寻找大三项式

Richard P. Brent　Paul Zimmermann

一个 *三项式* 是指一个含有三个非零项的单变量多项式. 例如 $P = 6x^7 + 3x^3 - 5$. 如果这个多项式 P 的系数 (在上述情形是 $6, 3, -5$) 在某个环或域 F 中, 我们就说 P 是 F 上的一个多项式, 记为 $P \in F[x]$, 在 $F[x]$ 上的多项式的加法和乘法运算通常定义为 F 上多项式系数的运算.

F 最通常的情形是 $F = \mathbb{Z}, \mathbb{Q}, \mathbb{R}$ 或 $\mathbb{C}$, 分别为整数域, 有理数域, 实数域或复数域. 然而, 有理数域上多项式在应用方面也是重要的. 我们局限于最简单有限域上的多项式, 即含有 2 个元素, 通常写成 0 和 1 的域 GF(2). 该域上的加法和乘法运算对整数用模 2 来定义, 所以 $0 + 1 = 1, 1 + 1 = 0, 0 \times 1 = 0, 1 \times 1 = 1$ 等.

这些定义的一个重要的结果是: 给定多项式 $P, Q \in \mathrm{GF}(2)[x]$, 我们有

$$(P + Q)^2 = P^2 + Q^2$$

由于 "交叉项" $2PQ$ 消失. 如果我们用域 GF(2) 上的多项式代替 $\mathbb{R}$ 上的多项式, 那么, 高中代数会变得更容易一些.

在域 GF(2) 上的三项式在密码系统和随机数生成方面是很重要的. 为了说明它的正确性, 考虑序列 $(z_0, z_1, z_2, \ldots)$ 满足下面的递推式:

$$z_n = z_{n-s} + z_{n-r} \mod 2, \tag{1}$$

这里 r 和 s 为给定的正整数且 $r > s > 0$, 初值 $z_0, z_1, \ldots, z_{r-1}$ 也是给定的. 递推式定义了序列中所有剩余的项 $z_r, z_{r+1}, \ldots$.

很容易建立硬件来实现递推式 (1). 我们需要一个能够存储 r 比特并且能够在电路中计算分开 $r - s$ 个位置的两个比特的模 2 加法 (等价地, "异或 (exclusive or)") 的移位寄存器. 另外, 这个寄存器可以把 r 到 s 位置的存储空间按 2 比特分开, 最后把它的输出再反馈到这个寄存器中. 它可以用图 1 ($r = 7$, $s = 3$) 来说明.

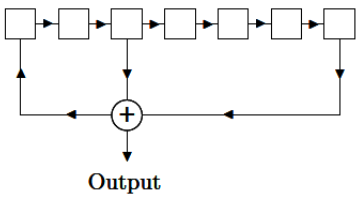

图 1　$z_n = z_{n-3} + z_{n-7} \mod 2$ 的硬件实现

递推式 (1) 看上去和著名的 Fibonacci (斐波那契) 递推式

$$F_n = F_{n-1} + F_{n-2};$$

相似. 的确, Fibonacci 数模 2 满足我们的递推式, 其中

译自: Notices of the AMS, Vol.58 (2011), No.2, p.233–239, The Great Trinomial Hunt, Richard P. Brent and Paul Zimmermann, figure number 1.
Richard Brent 是澳大利亚国立大学数学科学研究所的数学教授. 他的邮箱地址是 AMS@rpbrent.com.
Paul Zimmermann 是法国国立信息与自动化研究院 (INRIA) 研究主任. 他的邮箱地址是 Paul.Zimmermann@loria.fr.

$r = 2$, $s = 1$. 这种情形下给出了一个周期为 3 的序列 $(0, 1, 1, 0, 1, 1, \ldots)$: 不是很有意义. 然而我们如果取 r 更大些，我们能得到更长的周期.

该周期能够达到 $2^r - 1$, 用作伪随机数生成元或者流密码分量时，该序列会有意义. 事实上，如果初值不全为 0, 并且相关的三项式 $x^r + x^s + 1$ 作为在域 GF(2) 上的多项式是 *本原的* 时，周期为 $2^r - 1$. 一个本原多项式是一个不可约多项式 (该多项式没有非平凡因子) 且满足下节 "数学基础" 中的一个另外的条件.

Mersenne (*梅森*) *素数* 是指形式为 $2^r - 1$ 的素数. 这种素数是以 Marin Mersenne (1588—1648) 的名字来命名的. 他和许多同时代的学者交流，并在 1644 年给出了 $r \leq 257$ Mersnne 素数的清单 (并不完全准确).

Mersenne指数 是指 Mersenne 素数 $2^r - 1$ 中的指数 r. Mersenne 指数必须是素数. 但反之不成立. 例如，11 不是 Mersenne 指数，因为 $2^{11} - 1 = 23 \cdot 89$ 不是素数.

这篇文章的主题是寻找大指数的本原三项式，及与寻找大 Mersenne 素数之间的相互关系. 首先我们解释这两个主题之间的关系，并简单地描述一下 GIMPS 项目. 然后，我们描述我们寻找中使用的算法，该算法分为 "经典" 和 "现代" 两个不同的时期. 最后，我们叙述在现代时期获得的结果.

1. 数学基础

正如上文所述，我们考虑有限域 GF(2) 上的多项式. 一个 *不可约多项式* 就是一个除了它自身而不被任何非平凡多项式整除的多项式. 例如，$x^5 + x^2 + 1$ 是不可约的，而 x^5+x+1 不是, 因为 x^5+x+1 在域 $\mathrm{GF}(2)[x]$ 上可表示为 $x^5+x+1 = (x^2+x+1)(x^3+x^2+1)$. 我们不考虑二项式 $x^r + 1$, 因为它们能够被 $x + 1$ 整除，当 $r > 1$ 时是可约的.

一个次数为 $r > 1$ 的不可约多项式 P 产生包含 2^r 个元素的有限域 $\mathrm{GF}(2^r)$ 的一个表示：次数小于 r 的任一多项式可以表示一个元素，多项式相加后，次数仍小于 r. 乘法可以通过模多项式 P 来定义：首先两个多项式相乘，然后乘积模多项式 P. 这样 $\mathrm{GF}(2^r) \simeq \mathrm{GF}(2)[x]/P(x)$.

域 GF(2) 上的次数 $r > 0$ 的不可约多项式 P 称为是 *本原的*, 当且仅当 $P(x) \neq x$ 且模 P 的诸剩余类 x^k $(0 \leq k < 2^r - 1)$ 是互不相同的. 为了检验一个不可约多项式 P 是本原的只需检验模 P 剩余类 $x^k \neq 1$ 对 $2^r - 1$ 的最大非平凡因子 k 成立. 例如，$x^5 + x^2 + 1$ 是本原的，但 $x^6 + x^3 + 1$ 是不可约的但不是本原的，因为 $x^9 = 1 \bmod (x^6 + x^3 + 1)$. 这里 9 整除 $2^6 - 1 = 63$ 且是一个最大因子，因为 $63/9 = 7$ 是素数.

我们对本原多项式感兴趣是因为 x 是有限域 $\mathrm{GF}(2)[x]/P(x)$ 上乘积群的一个生成元，如果 $P(x)$ 是本原的.

如果 r 大且 $2^r - 1$ 不是素数，我们将很难验证次数为 r 的多项式的本原性，因为我们需要知道 $2^r - 1$ 的素数因子. 归于 Cunningham (坎宁安) 计划 [20], 对所有指数 $r < 929$ 是知道的，但对更大的 r, 一般就不清楚了. 另一方面，如果 $2^r - 1$ 是素数，那么所有的次数为 r 的不可约多项式是本原的，这就是我们研究 Mersenne 指数 r 的原因.

2. 开始寻找

在 2000 年，作者们通过邮件彼此进行交流，并且在与 Samuli Larvala 交流的同时，产生了用高效算法来检测 GF(2) 域上三项式不可约性和本原性的课题．由于伪随机数生成元的应用，第一个作者对这个课题感兴趣了很多年．Kumada 等人文章 [12] 的发表描述了一个指数为 859433 (一个 Mersenne 指数) 的本原三项式的寻找，促使我们 3 人开始从事寻找指数为 r 的本原三项式，r 跑遍所有已知的 Mersenne 指数．在那个时候，所知道的最大的 Mersenne 指数为 3021377 和 6972593. 已有的程序花费时间跟 r^3 成比例．因为 $(6972593/859433)^3 \approx 534$, 且由 Kamada 等人的计算在 19 个处理器上花费了 3 个月时间，这完全是一个挑战.

3. GIMPS 项目

GIMPS 代表 “伟大的互联网 Mersenne 素数寻找 (Great Internet Mersenne Prime Search)”, 这是由 George Woltman 在主页 `http://www.mersenne.org` 发布的计算计划. GIMPS 的目标是发现新的 Mersenne 素数. 自从在 1996 年发现 M_{35} 以来, 到 2010 年 12 月 GIMPS 在这 14 年里又发现了 13 个新的 Mersenne 素数, 并保持了最大 Mersenne 素数的记录．Mersenne 素数通常按大小呈递增的顺序排列为：$M_1 = 2^2 - 1 = 3$, $M_2 = 2^3 - 1 = 7$, $M_3 = 2^5 - 1 = 31$, $M_4 = 2^7 - 1 = 127$, ..., $M_{38} = 2^{6972593} - 1$, 等等.

由于 GIMPS 并不是按照顺序找到 Mersenne 素数的，对于已知最大的 Mersenne 素数的编号还存在某种不确定性．按发现的顺序，我们记 M'_n 为第 n 个 Mersenne 素数．在 $M_{39} = 2^{13466917}-1$ 之外还存在着一定的空隙. 这样对 $n > 39$ 可能有 $M'_n > M'_{n+1}$. 例如, 我们在找到 $M'_{46} = 2^{37156667}-1$ 和 $M'_{47} = 2^{42643801}-1$ 之前已经找到了 $M'_{45} = 2^{43112609}-1$. 在写这篇文章的时候, 第 47 个 Mersenne 素数是已知的, 且其中最大的为 $M'_{45} = 2^{43112609}-1$.

为了方便，我们记 r_n 为 M_n 的指数，r'_n 为 M'_n 的指数．例如 $r'_{45} = 43112609$.

4. Swan 定理

我们陈述一个有用的定理叫做 Swan 定理，尽管这个结果很早被 Pellet [14] 和 Stickelberger (斯蒂克贝格) [18] 发现．事实上，在 Swan 的文章 [19] 中有几个定理．我们陈述 Swan 推论 5 的一个简单形式.

定理1 令 $r > s > 0$, 假设 $r + s$ 是奇数，那么 $T_{r,s}(x) = x^r + x^s + 1$ 在下述情形在域 GF(2) 上有偶数个不可约因子：

a) r 为偶数，$r \neq 2s$, $rs/2 = 0$ 或 1 mod 4.

b) r 为奇数，s 不整除 $2r$, $r = \pm 3 \bmod 8$.

c) r 为奇数，s 整除 $2r$, $r = \pm 1 \bmod 8$.

在其他所有情形，$x^r + x^s + 1$ 含有奇数个不可约因子.

如果 r 和 s 都为偶数, 那么 $T_{r,s}$ 是一个平方且含有偶数个不可约因子. 如果 r 和 s 都为奇数, 我们可以把这个定理应用到 “倒数多项式” $T_{r,\,r-s}(x) = x^r T(1/x) = x^r + x^{r-s} + 1$, 因为 $T_{r,\,s}(x)$ 和 $T_{r,\,r-s}(x)$ 有相同数量的不可约因子.

对 r 是一个奇素数且超出容易检验的情形 $s = 2$ 或 $s = r - 2$ 时，情形 b) 说明三项

式含有偶数个不可约因子，所以在 $r = \pm 3 \bmod 8$ 时一定是可约的．这时，我们只需考虑那些 $r = \pm 1 \bmod 8$ 的 Mersenne 指数．$r > 10^6$ 的已知的 14 个 Mersenne 指数中，只有 8 个满足这个条件.

5．基本运算的成本

我们需要的基本运算有模三项式 $T = x^r + x^s + 1$ 的平方，模 T 的乘法以及 T 和小于 r 次多项式的最大公因子 (GCD). 根据实现这些运算需要的比特和字节数目来测量这些运算的成本．在 GF(2)[x] 中，由于 $x^i + x^j$ 的平方为 $x^{2i} + x^{2j}$, 所以平方的成本为 $O(r)$. 由 T 的稀疏性，低于 $2r$ 次多项式模 T 的化简成本为 $O(r)$; 因而模平方成本为 $O(r)$.

模乘法成本为 $O(M(r))$, 这里 $M(r)$ 是指在 GF(2) 上两个次数小于 r 的多项式乘法的成本；模 T 化简成本为 $O(r)$, 因此乘法成本会主导化简成本．“经典” 多项式乘法算法有 $M(r) = O(r^2)$, 但是归于 Schönhage 的一个算法 [1] 有 $M(r) = O(r \log r \log\log r)$[16].

通过使用 “分治算法” 结合 Schönhage 的快速多项式乘法, 次数不超过 r 的多项式的 GCD 计算成本为 $O(M(r) \log r)$．这些成本归纳在表 1 中.

表 1　基本运算的成本

modular squaring	$O(r)$
modular product	$O(M(r))$
GCD	$O(M(r) \log r)$

6．不可约测试

令 $\mathbf{P}_r(x) = x^{2^r} - x$. 周知 Gauss (高斯) 一个定理指出 $\mathbf{P}_r(x)$ 为所有 d 次不可约多项式的乘积，这里的 d 取遍 r 的所有因子．例如，在域 GF(2)[x] 上

$$\mathbf{P}_3(x) = x(x+1)(x^3 + x + 1)(x^3 + x^2 + 1).$$

这里 x 和 $x + 1$ 为 1 次不可约多项式，剩下的其他因子是 3 次不可约多项式．注意，在 GF(2) 上运算时，因为 $1 = -1$ (或等价地，$1 + 1 = 0$), 我们总可以用 “+” 代替 “−”.

特别地，若 r 是一个奇素数，则 r 次多项式 $P(x) \in \mathrm{GF}(2)[x]$ 不可约，当且仅当

$$x^{2^r} = x \bmod P(x). \tag{2}$$

(若 r 不是素数，那么 (2) 是必要的，但不是充分的；我们还需要检查一个进一步的条件来保证不可约性，见 [8].)

当 r 是一个素数时，方程 (2) 能够简单地测试不可约性 (或本原性，当 r 为 Mersenne 指数时): 只需要做 r 次模平方，由 x 开始，并检查结果是否是 x. 因为每个平方的成本为 $O(r)$, 因而不可约性测试的成本为 $O(r^2)$.

在模复合 (modular composition) [11] 和快速矩阵乘法 [3] 的基础上，我们有更复杂的算法来检验多项式的不可约性．然而，在次数小于 10^7 的三项式的应用中，这些算法实际上比经典算法要慢一些.

在寻找次数为 r 的不可约三项式时，我们可以假定 $s \le r/2$, 因为 $x^r + x^s + 1$ 是不可约的，当且仅当它的倒数多项式 $x^r + x^{r-s} + 1$ 是不可约的．这个简单的观察节省了 1/2 成本．在下面我们总是假定 $s \le r/2$.

1) 这个算法不同于 Schönhage-Strassen 整数乘法算法, 整数乘法算法不适用于 GF(2). 细节见 [2, 16]. —— 原注

7. 因子的次数

为了预测我们算法所期待的行为，我们需要知道不可约因子的次数预期分布. 我们的复杂性估计是基于假设次数为 r 的三项式的行为像所有同次多项式一样，相差一个常数因子.

假设 1 *在 GF(2) 上所有次数为 r 的三项式 $x^r + x^s + 1$ 中，一个三项式没有次数 $\le d$ 的非平凡因子的概率 π_d 至多为 c/d, 这里 c 为一个绝对常数，且 $1 < d \le r/\ln r$.*

这个假设虽然没有被证明，但是与实验一致，是合理的. 这个并不是关键的，因为我们算法的正确性并不依赖于这个假设，只有预测的运行时间依赖于这个假设. 在运行时间的预测中，d 的上界 $r/\ln r$ 是足够大的. r 作为 d 的上界可能是不正确的，因为它意味着至多 c 个 r 次的不可约多项式，但是我们期望的这个数是无界的.

表 2　对 $r = r_{38}$ 的统计

d	$d\pi_d$
1	1.00
2	1.33
3	1.43
4	1.52
5	1.54
6	1.60
7	1.60
8	1.67
9	1.64
10	1.65
100	1.77
1000	1.76
10000	1.88
226887	2.08

在 $r = r_{38}$ 的情形下，这个假设的一些证据呈现在表 2 中. $d\pi_d$ 在当 $d = 226887$ 时取到最大值，且最大值为 2.08. 我们试图解释 d 较小时 $d\pi_d$ 的准确值是有意义的，但这样会导致我们的研究更复杂.

8. 筛选

当我们测试一个较大的整数 N 是否素数时，在运用像 AKS, ECPP 或 (如果我们愿意接受一个小的误差概率) Rabin-Miller (米勒) 的素数测试之前，检查它是否有任何小的因子是明智的. 类似地，当测试一个高次多项式的不可约性时，在运用 $O(r^2)$ 测试之前检验其是否含有小的因子是明智的.

因为 d 次不可约多项式整除 $\mathbf{P}_d(x)$, 我们可以通过计算

$$\gcd(T, \mathbf{P}_d)$$

来检验一个三项式 T 是否含有一个 d (或 d 的某个因子) 次因子. 如果 $T = x^r + x^s + 1$ 且 $2^d < r$, 我们能够归结为一个次数小于 2^d 的多项式的 GCD 的计算. 令 $d' = 2^d - 1$, $r' = r \bmod d'$, $s' = s \bmod d'$, 那么 $\mathbf{P}_d = x(x^{d'} - 1)$,

$$T = x^{r'} + x^{s'} + 1 \mod (x^{d'} - 1),$$

所以，我们只需计算

$$\gcd(x^{r'} + x^{s'} + 1,\ x^{d'} - 1).$$

我们把这个过程叫做 “筛选”, 类似于筛选出整数的小的素数因子的过程，尽管那里使用的是 GCD 计算.

如果有小于 $\log_2(r)$ 次的因子的三项式通过筛选被排除,则由假设 1, 我们留下 $O(r/\log r)$ 三项式要检测，筛选的成本可以忽略不计. 这样，所有寻找的成本为 $O(r^3/\log r)$.

9. 证明的重要性

$r < r_{32} = 756839$ 次的本原三项式在 Heringa 等人的文章 [10] 中被列出. Kumada 等人在 [12] 中报道了对 $r_{33} = 859433$ 次的本原三项式的一个寻找 (他们没有考虑 r_{32}). 他们

找到了一个本原三项式，然而，由于筛选过程的一个缺陷，他们漏掉了三项式 $x^{859433}+x^{170340}+1$. 在 2000 年 6 月，当我们在已知的例子测试我们的程序时，发现了这个丢失的三项式.

这促使我们对所有我们测试过的三项式做可约性的 证明 (当然不包括少量证明是不可约的). 可约性的一个证明, 理想情形下是一个非平凡因子. 如果一个三项式 T 通过筛选发现有一个小因子, 那么, 很容易保持这个因子的记录. 如果我们不知道一个因子, 但是这个三项式没有通过不可约性测试 (2), 那么我们可以记录下剩余 $R(x)=x^{2^r}-x \bmod T$. 由于剩余可能很大，我们可以选择记录它的一部分. 例如，$R(x) \bmod x^{32}$.

10. 经典时期

2000—2003 年期间被称为是 经典 时期, 我们使用上面列出的经典算法来有效实现. 因为不同的三项式可以在不同的计算机上来测试，这容易进行并行寻找，通过使用尽可能多的处理器. 例如，当学生放假的时候，我们经常在假期充分利用本科教学实验室的计算机.

用这种方式，我们发现了 $r_{32}=756839$ 次的 3 个 (在 2000 年 6 月), 阶 $r_{37}=3021377$ 次的 2 个 (2000 年 8 月和 12 月), 和 $r_{38}=6972593$ 次的 1 个本原三项式 (2002 年 8 月).[1] 指数 r_{38} 的计算是在 2003 年 7 月完成并进行了双重检测.

对于指数 $r_{38}=6972593$, 证明只有唯一一个本原三项式 x^r+x^s+1[2](和通常一样, 假设 $s \le r/2$). 我们该怎么确定我们没有漏掉一些呢？对每个非本原多项式，我们有一个证明，并且这些证明被一个独立的计算进行检验. 事实上，我们发现了少量的差异，可能是由于我们使用较旧的计算机内存奇偶检验误差导致的. 在一任何长时间的计算中都存在风险，我们不应该假设计算机万无一失. 同样的现象在 Nicely [13] 中的 Brun (布伦) 常数的计算中也被发现 (也发现了声名狼藉的 “奔腾错误”).

因为我们已经赶上了 GIMPS 项目，我们认为 (不是最后一次) 这个项目已经完成，并且在 [4, 5] 中发表了我们的结果. 然而，GIMPS 不久超越了我们，通过寻找几个更大的 Mersenne 素数，他们所用的指数为 $\pm 1 \bmod 8$: $r'_{41}=24036583, \ldots, r'_{44}=32582657$.

指数 $r_{38}=6972593$ 的寻找花了两年多的时间 (2001 年 2 月—2003 年 7 月), 因此，处理新的 Mersenne 指数 $r'_{41}, \ldots, r'_{44}$ 似乎是不可行的.

11. 现代时期

我们意识到为了拓展计算，我们必须找到更加有效的算法. 计算中费用的大部分是通过方程 (2) 来测试不可约性. 如果我们进一步筛选，我们将会避免大部分的不可约性测试. 从假设 1 知，如果我们筛选到指数 $r/\ln r$, 那么我们将期望仅仅有 $O(\log r)$ 不可约性测试.

我们所需要的是在 平均 水平较快的时间下找到稀疏多项式 (特别地，一个三项式) 的最小因子的算法.

1) 次数为 r_{34}, r_{35} 和 r_{36} 的本原三项式通过 Swan 定理排除，就像 r_{39} 和 r'_{40} 一样.—— 原注

2) 次数为 6972593 的唯一一个本原三项式是 $x^{6972593}+x^{3037958}+1$. 为了纪念 [5] 的 3 个作者发现它时正在访问的村庄而命名为 Bibury.—— 原注

在有限域上有许多分解多项式的算法，如 [8], 它们中大多数的成本是通过 GCD 计算控制的. 然而, 用模乘法来代替大多数 GCD 的计算是可能的. 通过一个称为 *模块化* 的处理 (在整数分解背景由 Pollard [15] 中引入，对多项式分解在 von zur Gathen 和 Shoup [9] 中引入). 想法是简单的：替代计算 $\gcd(T, P_1), \ldots, \gcd(T, P_k)$, 希望找到一个非平凡 GCD (因此是 T 的一个因子), 如果它们不是不可约的，我们计算 $\gcd(T, P_1P_2\cdots P_k \bmod T)$, 且如果有必要就追踪分离因子. 对 $r \approx r'_{41}$, 一个 GCD 通常占用模乘法 40 倍长的时间，那么，模块化将会给出大的加速.

2007 年 2 月，在第 2 作者访问第 1 作者期间，我们意识到第 2 级别模块化可以通过平方来代替大多数的模乘法. 因为一个模乘法可能占用一个平方的 400 倍的时间 (对 $r \approx r'_{41}$), 这个第 2 级别模块化可以再一次提供一个大的加速，具体细节在 [6] 中有描述. 这里我们仅仅注意到 m 次乘法和 m 次平方能够被一次乘法和 m^2 个平方来替换. m 的最优值为 $m_0 \approx \sqrt{M(r)/S(r)}$, 这里 $M(r)$ 为一次模乘法的成本，$S(r)$ 为模平方的成本，导致加速大约 $m_0/2$. 如果 $M(r)/S(r) = 400$, 那么 $m_0 \approx 20$, 且在单级模块化上的速度大约提高了 10 倍.

到 2008 年 1 月，利用这些想法，结合多项式乘法的一个快速实现 [2] 和一个子二次 GCD 的算法，我们能够找到 10 个指数为 $r'_{41}, \ldots, r'_{44}$ 的本原三项式. 再次我们认为我们完成了且在 [7] 中发表了我们的结果，没想到在 2008 年 8 月发现了 M'_{45}, 相继不久又发现了 M'_{46} 和 M'_{47}, 使 GIMPS 项目又有了一次向前的飞跃. 指数 r'_{46} 被 Swan 定理排除了，但是我们不得不开始处理指数 $r'_{45} = 43112609$ 和之后稍小的指数 $r'_{47} = 42643801$.

对指数 r'_{45} 的寻找是于 2008 年 9 月—2009 年 5 月, 在 Dan Bernstein 和 Tanja Lange 的帮助下研究的. 他们非常友好地允许我们使用他们在 Eindhoven 的计算资源，导致 4 个本原三项式次数的记录.

表 3　次数 r 是 Mersenne 指数的本原三项式 $x^r + x^s + 1$, $s \le r/2$

r	s
24 036 583	8 412 642, 8 785 528
25 964 951	880 890, 4 627 670, 4 830 131, 6 383 880
30 402 457	2 162 059
32 582 657	5 110 722, 5 552 421, 7 545 455
42 643 801	55 981, 3 706 066, 3 896 488, 12 899 278, 20 150 445
43 112 609	3 569 337, 4 463 337, 17 212 521, 21 078 848

对指数 r'_{47} 的寻找是在 2009 年 6 月—2009 年 8 月发现了 5 个本原三项式. 在这个例子中，我们在澳大利亚国立大学很荣幸地进入一个有 224 个处理器的新计算机组，所以相对于早期的寻找，计算花费了较少的时间.

在 “现代时期” 我们的计算结果用表 3 给了出来. 对 s 值似乎没有任何可预测模式. 本原三项式的数目对给定的 Mersenne 指数 $r = \pm 1 \bmod 8$ 似乎是符合均值为 3.2 的 Poisson (泊松) 分布 (所以，它不可能被一个绝对常数界定 —— 参看上面假设 1 的讨论).

12. 现代算法 —— 一些细节

为了总结寻找本原三项式的 “现代” 算法，通过进一步筛选寻找最小次数的因子，使用基于快速乘法和两个级别的模块化的分解算法，我们在经典算法方面有所提高. 我们在下面段落中给出现代算法的一些细节，并且对现代算法和经典算法做个比较.

给定一个三项式 $T = x^r + x^s + 1$, 我们寻找最小次数 $d \le r/2$ 的一个因子 (事实上，

应用 Swan 定理，我们常常限定寻找到 $d \le r/3$, 因为我们知道三项式有奇数个不可约因子), 若找到了这样的因子，我们可知 T 是可约的，因此程序输出 “可约” 且保存这个因子作为可约性的一个证明. 通过计算 T 和 $x^{2^d}+x$ 的 GCD, 我们可以得到这个因子，若此 GCD 是非平凡的，那么 T 至少有一个次数为 d 的因数的因子，如果次数小于 d 的因子已经被排除，那么 GCD 只含有次数为 d 的因子 (可能是几个这样的因子的乘积), 这就是所谓的 *不同次分解* (DDF).

若 GCD 的次数为 λd, $\lambda > 1$, 且我们想把乘积分解成 λ 个次数为 d 的因子，这时我们需要用到 *同次分解* (EDF) 算法. 若 EDF 是必要的，这个算法一般费用低. 因为，若 $\lambda > 1$, 总次数 λd 常常较小.

在这种方式下, 仅仅由最小可能次数的一个非平凡因子我们就可得到可约性的证明，如果有几个这样因子，字典排序取最小的因子. [1)] 这些证明可以被检验，例如运用 NTL [17] 中一个独立的程序，比原始的计算要快得多 (对任何表 3 中的次数至少快一小时).

对大的 d, 当 $2^d \gg r$ 时，我们不去计算 $x^{2^d}+x$ 本身，而是计算模 T 的剩余，比方说 h. 事实上，$\gcd(T, x^{2^d}+x) = \gcd(T, h)$. 为了计算 h, 我们从 x 出发，执行 d 个模平方，且加上 x. 用这种方式，我们对次数小于 $2r$ 的多项式进行计算，检验次数为 d 的因子，需要 d 个模平方和一个 GCD 的成本. 因为我们按升序检验可能的次数 d, 我们计算 $x^{2^d} \bmod T$ 是从前一步得到的 $x^{2^{d-1}} \bmod T$, 结合一个额外的模平方. 这样，从表 1, 每个 d 值的成本为 $O(M(r)\log r)$, 这没有考虑上面讨论的基于模块化的加速.

最关键的事实是大多数三项式都有一个小因子，所以寻找的速度比平均快.

当寻找次数 $d < 10^6$ 的因子失败以后，比方说，我们需要转向经典的不可约检验标准 (2), 它在处理次数大于 10^6 的速度比分解快. 然而，在那种情形下，我们的证明清单是不完全的. 因为很少去寻找一个次数大于 10^6 的因子，我们运行程序直到找到因子或输出 “不可约”. 在后者的情形，当然，我们可以用经典的检测验证结果. 在我们寻找过程中找到的证明 (最小不可约因子) 中，三项式 $x^{42643801}+x^{3562191}+1$ 最大的因子是 $P(x) = x^{10199457}+x^{10199450}+\cdots+x^4+x+1$, 需要注意的是，尽管三项式是稀少的，并且有紧表示，但是因子是稠密的并且太大而不能在这里完全呈现.

13. 经典和现代的对比

为简单计，我们使用忽略对数因子的 $\tilde{O}$ 记号. “经典” 算法对每个三项式占用预期时间是 $\tilde{O}(r^2)$, 或对所有次数为 r 的三项式占用预期时间是 $\tilde{O}(r^3)$.

“现代” 算法对每个三项式占用预期时间是 $\tilde{O}(r)$, 或对所有次数为 r 的三项式占用预期时间是 $\tilde{O}(r^2)$.

在应用中，现代算法对 $r = r_{38} = 6972593$ 大约快 160 倍，对 $r = r'_{45} = 43112609$ 大约快 1000 倍.

因此，比较 $r = r'_{45}$ 和 $r = r_{38}$ 的两种计算：使用经典算法会花费 240 倍长的时间 (不现实的), 但用现代算法只需 1/1000 的时间.

1) 费力去找字典排序最小的因子是值得的，因为这使得证明是唯一的，且允许我们比较不同形式的程序和比不这么做更容易确定漏洞.—— 原注

14. 如何加快寻找

这里总结最关键的思想. 下述 (1)—(4) 适用于经典算法和现代算法. (5)—(6) 仅适用于现代算法.

(1) 因为每个三项式的计算可以独立进行, 容易使用尽可能多的计算机做并行寻找.

(2) 因为在 GF(2) 上的多项式的系数仅仅为 0 或 1, 在次数 $< d$ 的多项式和有 d 比特的二进制数之间存在一个一一对应. 因此, 在一个有 64 个字节的计算机上, 我们可以用 $\lceil (d+1)/64 \rceil$ 个计算机字来编码一个次数为 d 的多项式. 如果我们写程序仔细, 我们可以使用全字计算机运算, 做 64 次并行运算来处理这样的多项式.

(3) 在 GF(2) 上的多项式的平方可以在 线性时间 完成 (是多项式次数的线性函数), 因为平方的交叉项消失:

$$\Big(\sum_k a_k x^k\Big)^2 = \sum_k a_k x^{2k}.$$

(4) 一个 $2(r-1)$ 次的多项式模一个 r 次的三项式 $T = x^r + x^s + 1$ 的化简也可以在线性时间里完成. 简单使用恒等式 $x^n = x^{n+s-r} + x^{n-r} \bmod T$, $n = 2r-2, 2r-3, \ldots, r$, 用低次项替代 $\geq r$ 次的项.

(5) 大多数涉及多项式的 GCD 计算可以通过 "模块化" 的技术 (如上所描述) 由多项式的乘法来代替.

(6) 大多数多项式的乘积可以被多项式的平方代替, 使用另一级别的 "模块化", 如 [6] 中所描述的.

15. 结论

这 6 种思想的结合使得找到一个次数很大的本原三项式成为可能. 事实上, 目前所记录的次数和已知的最大的 Mersenne 指数 $r = r'_{45} = 43112609$ 一样大. 我们已经准备好去寻找更多的本原三项式, 只要 GIMPS 找到其它的不受 Swan 定理排除的 Mersenne 素数. 我们的任务比 GIMPS 更简单, 因为找到一个次数为 r 的本原三项式, 和验证单个 r 值是一个 Mersenne 指数, 二者成本大约一样: $\widetilde{O}(r^2)$.

三项式的寻找使得在 GF(2) 上的多项式的运算软件得到改善并且验证了理论上最好的算法在实际中并不一定总是最好. 也提供了一个较大的 GF(2) 上三项式因子的数据库, 这导致的几个有趣的猜测是未来研究的课题.

致谢 (略)

参考文献

[1] W. Bosma and J. Cannon, Handbook of Magma Functions, School of Mathematics and Statistics, University of Sydney, 1995. http://magma.maths.usyd.edu.au/.
[2] R. P. Brent, P. Gaudry, E. Thomé, and P. Zimmermann, Faster multiplication in GF(2)[x]., Proc. ANTS VIII 2008, Lecture Notes in Computer Science 5011, 153–166.
[3] R. P. Brent and H. T. Kung, Fast algorithms for manipulating formal power series, J. ACM 25 (1978), 581–595.
[4] R. P. Brent, S. Larvala, and P. Zimmermann, A fast algorithm for testing reducibility of trinomials mod 2 and some new primitive trinomials of degree 3021377, Math. Comp. 72 (2003), 1443–1452.

[5] ——, A primitive trinomial of degree 6972593, Math. Comp. 74 (2005), 1001–1002.
[6] R. P. Brent and P. Zimmermann, A multi-level blocking distinct-degree factorization algorithm, Finite Fields and Applications: Contemporary Mathematics 461 (2008), 47–58.
[7] ——, Ten new primitive binary trinomials, Math. Comp. 78 (2009), 1197–1199.
[8] J. von zur Gathen and J. Gerhard, Modern Computer Algebra, Cambridge Univ. Press, 1999.
[9] J. von zur Gathen and V. Shoup, Computing Frobenius maps and factoring polynomials, Computational Complexity 2 (1992), 187–224.
[10] J. R. Heringa, H. W. J. Blöte, and A. Compagner, New primitive trinomials of Mersenne-exponent degrees for random-number generation, International J. of Modern Physics C 3 (1992), 561–564.
[11] K. Kedlaya and C. Umans, Fast modular composition in any characteristic, Proc. FOCS 2008, 146–155.
[12] T. Kumada, H. Leeb, Y. Kurita, and M. Matsumoto, New primitive t-nomials ($t = 3, 5$) over GF(2) whose degree is a Mersenne exponent, Math. Comp. 69 (2000), 811–814. Corrigenda: ibid 71 (2002), 1337–1338.
[13] T. Nicely, A new error analysis for Brun's constant, Virginia Journal of Science 52 (2001), 45–55.
[14] A.-E. Pellet, Sur la décomposition d'une fonction entière en facteurs irréductibles suivant un module premier p, Comptes Rendus de l'Académie des Sciences Paris 86 (1878), 1071–1072.
[15] J. M. Pollard, A Monte Carlo method for factorization, BIT 15 (1975), 331–334.
[16] A. Schönhage, Schnelle Multiplikation von Polynomen über Körpern der Charakteristik 2, Acta Informatica 7 (1977), 395–398.
[17] V. Shoup, NTL: A library for doing number theory. http://www.shoup.net/ntl/.
[18] L. Stickelberger, Über eine neue Eigenschaft der Diskriminanten algebraischer Zahlkörper, Verhandlungen des ersten Internationalen Mathematiker-Kongresses, Zürich, 1897, 182–193.
[19] R. G. Swan, Factorization of polynomials over finite fields, Pacific J. Math. 12 (1962), 1099–1106.
[20] S. Wagstaff Jr., The Cunningham Project. http://homes.cerias.purdue.edu/∼ssw/cun/.
[21] G. Woltman et al., GIMPS, The Great Internet Mersenne Prime Search, http://www.mersenne.org/.

(韩伟 谌稳固 译 谌稳固 校)

****

(上接 288 页)

[2] A. DasGupta, The matching birthday and the strong birthday problem: a contemporary review, J. Statist. Plann. Inference 130 (2005) 377–389, available at `http://dx.doi.org/10.1016/j.jspi.2003.11.015`.
[3] P. Diaconis, F. Mosteller, Methods for studing coincidences, J. Amer. Statist. Assoc. 17 (1989) 853–861, available at `http://dx.doi.org/10.1080/01621459.1989.10478847`.
[4] R. L. Hocking, N. C. Schwertman, An extension of the birthday problem to exactly k matching, College Math. J. 17 (1986) 315–321, available at `http://dx.doi.org/10.2307/2686280`.
[5] W. Knight, D. M. Bloom, A birthday problem, Amer. Math. Monthly 80 (1973) 1141–1142, available at `http://dx.doi.org/10.2307/2318556`.
[6] E. H. McKinney, Generalized birthday problem, Amer. Math. Monthly 73 (1966) 385–387, available at `http://dx.doi.org/10.2307/2315408`.
[7] J. I. Naus, An extension of the birthday problem, Amer. Statist. 22 (1968) 27–29.

(陆柱家 译 陈凌宇 校)