陕西師範大學
SHAANXI NORMAL UNIVERSITY

整数分解报告

参赛题目：赛题二：整数分解

参赛院校：陕西师范大学

指导教师：李艳平

小组成员：蒋思阳 胡佳宇 吴姣姣

参赛日期：2018 年 6 月

目　录

大整数素因子分解报告

蒋思阳，胡佳宇，吴姣姣，李艳平(指导教师)
陕西师范大学，772922440@qq.com，15619263398

摘 要：整数分解是将一个整数 N 分解为素因子乘积。针对赛题二，本文主要涉及 8 种经典的大数分解算法和一个素性检测算法——Miller-Rabin 算法，将一个十进制位数为 432 位的整数 N 完全分解，共得到 14 个素因子(如下)。其中，利用试除法求得素因子 p_1 ，利用 Pollard rho 算法求得素因子 p_1 和 p_2，利用 Pollard p-1 算法求得素因子 p_2、 p_4 和 p_8，利用 Lenstra 椭圆曲线分解法求得素因子 p_3、 p_4、 p_5、 p_6、 p_7 和 p_8，利用初始化二次筛法和数域筛法求得剩余 6 个素因子，具体每个因子求解方法情况见文中图 2，并且 Dixon 随机平方法和二次筛法是作为算法理解的重要过渡。最终也将所得的素因子通过素性检测、计算乘积和个别算法结果对比来验证正确性，证明所得到的因子均是正确的。

p_1 = 439883
p_2 = 1234567891
p_3 = 1732792378957
p_4 = 832328713672817
p_5 = 952921534336737871
p_6 = 1309672216786572122317
p_7 = 1208925819614750508040051
p_8 = 170141183460469231731687303715884105727
p_9 = 27448314684429187698286359631001710l279
p_{10} = 274483146896724957848529135013585755443
p_{11} = 802874755424222799061978026497620694203777347803
p_{12} = 134160169824140540144895765720162546212l694828679
p_{13} = 52959220656257503334755619569082501981631902379203731956l7
p_{14} = 345764579608668445952537219644694709410707857553390847210l

1.问题提出与求解方法

1.1 大整数分解问题研究进展

算术基本定理指出，任何一个大于 1 的自然数 N，如果 N 不为素数，那么 N 可以唯一分解成有限个素数的乘积[1]。大整数的素因子分解问题是一个典型的单项问题且被认为是困难的。众所周知的公钥密码算法 RSA 的安全性正是基于大整数分解的困难性[2]。几百年来，研究者们不断寻找更快更高效的整数因子分解法，甚至为了激发更多的人研究大数分解问题，发起了 RSA 模数分解挑战赛，从而出现了很多高效的算法，也激励研究者们不断探索更快更好的大整数因子分解算法[3]。

目前，大数分解已经脱离了暴力试除的时代，已经有很多优秀的算法提出：Pollard rho 算法、Pollard p-1 算法、Lenstra 椭圆曲线分解法、Dixon 随机平方法、二次筛法以及数域筛法等分解算法。现有的十几种大数分解法也基于不同的理论知识，按照不同路线发展出不同的优化算法。按照目前的研究进展来看可以根据基于的理论知识主要分成如下图中的四类：

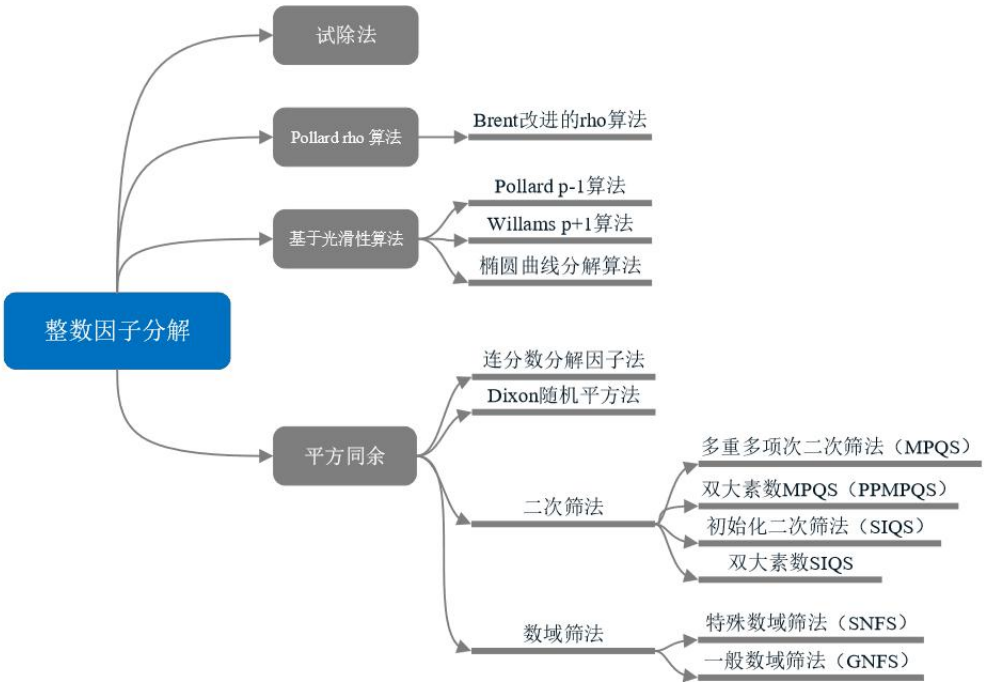

图 1　主流整数因子分解法的分类

1.2 求解思路

借鉴上述大整数分解法研究进展中的多种算法，本文针对本题附件的大整数 N 做了相应的分解尝试。本文按照循序渐进的解题方法，依次选择试除法、Pollard rho 算法、Pollard p-1 算法、Lenstra 椭圆曲线分解法、Dixon 随机平方法、二次筛法以及数域筛法来尝试完成题目，最终实现 N 的完全分解，得到 N 的所有素因子。在解题的过程中，对算法做了分析和尝试，并提出在分解过程中的问题。

表 1　算法分析与分解尝试

算法名称	时间复杂度	优点	缺点	分解尝试
试除法	$O(\sqrt{N})$	简单，适合分解拥有小素因子的整数	只适用于分解小规模整数	尝试用试除法分解时，分解出最小的因子，剩余分解效率太低，不可取
rho 算法	$O(N^{\frac{1}{4}}\log^2 N)$	相比试除法好	只适用于分解有小素因子的整数	利用 rho 算法分解出两个个小因子
$p-1$ 算法	$O(B\log B\log^2 N)$	p-1 具有小素因子的数容易被分解	通用性不好 实用性不强	利用 p-1 算法分解可以分解出 3 个因子；并且是理解椭圆曲线算法的基础
椭圆曲线法	$L_p[1,1/2,1/2,\sqrt{2}]$	结构简单，空间复杂度小，适用于中等规模的整数分解	通用性较差	椭圆曲线分解法是基于光滑性算法中最好的，利用其可分解出 N 的 6 个因子
二次筛法	$L_N[1,1/2,1/2,1]$	并行度高 通用性强	空间复杂度高 算法理解难度大	分解效率较高，利用初始化二次筛法将 3 个中间数分解出 N 的 6 个素因子
数域筛	$L_N[1/3,1/3,2/3,64/9]$	分解大规模整数最快的算法	算法复杂 空间复杂度高 通用性不够	算法难理解；利用一般数域筛法分解了两个中间数

注：定义 $L_q[\alpha_1,\alpha_2,\alpha_3,c]=O\left(\exp\left((c+o(1))^{\alpha_1}(\log q)^{\alpha_2}(\log\log q)^{\alpha_3}\right)\right)$

2.求解算法的描述、分析与尝试

本文分解整数 N 的过程中，从理论上和实现上主要涉及了 8 种因子分解算法和 1 种素性检测算法，以此实现了整数 N 的完全分解。这一部分主要阐述了算法的描述、对这些算法的理解以及针对题目对每种算法的尝试使用情况。

2.1 试除法

试除法(trial division)也称为穷举法，其基本思想非常简单，就是穷举搜索 N 所有可能的因子。当要分解的数 N 本身很小，或者 N 有较小的素因子时，试除法是不错的选择。但如果要分解的数 N 较大时，并且 N 没有小的因子，该方法就基本上无能为力了。试除法有如表 2 所示的五种境界，最常用的是境界 4 或 5。虽然本题所给整数 N 很大，但是在解决问题初期，作为实验，我们首先用试除法进行了尝试分解，并在二次筛法的实现中，分解较小的整数时运用了试除法。

表 2　试除法的 5 种境界

试除法	穷举搜索方式	时间复杂度
境界 1	尝试从 2 到 $N-1$ 的所有整数	$O(N)$
境界 2	尝试从 2 到 $N/2$ 的所有整数	$O(N/2)$
境界 3	尝试从 2 到 $N/2$ 的所有奇数	$O(N/4)$
境界 4	尝试从 2 到 $\sqrt{N}$ 的所有奇数	$O(\sqrt{N}/2)$
境界 5	尝试从 2 到 $\sqrt{N}$ 的所有素数	$O(\sqrt{N}/\ln N)$

2.2 Pollard rho 算法

Pollard rho 算法[4~6](见表 3)是一种基于随机数的方法，该算法实质上就是要找到伪随机序列中两个模 p 同余的数。算法的关键是这里的伪随机序列 $\{x_1, x_2, x_3, \cdots, x_i, \cdots\}$ 是通过多项式 $f(x) \in Z[x]$ 构造。虽然事先不知道 p，无法直接判断 $x_i \equiv x_j \pmod p$ 是否成立，但是可以通过计算 $d = \gcd(|x_i - x_j|, N)$，并判断 $1 < d < N$ 是否成立来间接知道这一点。事实上，该方法相比试除法好，但一般对有小素因子的整数更有效。本文在尝试用 rho 算法分解 N 时，分解出两个素因子。

表 3　Pollard rho 算法

Algorithm 1　Pollard rho algorithm for factoring (N, x_1)

external f
$x \leftarrow x_1$
$x' \leftarrow f(x) \bmod n$
$d \leftarrow \gcd(x - x', n)$
while $d = 1$
　　do { comment: in the ith iteration, $x = x_i$ and $x' = x_{2i}$
　　$x \leftarrow f(x) \bmod N$
　　$x' \leftarrow f(x') \bmod N$
　　$x' \leftarrow f(x') \bmod N$
　　$d \leftarrow \gcd(x - x', N)$
if $d = N$
　　then return ("failure")
　　else return (p)

2.3 基于光滑性算法

所谓基于光滑性的算法，表示这些算法的性能直接取决于待分解的数是否满足某些特殊的光滑性。例如，使得整数能分解为一些较小素数的乘积或者与有关的群的阶是否具有一定的光滑性等。本文的分解涉及此类方法中的 Pollard p-1 算法和椭圆曲线分解法。

2.3.1 Pollard p-1 算法

Pollard $p-1$ 算法[7~8]是利用费马小定理进行因子分解，可以找到合数满足如下条件的因子 p：$p-1$ 关于一个相对较小的界 B 是光滑的。该算法理论简单，且是关于 $\log N$ 的多项式时间算法。但是，B 的选择很大程度上决定了算法的成功概率，如果迅猛增加 B 的大小，比如说 $\sqrt{N}$，那么该算法成功的概率将是很高的，但是它并不比试除法快。针对本文的分解，除了利用该算法分解出 3 个因子外，还将此算法作为理解椭圆曲线法的准备，进而更好实现椭圆曲线分解法。

表 4　Pollard p-1 算法

Algorithm 2　Pollard $p-1$ algorithm for factoring (N, B)

$a \leftarrow 2$
for $j \leftarrow 2$ to B
　do $a \leftarrow a^j \bmod N$
$d \leftarrow \gcd(a-1, N)$
if $1 < d < n$
　then return (d)
　else return ("failure")

2.3.2 椭圆曲线分解法

椭圆曲线分解法[9,10](ECM) 由 H.W. Lenstra 提出，可看作 Pollard $p-1$ 算法的改进，两个算法成功分解的前提是它们的群的阶平滑，与 Pollard $p-1$ 算法不同的是，ECM 用随机选的椭圆曲线的加法群来取代 $p-1$ 算法中的乘法群 F_p。因此根据 Pollard $p-1$ 算法的理解，可以很快明白椭圆曲线算法的工作原理，并加以实现。事实上，ECM 是所有基于平滑性的分解算法中实际运行最快的。特别地，当 N 具有小素数因子时，ECM 运行更快。此外，ECM 还有一个显著的优点，即它占用的内存很少。本文借助优势显著的 ECM，共找到整数 N 的 6 个素因子。

表 5　椭圆曲线分解法

Algorithm 3　Lenstra's elliptic curves method for factoring

Step1：随机选择一条椭圆曲线 E：$y^2z = x^3 + axz^2 + bz^3$，$a, b \in Z_p$。

Step2：选择一个界 B，使得群 E 的阶$\#E$ 关于界 B 光滑，则有 $\#E \mid B!$。

Step3：随机选择椭圆曲线上一点 $P = (x, y, z)$，和 Pollard $p-1$ 中类似，循环后有 $P' = (x', y', z') \equiv B!P \bmod N$，根据费马小定理，$P'$ 等于加法零元 O（在椭圆曲线群 E 中的加法零元 $O = (0:1:0)$），此时 $z \equiv 0 \bmod p$，即 $p \mid z$，计算 $\gcd(z, N)$，若大于 1，得到 N 的一个非平凡因子。

Step4：若随机选的一条曲线未成功分解 N，根据 Hesse 定理，可选取多条曲线，提高$\#E$ 关于界 B 平滑的概率，从而提高找到 N 的因子的概率。

2.4 平方同余方法

基于平方同余的分解思想：得到平方同余式 $X^2 \equiv Y^2 \bmod N$ 两边的整数 X 和 Y（$X \neq Y \bmod N$），并通过计算 $\gcd(X \pm Y, N)$ 来得到非平凡因子。这是很多大整数分解算法共用的一个策略，只是找满足这样条件的 X 和 Y 的方式不同。本文分析并尝试了基于这一重要思想的 Dixon 随机平方法、二次筛法、初始化二次筛法和数域筛法，并利用初始化二次筛法和数域筛法分解出整数 N 的 6 个素因子。

基于平方同余方法的具体算法均略复杂，但都是基于 Kraitchik 提出的一般分解步骤：

(1)生成一系列同余式 $u_i \equiv v_i \bmod N$， $i = 1,2,\cdots$；

(2)对 u_i 和 v_i 进行分解；

(3)挑选子集 I，使得 $\prod_{i \in I} u_i$ 和 $\prod_{i \in I} v_i$ 为平方数，若令 $X = \left(\prod_{i \in I} u_i\right)^{1/2}$，$Y = \left(\prod_{i \in I} v_i\right)^{1/2}$，则 $X^2 \equiv Y^2 \bmod N$；

(4)计算 $\gcd(X \pm Y, N)$

本小结将对本文分解涉及到的平方同余思想算法按照分解的一般步骤进行描述。其中涉及到的基本概念有：

(1)因子基：令 $B = \{p_1, p_2, \cdots, p_b\}$，其中 $p_1, p_2, \cdots, p_b$ 是小于等于 m 的素数（B 中素数关于 m 平滑），称 B 为因子基。

(2)B-数：若一整数 u，$u^2 \bmod N$ 的素因子均在该因子基中，则称 u 为 B-数。

2.4.1Dixon 随机平方方法

Dixon 随机平方法[11~13]是通过一系列的随机数来找 B-数，进而寻找向量之间的线性相关来构造 $X^2 \equiv Y^2 \bmod N$。此算法在解题过程中是作为理解二次筛法的基础。如下是算法的具体描述：

表 6　Dixon 随机平方方法

Algorithm 4　Dixon's algorithm

Step1：初始化——设定因子基 *B*

令 L 为一个存储整数值在 $[1, N]$ 内的列表，$B = \{p_1, p_2, \cdots, p_b\}$ 是 b 个小于等于 m 的素数（B 中素数关于 m 光滑），且 A 和 U 均是空列表（A 表中值将作为 U 表的下标）

Step2：寻找 *B*-数

1.若 L 表为空，则退出；否则令 u 是 L 中第一个数，从 L 中删去 u，并执行 Step2-2

2.令 v 为 $u^2 (\bmod N)$ 的最小绝对剩余，分解 $v = v' \prod_i p_i^{\alpha_i}$，其中 v' 为在 B 中不能分解的数。若 $v' = 1$，则进行 Step3；否则返回执行 Step2-1（随机找 u，直到找到足够多 u 使得 $u^2 \equiv v \bmod N$ 在 B 中完全分解）

Step3：利用向量线性相关构造 $X^2 \equiv Y^2 \bmod N$

1.令 $\boldsymbol{a} \leftarrow (\alpha_1, \alpha_2, \cdots, \alpha_b)$，将向量 $\boldsymbol{a}$ 存入列表 A，$u = u_{\boldsymbol{a}}$ 存入 U 表；如果表 A 中至多有 b 个元素，则返回执行 Step2，继续找 B-数；否则执行 Step3-2

2.找到表 A 中一个向量 $\boldsymbol{c}$ 是在 mod 2 下与 A 中向量线性相关的，从表 A 中删去 $\boldsymbol{c}$，并从表 U 中删去 $u_{\boldsymbol{c}}$，计算使 $\boldsymbol{c} = \sum_{t \in A} f_t \boldsymbol{t} (\bmod 2)$ 成立的系数 f_t（=0 或 1），令 $\boldsymbol{d} = (d_1, \cdots, d_b) \leftarrow \left(\boldsymbol{c} + \sum f_t \boldsymbol{t}\right)/2$，并执行 Step4.

Step4：求 *N* 的非平凡因子

令 $X \leftarrow u_c \prod_t u_t^{f_t}$，$Y \leftarrow \prod_i p_i^{d_i}$，则有 $X^2 \equiv \prod_i p_i^{2d_i} \equiv Y^2 \bmod N$.若 $X \equiv \pm Y \bmod N$，则返回 Step2；反之，计算 $\gcd(X \pm Y, N)$ 即为 N 的一个因子

2.4.2 二次筛法与初始化二次筛法

二次筛法[14,15]和初始化二次筛法[16]是本文分解的主要解题方法。主要想法和Dixon 算法类似，改进在因子基上寻找平滑整数(Step2)和 Gauss 消去求向量相关性(Step3)这两个主要步骤。二次筛法最关键的一步是用因子基中的素数对某个区间(筛区间)内的二次函数值进行筛选，从而找到 *B*-数，算法的具体描述如下：

表 7　二次筛法

Algorithm 5　Quadratic sieve

Step1：构造因子基 *B*

取定 m，构造因子基 $B=\{p_0,p_1,\cdots,p_b\}$，其中 $p_0=-1$，$p_1,p_2,\cdots p_b$ 是不超过 m 的所有满足条件 $\left(\frac{N}{p_j}\right)=1$ 的素数.

Step2：通过多项式 $Q(x)=\left(x+\lfloor\sqrt{N}\rfloor\right)^2-N$ 的解寻找 *B*-数

候选的 *B*-数 $u_i=\lfloor\sqrt{N}\rfloor+i$，$i\in[-M,M]$，则 $v_i \overset{def}{\equiv} u_i^2 \bmod N=\left(\lfloor\sqrt{N}\rfloor+i\right)^2 \bmod N=\left(\lfloor\sqrt{N}\rfloor+i\right)^2-N$

对因子基 B 中每个素数 $p_j\in B$ 的幂次 $p_j^h\le m$ $\left(j=1,2,\cdots,b;h=1,2,\cdots\lfloor\log m/\log p_j\rfloor\right)$ 求解同余方程 $\left(\lfloor\sqrt{N}\rfloor+x\right)^2\equiv N \bmod p_j^h$，有解为 $x_1^{(p_j^h)}$，$x_2^{(p_j^h)}$（$p=2$ 时另行处理）

计算每个 $v_i=\left(\lfloor\sqrt{N}\rfloor+i\right)^2-N$ 的对数值，即 $\log v_i$，$i\in[-M,M]$

将这些对数值存入对应下表为 i 的数组中

对同余方程的所有解 $x_1^{(p_j^h)}$，$x_2^{(p_j^h)}$，将数组下标为 $x_1^{(p_j^h)}$，$x_2^{(p_j^h)}$ 对应位置的对数值均减去 $\log p_j^h$

扫描最终的数组，如果某一位置 k 存的对数值经过减去之后剩余值接近 0，则下标 k 对应的 v_k 在因子基 B 中完全分解，将完全分解的 v_k 按顺序记为 v_k $(k=1,2,\cdots)$

此时对应的 u_k $(k=1,2,\cdots)$ 均为 *B*-数，即 $u_k^2 \bmod N$ 在因子基 B 中完全分解

Step3：利用 Gauss 消去法构造 $X^2\equiv Y^2 \bmod N$

对完全分解的 v_k 利用试除法进行素因子分解 $u_k^2\equiv v_k\equiv p_0^{\alpha_{0k}}\times p_1^{\alpha_{1k}}\times\cdots\times p_b^{\alpha_{bk}} \bmod N$（若 Step2 中得到的 *B*-数少于 *b*+2 个，可扩大筛区间，直到找到 *b*+2 个为止）

对每一个 k，考虑向量 $\boldsymbol{a}_k=(\alpha_{0k},\alpha_{1k},\cdots,\alpha_{bk}) \bmod 2\in Z_2^{b+1}$

当有足够多的向量 $\boldsymbol{a}_k$，利用 Gauss 消去法，找出这些向量(在 Z_2 上)的线性相关的向量组，即 $\boldsymbol{a}_{k_1}+\boldsymbol{a}_{k_2}+\cdots+\boldsymbol{a}_{k_t}=\boldsymbol{0}$

计算 $Y^2\equiv v_{k_1}\cdots\cdots v_{k_t}$，$X\equiv u_{k_1}\cdots\cdots u_{k_t}$，则有 $X^2\equiv Y^2 \bmod N$

Step4：求 N 的非平凡因子

利用 Euclidean 算法计算 $\gcd(X\pm Y,N)$，得到 N 的一个分解

在基本二次筛法中只使用一个二次多项式，有一种较好的改进方法称为多重多项式二次筛法(multiple polynomial quadratic sieve, MPQS)。MPQS 方法顾名思义，使用了多个多项式，这些多项式形如

$$Q(x)=ax^2+2bx+c，\text{其中 } N\mid b^2-ac$$

这样

$$aQ(x)=a^2x^2+2abx+ac=(ax+b)^2-(b^2-ac)\equiv(ax+b)^2 \bmod N$$

而初始化二次筛法(self initializing quadratic sieve, SIQS)是在 MPQS 的基础上，提出了一种更快改变多项式的方法，从而在更小筛区间内，使得平方剩余更小，更容易找到一系列平滑值，其初始化阶段的改进算法如下：

表 8　初始化二次筛法的初始化步骤

Algorithm 6　SIQS initialization stages

Initialization stage for first polynomial:

找因子基中乘积约为 $\sqrt{2N}/M$ 的素数 $q_1,\cdots,q_s$，令 $a=\prod_{l=1}^{s}q_l$.

For l=1 to s

Compute $\gamma=tmem_p\times(a/q_l)^{-1}\bmod q_l$.

If $\gamma>q_l/2$ then replace γ with $q_l-\gamma$.

Let $B_l=a/q_l\times\gamma$.

For each prime p in the factor base that does not divide a

Compute $ainv_p=a^{-1}\bmod p$

For j=1 to s

Compute $Bainv2_{j,p}=2\times B_j\times ainv_p\bmod p$

Let $b=B_1+\cdots+B_s$ and $g_{a,b}(x)=(ax+b)^2-N$.

For each prime p in the factor base that does not divide a

Compute $soln1_p=ainv\times(tmem_p-b)\bmod p$ and $soln2_p=ainv\times(-tmem_p-b)\bmod p$

Initialization stage for the next $2^{s-1}-1$ polynomial:

Let v be the integer satisfying $2^v\|2i$. Let $b=b+2\times(-1)^{\lceil i/2^i\rceil}\times B_v$ and $g_{a,b}(x)=(ax+b)^2-N$.

For each prime p in the factor base that does not divide a

Compute $soln1_p=soln1_p+(-1)^{\lceil i/2^v\rceil}\times Bainv2_{v,p}\bmod p$

Compute $soln2_p=soln2_p+(-1)^{\lceil i/2^v\rceil}\times Bainv2_{v,p}\bmod p$

2.4.3 数域筛法

数域筛法[17](number field sieve, NFS)分为特殊数域筛法(SNFS)和一般数域筛法[18](GNFS)。GNFS 可以分解一般形式的整数，是对二次筛法进行了扩展，允许更高次的多项式。数域筛法被认为是当前最好的因式分解算法，分解大整数有极高的效率。数域筛法主要思想分为四个步骤：

(1)构造代数数域，选取两个互质的且模 N 时有相同根的多项式 f_1、f_2；

(2)筛选出足够多的 (a,b) 对，满足 $a+bm$ 和 $a+b\alpha$ 在相应因子基下完全分解；

(3)与二次筛法一样，通过矩阵求解线性相关得到平方数；

(4)求平方数(涉及求解代数域上的代数平方根)，从而得到 N 的非平凡因子。

2.5 素性检测

在整个分解过程中，一直需要对算法实现的分解结果进行素性检测，因此在解题过程中本文采用一个多项式时间的偏是的 Monte Carlo 算法——Miller-Rabin 素性检测算法，算法具体描述如下：

表 9　Miller-Rabin 素性检测算法

Algorithm 7　Miller-Rabin (N)

把 $N-1$ 写成 $N-1=2^k m$，其中 m 是一个奇数，随机选取整数 a，使得 $1\le a\le N-1$

$b\leftarrow a^m\bmod N$

if $b\equiv 1(\bmod N)$

then return ("N is prime")

For $i\leftarrow 0$ to $k-1$

do
- if $b\equiv -1(\bmod N)$
- then return("N is prime")
- else $b\leftarrow b^2\bmod N$

return("N is composite")

3.分解结果与检验

3.1 分解结果

本文将一个十进制位数为 432 位的整数 N 完全分解，共得到 14 个素因子，分解结果见表 10。在整个解题的过程中，涉及到的分解算法有试除法、Pollard rho 算法、Pollard p-1 算法、Lenstra 椭圆曲线分解法(ECM)、Dixon 随机平方法、二次筛法(QS)、初始化二次筛法(SIQS)以及数域筛法(NFS)，并使用了素性检测算法——Miller-Rabin 算法；其中 Dixon 随机平方法是作为理解二次筛法系列的基本算法，其余算法在实际分解过程中均使用到，图 2 已给出分解每个素因子的具体算法使用情况(注：图 2 中 N 表示题中要分解的大整数，p_* 是分解出 N 的某个素因子，N_* 是分解过程中剩余的十进制位数为*的中间合数)。

表 10　分解结果

序号	分解出 N 的素因子	十进制位数
p_1	439883	6
p_2	1234567891	10
p_3	1732792378957	13
p_4	832328713672817	15
p_5	952921534336737871	18
p_6	1309672216786572122317	22
p_7	1208925819614750508040051	25
p_8	1701411834604692317316873037158841057 27	39
p_9	2744831468442918769828635963100171012 79	39
p_{10}	2744831468967249578485291350135857554 43	39
p_{11}	8028747554242227990619780264976206942037773478 03	48
p_{12}	1341601698241405401448957657201625462121694828679	49
p_{13}	5295922065625750333475561956908250198163190237920373195647	58
p_{14}	3457645796086684459525372196446947094107078575533908472103	58

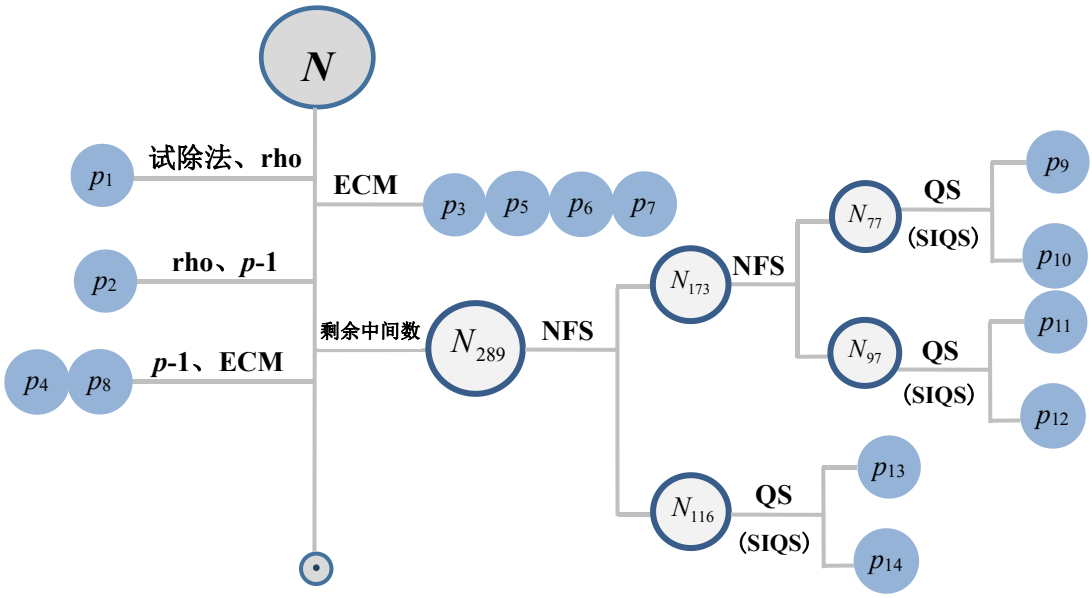

图 2　分解每个素因子的算法使用情况

3.2 分解过程

根据每种算法的理论背景和在实际分解时的情况，本文选取了不同参数进行尝试，表 11 列出在分解过程中每个因子产生时所使用的基本参数情况。

表 11　分解过程中参数情况

算法名称	所需参数	参数情况
试除法	尝试从 2 开始的素数	在一定时间内只找到 p_1
rho 算法	伪随机序列的构造函数 f 序列初始值 x_1	p_1: $f=x^2+2$　$x_1=1$ p_2: $f=x^2+1$　$x_1=1$
$p-1$ 算法	界 B	p_2: $B=1.5\times10^5$　p_4: $B=3.75\times10^6$　p_8: $B=1.5\times10^7$
椭圆曲线法	界 B 随机选用的曲线个数	p_3: $B=1.5\times10^5$　选用 10 条曲线 p_4: $B=5\times10^4$　选用 214 条曲线 p_5: $B=2.5\times10^5$　选用 4 条曲线 p_6: $B=2.5\times10^5$　选用 15 条曲线 p_7: $B=2.5\times10^5$　选用 2 条曲线 p_8: $B=2.5\times10^5$　选用 409 条曲线
二次筛法（SIQS）	因子基 $B=\{p_1,p_2,\cdots,p_b\}$，其中 $p_j\le$界 m，$\left(\frac{N}{p_j}\right)=1$	$N_{77}\to p_9\times p_{10}$　因子基最大素数为 908197 $N_{97}\to p_{11}\times p_{12}$　因子基最大素数为 2738423 $N_{116}\to p_{13}\times p_{14}$　因子基最大素数为 9100669

3.3 结果检验

将十进制位数为 432 位的整数 N 完分解成 14 个素因子后，通过下列三点简单证明，所得到的 14 个素因子是正确的：

(1) 利用 Miller-Rabin 素性检测算法，分别迭代 100 次，以高正确概率判别出 14 个因子均为素数，见图 3。

(2) 计算 14 个因子的乘积，与 N 完全相等，见图 4。

(3) 例如因子 p_1、p_2、p_4 和 p_8 都是用两种算法得出的结果，且均一致。

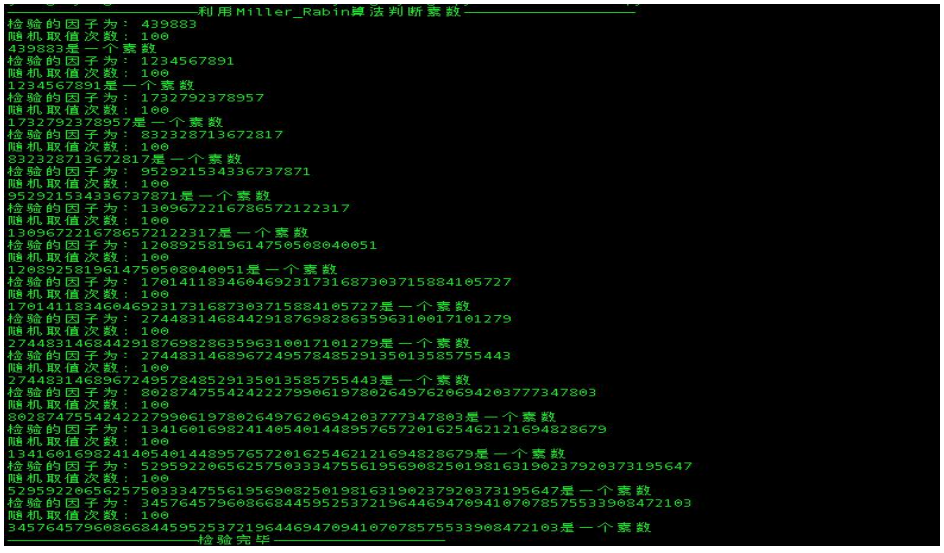

图 3　Miller-Rabin 算法检测所求因子是否为素数

图 4　14 个素因子乘积等于 *N*

4.小结

本文通过一些经典的大数分解算法将一个十进制位数为 432 位的整数 *N* 完全分解，共得到 14 个素因子。在整个解题的过程中，涉及到的分解算法有试除法、Pollard rho 算法、Pollard *p*-1 算法、Lenstra 椭圆曲线分解法（ECM）、Dixon 随机平方法、二次筛法（QS）、初始化二次筛法（SIQS）以及数域筛法（NFS），且使用了素性检测算法——Miller-Rabin 算法。为了将算法的工作原理理解的更透彻，在实际的解题过程中，通过学习 Pollard *p*-1 算法的理论原理去理解椭圆曲线算法的工作原理；同时还学习了连分数因子分解法和 Dixon 随机平方法，作为理解二次筛法及其一系列优化算法的基础。因此，整个解题是一个循序渐进，由浅入深的过程，而且较好地实现了整数 *N* 的完全分解。最终也将所得的素因子通过素性检测、计算乘积和个别算法结果对比来验证正确性，证明所得到的因子均是正确的。

大整数因子分解算法的研究一直在发展，更好更高效且简单易懂的算法是研究者们的向往。本小组循序渐进地深入大数分解这一研究课题，尽最大努力通过本题了解大数分解，并做出努力尝试，从零开始，学习算法，分解整数。希望未来能通过对算法的更深刻理解，提出进一步的改进和优化。

5.参考文献

[1] Henri Cohen. A course in computational algebraic number theory, volume 138 of Graduate texts in mathematics. Springer, 1993.

[2] 董青, 吴楠. 整数质因子分解算法新进展与传统密码学面临的挑战[J]. 计算机科学, 2008, 35(8): 17-20.

[3] 刘新星, 邹潇湘, 谭建龙. 大数因子分解综述[J]. 计算机应用研究, 2014, 31(11): 3201-3207.
[4] Douglas R. Stinson 著. 冯登国等译. 密码学原理与实践（第 3 版）[M]. 北京：电子工业出版社，2009.7.
[5] Pollard J M.A Monte Carlo method for factorization.BIT Numerical Mathematics,1975,15(3):331-334
[6] Brent R P. An Improved Monte Carlo Factorization Algorithm. BIT, 1980, 20:176-184.
[7] Pollard J M.Theorems of Factorization and Primality Testing.Cambridge Philosophical Society,1974,76:521-528
[8] Williams H C.A p+1 method of factoring.Mathematics of Computation,1982, 39:225-234
[9] Lenstra H W Jr.Factoring integers with elliptic curves.Annals of Mathematics,1987,126(2):649-673
[10] 刘如玉, 周锦君. 椭圆曲线在整数分解中的应用[J]. 信息工程学院学报, 1999, 18(2): 53-55.
[11] Lehmer D H,Powers R E.On Factoring Large Numbers.Bulletin of the American Mathematical Society,1931,37:770-776.
[12] Morrison M A,Brillhart J.A Method of Factoring and the Factorization of F7.Mathematics of Computation,1975,29(129):183-205
[13] Dixon J D.Asymptotically fast factorization of integers.Mathematics of Computation,1981,36:255-260
[14] Pomerance C. The Quadratic Sieve Factoring Algorithm.EUROCRYPT1984 //Pomerance C,ed.A Tale of Two Sieves.Not.Amer.Math.Soc.,1996,43:1473-1485
[15] 周敏, 蒋增荣. 大整数分解的二次筛法及其微机实现[J]. 中山大学学报论丛, 1996, 5(3): 54-56.
[16] Contini S. Factoring integers with the self-initializing quadratic sieve. Masters thesis.University of Georgia,1997
[17] Michael C. A Beginner’s Guide To The General Number Field Sieve. Oregon State University.
[18] 倪谷炎. 用数域筛法分解大整数[J]. 国防科技大学学报, 1998, 20(4): 103-108.