浅谈几种分解质因数方法

重庆市巴蜀中学 罗思远

摘 要

数论是信息学竞赛中一大板块，其中有不少题目都涉及分解质因数，分解质因数也是学术界具有重要地位的问题之一。本文介绍了三种在 OI 范围内认知度尚且较低，算法本身也比较简单的质因数分解算法：SQUFOF，CFRAC 和二次筛法。

1 约定

本文中，涉及到分解质因数时，设 N 为要分解的数。定义对 N 分解质因数的过程为将 N 写为素数之积的形式，也即求出 $p_1, p_2, \ldots, p_k$ 使得 $\forall 1 \le i \le k$，p_i 均为素数，且 $\prod_{1\le i\le k} p_i = n$。

称正整数 d 是正整数 n 的非平凡因子当且仅当 $d|n$ 且 $1 < d < n$。

2 前置知识

2.1 Miller Rabin 素数判断

先简要介绍 Miller Rabin 素数判断。设待判断的数为 n（不妨假设 n 为奇数），设 $n-1 = 2^d \cdot r$，这里 r 为奇数，d 为正整数。选取 a，用快速幂计算 $a^r \bmod n$ 的值 x，接着对 x 执行至多 d 次自乘。最终，若 n 为素数，x 的值一定为 1，这是因为由费马小定理，当 n 为素数时 $a^{n-1} \equiv 1 \pmod n$ 总成立。当 x 变为 1 时算法终止。

如果在某次自乘后 x 变为 1，自乘前 x 的值在模 n 意义下非 1 也非 $n-1$，则 n 一定不是素数。这是因为，若 $2 \le x \le n-2$ 满足 $x^2 \equiv 1 \pmod n$，就有 $n|(x^2-1) = (x+1)(x-1)$，若 n 为素数，就有 $n|x-1$ 或 $n|x+1$ 而 $x+1, x-1$ 均在 $[1, n-1]$ 内，不是 n 的倍数，矛盾。

若对于某个底数 a 与合数 n，上面的过程不能确认 n 为合数，我们称 a 是 n 的一个 strong liar。[3] 指出，对于合数 n，strong liar 的数量不超过 $\frac{n}{4}$，故如果重复执行 k 轮上面的算法，每轮算法均随机选取 a，则错误率不超过 4^{-k}。[3] 还指出，对于 $3,825,123,056,546,413,051 > 10^{18}$ 以内的正整数 n，选取前 9 个素数作为 a 即可保证结果正确；对于 2^{64} 以内的正整数 n，选取前 12 个素数作为 a 即可保证结果正确。

Miller Rabin 素数判断共需执行 $O(k \log n)$ 次模 n 意义下的乘法，其中 k 是轮数。

2.2 试除法

设 n 为要分解的数。若 n 不是素数，则 n 的最小质因子一定小于等于 $\sqrt{n}$。先用 $O(\sqrt{n})$ 的线性筛求出所有 $\sqrt{n}$ 以内的所有素数，分解时只需枚举 $\sqrt{n}$ 以内的所有素数即可。根据素数定理 [1]，小于等于 x 的素数有 $O(\frac{x}{\log x})$ 个，因此时间复杂度为预处理 $O(\sqrt{n})$，单次分解 $O(\frac{\sqrt{n}}{\log n})$。

例 2.2.1. 给出 n 个正整数 $a_1, \ldots, a_n$，判断其中是否存在两个数 a_i, a_j $(i \neq j)$ 使得 a_i, a_j 不互素。$1 \leq n \leq 10^5$，$1 \leq a_i \leq 10^9$，时间限制 *3* 秒。[1]

解法　直接用试除法对所有 a_i 分解质因数，判断是否有质因子出现在两个以上 a_i 中即可。时间复杂度 $O(n\frac{\sqrt{m}}{\log m})$，其中 $m = \max a_i$。特别地，由于需要试除的质数个数很少，可以在适当预处理后使用 Barrete 模乘 [2] 优化除法运算的常数。 ■

需要注意的是，某些题目中，根据题目的特殊性质，可能可以调整试除上界来取得更优秀的复杂度。

例 2.2.2. T 次询问：给出 x 求 $\mu(x)$。$T \leq 5000, 1 \leq x \leq 10^{18}$。时间限制 *2* 秒。[2]

解法　直接用试除法对所有 a_i 分解质因数，但试除的质因子只枚举 $\sqrt[3]{x}$ 以内的。若除完后还有剩余，只可能是质数，质数的平方，或两个质数相乘。前两种情况分别用 Miller Rabin 素数判断与直接计算平方根判出，若都不是即可确定是两个质数相乘。容易发现，只凭以上信息，就能确定 $\mu(x)$ 的值。时间复杂度 $O(T\frac{\sqrt[3]{m}}{\log m})$，其中 $m = \max x$。 ■

2.3 连分数的简单介绍

在介绍下面的质因数分解算法之前，先要了解连分数的概念以及几个性质。

定义 对任意实数 x，都存在唯一的有限或无限的数列 $[a_0; a_1, a_2, \ldots]$ 使得

- $a_0 \in \mathbb{Z}$；$\forall i > 0$，$a_i \in \mathbb{N}^*$；
- $x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots}}}$。

称 $[a_0; a_1, a_2, \ldots]$ 为 x 的一般连分数表示。以下，如果没有特殊说明，所说的“连分数”均为“一般连分数”之意。

求出连分数表示 假设要求出实数 x 的连分数表示 $[a_0; a_1, a_2, \ldots]$，可以这样计算：

[1] https://codeforces.com/contest/1771/problem/C

[2] 修改自 https://www.luogu.com.cn/problem/T129723

- 令 $x_0 = x$。
- 令 $a_0 = \lfloor x_0 \rfloor$，$x_1 = \dfrac{1}{x_0 - a_0}$。
- 令 $a_1 = \lfloor x_1 \rfloor$，$x_2 = \dfrac{1}{x_1 - a_1}$。
- 令 $a_2 = \lfloor x_2 \rfloor$，$x_3 = \dfrac{1}{x_2 - a_2}$。
- 以此类推。若中途某时刻 $a_i = x_i$（也就是 x_i 为整数），则算法终止。

性质 2.3.1. 算法会终止当且仅当 x 是有理数。

证明. 如果算法终止了，x 一定是有理数，因为此时连分数表示只有有限项，可以直接计算出 x 的值。

如果 x 是有理数，不妨假设 $x = \dfrac{P}{Q}$，其中 $Q > P$，$\gcd(P, Q) = 1$。则不难发现，从 x_0 到 x_1 的过程，就是令 $Q \to P'$，$P \bmod Q \to Q'$，$\dfrac{P'}{Q'} \to x_1$。$(P, Q) \to (P', Q')$ 的过程与计算最大公约数的欧几里得算法完全一致。由于欧几里得算法只会进行有限步，连分数表示自然也只有有限项，故算法会终止。 □

$\sqrt{n}$ 连分数表示的性质 考察 $\sqrt{n}$ 的连分数表示（不妨假设 n 不是完全平方数）$[a_0; a_1, a_2, \dots]$。

将 x_i 写成 $\dfrac{\sqrt{n} + P_i}{Q_i}$ 的形式（$P_0 = 0, Q_0 = 1, Q_i \neq 0$）。由此可得，$a_i = \lfloor \dfrac{\sqrt{n} + P_i}{Q_i} \rfloor$，$x_{i+1} = \dfrac{1}{x_i - a_i} = \dfrac{1}{\frac{\sqrt{n}+P_i}{Q_i} - a_i} = \dfrac{1}{\frac{\sqrt{n}+P_i - a_i Q_i}{Q_i}} = \dfrac{\sqrt{n} + (a_i Q_i - P_i)}{\frac{n-(P_i - a_i Q_i)^2}{Q_i}} = \dfrac{\sqrt{n} + (a_i Q_i - P_i)}{\frac{n - P_{i+1}^2}{Q_i}}$。

因此我们得到递推式：

$$P_{i+1} = a_i Q_i - P_i, Q_{i+1} = \frac{n - P_{i+1}^2}{Q_i} \ (i \geq 0) \tag{1}$$

性质 2.3.2. P_i, Q_i 总是整数。

证明. 用数学归纳法。当 $i = 0$ 时成立。当 $i = 1$ 时，$x_1 = \dfrac{1}{\sqrt{n} - a_0} = \dfrac{\sqrt{n} + a_0}{n - a_0^2}$，由此 $P_1 = a_0, Q_1 = n - a_0^2$，成立。

假设 $i = k-1, k$ 时 P_i, Q_i 均为整数，显然 P_{k+1} 也是整数。而 $n - P_{k+1}^2 = n - (a_k Q_k - P_k)^2 \equiv n - P_k^2 \pmod{Q_k}$。而由递推式，$n - P_k^2 = Q_k Q_{k-1}$，故 $n - P_k^2 \equiv 0 \pmod{Q_k}$，故 $n - P_{k+1}^2 \equiv 0 \pmod{Q_k}$，所以 Q_{k+1} 是整数。故 P_{k+1}, Q_{k+1} 都是整数。

综上，P_i, Q_i 总是整数。 □

性质 2.3.3. 对所有整数 $i \geq 0$ 都有 $0 < Q_i < 2\sqrt{n}, 0 \leq P_i < \sqrt{n}$。

证明. 由连分数的计算过程容易发现，当 $i \geq 1$ 时，总有 $x_i > 0$，$a_i \geq 1$。

下面用数学归纳法证明 $\forall i \geq 0$，$Q_i > 0, 0 \leq P_i < \sqrt{n}$。当 $i = 0$ 时成立。

假设 $i = k$ 时成立，分类讨论：

- $Q_k > \sqrt{n}$，注意 $a_k \geq 1$，因此 $P_{k+1} \geq Q_k - P_k > \sqrt{n} - P_k > 0$。
- $Q_k < \sqrt{n}$，则 $a_k = \lfloor \dfrac{\sqrt{n} + P_k}{Q_k} \rfloor \geq \lfloor \dfrac{Q_k + P_k}{Q_k} \rfloor = 1 + \lfloor \dfrac{P_k}{Q_k} \rfloor$。所以 $a_k Q_k > P_k$，因此 $P_{k+1} > 0$。

由于 $x_{k+1} > 0$，$P_{k+1} > 0$，所以 $Q_{k+1} > 0$。这意味着 $n - P_{k+1}^2 > 0$，也即 $P_{k+1} < \sqrt{n}$。

综上，总有 $Q_i > 0, 0 \leq P_i < \sqrt{n}$。

再结合 $P_{i+1} = a_i Q_i - P_i$，也即 $Q_i = \dfrac{P_i + P_{i+1}}{a_i} \leq P_i + P_{i+1} < 2\sqrt{n}$，就有 $0 < Q_i < 2\sqrt{n}$。 □

如果无理数 x 的连分数表示 $[a_0; a_1, a_2, \dots]$ 满足，存在正整数 len, s，使得对于任意正整数 $i \geq s$，都有 $a_i = a_{i-len}$，则称 x 的连分数表示有循环节。上述结论也说明，当 x 可以写成 $\sqrt{n}$ 的形式（其中 n 是非完全平方数的正整数）时，x 的连分数表示一定有循环节。实际上可以证明，无理数 x 的连分数表示存在循环节，当且仅当存在一元二次方程 $ax^2 + bx + c = 0$ 使得其存在一个实根为 x。证明与本文无关，故不展开。[4]

下面当提到 $\sqrt{n}$ 的连分数表示时，我们会沿用 P_i, Q_i 的记号。

连分数收敛子及其性质 设大于 1 的无理数 x 的的连分数表示为 $[a_0; a_1, a_2, \dots]$。设 $\{A_k\}, \{B_k\}$ 为两个正整数数列，满足 $[a_0; a_1, a_2, \dots, a_k] = \dfrac{A_k}{B_k}$。称 $\dfrac{A_k}{B_k}$ 为该连分数的第 k 个收敛子。

性质 2.3.4. 对于 $i \geq 2$，

$$A_i = a_i A_{i-1} + A_{i-2}, B_i = a_i B_{i-1} + B_{i-2} \tag{2}$$

证明. 用数学归纳法。对 $i = 2$ 容易直接验证成立。假设 $i = k$ 成立，考虑 $i = k + 1$ 时。

注意到，如果把在连分数表示的第 $0 \sim k$ 项里，令 $a_k + \dfrac{1}{a_{k+1}} \to a_k$，则最终算出来的值就会是 $\dfrac{A_{k+1}}{B_{k+1}}$。所以，

$$\frac{A_k}{B_k} = \frac{A_{k-1}a_k + A_{k-2}}{B_{k-1}a_k + B_{k-2}} \to \frac{A_{k+1}}{B_{k+1}} = \frac{A_{k-1}(a_k + \frac{1}{a_{k+1}}) + A_{k-2}}{B_{k-1}(a_k + \frac{1}{a_{k+1}}) + B_{k-2}} = \frac{A_k a_{k+1} + A_{k-1}}{B_k a_{k+1} + B_{k-1}}$$

因此上述递推式对所有 i 都成立。 □

上述过程求出的 A_k, B_k 是否可能不互质？事实上有下述性质：

性质 2.3.5. 对任意 $i \geq 1$，都有 $A_{i-1}B_i - A_i B_{i-1} = (-1)^i$。

该性质的证明只需使用数学归纳法后直接代入性质 4 即得，这里就不赘述了。结合 Bézout's identity [5] 即得 $\gcd(A_i, B_i) = 1$ 对 $i \geq 0$ 总成立。

性质 2.3.6. 对任意 $i \geq 2$，$x = \dfrac{A_{i-2} + A_{i-1}x_i}{B_{i-2} + B_{i-1}x_i}$ 总成立。

证明. 注意到，如果把在连分数表示的第 $0 \sim k$ 项里，令 $x_k \to a_k$，最终的值就会是 x。再结合性质 4，即证毕。 □

3 Square Form Factorization (SQUFOF)

Square Form Factorization，简称 SQUFOF，由 Shanks 提出。[13]

3.1 SQUFOF 算法梗概

有了上述准备，我们可以开始介绍 SQUFOF 的算法流程了。它基于下述性质：

性质 3.1.1. 对任意 $i \geq 1$，都有 $A_{i-1}^2 - nB_{i-1}^2 = (-1)^i Q_i$。

证明. 由性质 6：$\sqrt{n} = \dfrac{A_{i-2} + A_{i-1}x_i}{B_{i-2} + B_{i-1}x_i}$。代入 $x_i = \dfrac{\sqrt{n} + P_i}{Q_i}$ 得

$$\sqrt{n} = \frac{Q_iA_{i-2} + P_iA_{i-1} + \sqrt{n}A_{i-1}}{Q_iB_{i-2} + P_iB_{i-1} + \sqrt{n}B_{i-1}} \to \sqrt{n}(Q_iB_{i-2} + P_iB_{i-1} - A_{i-1}) = Q_iA_{i-2} + P_iA_{i-1} - nB_{i-1}$$

由于 $\sqrt{n}$ 是无理数，P, Q, A, B 均为整数数列，所以 $Q_iB_{i-2} + P_iB_{i-1} - A_{i-1} = 0, Q_iA_{i-2} + P_iA_{i-1} - nB_{i-1} = 0$。

前式乘以 A_{i-1}，后式乘以 B_{i-1}，相减得 $Q_i(B_{i-2}A_{i-1} - A_{i-2}B_{i-1}) = A_{i-1}^2 - nB_{i-1}^2$。代入性质 5 即得到待证式。 □

因此，$A_{i-1}^2 \equiv (-1)^i Q_i \pmod{n}$。而 SQUFOF 算法的思想即：找到某个偶数 i，使得 Q_i 为完全平方数 t^2，这样就有 $A_{i-1}^2 - t^2 \equiv 0 \pmod{n}$，则很大概率 $\gcd(n, A_{i-1} \pm t)$ 是 n 的一个非平凡因子。

据此可以得出下面的算法流程：

算法流程（一）

- 初始化 $P_0, Q_0, A_0, a_0, P_1, Q_1, A_1, a_1$ 的值。
- 根据递推式 (1) (2) 递推求出 P, Q, A, a 的每一项，直到在某个偶数位置 i，Q_i 是完全平方数。
- 返回 $\gcd(n, A_{i-1} \pm \sqrt{Q_i})$。

有概率上述过程返回的仍是平凡因子，这时可以选取一个较小的正整数 k 作为 multiplier，对 $n' = kn$ 重复上述算法。

3.2 优化

常数优化

- n 不太大时，可以用 Barrete 模乘或类似方式优化 A_i 的计算。注意：无论事先选取的 multiplier 是什么，A_i 都只对 n 取模即可（而不是 kn）。
- 判断 x 是否是平方数：选取一个值 M，先判断 x 模 M 的余数是不是二次剩余，若是再调用平方根函数。
- 过程中，P, Q, a 的计算共用到了两次除法（Q 一次，a 一次），但实际上可以省略 Q 这一次：$Q_iQ_{i-1} = n - P_i^2, Q_iQ_{i+1} = n - P_{i+1}^2$，两式相减得 $Q_i(Q_{i+1} - Q_{i-1}) = P_i^2 - P_{i+1}^2 = (P_i + P_{i+1})(P_i - P_{i+1}) = a_iQ_i(P_i - P_{i+1})$。也即，

$$Q_{i+1} = Q_{i-1} + a_i(P_i - P_{i+1}) \tag{3}$$

  这样，计算 Q_{i+1} 就不需要除法了。
- 所有变量都可以滚动计算，不需要开数组。

避免计算 A_k 的优化

受到计算机字长的限制，当 n^2 超过计算机能一次性处理的范围但 n 未超过时，P, Q, a 的计算仍然可以照常进行，但 A 就不便于计算了。事实上，存在能够避免直接计算 A_k 的方法。下面我们先给出算法流程再证明正确性。

算法流程（二）

- 初始化 $P_0, Q_0, a_0, P_1, Q_1, a_1$ 的值。
- 根据递推式 (1)(3) 递推求出 P, Q, a 的每一项，直到在某个偶数位置 k，Q_k 是完全平方数 w^2。
- 从 $\dfrac{\sqrt{n} - P_k}{w}$ 开始，按照与上面相同的方法计算 P', Q', a'。具体地，令 $P'_0 = -P_k$，$Q'_0 = w$，并用与递推式 (1)(3) 一样的方法递推 P', Q'，直到某处 $P'_i = P'_{i+1}$。
- 返回 $\gcd(Q'_i, n)$。

为说明算法的正确性，我们先证明以下性质：

性质 3.2.1. 若某个 $i \geq 0$ 满足 $P'_i = P'_{i+1}$，就有 $n = Q'_i(Q'_{i+1} + \dfrac{{a'_i}^2 Q'_i}{4})$。

证明. 由算法流程，a', P', Q' 也服从递推式 (1)(3)。由 $P'_i + P'_{i+1} = a'_iQ'_i$，得 $P'_{i+1} = \dfrac{a'_iQ'_i}{2}$。再代入 $n = {P'_{i+1}}^2 + Q'_iQ'_{i+1}$，化简即得上式。 □

性质 8 告诉我们，最后一步中返回的 $\gcd(Q'_i, n)$ 的确很有可能就是 n 的因数。之所以不直接返回 Q'_i，是因为 $\frac{a_i'^2 Q'_i}{4}$ 可能不是整数。

性质 3.2.2. 递推 P', Q' 的过程中，对任意 $i \geq 1$ 都有 $0 < Q'_i < 2\sqrt{n}, 0 \leq P'_i < \sqrt{n}$。

证明. 由算法流程，$a'_0 = \lfloor \frac{\sqrt{n} - P_k}{w} \rfloor$。$P'_1 = a'_0 \cdot w + P_k \leq \sqrt{n} - P_k + P_k = \sqrt{n}$，又因为 $\sqrt{n}$ 不是整数，所以 $P'_1 < \sqrt{n}$，而显然 $P'_1 \geq 0$。由 $Q'_1 = \frac{n - P_1'^2}{Q'_0}$ 也可得到 $Q'_1 > 0$。这样，套用性质 3 证明中的归纳，即可证明结论。 □

这样我们知道第二次递推过程中 P', Q' 的大小仍为 $\sqrt{n}$ 级别，因此两次递推在实现上完全没有区别。

通过性质 8, 9，我们已经能看出算法的正确性，接下来考虑第二次递推的时间复杂度。

性质 3.2.3. 第二次递推的下标数量约为第一次递推下标数量的一半。

该性质的证明（参见 [13]）涉及到二元二次型 (Binary Quadratic Forms) 的性质，由于作者学识浅薄，这里略去。由性质 10，分析时间复杂度时就可以只分析第一次递推的复杂度。

[13] 中还指出了可以维护一个列表以在不计算 A 的前提下判断某个完全平方数 Q 是否仅会带来平凡因子，这里不具体介绍。

3.3 复杂度分析与运行效果

注意到 $Q_i < 2\sqrt{n}$，而 $2\sqrt{n}$ 以内的完全平方数约占 $\frac{1}{O(\sqrt[4]{n})}$，因此生成 $O(\sqrt[4]{n})$ 项后很可能就能找到一个完全平方数 [6]（注意这依赖于 Q_i 几乎是一列随机整数的假设）。实践中，SQUFOF 约能在 1 秒内分解 10^{30} 以内的随机整数（随机方式为随机两个 10^{15} 左右的质数并相乘得到 n），可参考笔者在此处[3] 的提交。

4 Continued Fraction Method (CFRAC)

CFRAC 是另一种基于 $\sqrt{n}$ 的连分数表示的质因数分解算法，由 Morrison 和 Brillhart 提出。[14]

4.1 算法思想

与 SQUFOF 一样，它也基于性质七；然而，SQUFOF 主要思想是“等待”一个完全平方 Q_i 的出现，而 CFRAC 尝试从已有的 Q_i 构造出一个完全平方数。具体地，若下标集合 $U =$

[3] https://loj.ac/p/6466

$\{i_1, i_2, \ldots, i_k\}$ 满足 $S = \prod_{i\in U}(-1)^i Q_i$ 是完全平方数，则取 $x = \sqrt{S} \bmod n$，$y = \prod_{i\in U} A_{i-1} \bmod n$，就有 $x^2 - y^2 \equiv 0 \pmod n$。这样，$\gcd(n, x \pm y)$ 很可能就是一个非平凡因子。

如何找到一个 Q_i 的子集乘积是完全平方数？如果可以分解 Q_i 的质因数，问题就比较清楚了：将分解质因数的结果看成一个无限维的向量，向量的第 j 位是在分解结果中，第 j 个质数的幂次模二的结果，只需要找到一个在模二意义下和为零向量的向量子集，就找到了一个合法子集。

称一个数是 B-smooth 的，当且仅当其所有质因数都不超过 B。CFRAC 的思想即，设置阈值 B，只用 $\le B$ 的质数去试除 Q_i，只考虑能够在这个过程中就完全分解的 Q_i（也即只考虑 B-smooth 的 Q_i），并用高斯消元求解和为零向量的向量子集。注意 Q_i 有时会带有额外的 -1 系数，可以将 -1 也视为一个质数作为高斯消元的一部分。一个重要的优化是，由性质 7，$A_{i-1}^2 - nB_{i-1}^2 = (-1)^i Q_i$，若 Q_i 有质因子 p，就有 $A_{i-1}^2 - nB_{i-1}^2 \equiv 0 \pmod p$。因为 $\gcd(A_{i-1}, B_{i-1}) = 1$，所以不可能 $B_{i-1} \equiv 0 \pmod p$，因此可以在等式两边同除 B_{i-1}^2 即得 $(\frac{A_{i-1}}{B_{i-1}})^2 \equiv n \pmod p$，这说明 n 是模 p 意义下的二次剩余。因此可以先用快速幂计算勒让德符号 (n/p) 判断 n 是否是模 p 意义下的二次剩余，提前排除约一半无用质数。

CFRAC 还存在一个效果较好优化：设执行完毕试除后，剩余的数为 p。若存在两个数剩余的数相同，均为 p，不妨设已知的两个同余式为 $a_1^2 \equiv p \cdot q_1 \pmod n$，$a_2^2 \equiv p \cdot q_2 \pmod n$（$q_1, q_2$ 都是已经确定质因数分解的 B-smooth 数），结合两式可得 $(\frac{a_1 a_2}{p})^2 \equiv q_1 q_2 \pmod n$，这样我们又获得了一条可以直接用于高斯消元的等式。这样可以不小地提高 Q_i 的利用率，进而优化效率。由于存储剩余数 p 对应的 Q 及质因数分解也需要不小的时间，所以可以预先设置阈值 $B' \approx B^2$，仅当 $p < B'$ 时才执行上述优化。

4.2 复杂度分析与运行效果

实践上，由于同一个数质因子个数不多，所以高斯消元的矩阵非常稀疏，而且可以压位，几乎不会成为耗时瓶颈，程序大部分时间都消耗在试除上 [12]。这里我们分析未加入最后一个优化的 CFRAC 的试除部分复杂度（注意加入最后一个优化并不等价于直接将 B 设为 B'，所以不能直接套用下述分析！）。

这里仍然需要假设 $\sqrt{n}$ 的连分数表示中求出的 Q_i 是在 $(0, 2\sqrt{n})$ 之间的一列随机整数，设 B-smooth 的整数在 $(0, 2\sqrt{n})$ 中的占比为 $\frac{1}{p(B)}$，则粗略估计算法越需要执行 $p(B)\cdot\pi(B)^2$ 次试除以获得一个线性相关向量子集（期望 $p(B)$ 次找到一个 B-smooth 的整数；需要大约 $\pi(B)$ 个这样的整数才能找到一个线性相关子集；每个整数执行 $\pi(B)$ 次试除）。我们即需要最小化 $p(B)\cdot\pi(B)^2$。

在上述假设下，我们粗略估计 CFRAC 的复杂度：首先不妨忽略乘积中的 log 因子，这样，复杂度约为 $p(B)\cdot B^2$。根据 [7]，x 以内 $x^{1/\alpha}$-smooth 数的占比的一个粗糙估计为 $\alpha^{-\alpha}$。因此可以估计复杂度约为 $O((\frac{\log n}{2\log B})^{\frac{\log n}{2\log B}}\cdot B^2)$。取对数得到 $2\log B + \frac{\log n}{2\log B}(\log\log n - \log(2\log B))$。

可以发现整个式子形如 $A(n)\dfrac{\log n}{2\log B}+B(n)\cdot 2\log B$，$A(n),B(n)$ 的级别都远小于 $\log n$，因此粗略估计得 $2\log B$ 大体约为 $\sqrt{\log n}$，所以 $\log\log n-\log(2\log B)\approx\dfrac{1}{2}\log\log n$，所以原式约等于 $2\log B+\dfrac{\log n}{4\log B}\log\log n$，最小值为 $\sqrt{2\log n\log\log n}$。因此时间复杂度可以估计为 $O(\exp((1+o(1))\sqrt{2\log n\log\log n}))$，在 B 为 $\exp(\sqrt{\log n\log\log n/(2\sqrt{2})})$ 级别时取到。因此，CFRAC 是一种亚指数的质因数分解算法。

该算法实现非常简洁，运行速度也较快，实现可参考笔者在此处[4] 的提交。CFRAC 可以在几秒内分解 `__int128` 范围内的整数。

5 The Quadratic Sieve (QS)

二次筛法（The Quadratic Sieve，简称 QS）是一种质因数分解算法，由 Carl Pomerance 发明。[12]

5.1 算法思想

事实上，如果仅仅只需要快速生成很多 $x^2\equiv y\pmod n$ 形式的同余式，最简便的方法就是从 $\lfloor\sqrt{n}\rfloor+1$ 开始从小到大枚举 x，令 $y=x^2\bmod n$。而且，如果用该过程替换 CFRAC 中连分数计算过程，可以发现指数的级别是相同的！这被称为 Kraitchik's method[12]。然而，这种算法在效率上比 CFRAC 差了不少，因为需要判断是否 B-smooth 的数的大小大约是 $C\times\sqrt{n}$ 级别（其中 C 是需要判断的数的个数，约为 $B\cdot p(B)$），比 CFRAC 的 $2\sqrt{n}$ 大了不少倍，这会导致 smooth 的数相对稀疏很多。

然而，上述方法还有优化的余地。注意有不少时间都浪费在了对无用的数执行试除法上，如果能直接算出哪些数是 B-smooth 的，时间复杂度中，立马可以少乘以一个 $\pi(B)$。而 Kraitchik's method 生成的 x^2-n 形式的数具有特别的形式，这为我们快速判断哪些数是 B-smooth 的提供了便利。

注意到，$p|x^2-n$ 等价于 $x^2\equiv n\pmod p$，用计算二次剩余的算法（如 Cipolla 算法）可以快速求出 x 在模 p 意义下的 $O(1)$ 个可能值，我们如果只用 p 去分解模 p 意义下合法的 y，根据埃氏筛 [8] 的复杂度分析，该过程的时间复杂度为 $O(C\log\log C)$。

直接用上述筛法分解 x^2-n，我们就得到了一种分解质因数部分时间复杂度为 $O(C\log\log C)$ 的算法。不过，直接这样实现的实际运行效果并不好，因为待分解的数的大小随着它的下标是线性增长的，当待分解的数越来越多、越来越大时，B-smooth 的比例会越来越少。所以，实践中通常会同时取多个二次函数 $f(x)$，每个二次函数仅计算相对少量的 $f(x)$ 值，以

[4] http://119.27.163.117/problem/226

此控制 $f(x)$ 的范围 [10]。具体地，取 $f(x) = (Ax + B)^2 - n$，且整数 C 满足 $B^2 - n = AC$，就有 $f(x) = A(Ax^2 + 2Bx + C)$。若 A 是完全平方数 p^2，就有 $(Ax^2 + 2Bx + C) \equiv (\frac{Ax + B}{p})^2 \pmod{n}$。

这样，我们得到下列新的算法流程：多次选取 $A = p^2$。取 B 使得 $b^2 \equiv n \pmod{A}$。取 $C = \frac{B^2 - n}{A}$。这样，对于 x 而言，需要被筛的数值即 $Ax^2 + 2Bx + C$。通过适当选取 A, B, C，可以使得 $Ax^2 + 2Bx + C$ 的绝对值（若为负数，将 -1 视为一个质数即可）较小，从而增加 smooth 数的占比。

与 CFRAC 一样地，若存在两个数在分解后剩余的数相同，同样可以将其结合得到新的可以用于高斯消元的关系。

5.2 复杂度分析与运行效果

使用与 CFRAC 相同的复杂度分析方法，容易得到分解质因数部分最终的时间复杂度为 $O(\exp((1 + o(1))\sqrt{\log n \log\log n}))$，在阈值 B 取 $\exp(\sqrt{\log n \log\log n}/2)$ 级别时取到。在 B 较大时，可以将高斯消元换为针对稀疏矩阵的其它算法。

已有的一些 QS 实现，如 [9] 是一份 C 语言的实现，可以在几秒内分解 2^{200} 左右的整数。

6 总结

本文简单介绍了几种在 OI 界认知度较低的质因数分解算法，也展现了目前对分解质因数问题的一些思路：通过分析连分数表示的性质、对 smooth numbers 分解质因数后用消元寻找同余式、将数论中的“筛法”一般化等方式，得到了几种分解算法。希望本文中的介绍能起到抛砖引玉的作用，为 OI 中的数论问题带来一些启发。

致谢

感谢中国计算机学会提供学习交流的平台。

感谢父母的养育之恩。

感谢黄新军老师的关心和指导。

感谢郭雨豪同学为本文验稿。

参考文献

[1] Wikipedia, Prime Number Theorem, https://en.wikipedia.org/wiki/Prime_number_theorem

[2] Wikipedia, Barrett Reduction, https://en.wikipedia.org/wiki/Barrett_reduction

[3] Wikipedia, Miller-Rabin primality test, https://en.wikipedia.org/wiki/Miller-Rabin_primality_test

[4] OI Wiki, 连分数, https://oi-wiki.org/math/number-theory/continued-fraction/

[5] Wikipedia, Bézout's identity, https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity

[6] Wikipedia, Shanks's square forms factorization, https://en.wikipedia.org/wiki/Shanks%27s_square_forms_factorization

[7] Wikipedia, Dickman function, https://en.wikipedia.org/wiki/Dickman_function

[8] OI Wiki, 筛法, https://oi-wiki.org/math/number-theory/sieve/

[9] michel-leonard, C Factorization using Quadratic Sieve, https://github.com/michel-leonard/C-Quadratic-Sieve

[10] 钟子谦, 二次筛法 (Quadratic Sieve), https://zhuanlan.zhihu.com/p/106650020

[11] Hans Riesel. Prime Numbers and Computer Methods for Factorization. Springer Science + Business Media, LLC, 1994.

[12] Carl Pomerance. A Tale of Two Sieves. Notices of the AMS, Volume 43, Number 12, 1473-1485, 1996.

[13] Jason E. Gower; Samuel S. Wagstaff, Jr. Square Form Factorization. Mathematics of Computation, Volume 77, Number 261, 551-588, January 2008.

[14] Michael A. Morrison; John Brillhart. A Method of Factoring and the Factorization of F_7. Mathematics of Computation, Volume 29, Number 129, 183-205, January 1975.