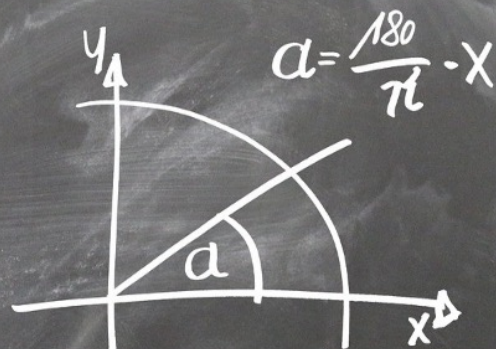
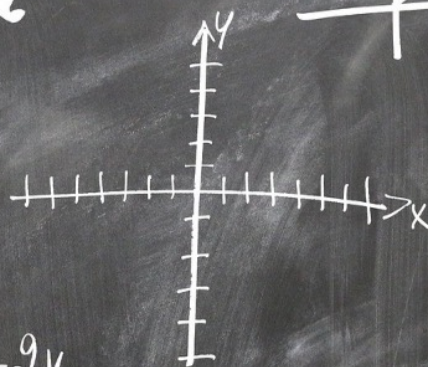


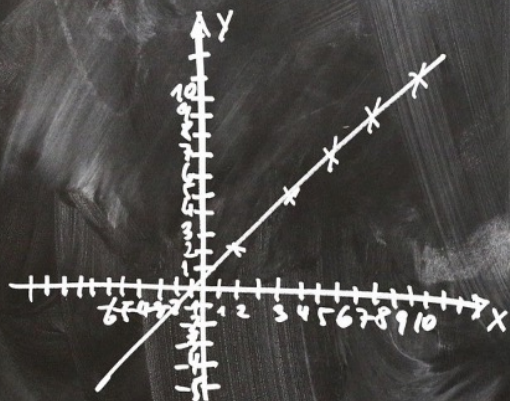
$$X_{1/2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



$$X^2 + px + q = 0$$



$$X_{1/2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}$$



$$X = 6 - 2y$$

$$X + a = b$$

$$f(x) = \tan x$$

$$f(x) = \sin x$$

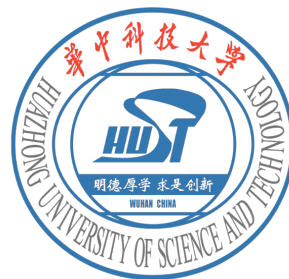
大整数拆分算法研究

作者：王清雨，李思岑，李书涵

组织：计卓 1801

时间：December 9, 2019

版本：1.00



目 录

1	Pollard ρ Algorithm	2
1.1	前置知识	2
1.2	算法简述	5
1.3	总结	7
2	二次筛选法	8
2.1	前置知识	8
2.2	算法简述	9
2.3	算法案例	10
3	Pollard $p - 1$ Algorithm	12
3.1	前置知识	12
3.2	算法简述	12
3.3	总结	13
4	Lenstra 椭圆曲线方法 (Elliptic Curve Method(ECM))	14
4.1	前置知识	14
4.2	Lenstra ECM 具体流程	17
4.3	时间复杂度	18
5	数域筛法 (Number Field Sieve)	19
5.1	前置知识	19
5.2	有理筛法	19
5.3	狭义数域筛法 (Special Number Field Sieve)	20
5.4	广义数域筛法 (General Number Field Sieve)	21
A	数据处理代码	24

第 1 章 Polland ρ Algorithm

1.1 前置知识

内容提要


❑ 伪随机序列 (pesudorandom sequence)

❑ 生日悖论 (birthday paradox)

1.1.1 伪随机序列

定义 1.1. 伪随机序列

伪随机序列 (pesudorandom sequence) 是一种特殊的数列，其中的数字出现看上去就像是随机数一样，没有特定的规律可言。

 **注意** 如同计算机中的伪随机数一样，伪随机序列并非真正的随机，整个伪随机序列可以通过一个初始值完全确定，该初始值也叫该序列的种子 (seed) (这点与我们以前在信息技术导论课上学到的伪随机数一样)

而在本算法中，我们要使伪随机序列中每个数的值小于 n ，因此我们需要一个多项式作为伪随机数生成器，并对得到的值对 n 取模。如例 1.1 所示。


例 1.1 (生成伪随机序列)

对于函数 $f(x) = (x^2 + 1) \bmod n$ ，我们可以得到一个序列 $\{x_k\}$ ，其中，

$$\begin{aligned}x_1 &= g(i) \\x_2 &= g(g(i)) \\x_3 &= g(g(g(i))) \\&\dots\end{aligned}$$

上述公式中对 i 即该伪随机序列的种子

在得到 $\{x_k\}$ 之后，我们可以立即得到另一个伪随机序列 $\{x_k \bmod p\}$ ，这两个序列是算法的关键之处。

 **注意** 我们一开始并不知道 p 的值，因此我们并不能立刻计算出第二个序列，目前我们只需要知道有这样两个序列即可。

对于上述的序列 $\{x_n\}$ 我们有如下性质

性质 $\{x_n\}$ 的取值是**有限**的，一定会出现循环。

证明 由 $x_i = g(x) \bmod n$ 可知， $\forall i \in (1, \infty), x_i < n$ ，因此由鸽巢原理可以得到上述性质。

性质 $\{x_n\}$ 中的每一项 x_i 的值只依赖于其前一项 x_{i-1} 的值。

证明 由 $\{x_n\}$ 的定义可以立即得到上述性质。

这个性质使得序列 $\{x_n\}$ 像递推式一样，只要确定了种子，就能确定整个序列。

在 1.2 的论述中，我们可以看到，上面的这些性质很有作用。

表 1.1: 生日悖论概率分布

n	$p(n)$
10	12%
20	14%
30	70%
50	97%
100	99.99996%
200	99.99999999999999999999999999998%
300	$1 - (7 \times 10^{-73})$
350	$1 - (3 \times 10^{-131})$
≥ 366	100%

1.1.2 生日悖论

命题 1.1. 生日悖论

生日悖论 (*birthday paradox*) 是指, 尽管一年有多达 365 天, 但只需要 23 人就能使有两人生日相同的概率达到 50%, 而对于 60 或更多的人, 概率要大于 99%。

注 尽管被称为悖论, 但实际上该命题在逻辑和概率上是完全正确的, 它被称为悖论只是因为这个数学事实与一般人的直觉相抵触而已。

我们可以利用刚刚学过的概率论的知识来对这个命题进行简单的证明 (验证)。

证明 假设一年 365 天中, 出生概率是均匀分布的。

我们首先来计算 n 个人中, 每个人生日都不相同的概率。

假如 $n > 365$, 由鸽巢原理可以知道, 该概率为

$$\bar{p}(n) = 0$$

而对于 $n \leq 365$, 该概率为

$$\bar{p}(n) = 1 \cdot \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdots \left(1 - \frac{n-1}{365}\right)$$

即

$$\bar{p}(n) = \frac{365!}{365^n (365 - n)!}$$

从而我们可以得到至少有两个人生日相同的概率

$$p(n) = \begin{cases} 1 - \frac{365!}{365^n (365 - n)!} & n \leq 365 \\ 1 & n > 365 \end{cases}$$

这样就验证了生日悖论的正确性。

从上述公式中, 我们可以通过代码¹计算人数对应的概率, 并作出图像, 如表 1.1, 图 1.1, 图 1.2, 图 1.3 所示

从图 1.1 中我们可以看出, 一开始的概率 $p(n)$ 很小, 但随着 n 的增大, 概率迅速增大, 且增长速率越来越快。

而从图 1.2 中, 我们可以看到概率趋向于 1 的全过程。正如命题 1.1 所述, 概率 $p(n)$ 在 $n = 23$ 左右达到 0.5, 在 $n = 60$ 左右达到 0.9。

同样的, 图 1.3 则展示了概率 $p(n)$ 在全部定义域上的形状。可以看出, 在大部分情况下, 概

¹相关代码见附录 A

图 1.1: 生日悖论概率分布

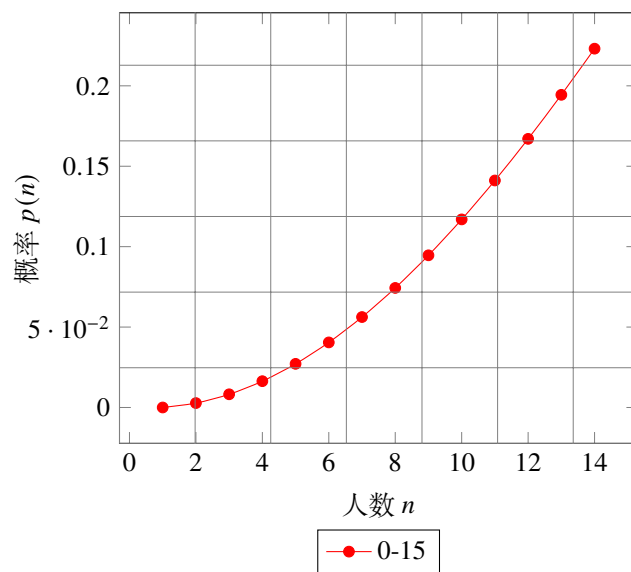


图 1.2: 生日悖论概率分布

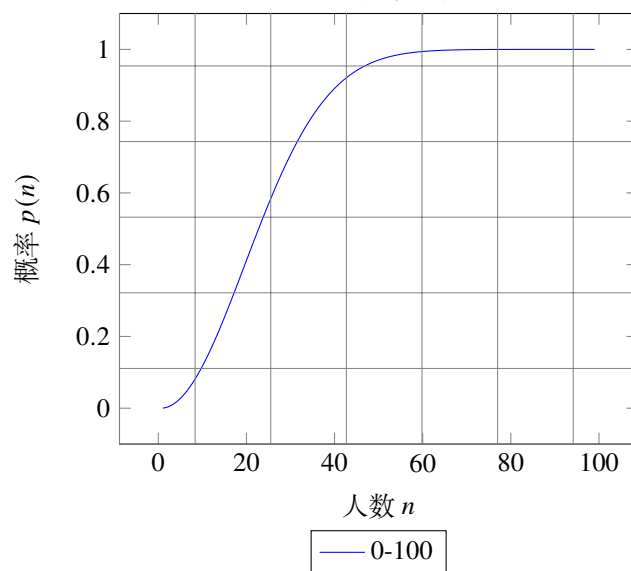
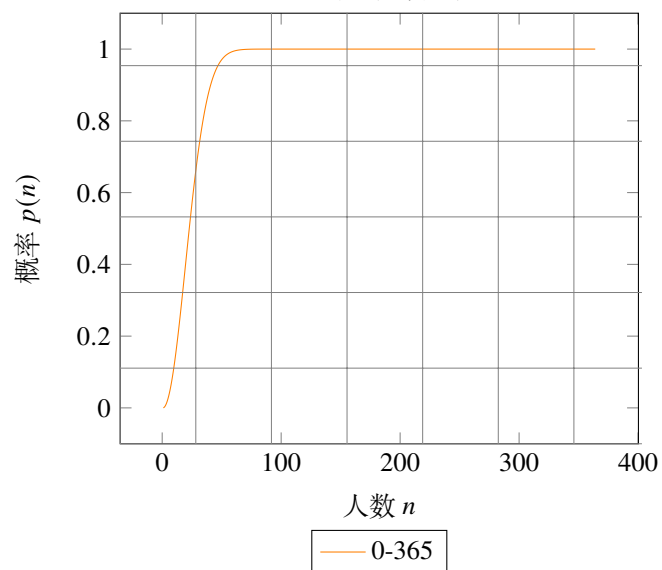


图 1.3: 生日悖论概率分布



率 $p(n)$ 都趋近于 1，这正符合命题 1.1 所描述的情形。

1.2 算法简述

1.2.1 核心思想

Pollard ρ 算法是一种启发式算法²因此算法中有一定的随机成分，体现在算法中则是需要用到之前提到的伪随机序列 $\{x_n\}$ 。

根据 1.1 中的描述，我们能得到引理 1.1

引理 1.1

如 1.1.1 所述，伪随机序列 $\{x_n\}$ 的值是有限的，因此会出现重复。

而又根据 1.1.2 中提到的生日悖论与 [1] 中的论述，出现重复之前预计要执行的步数为 $\Theta(\sqrt{n})$ 。

此外，序列 $\{x_n\}$ 包含一个对应的模 p 的序列 $\{x'_n\}$ ，即对于所有的 i ，有：

$$x'_i = x_i \bmod p$$

更进一步，我们能得到引理 1.2

引理 1.2

若 $f_n(x) = f(x) \bmod n$ 为我们用来生成 $\{x_n\}$ 的函数，那么有

$$x'_i = f_p(x'_{i-1})$$

其中， $f_p(x) = f(x) \bmod p$

证明

$$\begin{aligned} x'_i &= x_i \bmod p \\ &= f_n(x_{i-1}) \bmod p \\ &= ((x_{i-1}^2 - 1) \bmod n) \bmod p \\ &= (x_{i-1}^2) \bmod p \\ &= ((x'_{i-1})^2 - 1) \bmod p \\ &= f_p(x'_{i-1}) \end{aligned}$$

因此我们由引理 1.1 可以看出， $\{x'_n\}$ 与 $\{x_n\}$ 有相同的递推式，也都会满足引理 1.2。并且由于 p 是 n 的因子，因此 $\{x'_i\}$ 开始循环的步数的期望要小于 $\{x_i\}$ 开始循环的步数。也就是说 $\{x'_i\}$ 会更早进入循环。

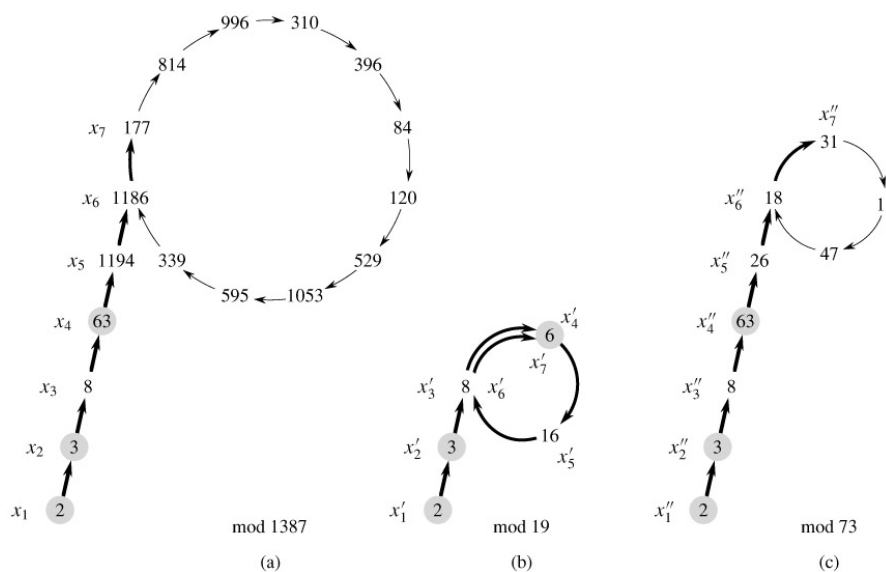


注意 尽管此时我们并不知道 p 的具体值，但是我们通过上述分析可以了解到有关 p 的性质，并利用这些性质来求解 p 。

我们构造一快 (y)、一慢 (x_i) 两个变量来遍历伪随机序列。当 y 与 x_i 在 $\{x'_i\}$ 相遇时，有

$$y \equiv x_i \pmod{p}$$

²又称蒙特卡洛方法 (Monte Carlo method)，是以概率和统计的理论为基础的一种计算方法，将所求解的问题同一定的概率模型相联系，用计算机实现统计模拟或抽样，以获得问题的近似解。

图 1.4: Pollard- ρ 启发式方法 [1]

即

$$y - x_i \equiv 0 \pmod{p}$$

又因为 p 为 n 的一个因数，因此有

$$\gcd(y - x_i, n) \neq 1$$

注意 上述过程描述的是正向的思维过程，而在算法中应当将上述过程反过来，找到使得 $\gcd(y - x_i, n) \neq 1$ 的值，从而求出 p 。

下面通过一个例子来进一步解释算法过程。

例 1.2 对整数 1387 进行分解。[1]

整个过程如图 1.4(a) 所示。粗箭头指出因子 19 被发现以前所执行的迭代步骤，细箭头指出在迭代中未达到的值。

因子 19 是在到达 $x_7 = 177$ 时被发现，此时计算 $\gcd(63 - 177, 1387) = 19$ 。

而图 1.4 (b) 则很好的描述了上面所说的 $\{x'_i\}$ 的情况。图 1.4 (a) 中给出的每个值 x_i 都对应着这里的 x'_i 例如， $x_4 = 63$ 和 $x_7 = 177$ 模 19 都等于 6，也就是上面论述中所说的在 $\{x'_i\}$ 中相遇的情况。

同样的，图 1.4 (c) 则描述了对于 $p = 73$ 时的情形。

1.2.2 算法描述

算法的执行过程如算法 1 所示。第 1 行到第 4 行进行初始化，将 i 初始化为 1，将 x_1 初始化为 Z_n 中一个随机的值。第 5 行开始循环，来搜索 n 的因子，在每一次迭代中，第 7 行用递归式

$$x_i = (x_{i-1}^2 - 1) \bmod n$$

计算无穷序列

$$x_1, x_2, x_3, \dots$$

中 x_i 的下一个值。

注 为了清楚，伪代码中使用了下标变量 x_i ，但即使去掉下标，程序也以同样的方式执行，因为仅仅需要保持最近计算出的 x_i 值。因此整个程序只需要常量级的存储空间。

Algorithm 1: Polland- ρ

输入: 需要分解的大整数 n
输出: n 的非平凡因子

```

1  $i \leftarrow 1$ 
2  $x_1 \leftarrow \text{Random}(0, n-1)$ 
3  $y \leftarrow x_1$ 
4  $k \leftarrow 1$ 
5 while TRUE do
6    $i = i + 1$ 
7    $x_i = (x_{i-1}^2 - 1) \bmod n$ 
8    $d = \text{gcd}(y - x, n)$ 
9   if  $d \neq 1$  and  $d \neq n$  then
10    | Print  $d$ 
11  end
12  if  $i == k$  then
13     $y = x_i$ 
14     $k = 2k$ 
15  end
16 end

```

程序不时地将最近计算出的 x_i 存入变量 y 中。具体来说, 存储的值是那些下标为 2 的幂的变量:

$$x_1, x_2, x_4, x_8, x_{16} \dots$$

第9行尝试根据 y 和 x_i 的值找出 n 的一个因子, 若发现 d 是 n 的非平凡约数, 则输出 d 的值。

1.2.3 算法分析

起初看时, 会觉得算法有一些神秘, 而且有一定的随机性, 会让人觉得不是很靠谱³。但实际上真正地追究其中的统计学与概率学原理, 会发现其中确实蕴含着深刻的数学道理。

实际上, Polland- ρ 算法绝对不会输出错误的答案, 它的任何输出都一定是 n 的非平凡约数。尽管该算法并不保证一定能得到输出⁴, 但我们根据统计学与概率学原理可以预计, 大约在执行 $\Theta(\sqrt{p})$ 次迭代后, 会输出一个 n 的因子。[1]

于是, 我们可以在这里给出一个命题

命题 1.2. Polland- ρ 算法的结果

Polland- ρ 算法可以在期望的 $\Theta(\sqrt{p})$ 次算数运算中, 找到 n 的一个小因子 p 。

具体的推导与启发式证明见 [1]。

1.3 总结

这个算法最早由 John Pollard 在 1975 年提出 [2]。并且在 [1-2] 中都有提到, Polland- ρ 算法在实践中的表现很好。

总体来说的话, 这个启发式算法很好的解决了大数拆分的问题。

后来也有人给出了一些改进的算法, 详细的研究可以参见 [3-4]

³蒙特卡洛方法普遍都给人以这种感觉

⁴有极小的概率会发生两个伪随机序列同时重复的情况, 这时只需要更换一个种子重新执行即可

第2章 二次筛选法

2.1 前置知识

内容提要

□ 费马小定理

□ 费马因式分解

2.1.1 费马小定理

定理 2.1. 费马小定理

若 p 是质数，且 $\gcd(a, p) = 1$ ，那么 $a^{p-1} \equiv 1 \pmod{p}$ 。

注 由于费马小定理在课内讲过，因此这里就不做证明和解释，更多关于费马小定理的内容可以参见 [1]

2.1.2 费马因式分解

定理 2.2. 费马因式分解

对于任意一个奇数 n ，有 $n = ab = x^2 - y^2$ ，其中 x 和 y 必为整数。

上述推导是很容易的，如下

证明 令

$$\begin{aligned}x &= \frac{a+b}{2} \\y &= \frac{a-b}{2}\end{aligned}$$

其中 n 为奇数，则 a, b 必为奇数，所以 $(a-b), (a+b)$ 必为偶数，进而 x, y 必为整数，且 $x > y$ 。

进而则有

$$\begin{aligned}n &= ab \\&= (x-y)(x+y) \\&= x^2 - y^2\end{aligned}$$

进而我们可以进行变换得到 $y^2 = x^2 - n$ ，因此有如下性质

性质 可以从 $x = \lceil \sqrt{n} \rceil$ 开始枚举，直到 $x^2 - n$ 为完全平方数为止，即可求得 x 与 y ，从而获得 a 和 b 。

证明 这个命题的证明比较简单，直接对定理2.2的结论进行变形即可

$$\begin{aligned}y^2 &\geq 0 \\ \Rightarrow x^2 - n &\geq 0 \\ \Rightarrow x &\geq \sqrt{n}\end{aligned}$$

而对于质数，我们则有更进一步的性质

性质 可以在区间 $[\lceil\sqrt{n}\rceil, \frac{n+1}{2}]$ 内进行枚举，从而判定 n 是否为质数。

证明 显然我们知道质数（除2以外）都是奇数，所以质数 n 的费马因式分解一定是 $n = n \cdot 1$ ，故 $a = n, b = 1$ ，这时我们枚举的区间即为 $[\lceil\sqrt{n}\rceil, \frac{n+1}{2}]$ 。

2.2 算法简述

2.2.1 算法思路

二次筛法即是来源于定理2.2的。由费马分解法我们有

$$x^2 - y^2 = n$$

即

$$x^2 \equiv y^2 \pmod{n}$$

这指导我们去寻找一些完全平方数之间的关系，而不是直接的去试除或者枚举。

我们构造二次函数

$$Q(x) = (x + \lceil\sqrt{n}\rceil)^2 - n$$

其中 x 为非负整数。

之所以构造这样一个二次函数，是因为它自然地满足我们想要的一些形式上的需求：

$$Q(x) \equiv (x + \lceil\sqrt{n}\rceil)^2 \pmod{n}$$

进一步明确，我们只需要 $Q(x)$ 是一个（模 n 的）平方数，即可以用费马分解法分解。但是在这个过程中，我们不会试图去直接寻找一个满足条件的完全平方数，而是去找一系列能够被某个质数集合完全分解的数，进而由这些数，拼凑出 $Q(x)$ 。这样思考是因为平方数的所有质因子的幂都是偶数（如 $400 = 2^4 \times 5^2$ ）。所以我们接下来需要做的便很明晰了。

2.2.2 名词定义

针对上文中提到的“某个质数集合”，我们给出如下定义

定义 2.1. 因子基 (Factor Base)

若一个集合中的所有数都为质数，则称其为因子基。

例 2.1 上面提到的集合 $\{2, 5\}$ 就是一个因子基。

定义 2.2. 光滑数 (Smothness)

若一个数能够被一个因子基中的质数完全分解，则称其为光滑数。

例 2.2 对于因子基 $\{2, 5\}$ ，400 相对于它就是光滑的，而 30 则是不光滑的。


2.2.3 算法步骤

2.2.3.1 确定因子基

因子基 $S = \{p_1, p_2, \dots, p_m\}$ ，其中要求 $p_m (m \geq 2)$ 满足两条性质：


- p_m 是素数

- np_m 是二次剩余的, 即满足表达式 $\exists x \text{ s.t. } x^2 \equiv n \pmod{p_n}$

 **注意** 这里根据 n 的大小取一定数量的满足性质的即可。

2.2.3.2 获取 $Q(x_i)$

根据二次函数 $Q(x) = (x + \lceil \sqrt{n} \rceil)^2 - n$, 我们取一定范围的 x , 计算其函数值构成初始的光滑数选取集合。

 **注意** 此范围根据 n 的大小来定, 若接下来算法未能分解, 则可进一步扩大。

假定我们在经历过步骤 2 之后获取的光滑数 $Q(x_i)$ 如下

$$T = \{Q(x_1), Q(x_2), Q(x_3) \dots Q(x_i)\}$$

则它们的乘积一定满足

$$Q(x_1)Q(x_2) \dots Q(x_i) \equiv ((x_1 + \lceil \sqrt{n} \rceil)(x_2 + \lceil \sqrt{n} \rceil) \dots (x_i + \lceil \sqrt{n} \rceil))^2 \pmod{n}$$

这时, 右边已经符合完全平方数, 而左边, 因为 $Q(x_i)$ 均是光滑的, 所以我们只需要找到合适的 $Q(x_i)$ 相乘来确保所得到的乘积 $Q(x)$ 的所有质因子的幂均为偶数即可, 而这一过程本质上是一个求解线性方程的过程。

我们将获取的这些光滑数做质因子分解 (实际上这一步已经在步骤 2 中完成), 如下

$$Q(x_i) = p_1^{a_{i1}} p_2^{a_{i2}} \dots p_m^{a_{im}}$$

然后我们就得到了指数矩阵 $M = \{a_{im} \bmod 2\}$, 接着选取这些 $Q(x_i)$ 只需要解如下线性方程即可:

$$RM = \vec{0}_m$$

所解得的列向量 $R = r_j$ 实际上是一个 01 串, 它表示的就是光滑数集合的一个子集, 即我们想要求取的用来拼凑结果的“积木”, 于是我们有

$$Q(x) = \prod_{r_j=1} Q(x_j)$$

这便是我们最初的等式 $x^2 \equiv y^2 \pmod{n}$ 中的 x , 而 y 则是

$$\prod_{r_j=1} (x_j + \lceil \sqrt{n} \rceil)^2$$

2.2.3.3 分解 n

$$n = \gcd(x - y, n) \cdot \gcd(x + y, n)$$

2.2.4 算法分析

命题 2.1. 二次筛选法的时间复杂度

$$e^{(1+o(1))\sqrt{\ln n \ln \ln n}} = L_n[1/2, 1]$$

具体的分析过程可以参见 [5]

2.3 算案例

例 2.3 接下来我们针对 15347 进行二次筛选分解。

- 因为 15347 比较小, 所以首先选择因子基 $S = \{2, 17, 23, 29, 31\}$ (满足要求)。
- 计算 $Q(x_i)$, 这里我们取 $0 \leq x_i \leq 99$, 接着有

$$\begin{aligned} T &= \{Q(x_1), Q(x_2), Q(x_3), Q(x_4), Q(x_5), Q(x_6), \dots, Q(x_{99})\} \\ &= \{29, 278, 529, 782, 1037, 1294, \dots, 34382\} \end{aligned}$$

- 使用筛法进行筛选:

1. 将 T 中所有数中含有 2 的因子全部除去, 得到 T_1 , 如下:

$$T_1 = \{29, 139, 529, 391, 1037, 647, \dots, 17191\}$$

2. 将 T_1 中所有数中含有 17 的因子全部除去, 得到 T_2 , 如下:

$$T_2 = \{29, 139, 529, 23, 61, 647, \dots, 17191\}$$

3. 依次类推, 遍历整个因子基 S , 最后得到

$$T_n = \{1, 139, 1, 1, 61, 647, \dots, 17191\}$$

- 最后 T_n 中出现了若干个 1, 这是预料之内的。
- 事实上, 在该过程中我们发现

$$Q(2) = (2 + \lceil 15347 \rceil)^2 - 15347 = 126^2 - 15347 = 529 = 23^2$$

这意味着 $126^2 \equiv 23^2 \pmod{15347}$, 则我们直接可以得到

$$15347 = \gcd(126 + 23, 15347) \cdot \gcd(126 - 23, 15347) = 149 \times 103$$

但一般情况下, 我们不会有这么好运, 所以抛去该数, 我们希望用剩下的数据走完整个算法流程。因此我们对拼凑因子 $Q(x_i)$ 有如下的选取

$$Q(0) = 29, Q(3) = 782, Q(54) = 16337, Q(71) = 22678$$

他们对应的指数矩阵 (mod 2), 如下:

$$M = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

- 最后的任务便是解如下方程

$$RM \equiv [0, 0, 0, 0, 0] \pmod{2}$$

线性代数方面的解方程这里从略, 因此解得

$$R = [1, 1, 0, 1]$$

则我们需要构造的 $Q(x)$ 如下

$$Q(x) = Q(0)Q(3)Q(71) = 29 \times 782 \times 22678 = 22678^2$$

相对应的有

$$(0 + \lceil 15347 \rceil)^2 (3 + \lceil 15347 \rceil)^2 (71 + \lceil 15347 \rceil)^2 = (124 \times 127 \times 195)^2 = 3070860^2$$

构造出预期的等式

$$22678^2 \equiv 3070860^2 \pmod{15347}$$

- 分解 15347

$$15347 = \gcd(3070860 + 22678, 15347) \cdot \gcd(3070860 - 22678, 15347) = 149 \times 103$$



第 3 章 Polland $p - 1$ Algorithm

Polland $p - 1$ 算法，要早于 Polland ρ 算法，在 1984 年同样由 Polland 在 [7] 提出，相比于 Polland ρ 算法，Polland $p - 1$ 算法在理解和实现上更为简单，但局限性更大。

3.1 前置知识

Polland $p - 1$ 算法是基于定理 2.1 的算法，同时，其正确性证明及结果分析还需要用到定义 2.2 以及与之有关的定义 3.1。

定义 3.1. 幂次光滑数

对于一个光滑数 $n = \prod_i p_i^{n_i}$ ，如果有 $\forall i, p_i^{n_i} \leq B$ ，则称 n 为 B -幂次光滑数。

例 3.1 $2^4 \cdot 3^2 \cdot 5^1$ 为 5-光滑数，但不是 5-幂次光滑数，因为其最大素数幂次为 $2^4 = 16$ 。¹

3.2 算法简述

假定我们要分解的整数 n 有素因子 p ，根据费马小定理 2.1，对于素数 p 和任意与 p 互素的整数 a ，我们有

$$a^{K(p-1)} \equiv 1 \pmod{p}$$

其中 K 为任意正整数。

我们想通过将指数表示为许多个素数的乘积使其是 $p - 1$ 的倍数，从而找到 p 。

3.2.1 算法描述



注意 对于 a 的选取，并不一定需要随机，可以固定选取同一个 a ，算法也同样工作。（如只要 n 为奇数，则我们可以一直选取 $a = 2$ ）

例 3.2 我们想通过 Polland $p - 1$ 算法来分解整数 $n = 299$ 。

1. 我们选择 $B = 5$
2. 因此 $M = 2^2 \cdot 3^1 \cdot 5^1$
3. 我们选择 $a = 2$
4. $g = \gcd(a^M - 1, n) = 13$
5. 因为 $1 < 13 < 299$ ，因此 13 就是 n 的一个因数

如果有 $g = 1$ ，那么说明不存在 n 的素因子 p ，使得 $p - 1$ 是 B -幂次光滑数，因此要扩大 B 的选取。同理，如果 $g = n$ ，说明选取的 B 过大，应当选择更小的 B 。[8]

¹该数是 16-幂次光滑数，也是 17-幂次光滑数，18-幂次光滑数

Algorithm 2: Polland $p - 1$

输入: 需要分解的大整数 n
输出: n 的非平凡因子

- 1 选取一个光滑数上界 B
- 2 $M \leftarrow \prod_{\text{primes } q \leq B} q^{\lceil \log_q B \rceil}$ // M 不用显式地计算出来
- 3 随机选取一个与 n 互素的整数 a
- 4 $g \leftarrow \gcd(a^M - 1, n)$ // 可以使用模指数运算来加快计算
- 5 **if** $1 < g < n$ **then**
- 6 Print(g)
- 7 **return**
- 8 **end**
- 9 **if** $g = 1$ **then**
- 10 **end**
- 11 选择更大的 B
- 12 **if** $g = n$ **then**
- 13 **end**
- 14 选择更小的 B

3.2.2 算法分析

3.2.2.1 选取合适的 B

由上面的分析可见，算法的关键即在于选取一个合适大小的 B ，如果 B 选取过大了，会导致算法运行过慢，如果 B 选取过小了，会导致算法失败。[7-8] 中都有提到如何选取合适的 B 。

3.2.2.2 时间复杂度

显然，算法的时间复杂度与 B 紧密相关，具体而言，其时间复杂度为 $O(B \times \log B \times \log^2 B)$ 。详细的推导参见 [2, 8]

3.3 总结

相较于前两个算法，这个算法在思想上更加简单，感觉上只是对试除法的一个简单的推广，自然地在运行效率上也就不如前两个算法了。²

²值得一提的是，在发明这个算法的一年后，Polland 就发明了 Polland ρ 算法

第 4 章 Lenstra 椭圆曲线方法 (Elliptic Curve Method(ECM))

4.1 前置知识


4.1.1 射影平面

时间回到古希腊时代，欧几里得在《几何原本》中提出了五条公设：

1. 任意两点可确定一条直线。
2. 任意一条线段可无限延伸成一条直线。
3. 给定任一点，已知半径，可作一个圆。
4. 所有直角都相等。
5. 两条直线被一条直线所截，若截线一侧的同旁内角之和小于两直角之和，则两条直线在这一侧相交。

在该书中，第五条公设仅在第 29 个命题的证明中使用到了，学界对此公设的存在意义产生了长达两千多年的争论，最终产生了一些几何分支

- 欧式几何：满足以上公设
- 罗氏几何（双曲几何）：过直线外一点至少有两不同的直线和已知直线平行
- 黎曼几何（椭圆几何）：在同一平面内任何两条直线都有公共点（交点），也就是说没有欧式几何下定义的平行线

 **注意** 这里我们主要说明平行线公设这一点存在的不同，其余不同点从略

我们定义**无穷远点** ∞ 是所谓的平行线的交点，这样一来平面上两两直线都统一由唯一的交点，性质如下：

1. 一条直线只有一个无穷远点，一对平行线有公共的无穷远点。
2. 任何两条不平行的直线有不同的无穷远点（否则会造成有两个交点）
3. 平面上全体无穷远点构成一条无穷远直线

进一步，**射影平面**很自然地有如下描述：

定义 4.1. 射影平面

平面上全体无穷远点与全体平常点构成射影平面。

射影平面点的定义如下：

定义 4.2. 射影平面点

对普通平面上点 (x, y) ，令 $x = X/Z, y = Y/Z, Z \neq 0$ ，则投影为射影平面上的点 $(X : Y : Z)$ 。

4.1.2 射影平面上的椭圆曲线

在射影平面上定义椭圆曲线如下：

定义 4.3. 射影平面上的椭圆曲线

一条椭圆曲线是在射影平面上满足 Weierstrass 方程（如下）所有点的集合。

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3$$

注 对上式的几点说明：

1. 椭圆曲线方程是一个齐次方程。
2. 曲线上每一个点都是光滑的（即各分量 X, Y, Z 的偏导数不同时为零）。

4.1.3 有限域

这里直接给出有限域 F_p 的定义：

定义 4.4. 有限域

- 有限域 F_p (p 为质数) 中含有元素: $0, 1, 2, \dots, p-1$
- 有限域 F_p 的运算定义：

$$a + b \equiv c \pmod{p}$$

$$a \cdot b \equiv c \pmod{p}$$

$$a \div b \equiv c \pmod{p}$$

- 单位元 1, 零元 0。
- 以上运算满足交换律, 结合律, 分配律。

4.1.4 有限域上的椭圆曲线

取 $Z = 1$ 则对应的椭圆曲线的普通方程¹如下：

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

一方面为了方便我们对曲线方程进行操作, 我们可以通过平移等操作将 y, xy, x^2 项消去, 另一方面椭圆曲线是连续的, 并不适合用于加密, 我们必须把椭圆曲线变成离散的点, 所以得到如下公理化的**有限域椭圆曲线**定义：

定义 4.5. 有限域椭圆曲线

设 F 是特征不为 2, 3 的域, $x^3 + ax + b \in F[x]$ 无平方因子, O 表示无穷远点, 则

$$E = \{(x, y) \in F^2 \mid y^2 = x^3 + ax + b\} \cup \{O\}$$

称为 F 上的一条椭圆曲线。

其中 a, b 为满足以下条件的小于 p 的非负整数

$$4a^3 + 27b^2 \not\equiv 0 \pmod{p}$$

需要说明的是, 特征不为 2, 3 以及无平方因子是由我们选取的 a, b 所满足的上述条件决定的。从图像上看, 椭圆曲线没有尖点。

椭圆曲线图象如图4.1和图4.2所示：

¹这是我们算法中要使用到的方程

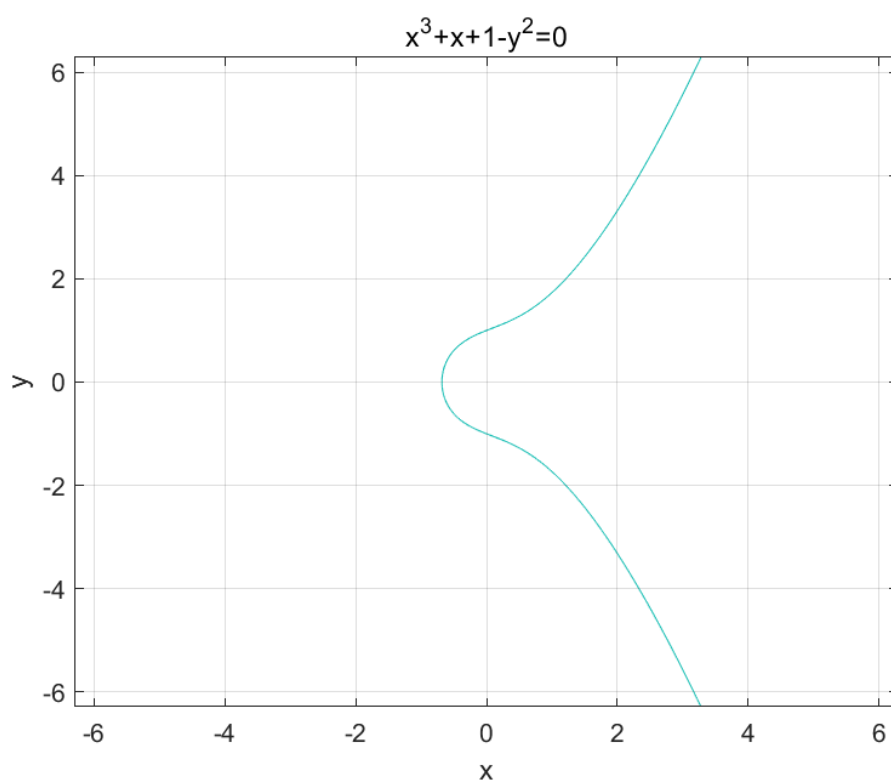


图 4.1: 椭圆曲线图像

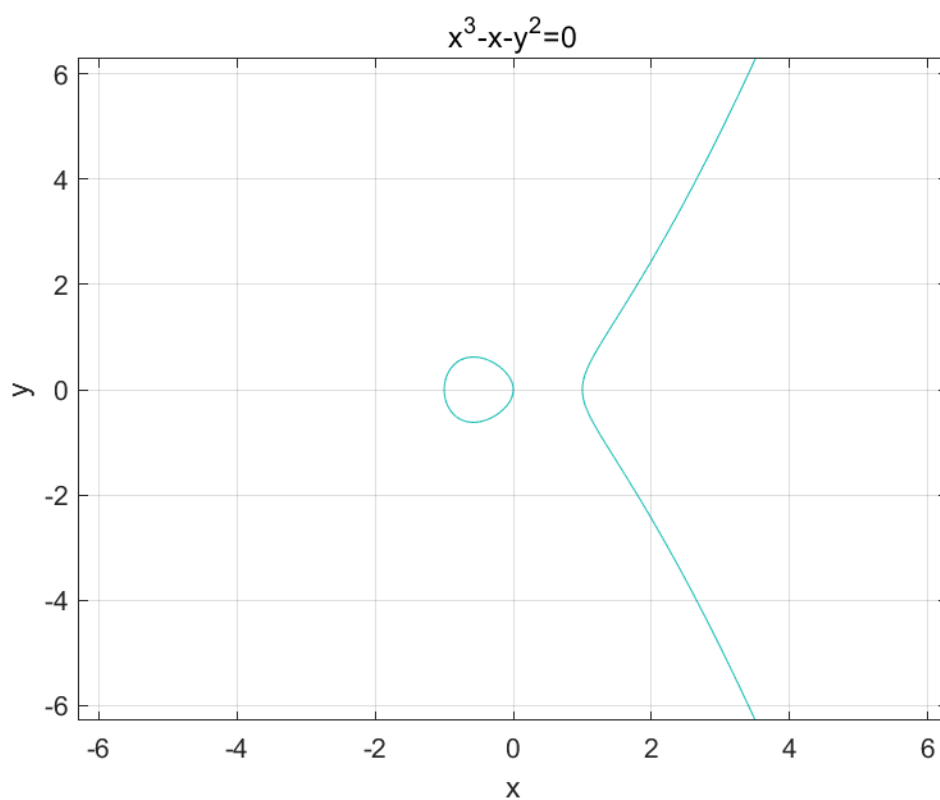


图 4.2: 椭圆曲线图像

4.1.5 阿贝尔群

群是一种代数结构，由一个集合以及一个二元运算所组成。已知集合和运算 $(G, *)$ 如果是群则必须满足如下要求：

定义 4.6. 群

1. 封闭性： $\forall a, b \in G, a * b \in G$
2. 结合性： $\forall a, b, c \in G, (a * b) * c = a * (b * c)$
3. 存在单位元： $e \in G, \forall a \in G, e * a = a * e = a$
4. 存在逆元： $\forall a \in G, b \in G, ab = ba = e$



注意 阿贝尔群在满足以上四条的同时，额外满足交换率 $a * b = b * a$ 。

4.1.6 椭圆曲线上的加法运算

设椭圆曲线 E ，点 $P, Q \in E$ ，经过 P, Q 的直线 l 交 E 于三点 $\{P, Q, S\}$ ，令 $-S$ 为点 S 关于 x 轴的对称点，则我们定义加法如下：

$$P + Q = -S$$

注 针对特殊情况的说明：

1. $P = Q$ 时， l 为 P 点处切线。
2. $P = -Q$ 时， $-S$ 为无穷远点 ∞ 。
3. $Q = \infty$ ，则 $P + \infty = \underbrace{-(-P)}_{k \text{ times}} = P$ 。
4. $kP = \overbrace{P + P + \dots + P}^{k \text{ times}}$ ，其中 k 为整数。

由上述定义我们显然可以证明该运算满足阿贝尔群的性质，这里不再赘述，但需要指出的是，该性质保证了算法中递推计算的合法性。

设 $P = (x_1, y_1), Q = (x_2, y_2), -S = (x_3, y_3)$ 则有：

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \quad \text{where } \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & P = Q \end{cases}$$

这里我们考虑了域 F_p 上的椭圆曲线，然而对于因子分解的认为来说，我们需要考虑 $\mathbb{Z}/N\mathbb{Z}$ 上的椭圆曲线。这里 $x_1 - x_2$ 等运算在 $\mathbb{Z}/N\mathbb{Z}$ 上未必可逆，但无伤大雅，有如下解释消除顾虑：

- 当 $x_1 - x_2$ 不可逆时，我们可以通过计算 $\gcd(x_1 - x_2, N)$ 来计算 N 的非平凡因子，从而直接完成因子分解。
- 当 $x_1 - x_2$ 可逆时，我们可以按照有限域上正常的运算进行算法。

因此我们不用花更多的功夫来定义 $\mathbb{Z}/N\mathbb{Z}$ 上的椭圆曲线，同时我们通过判断是否可逆来确定是否完成分解。需要指出的是，从几何上看， λ 是直线 PQ 的斜率，不可逆时 $\lambda = \infty$ ，也就是说我们需找到了一条竖直的线。

4.2 Lenstra ECM 具体流程

设要分解的整数 n ，满足 $\gcd(n, 6) = 1^2$ 。

- 随机选取整数 ab 满足条件 $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ ， E 为曲线 $y^2 = x^3 + ax + b$ 。

²保证域的特征不为 2, 3

- 取点 $P = (x_1, x_2)$, 以及指数 $e_k = \lfloor \ln_{p_k} B \rfloor$, 根据

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \quad \text{where } \lambda = \begin{cases} \frac{y_1 - y_2}{x_1 - x_2} & P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & P = Q \end{cases}$$

递推计算

$$P_i = i!P(2 \leq i \leq n)$$

如果计算中出现上述不可逆的 $t = x_1 - x_2$ 情况, 则直接分解, 否则返回上一步, 重新选取 a 。

- 计算 $d = \gcd(t, N)$ 。如果 $d \neq N$, 则完成分解, 否则返回第一步。

4.3 时间复杂度

以下论断摘自维基百科 [9]。

ECM 算法的时间复杂度依赖于我们所分解出的因子中最小的一个 p , 具体表达如下:

$$e^{(\sqrt{2}+o(1))\sqrt{\ln p \ln \ln p}}$$

同时, 需要指出的是, 该算法中多次用到扩展欧几里得算法³, 并且是针对不同的椭圆曲线求取出的不可逆点处, 如何快速求取模逆是该算法的一大优化要点, 具体算法有 Montgomery 加速算法, 它能够同时对多条椭圆曲线进行求模逆运算, 具体内容读者可参见 [10]

³实际上是求模逆

第 5 章 数域筛法 (Number Field Sieve)

5.1 前置知识

5.1.1 算术基本定理

定理 5.1. 算术基本定理

对于任何一个大于 1 的数 n ，有且仅有一种方法，将其写成有限个素数幂的乘积。

5.1.1.1 因子基

定义 5.1. 因子基

由算术基本定理，对于任何一个数 n ，我们可以找到有限个素数，将 n 写为这些素数幂的乘积；这些素数构成的集合 P ，称为 n 的因子基。

5.1.1.2 光滑数

定义 5.2. 光滑数

能被一系列较小的素数分解的数，称为光滑数。

注 光滑的概念通常伴随一个确定的因子基出现，因此常说：

命题 5.1

对于一个确定的因子基 P ，若某个合数 t 能被 P 分解，则称 t 是 P 上的光滑数。

在理论应用中，通常设置一个上界 B ，使 P 包含所有不大于 B 的素数；此时能被这样的 P 分解的数称为 B -光滑数。

5.1.2 素多项式

定义 5.3. 素多项式

指不能被分解为两个多项式乘积的多项式 [12]。

5.2 有理筛法

同此前的二次筛法，这一方法的核心思想仍然是通过构造平方数关于 n 的同余关系式

$$x^2 \equiv y^2 \pmod{n}$$

利用式子

$$n = \gcd(x + y, n) \cdot \gcd(x - y, n)$$

完成对 n 的质数分解 [13]。

- 对于需要分解的一个合数 n ，取一个合理的初始上界 B ，从而得到一个对应的因子基 P ；
- 若 P 中的一个或几个元素可以整除 n ，则可以退化到一个更小数的有理筛分解问题；若不能，则继续步骤 3；
- 我们可以寻找正整数 z ，使得 z 与 $z+n$ 都是 B -光滑数，且显然有

$$z \equiv z+n \pmod{n}$$

- 从小到大寻找满足条件的 z ；一般情况下，当寻找数量与因子基 P 的规模相当时，可以从对应的多个同余方程式中，选取一个或几个相乘，得到的新同余式即为，或可化简为两个完全平方数 x^2 与 y^2 的同余式。
- 根据得到的 x^2 与 y^2 ，利用上述完全平方同余式完成对 n 的质数分解。

例 5.1 分解合数 187；

不妨取 $B = 7$ ，得到对应的因子基 $P = \{2, 3, 5, 7\}$ ；不难验证，187 无法被 P 中任何一个元素整除。随后我们开始寻找正整数 z ，使得 z 与 $z+n$ 都是 7-光滑数。由于 $|P| = 4$ ，按顺序找到的前 4 个 z 分别为 2, 5, 9, 56，可以得到 4 个同余关系式：

$$2^1 3^0 5^0 7^0 = 2 \equiv 189 = 2^0 3^3 5^0 7^1 \pmod{187}$$

$$2^0 3^0 5^1 7^0 = 5 \equiv 192 = 2^6 3^1 5^0 7^0 \pmod{187}$$

$$2^0 3^2 5^0 7^0 = 9 \equiv 196 = 2^2 3^0 5^0 7^2 \pmod{187}$$

$$2^3 3^0 5^0 7^1 = 56 \equiv 243 = 2^0 3^5 5^0 7^0 \pmod{187}$$

可以发现，第三个式子直接满足了完全平方数的同余关系，可以直接得到

$$3^2 \equiv 14^2 \pmod{187}$$

则有

$$n = \gcd(14 + 3, 187) \cdot \gcd(14 - 3, 187) = 11 \times 17$$

也可以利用第一个和第四个式子相乘得到

$$2^4 \times 7 \equiv 3^8 \times 7 \pmod{187}$$

即

$$4^2 \equiv 81^2 \pmod{187}$$

则

$$n = \gcd(81 + 4, 187) \cdot \gcd(81 - 4, 187) = 11 \times 17$$

5.2.1 有理筛法的局限性

若某个数本身就是某个素数的幂，则有理筛法无法对其进行分解，但在统计角度看来，这样的数很少，且容易验证为素数的幂，故不会对实际的应用造成太大影响。

5.3 狭义数域筛法 (Special Number Field Sieve)

在引言中，我们介绍了有理筛法，主要由两个步骤实现：

- 寻找关于因子基 P 光滑的 z 和 $z+n$ ，且至少找到 $|P|$ 个这样的 z ；
- 根据上述 z 与 $z+n$ 得到同余关系式，并选取其中的某些相乘，得到完全平方数的同余式，

即形如 $x^2 \equiv y^2 \pmod{n}$ 的式子，最后利用式子

$$n = \gcd(x + y, n) \cdot \gcd(x - y, n)$$

完成对 n 的质数分解。

在要接下来即将介绍的狭义数域筛法中，仅有第一步操作与有理筛法不同。我们可以利用数域的特性，构建比有理筛法更高效的整数分解算法。

5.3.1 基本步骤

- 选取一个系数均为整数的素多项式 f ，以及一个整数 m 使得 $f(m) \equiv 0 \pmod{n}$ ；选定后，令 α 为 f 的根；由此可以产生一个环域 $[\alpha]$ ，且存在环同态函数 $\varphi: [\alpha] \rightarrow \mathbb{Z}/n\mathbb{Z}$ ，将 α 映射至 m 。为了讨论的简便，假设 $[\alpha]$ 是一个唯一分解环
- 在两个环域中，分别取两组因子基 P_1 与 P_2 ，在 $[\alpha]$ 上的因子基 P_1 由 N_{\max} 确定上界，在上 P_2 则由 B 确定上界；同上述有理筛法，两组因子基均包含环域中不超过上界的所有质数；
- 开始寻找互质的数对 (a, b) ，使在两个环域中分别满足： $a + bm$ 是关于 P_2 的光滑数； $a + b\alpha$ 是关于 P_1 的光滑数；
- 当找到这样足够多的数对时，可以生成一个幂系数矩阵，使用线性代数中求解线性方程组的方法，可以解得形如 $x^2 \equiv y^2 \pmod{n}$ 的完全平方同余式。

这种通过环域上的性质进行筛选得到完全平方同余式的过程，即为狭义数域筛法，简称 SNFS。

5.3.2 局限性

若一个数可以写为 $r^e \pm s$ 形式，且 r 和 s 都不太大时，或者更一般地写为 $ar^e \pm bs^f$ 时，狭义数域筛法可以较为高效地对其进行分解；如若不能，则其效率受限；大致原因，对于一个广义多项式，若其系数较大，则其对应的范数也更大；由于 SNFS 在进行分解时，依赖于固定的素数集，范数过大时，不易进行分解，导致算法效率很低；更加细致的分析需要更多关于抽象代数和环论的知识，再此不做赘述。

5.3.3 时间复杂度

摘自维基百科 [14]，对于 n 位整数，此算法的时间复杂度为：

$$\exp\left((1 + o(1))\left(\frac{32}{9} \log n\right)^{1/3} (\log \log n)^{2/3}\right) = L_n\left[1/3, (32/9)^{1/3}\right]$$

5.4 广义数域筛法 (General Number Field Sieve)

与狭义数域筛法思想相同，广义数域筛在有理筛法的基础上，也只对寻找同余关系式的方法进行改进，通过这些同余关系式获得完全平方同余式的方法是线性代数中主要研究的问题，在此略去。

广义数域筛法是狭义数域筛法的推广，通过使用两个素多项式，同时确定两个环域，来更高效地进行筛选。

5.4.1 具体步骤

选取两个多项式 $f(x)$ 与 $g(x)$ ，使它们满足：

- 系数均为整数
- 是有理数域上的素多项式
- 次数分别为确定的常数 d 和 e ，且都不太大
- $f(x)$ 与 $g(x)$ 对 n 取模后，有共同的整数根 m

对于上述 $f(x)$ 与 $g(x)$ ，可以分别求得根 r_1 与 r_2 。对于某个整数对 (a, b) ， $r = b^d f(\frac{a}{b})$, $s = b^e g(\frac{a}{b})$ ，不难验证， r 与 s 均为整数；考虑数域环 $[r_1]$ 与 $[r_2]$ ，设在数域环 $[r_1]$ 与 $[r_2]$ 上所选因子基分别为 P_1 与 P_2 ，该算法的目标是寻找恰当的整数对 (a, b) ，使 r 与 s 分别关于 P_1 和 P_2 光滑。

在找到足够多的整数对 (a, b) 后，使用高斯消元法，可以使 r 和 s 同时为平方数，从而得到所需的完全平方同余式。

5.4.2 时间复杂度

摘自维基百科 [15] 对于 n 位整数，此算法的时间复杂度为：

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}} + o(1)\right)(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$$

参考文献

- [1] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to algorithms[M]. [S.l.]: MIT press, 2009.
- [2] POLLALRD J. A monte oarlo method for factorization[J]. 1975.
- [3] TESKE E. Speeding up pollard's rho method for computing discrete logarithms[C]//International Algorithmic Number Theory Symposium. [S.l.]: Springer, 1998: 541-554.
- [4] CHRISTENSEN S. Speeding up the pollard's rho algorithm[C]//[S.l.: s.n.], 2015.
- [5] Wikipedia contributors. Quadratic sieve — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Quadratic_sieve&oldid=918872647.
- [6] POMERANCE C. Analysis and comparison of some integer factoring algorithms[J]. Computational methods in number theory, 1982:89-139.
- [7] POLLARD J M. Theorems on factorization and primality testing[C]//Mathematical Proceedings of the Cambridge Philosophical Society: volume 76. [S.l.]: Cambridge University Press, 1974: 521-528.
- [8] Wikipedia contributors. Pollard's p-1 algorithm — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Pollard%27s_p-1_algorithm&oldid=907113548.
- [9] Wikipedia contributors. Lenstra elliptic-curve factorization — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Lenstra_elliptic-curve_factorization&oldid=920275405.
- [10] Wikipedia contributors. Montgomery modular multiplication — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Montgomery_modular_multiplication&oldid=909117863.
- [11] *maTHμ* 计算机代数系统. 整数因子分解[EB/OL]. 2009. <http://mathmu.github.io/MTCAS/doc/IntegerFactorization.html#sec>.
- [12] Wikipedia contributors. Irreducible polynomial — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Irreducible_polynomial&oldid=926946608.
- [13] Wikipedia contributors. Rational sieve — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Rational_sieve&oldid=910692239.
- [14] Wikipedia contributors. Special number field sieve — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Special_number_field_sieve&oldid=908206397.
- [15] Wikipedia contributors. General number field sieve — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=General_number_field_sieve&oldid=916877592.
- [16] Wikipedia contributors. Pollard's rho algorithm — Wikipedia, the free encyclopedia[EB/OL]. 2019. https://en.wikipedia.org/w/index.php?title=Pollard%27s_rho_algorithm&oldid=921200660.

附录 数据处理代码

```
process.js

const fact = x => {
  if (x === 0n) return 1n;
  return fact(x - 1n) * x;
};

const factor = BigInt(10) ** 700n;
const fn = factor.toString().length;

const f = x => {
  const fm = (365n ** x) * fact(365n - x);
  return factor - fact(365n) * factor / fm;
};

for (var i = 1; i <15; i++) {
  const big = f(BigInt(i)).toString();
  const n = big.length;
  const dn = fn-n;
  const prefix = '0.' + Array(dn-1).fill('0').join('');
  const res = prefix + big.substr(0,20);
  console.log(i, res);
};
```
