \leq L_i''$ 的 f_F。在上面的过程中代替 g 维护 f，依次分析每一步操作对 f 的影响（设 f' 为操作后的 f）：

对于第一步操作，g 不会改变。设 $L_i = (l_1, l_2, ..., l_k)$，因为所有满足非零的 g_F 都满足 $F \leq L_i$，因此对于任意状态 F，有 $f_F' = f_{\min(F,L_i)}$。

对于加入一个数 i 的转移对 f 的影响，根据 2.3 以及算法二中的分析，可以得到：

$$f'_F = \begin{cases} 2 * f_F, & F_{i+1} \le F \\ f_F, & F_{i+1} \not\le F \end{cases} \tag{1}$$

对于第三步操作，设 $L'_i = (l'_1, l'_2, ..., l'_k)$，将操作分为若干步，每一步选择一维 a，对所有第 a 维上的值等于 l'_a 的状态 F 进行操作，使非零的 $g_{i,F}$ 都满足 F 第 a 维上的值小于 l'_a。

考虑这样的一步操作对 g 的影响（设 g' 为操作后的 g）：

$$g'_{(q_1,q_2,...,q_k)} = \begin{cases} g_{(q_1,q_2,...,q_k)}, & q_a < l_a - 1 \\ g_{(q_1,q_2,...,q_k)} + p_a * g_{(q_1,q_2,...q_{a-1},q_a+1,q_{a+1},...,q_k)}, & q_a = l_a - 1 \\ 0, & q_a = l_a \end{cases}$$

根据 f 的定义，可以求出操作对 f 的影响：

$$f'_{(q_1,q_2,...,q_k)} = \begin{cases} f_{(q_1,q_2,...,q_k)}, & q_a < l_a - 1 \\ p_a * f_{(q_1,q_2,...q_{a-1},q_a+1,q_{a+1},...,q_k)} - (p_a - 1) * f_{(q_1,q_2,...,q_k)}, & q_a = l_a - 1 \end{cases}$$

(在操作后所有值非零的状态第 a 维上的值都小于 l_a，因此不需要保留满足 $q_a = l_a$ 的 f)

通过对操作次数归纳可得，在进行 x 次这样的操作后，得到的 f' 为：

$$f'_{(q_1,q_2,...,q_k)} = \begin{cases} f_{(q_1,q_2,...,q_k)}, & q_a < l_a - x \\ p_a^x * f_{(q_1,q_2,...q_{a-1},l_a,q_{a+1},...,q_k)} - \sum_{i=0}^{x-1} p_a^i (p_a - 1) * f_{(q_1,q_2,...q_{a-1},l_a-x+i,q_{a+1},...,q_k)}, & q_a = l_a - x \end{cases}$$

依次进行每一维上的操作即可。

通过这三步操作即可维护 F 从而求出答案，接下来分析这个算法的实现与复杂度。

2.4.2 实现与复杂度分析

在算法的过程中，任意时刻都存在一个各维的值都为非负整数的向量 $L'' = (l_1, l_2, ..., l_k)$，满足当前维护的所有 f_F 满足 $F \le L''$。

定义一个从当前所有的状态 $\{(q_1, q_2, ..., q_k) \mid \forall j \in \{1, 2, ..., k\}, 0 \le q_j \le l_j, q_j \in \mathbb{N}\}$ 到 $\{0, 1, ..., \prod_{j=1}^{k}(l_j + 1) - 1\}$ 的映射：$f1 : \{(q_1, q_2, ..., q_k) \mid \forall j \in \{1, 2, ..., k\}, 0 \le q_j \le l_j, q_j \in \mathbb{N}\} \longrightarrow \{0, 1, ..., \prod_{j=1}^{k}(l_j + 1) - 1\}$：

$$f1((q_1, q_2, ..., q_k)) = \sum_{j=1}^{k} q_j * (\prod_{j'=j+1}^{k} (l_{j'} + 1))$$

Lemma 2.8. $f1$ 是一个双射。

证明. 每一个 $(q_1, q_2, ..., q_k)$ 只会对应一个 $\{0, 1, ..., \prod_{j=1}^{k}(l_j+1)-1\}$ 中的数。接下来证明 $\forall s \in \{0, 1, ..., \prod_{j=1}^{k}(l_j+1)\}$，存在唯一一个满足 $\forall j \in \{1, 2, ..., k\}, 0 \le q_j \le l_j, q_j \in \mathbb{N}$ 的 $(q_1, q_2, ..., q_k)$。

由 $f1$ 的定义可以得到 $(\sum_{j=1}^{k-1} q_j * (\prod_{j'=j+1}^{k-1}(l_{j'}+1))) * (l_k+1) + q_k = s$，则 $q_k \equiv s (\bmod l_k+1)$。因为 $q_k \in \{0, 1, ..., l_k\}$，所以存在唯一一个满足条件的 q_k。q_k 固定后，剩余部分相当于 $\sum_{j=1}^{k-1} q_j * (\prod_{j'=j+1}^{k-1}(l_{j'}+1)) = \frac{s-q_k}{l_k+1}$，且满足 $\frac{s-q_k}{l_k+1} \in \{0, 1, 2, ..., \prod_{j=1}^{k-1}(l_j+1)-1\}$。使用归纳法即可证明引理。□

根据 Lemma 2.8，可以对于所有的 $s \in \{0, 1, ..., \prod_{j=1}^{k}(l_j+1)\}$，记录 s 通过 $f1$ 对应的 F 对应的 f_F。依次考虑关于 F 的三步操作：

设操作前的状态为 f，操作后的状态为 f'。第一步操作相当于对于每一个满足 $F \le L_i' = (l_1', l_2', ..., l_k')$ 的 F，设 $F = (q_1, q_2, ..., q_k)$，则 $f_F' = f_{(\min(q_1, l_1'), \min(q_2, l_2'), ..., \min(q_k, l_k'))}$。

使用类似深度优先搜索的方式枚举 F 每一维上的值，在枚举每一维上的值时，可以求出 $(\min(q_1, l_1'), \min(q_2, l_2'), ..., \min(q_k, l_k'))$ 每一维上的值以及这两个状态通过 $f1$ 对应的值。如果记录所有满足 $l_i' > 0$ 的维，只枚举这些维上的值，则枚举的复杂度为 $O(\prod_{j=1}^{k}(l_j'+1))$。

对于后两个操作，每一个 f_F 只会影响一个 f' 中的值。使用类似深度优先搜索的方式枚举每一个 F，可以在这个过程中求出它影响的 f' 中的状态以及系数。这两步操作的复杂度同样为 $O(\prod_{j=1}^{k}(l_j'+1))$。

因此这个实现的复杂度与每一个 L_i' 的 $\prod_{j=1}^{k}(l_j'+1)$ 的和成正比。记这个值为 T_n。可以得到 T_n 的大小（所有大小保留三位有效数字）：

n	50	100	200	500	1000
S_n	4.42×10^5	6.61×10^8	4.75×10^{15}	1.08×10^{31}	5.59×10^{53}
T_n	5.69×10^4	1.83×10^7	4.00×10^{10}	8.79×10^{19}	2.43×10^{33}

这样的效率不优于算法三，但根据算法三中优化的方式，可以对算法四进行修改：

将所有数按照它们的最大质因子从大到小排序（最大质因子相同时从小到大排序），按照这个顺序进行算法四中的做法。

按照最大质因子大小排序后进行算法四的做法，设这时每一个 L_i' 的 $\prod_{j=1}^{k}(l_j'+1)$ 的和为 T_n'，有：

n	500	1000	2000	3000	5000	10^4
T_n'	1.35×10^5	1.44×10^6	2.45×10^7	1.97×10^8	3.60×10^9	4.66×10^{11}

T_n' 的增长速度仍然为指数级，但 T_n' 的增长速度相对 S_n 较慢，且在 $n \le 2000$ 时 $O(T_n')$ 的复杂度是可以接受的。因此算法四可以在 1 秒内通过 $n = 2000 \sim 2500$ 的数据。

实际上这个算法的复杂度并不是完全满的，实际效率可以更快一点。

通过按这个顺序进行动态规划，可以得到相对小的状态数 T_n'。但这样的状态数不是最优的，通过使用部分随机化算法 [2,3]，可以得到更小的状态数。例如作者使用随机化算法得到的部分状态数 T_n''：

n	500	1000	2000	3000
T_n'	1.35×10^5	1.44×10^6	2.46×10^7	1.97×10^8
T_n''	1.25×10^5	1.35×10^6	2.34×10^7	1.93×10^8

但因为作者水平有限，并没有得到优于指数级复杂度的求出最优状态数的算法。在这里不再进行分析。

2.5 拓展

算法四可以拓展到相对一般的情况，即：

给定 n 个非负整数 $a_1, a_2, ..., a_n$ 以及一个满足 $\forall i \in \mathbb{N}^+, f(i) \neq 0$ 的积性函数 f，求：

$$\sum_{S\subset\{1,2,...,n\}} (\prod_{i\in S} a_i) * f(LCM(S)) \tag{2}$$

重新定义 $v_{i,F}$：$v_{i,(q_1,q_2,...,q_k)} = \sum_{S\subset\{i+1,i+2,...,n\}} (\prod_{i\in S} a_i) * f(LCM(\prod_{j=1}^{k} p_j^{q_j}, LCM(S)))$。

在这个定义下，Lemma 2.7 的结论变为 $v_{i,F} = \frac{f(p_a^{q_a})}{f(p_a^{q_a-1})} * v_{i,F'}$，(1) 中 $2 * f_F$ 变为 $(a_i+1) * f_F$。剩余部分使用与算法四完全相同的方式推导即可。

例题 2 Math is Fun [1]

给一个长度为 n 的正整数序列 v，定义一个序列的权值为序列中所有元素的最小公倍数的平方与序列中所有元素的最大公约数的乘积。求出这个序列所有 $2^n - 1$ 个非空子序列的权值之和模 $10^9 + 7$ 的结果。一共有 T 组数据。

$n \leq 100, v_i \leq 1000, T \leq 50$。时限 4 秒。

设 f_i 表示所有满足最大公约数等于 i 的非空子序列的序列元素最小公倍数的平方的和，g_i 表示所有满足最大公约数为 i 的倍数的非空子序列的序列元素最小公倍数的平方的和。

首先求出 g_i。所有数的最大公约数为 i 的倍数当且仅当所有数都是 i 的倍数。因此 g_i 等于这些数的所有非空子序列的序列元素最小公倍数的平方和。显然若 $g|s_1, s_2, ..., s_l$，则 $LCM(s_1, s_2, ..., s_l) = LCM(\frac{s_1}{g}, \frac{s_2}{g}, ..., \frac{s_l}{g}) * g$。则 g_i 等于只考虑序列中是 i 的倍数的数得到的序列，将这个序列所有元素除以 i 后，所有非空子序列的序列元素最小公倍数的平方和乘 i^2 的结果。

[1] 来源：Petrozavodsk Programming Camp,Summer 2016,Day 8,DPRK Contest

序列中所有元素都不会超过 $\frac{1000}{i}$。设数 a 出现了 c_a 次。令 $v_a = 2^{c_a} - 1$，再设函数 $f(n) = n^2$，则问题变为 (2) 的形式，可以使用算法四解决。计算所有 g_i 的复杂度与 $\sum_{i=1}^{1000} T'_{\frac{1000}{i}}$ 成正比。

然后通过 g_i 求出所有 f_i，有 $g_i = \sum_{i|j} f_j$，可以通过对 g 容斥求出 f。这一部分的复杂度相对于上一部分可以忽略。

$\sum_{i=1}^{1000} T'_{\frac{1000}{i}} \leq 1.7 \times 10^6$，这个复杂度可以接受。作者的实现在此问题上运行时间约为 0.1 秒。

3 问题的拓展

对于更为一般的情况，可以将问题描述为如下形式：

给出 n 个 d 维向量 $V_1, V_2, ..., V_n$ 以及 n 个非负整数 $a_1, a_2, ..., a_n$。给出的所有向量满足每一维上的值都为非负整数。对两个向量取 max 的定义同上一节。求：

$$\sum_{S \subset \{1,2,...,n\}} (\prod_{i \in S} a_i) * f(max_{i \in S} V_i)$$

其中 f 为一个值域为正整数的函数，满足 $f((q_1, q_2, ..., q_d)) = \prod_{i=1}^{d} f_i(q_i)$，且所有 f_i 均为值域为正整数的函数。

上一个问题为这个问题的一个特例，满足给出的所有向量 $V = (q_1, q_2, ..., q_d)$ 都满足 $\sum_{i=1}^{d} q_i * \log p_i \leq \log n$。

在不存在特殊限制的情况下，算法四中的状态数总和 T'_n 与 $S_n * n$ 接近。但在存在类似上一个限制的限制时，通过与算法四中类似的调整顺序的方式，得到的 T'_n 可以远小于 $S_n * d$，例如：

满足给出的所有向量 $V = (q_1, q_2, ..., q_d)$ 都满足 $\sum_{i=1}^{d} q_i * i \leq m$ 时（$d = m$，此处 S_n, T'_n 的定义同算法四中的定义，将所有向量按照最后一个非零位置排序）：

m	15	18	20	23	25
$S_n * d$	4.25×10^8	2.65×10^{10}	3.22×10^{11}	8.44×10^{12}	1.70×10^{14}
T'_n	7.63×10^5	9.31×10^6	4.99×10^7	5.83×10^8	2.96×10^9

在类似的问题上，算法四有优于算法一，二等做法的表现。与上一个问题相同，这里也存在使 T'_n 更小的向量顺序。

4 总结

本文描述的算法在动态规划的过程中，通过进行变换，将状态中不会被剩余的动态规划过程中改变的部分进行合并，从而达到减小状态数的结果。

由于最小公倍数问题的特殊性质，在改变动态规划的顺序后，该算法可以在这个问题上得到较优秀的结果。

同时，这个思想可以对更多的动态规划的问题进行优化。

感谢

感谢中国计算机学会提供学习和交流的平台。

感谢学校以及教练对我的支持。

感谢家人对我的关心。

参考文献

[1] OI Wiki，” 基于 DP 计算高维前缀和”.

[2] Wikipedia，”Simulated annealing”.

[3] 任一恒，《非完美算法初探》，IOI2009 中国国家队候选队员论文.