蒙哥马利归约

sammyne

2020 年 9 月 19 日

1 归约

蒙哥马利归约是一种能够高效地实现模乘而无须显式执行传统的取模归约操作的技巧。

给定 m，R 和 T 为正整数，$R > m$，$gcd(m, R) = 1$, $0 \leq T < mR$。$TR^{-1} \bmod m$ 称为 T 关于 R 模 m 的 *蒙哥马利归约*。选择合适的 R，可以高效地执行蒙哥马利归约。

给定整数 x 和 y $(0 \leq x, y < m)$，令 $\tilde{x} = xR \bmod m$，$\tilde{y} = yR \bmod m$，则有 $\tilde{x}\tilde{y}$ 的蒙哥马利归约为

$$\tilde{x}\tilde{y}R^{-1} \bmod m = xyR \bmod m$$

这个等式可用于实现模幂运算的高效方法。例如，计算 $x^5 \bmod m$ $(1 \leq x < m)$

1. 计算 $\tilde{x} = xR \bmod m$
2. 计算 $\tilde{x}\tilde{x}$ 的蒙哥马利归约 $A = \tilde{x}^2 R^{-1} \bmod m$
3. 计算 A^2 的蒙哥马利归约 $A^2 R^{-1} \bmod m = \tilde{x}^4 R^{-3} \bmod m$
4. 计算 $(A^2 R^{-1} \bmod m)\tilde{x}$ 的蒙哥马利归约

$$(A^2 R^{-1})\tilde{x}R^{-1} \bmod m = \tilde{x}^5 R^{-4} \bmod m = (xR \bmod m)^5 R^{-4} \bmod m = x^5 R \bmod m$$

5. 将上述值乘以 $R^{-1} \bmod m$，并基于 m 归约即可得到 $x^5 \bmod m$

如果 m 看做长度为 n 的基底为 b 的整数，R 通常选为 b^n。$R > m$ 很明显是满足的，但是只有 $gcd(b, m) = 1$ 时才会有 $gcd(R, m) = 1$。因此，R 并不是对于所有模数都有合理的 R。对于特别的模数（例如 RSA 的模数），m 会为奇数，b 为 2 的次幂，这样 $R = b^n$ 就足够了。

事实 1.1 (蒙哥马利归约)**.** *给定整数 m 和 R，$gcd(m, R) = 1$，令 $m' = -m^{-1} \bmod R$，T 为整数，满足 $0 \leq T < mR$。如果 $U = Tm' \bmod R$，则有 $(T + Um)/R$ 为整数，且 $(T + Um)/R \equiv TR^{-1} (\bmod\ m)$。*

证明.

$\because\ T + Um \equiv T\ (\bmod\ m)$

$\therefore\ (T + Um)R^{-1} \equiv TR^{-1}\ (\bmod\ m)$

$\because\ U = Tm' \bmod R$

$\therefore\ \exists k \in \mathbb{Z}, U = Tm' + kR$

$\because\ m' = -m^{-1} \bmod R$

$\therefore\ \exists l \in \mathbb{Z}, m'm = -1 + lR$

$\therefore$

$$\begin{aligned}(T+Um)/R &= (T+(Tm'+kR)m)/R \\ &= (T+Tm'm+kRm)/R \\ &= (T+T(-1+lR)+kRm)/R \\ &= lT+km\end{aligned}$$

□

推论 1.1 (关于事实 1.1 的推论).

1. $T < mR, U < R$
   $\Rightarrow (T+Um)/R < (mR+mR)/R = 2m$
   $\Rightarrow (T+Um)/R = TR^{-1} \bmod m \quad or$
   $(T+Um)/R = TR^{-1} \bmod m + m$
2. 如果所有整数的基底均为 b 且 $R = b^n$，则 $TR^{-1} \bmod m$ 可借助两个多精度乘法（即 $U = T \cdot m'$ 和 $U \cdot m$）和 $T+Um$ 除以 R 的简单右移实现

例 1.1 (蒙哥马利归约). 令 $m = 187$，$R = 190$，则有 $R^{-1} \bmod m = 125$，$m^{-1} \bmod R = 63$，$m' = 127$。

- 如果 $T = 563$，则 $U = Tm' \bmod R = 61 \Rightarrow (T+Um)/R = 63 = TR^{-1} \bmod m$
- 如果 $T = 1125$，则 $U = Tm' \bmod R = 185 \Rightarrow (T+Um)/R = 188 = TR^{-1} \bmod m + m$

算法 1: 蒙哥马利归约

Input: $m = (m_{n-1} \dots m_1 m_0)_b$，其中 $gcd(m, b) = 1$
Input: $R = b^n$
Input: $m' = -m^{-1} \bmod b$
Input: $T = (t_{2n-1} \dots t_1 t_0)_b < mR$
Output: $TR^{-1} \bmod m$

1 $A \leftarrow T$，记 $A = (a_{2n-1} \dots a_1 a_0)_b$
2 **for** $i = 0$ **to** $(n-1)$ **do**
3 $\quad u_i \leftarrow a_i m' \bmod b$
4 $\quad A \leftarrow A + u_i m b^i$（注：这一步会更新 $a_{i+1} \dots a_{2n-1}$）
5 $A \leftarrow A/b^n$
6 **if** $A \geq m$ **then**
7 $\quad A \leftarrow A - m$
8 **return** A

小记 1.1 (关于蒙哥马利归约算法 1).

1. 不同于事实 *1.1*，算法不要求 $m' = -m^{-1} \bmod R$，而是要求 $m' = -m^{-1} \bmod b$，这是由于 $R = b^n$
2. 算法的第 *3* 行，对于 $0 \leq j \leq i-1$ 有 $a_j = 0$。第 *4* 行并没有改变这些值，只是将 a_i 置为 *0*。因此，第 *5* 行有 b^n 整除 A

3. 进入第 5 行，A 的值等于 T 加上 m 的整数倍（由第 4 可得）。此时 $T=(T+km)/b^n$ 为整数，$A\equiv TR^{-1}(\mathrm{mod}\ m)$。剩下只需要证明 $A<2m$ 即可确保第 7 行的一次减法（而不是除法）就够了。推导如下，第 5 行前

$$A=T+\sum_{i=0}^{n-1}u_ib^im\le T+b^nm<Rm+Rm=2Rm$$

执行第 5 行，得 $A\leftarrow A/b^n=A/R<2m$

小记 1.2 (蒙哥马利归约的计算效率)**.** 算法 1 的第 3 和 4 行共需 $n+1$（a_im' 消耗 1 次，u_imb^i 消耗 n 次）次单精度乘法。由于这两步执行 n 次，所以共需单精度乘法次数为 $n(n+1)$。算法 1 不需要任何单精度除法。

例 1.2 (蒙哥马利归约)**.** 给定 $m=72639$，$b=10$，$R=10^5$，和 $T=7118368$，有 $n=5$，$m'=-m^{-1}\bmod 10=1$，$T\bmod m=72385$，$TR^{-1}\bmod m=39796$，具体计算过程如下表 1

i	a_i	$u_i=a_im'\bmod 10$	u_imb^i	A
-	-	-	-	7118368
0	8	8	581112	7699480
1	8	8	5811120	13510600
2	6	6	43583400	57094000
3	4	4	290556000	347650000
4	5	5	3631950000	3979600000

表 1: 蒙哥马利归约算法示例

2 模乘

算法 2 将归约算法 1 和多精度乘法结合，计算两个整数乘积的蒙哥马利归约。

算法 2: 蒙哥马利乘法

Input: $m=(m_{n-1}\dots m_1m_0)_b$，其中 $gcd(m,b)=1$
Input: $x=(x_{n-1}\dots x_1x_0)_b$，且 $0\le x<m$
Input: $y=(y_{n-1}\dots y_1y_0)_b$，且 $0\le y<m$
Input: $m'=-m^{-1}\bmod b$
Output: $xyR^{-1}\bmod m$

1 $A\leftarrow 0$，记 $A=(a_na_{n-1}\dots a_1a_0)_b$
2 **for** $i=0$ **to** $(n-1)$ **do**
3 $\quad u_i\leftarrow(a_0+x_iy_0)m'\bmod b$
4 $\quad A\leftarrow(A+x_iy+u_im)/b$（注：这一步会更新 $a_0\dots a_n$）
5 **if** $A\ge m$ **then**
6 $\quad A\leftarrow A-m$
7 **return** A

小记 2.1 (蒙哥马利乘法 2的部分证明)**.** 假设算法 2 第 2 行的第 i 次循环时，$0\le A<2m-1$，

第 4 行将 A 更新为

$$
\begin{aligned}
&(A + x_i y + u_i m)/b \\
&\le (2m - 2 + (b-1)(m-1) + (b-1)m)/b \\
&= (2m - 2 + (b-1)m - (b-1) + (b-1)m)/b \\
&= (2m - 1 + 2m(b-1) - b)/b \\
&= (2mb - b - 1)/b \\
&= 2m - 1 - 1/b \\
&\le 2m - 1
\end{aligned}
$$

小记 2.2 (蒙哥马利乘法 2的第 4 行所得 A 为整数的证明).

$\because\ m' = -m^{-1} \bmod m$

$\therefore \exists k_1 \in \mathbb{Z}, m' = -m^{-1} + k_1 b$

$\therefore$

$$
\begin{aligned}
u_i &= (a_0 + x_i y_0) m' \bmod b \\
&= (a_0 + x_i y_0)(-m^{-1} + k_1 b) \bmod b \\
&= [-m^{-1}(a_0 + x_i y_0) + k_1 b(a_0 + x_i y_0)] \bmod b \\
&= -m^{-1}(a_0 + x_i y_0) \bmod b \\
&\Rightarrow \exists k_2 \in \mathbb{Z}, u_i = -m^{-1}(a_0 + x_i y_0) + k_2 b
\end{aligned}
$$

$\therefore$

$$
\begin{aligned}
&(A + x_i y + u_i m)/b \\
&= \left(a_0 + \sum_{i=1}^{n-1} a_i b^i + x_i y_0 + \sum_{j=1}^{n-1} x_i y_j b^j + u_i m \right) /b \\
&= \left(a_0 + \sum_{i=1}^{n-1} a_i b^i + x_i y_0 + \sum_{j=1}^{n-1} x_i y_j b^j + m(-m^{-1}(a_0 + x_i y_0) + k_2 b) \right) /b \\
&= \left(a_0 + \sum_{i=1}^{n-1} a_i b^i + x_i y_0 + \sum_{j=1}^{n-1} x_i y_j b^j - (a_0 + x_i y_0) + m k_2 b) \right) /b \\
&= \left(\sum_{i=1}^{n-1} a_i b^i + \sum_{j=1}^{n-1} x_i y_j b^j + m k_2 b \right) /b \\
&= \sum_{i=1}^{n-1} a_i b^{i-1} + \sum_{j=1}^{n-1} x_i y_j b^{j-1} + m k_2 \in \mathbb{Z}
\end{aligned}
$$

以上过程本质上是逐步把 $x_i y \quad (0 \le i < n)$ 逐步加到 A 上，并在每一步加法操作时，执行归约操作

小记 2.3 (蒙哥马利乘法 2 的效率). 因为 $A + x_i y + u_i m$ 是 b 的整数倍，算法的第 4 行执行右移操作即可实现除以 b。第 3 行需要两次单精度乘法，第 4 行则需要 $2n$ 次。由于第 2 行需要执行 n 次，所以共需 $n(2 + 2n) = 2n(n + 1)$ 次单精度乘法。

小记 2.4 (基于蒙哥马利乘法计算 $xy \bmod m$). 假设 x，y 和 m 都是基底为 b 的 n 位整数，且 $0 \le x, y < m$。忽略输入中的预计算的代价，算法 2 计算 $xyR^{-1} \bmod m$ 需要 $2n(n+1)$ 次单精度乘法。忽略计算 $R^2 \bmod m$ 的代价，对 $xyR^{-1} \bmod m$ 和 $R^2 \bmod m$ 应用算法 2，$xy \bmod m = (xyR^{-1} \bmod m)(R^2 \bmod m)R^{-1} \bmod m$ 可在共 $2n(n+1) + 2n(n+1) = 4n(n+1)$ 次单精度乘

法内算得。使用传统的模乘需要 $2n(n+1)$ 次单精度乘法，不需要预计算。因此，传统算法在处理单次模乘时更优。然而，蒙哥马利模乘在执行模幂运算时非常高效（具体算法参见《*Handbook of Applied Cryptography*》一书的算法 *14.94*）。

小记 2.5 (蒙哥马利归约和蒙哥马利乘法)**.** 算法 *2* 接收两个 n 位的整数，然后交叉执行乘法和归约操作。因此，算法 *2* 无法有效利用两个整数相等（例如，求平方）这种特殊情况。而算法 *1*（蒙哥马利归约）假设输入的是两个整数乘积，每个最多 n 位。由于算法 *1* 独立于多精度乘法，在归约之前可以采用更快的平方算法。

例 2.1 (蒙哥马利乘法)**.** 给定 $m = 72639$，$R = 10^5$，$x = 5792$，$y = 1229$。则有 $n = 5$，$m' = -m^{-1} \bmod 10 = 1$，$xyR^{-1} \bmod m = 39796$，具体计算过程如下

i	x_i	$x_i y_0$	u_i	$x_i y$	$u_i m$	A
0	2	18	8	2458	581112	58357
1	9	81	8	11061	581112	65053
2	7	63	6	8603	435834	50949
3	5	45	4	6145	290556	34765
4	0	0	5	0	363195	39796

3 参考文献

- Handbook of Applied Cryptography - 14.3.2 Montgomery reduction