The Montgomery Reduction Algorithm

Rui-Jie Fang

1 Preliminaries

We would like to compute $a \cdot b \mod n$, $a, b, n \in \mathbb{N}$. The standard way of doing so involves a single multiplication to get ab, and then a division operation to find the remainder. Montgomery reduction uses low-bit cancellation to provide a faster alternative.

1.1 Outline

Briefly speaking, Montgomery reduction works as follows. We first convert a, b into Montgomery form in order to multiply. Montgomery form: $a' = aR \mod n$, $b' = bR \mod n$. Then $a'b' = abR^2 \mod n$. Then we do a first Montgomery reduction, converting $a'b'$ into the form $abR \mod n$; then we use a second Montgomery reduction to convert out of Montgomery form, reaching the result $ab \mod n$.

2 Basic Arithmetics on Z/nZ

2.1 The Ring Z/nZ

Let n be our positive modulus. The ring of residue classes modulo $n (Z/nZ)$ has elements which are sets of the form $\{a + k \cdot n | k \in \mathbb{Z}\}$, where $a \in \mathbb{Z}$. A residue class in such a ring is a set S of integers such that $\forall a, b \in S, (a-b)|n$. Such a set is maximal in that no integers that satisfy this property are left out (Remark: It is easy to see why; as $a + k_a \cdot n - (b + k_b \cdot n) = (k_a - k_b)n$ which must divide n).

The residue corresponding to an integer a is denoted as $\overline{a}$. The equality operation of residue classes is congruence; it is denoted as $\overline{a} \equiv \overline{b} \pmod n$. Each residue classes has infinitely many elements (therefore impossible to be stored on a computer). Instead, when computing residue classes we store representatives of modulo classes; by convention the representatives are the integers $0 \leq a \leq n - 1$. Therefore, for any integer a' the representative of $\overline{a}$ is written as $a' \mod n$ which is in $[0, n-1]$. Therefore congruence between class representatives can be written as $a \equiv b \pmod n$.

2.2 Arithmetic on Z/nZ

Arithmetic on the residue classes of ring Z/nZ is done by performing interger arithmetic on the class representatives. Each arithmetic operation is done in two steps:

1. Compute the arithmetic result using an integer operator, which maps to third positive integer $\mod n$, thereby belonging to one of the residue classes.

2. Compute the result $\mod n$ using representative of that class in range $[0, n-1]$, thereby determining the output of the modular operation. More specifically, let $c \mod n = a * b \mod n$, where $*$ is our integer operation. Then we may find the output of the modular operation as a representative $k \in [0, n-1]$, $(c-k)|n$ (this is just standard division found via a search on the representatives; specifically, $c = qn + k$).

We illustrate the principle through an example using the addition operation. We want to compute the sum of residue classes $\overline{8}$ and $\overline{25}$ with $n = 29$. First, we compute $c = 8 + 25 = 33$. We then attempt to find $k \in [0, 28]$ such that $33 - k$ divides n. We find that $k = 4$, and that $33 - 4 = 29; 29/29 = 1$. Therefore $8 + 25(\mod 29) = 4$, with k being our answer.

2.3 Determining representatives

For different arithmetic operations such as addition, division, multiplication, etc. the number of operations we have to go through before determining the representative (and therefore the output of the modular operation) is different. For simple addition of two residue classes $a, b \in [0, n-1]$ in Z/nZ, the output c is always in the range $[0, 2n-2]$; hence, only a single subtraction is needed to determine the representative of c. For subtraction, we have $a, b \in [0, n-1]$ and $c \in [-n+1, n-1]$. By performing a single addition (for $c < 0$) we may determine the representative.

We now analyze the product $a \cdot b$ with $a, b \in [0, n-1]$. Then $c \in [0, n^2 - 2n + 1]$. In terms of space complexity, the multiplication operation leads c to have twice the number of bits than a, b. Proceeding in the regular fashion, To determine a representative k in $[0, n-1]$ for such a c, we need division:

$$k = c \mod n$$

$$k = ab \mod n$$

In other words,

$$ab = q \cdot n + k$$

$$k = ab - qn$$

(By standard Euclidean division). Note that the quotient $q = \text{floor}(ab/n)$ by standard machine integer division convension, and $k \in [0, n-1]$. The dilemma of implementation here is the determination of $ab \mod n$ requires a division, which is costly in terms of speed.

Example 1 (Multiplication on Z/nZ.)**.** We want to compute $a \cdot b \mod n$, $a = 7$, $b = 8$, $n = 10$. Then $c = ab = 7 \cdot 8 = 56$; then we find $q = 5, k = 6$ such that $c = qn + k = 5 \cdot 10 + 6 = 56$. Therefore we determined that $k = 6$.

From the above example we can see that by choosing a "good" n, i.e. $n = 10$, it is very easy for humans to compute the result of the division. The idea of Montgomery Reduction stems from exactly this: If we have a difficult n such that we *have* to use Euclidean division, can we find another base R that is easier for *machines* to work with (i.e. $r = 2$)?

2.4 A faster alternative for Euclidean division

On computers where everything is implemented as base 2, binary divisions and multiplications can be done as left shifts and right shifts, and are in general, very fast (~ 0.1cycle/ops). A Montgomery form is a different way of expressing elements of Z/nZ such that no expensive divisions are required; and when divisions are indeed required, they are expressed in a convenient base R, which can be a power of two, enabling fast computation via left shifts. This improvement makes the speed of computing $ab \mod n$ roughly equal to the speed of multiplication.

3 The Montgomery Reduction

3.1 Auxiliary modulus and Montgomry form

We start by determining R, our auxiliary modulus, the divisior which we would like our future divisions to be based upon. R is a positive integer and is coprime to n, making $\gcd(n, R) = 1$. We also maintain that R must be a number in which it is less expensive to perform divisions (i.e. power of 2) and that $R > n$ because it would not help with determining representatives otherwise. For a residue class $\overline{a}$, its associated Montgomery form with respect to auxilliary modulus R is the representative of the residue class $\overline{aR}$, which is $aR \mod n$.

Example 2 (Montgomery form of residue classes.)**.** Let $a = 17$, $n = 21$, and $R = 32$. Then the representative of $\overline{a}$ is 17, and the Montgomery form of $\overline{a}$ is the representative of $\overline{aR}$, which is $aR \mod n = 17 \cdot 32 \mod 21 = 544 \mod 20 = 19$.

3.2 Operations under Montgomery form

Additions and subtractions under Montgomery form are distributive:

Algorithm 3. *The Extended Euclidean Algorithm.*

`Input:` $a, b \in \mathbb{N}$;
`Output:` r, s, t `such that` $m = \gcd(a, b)$, $as + bt = r$.

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} m & n \\ 1 & 0 \\ 0 & 1 \end{pmatrix};$$

`while` $r_1 \neq 0$:

$q \coloneqq \texttt{floor}(r_0/r_1)$;

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} r_1 & r_0 - qr_1 \\ s_1 & s_0 - qs_1 \\ t_1 & t_0 - qt_1 \end{pmatrix};$$

`return` $\begin{pmatrix} r_0 & s_0 & t_0 \end{pmatrix}$;

Example 4 (Computing the Bezout coefficients through $\gcd(240, 46)$.)**.**

$i = 0$, $r = 240$, $s = 1$, $t = 0$ $(240 = 1 \cdot 240 + 0)$
$i = 1$, $r = 46$, $s = 0$, $t = 1$ $(46 = 1 \cdot 46 + 0)$
$i = 2$, $r = 240 - 5 \cdot 46 = 10$, $s = 1 - 5 \cdot 0 = 1$, $t = 0 - 5 \cdot 1 = -5$.
$i = 3$, $r = 46 - 4 \cdot 10 = 6$, $s = 0 - 4 \cdot 1 = -4$, $t = 1 - (4 \cdot -5) = 21$.
$i = 4$, $r = 10 - 1 \cdot 6 = 4$, $s = 1 - (-4 \cdot 1) = 5$, $t = -5 - 1 \cdot 21 = -26$.
$i = 5$, $r = 6 - 1 \cdot 4 = 2$, $s = -4 - 5 \cdot 1 = -9$, $t = 21 - (-26 \cdot 1) = 47$.
$i = 6$, $r = 4 - 2 \cdot 2 = 0$, $s = 5 - (-9) \cdot 2 = 23$, $t = -26 - 47 \cdot 2 = -120$.

Conclusion. At the second to last step we have $s_k a + t_k b = -9 \cdot 240 + 47 \cdot 46 = 2 = \gcd(240, 46)$.
At the last step we have $240 \cdot 23 - 46 \cdot 120 = 0$.

Figure 1: Recap for the Extended Euclidean Algorithm.

$$aR + bR = (a + b)R$$

$$aR - bR = (a - b)R$$

Multiplication brings along the problem of an extra factor:

$$aR \mod n \cdot bR \mod n = (abR)R \mod n$$

Therefore we need to remove the extra factor of R in order to compute the the product correctly.

3.3 Modular multiplication by extended gcd

To remove the extra factor of R in multiplication, we can choose to multiply $abR^2 \mod n$ by R', the modular inverse of R: $RR' \equiv 1(\mod n)$. Then we have the following:

$$aR \mod n \cdot bR \mod n \cdot R' = (abR^2 \mod n)R' = abR \mod n$$

We have $RR' \equiv 1(\mod n)$ and R' exactly because $\gcd(n, R) = 1$. Then by using extended gcd and Bezout's Identity we can determine R' and n' s of the form:

$$RR' - nn' = 1$$

Note that we need to apply this transformation again by R' to go out of the Montgomery form.

Example 5 (Modular Multiplication under Montgomery form.)**.** Following example 2, choose $a = 17$, $b = 5$, $n = 21$, and $R = 32$. Compute $aR \mod n = 17 \cdot 32 \mod 21 = 19$, $bR \mod n = 5 \cdot 32 \mod 20 = 13$. Then $(aR \mod n)(bR \mod n) = 19 \cdot 13 = 247$ (This is $abR^2 \mod n$). By extended Euclidean algorithm we find $R' = 2, n' = -3$ such that $RR' - nn' = 32 \cdot 2 - 21 \cdot 3 = 1$ (See figure 2). Then we compute $(abR^2 \mod n) \cdot R' = 247 \cdot 2 \mod 21 = 494 \mod 21 = 11$. To verify, we compute $abR \mod n = 17 \cdot 5 \cdot 32 \mod 21 = 11 = abR^2R' \mod n$.

This is, however, one (slower) way to do multiplication under Montgomery form because of the amount of computation involved. To do faster modular multiplication we must use a technique called Montgomery Reduction.

3.4 Modular multiplication via Montgomery reduction

Montgomery Reduction computes $(abR^2) \cdot R' \mod n$ quickly; more specifically, it finds R' and reduces to modulo n in time less than the cost of division. The procedure `REDC` below is the procedure for Montgomery Reduction.

Algorithm 6 (Montgomery Reduction)**.**

`Input:` R, n, n', T

`Requires:` $R > n \wedge \gcd(R, n) = 1$, $n' \in [0, R-1] \wedge nn' \equiv -1 \mod R$, $T = abR^2 \mod n$.

`Output:` $TR^{-1} \mod n$.

$m \coloneqq (T \mod R) \cdot n' \mod R$;
$t \coloneqq (T + m \cdot n)/R$;
`if` $t \geq n$:
    `return` $t - n$;
`else:`

Input: $a = 32, b = 21$.
Step 0. (Initialize)

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} a = 32 & b = 21 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}; \, q = 0$$

Step 1. (Verify: $as_0 + bt_0 = 32 \cdot 1 + 21 \cdot 0 = 32 = r_0$)

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} 21 & 32 - 1 \cdot 21 = 11 \\ 0 & 1 - 1 \cdot 0 = 1 \\ 1 & 0 - 1 \cdot 1 = -1 \end{pmatrix}; q = 1$$

Step 2. (Verify: $as_0 + bt_0 = 32 \cdot 0 + 21 \cdot 1 = 21 = r_0$)

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} 11 & 21 - 11 \cdot 1 = 10 \\ 1 & 0 - 1 \cdot 1 = -1 \\ -1 & 1 - (-1 \cdot 1) = 2 \end{pmatrix}; q = 1$$

Step 3. (Verify: $as_0 + bt_0 = 32 \cdot 1 + 21(-1) = 11 = r_0$)

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} 10 & 11 - 10 = 1 \\ -1 & 1 - (-1 \cdot 1) = 2 \\ 2 & -1 - (2 \cdot 1) = -3 \end{pmatrix}; q = 1$$

Step 4. (Verify: $as_0 + bt_0 = 32(-1) + 21 \cdot 2 = 10 = r_0$)

$$\begin{pmatrix} r_0 & r_1 \\ s_0 & s_1 \\ t_0 & t_1 \end{pmatrix} \coloneqq \begin{pmatrix} 1 & 10 - 10 \cdot 1 = 0 \\ 2 & -1 - (2 \cdot 10) = -21 \\ -3 & 2 - (-3 \cdot 10) = 32 \end{pmatrix}; q = 10$$

Step 5. (Output) $r_0 = 1$, $s_0 = 2$, $t_0 = -3$. (Verify: $as_0 + bt_0 = 2 \cdot 32 - 3 \cdot 21 = 1 = r_0$)

Figure 2: The Extended Euclidean Algorithm's quotient sequence for Example 5.

```
        return t;
```

TODO: Proof of correctness, explanation, and example. TODO: Discuss time complexity.