B. Basic arithmetic

Multiplication of integers by computers used to be slower than addition by factor of 10 and more for many decades after the first computers have been constructed. Nowadays the speed of addition and multiplication does not much differ when performed in the length of computer word. This is not because the complexity of multiplication has diminished, but because this complexity has been transformed to hardware. In terms of tacts of the processor the multiplication needs two or three tacts while the addition only one tact. However, due to pipelining the actually observed behaviour may give an impression that the speed of multiplication is nearly the same as the speed of addition. The speed-up of division does not seem to have kept pace with the speed-up of multiplication. In microprocessors the divison takes 10 times more time than the multiplication, while the ratio in historical mainframes used to be around 3.

Computations with very long integers rely upon software packages of *multiple-precision arithmetic*. Since this arithmetic is realized by software and not by hardware, there is no decline in importance of replacing multiplication by addition whenever possible, and replacing division by multiplication whenever possible.

The division occurs naturally when working modulo p, p a large prime. The naive algorithm of computing x times y modulo p goes by performing first the multiplication of integers, and then taking the remainder of division by p.

The *Montgomery arithmetic* described below replaces the division by p by multiplications. The key concept is to replace each $x \bmod p$ by $xR \bmod p$, where R is an integer of special properties. Leaving implementation details aside, consider the situation when each $x \in \mathbb{Z}_p$ is represented in the memory of the computer by $X \equiv xR \bmod p$. To represent $z \equiv x + y \bmod p$ an algorithm is needed that derives $Z \equiv zR \bmod p$ from $X \equiv xR \bmod p$ and $Y \equiv yR \bmod p$. That is trivial since $Z \equiv X + Y \bmod p$ as $xR + yR = (x + y)R$.

What about $xy \bmod p$? Multiplying xR and yR modulo p yields ZR, where $Z \equiv (xy)R \bmod p$. Finding an efficient method that derives Z from ZR hence results into finding a way how to multiply efficiently modulo p, circumventing thus the division by p.

The goal hence is to devise an algorithm that transforms an integer, say x, that corresponds to ZR to an integer, say y, that corresponds to Z. The input restriction is $0 \le x < pR$. The output requires $0 \le y < p$ and $x \equiv yR \bmod p$, i.e. $y = xR^{-1} \pmod p$. Such a transformation is known as *Montgomery reduction*.

Of course, an efficient Montgomery reduction is conceivable only under some external assumption. The assumption here comes from the reality of computers. The division by R requires much less resources if R is a power of two or, even better, if $R = b^t$, where b is the extent of the computer word ($b = 2^{32}$ or 2^{64} etc.).

It will be thus assumed that $R = b^t > p$ and that the division by b (and thus also by R) is 'cheap'. No other external assumption is being made. The integer b is considered as a *basis* and integers $< p$ are represented as $(a_{t-1}, \ldots, a_1, a_0)_b = \sum a_i b^i$. Examples that rely on the pen and mental arithmetic may have $b = 10$ or $b = 100$ etc.

The idea of Montgomery reduction is as follows: The residue class modulo p does not change if x is replaced by $x + xpq$. Choose q so that $pq \equiv -1 \bmod R$. That makes $x + xpq$ divisible by R. Change now $x + xpq$ in such a way that xq is replaced by $u = xq \pmod R$. This affects neither the residue class nor the divisibility by R. Hence $y = (x + up)/R$ is an integer, and $yR \equiv x \bmod p$. While there does not have to be $y < p$, there has to be $y < 2p$ if $x < pR$ is assumed. This is because $u < R$ and because $y < 2p$ may be expressed as $2pR > yR = x + up$.

The preceding observation will be recorded as a statement:

Lemma B.1. *Let $R > 1$ be an integer, and let $p, q \in \mathbb{Z}_R$ be such that $pq \equiv -1 \bmod R$, i.e. $q = -p^{-1} \pmod R$. Let x be an integer such that $0 \le x < pR$. Put $u = xq \pmod R$. Then $R \mid up+x$, and $y = (up+x)/R$ fulfils both $y < 2p$ and $yR \equiv x \bmod p$.*

Proof. Indeed, $up+x \equiv xpq + x \equiv 0 \bmod R$, and $yR = up+x \equiv x \bmod p$. Furthermore, $yR = pu + x < pR + pR = 2pR$. □

B.1. **Montgomery arithmetic.** Consider an algorithm that performs some task in the arithmetic modulo p, with inputs $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$. To implement the algorithm by means of Montgomery arithmetic requires to determine $R = b^t > p$ and $q = -p^{-1} \pmod R$ in advance, and then, whenever the procedure is invoked, to convert the inputs a_i to $A_i \equiv a_i R \bmod p$, to perform all arithmetical operations of the procedure in this representation, and finally to convert each $B_j \equiv b_j R \bmod p$ to b_j at the time of output.

The Montgomery reduction $x \to y$, where $0 \le x < pR$, $0 \le y < p$ and $x \equiv yR \bmod p$, may be executed as suggested by Lemma B.1. That means to multiply x and q, and reduce it modulo R. The computations are exercised in the basis b. The reduction modulo R thus means to take the last t positions (i.e., the last t computer words) of the product. This is denoted by u. The output is equal to $x = (x + up)/R$ if $x < p$. If $x \ge p$, then the output is equal to $x - p$.

The disadvantage of this approach is that it requires two long multiplications (of x with q, and of u with p). A more efficient solution reduces this to a linear number of multiplications of a long integer with an integer in the size of the computer word (i.e., $< b$). It turns out that knowledge of q is not necessary. It suffices to know $q' = -p^{-1} \pmod b$.

Suppose that $x = \sum x_i b^i$, $0 \le x_i < b$. Let x be divisible by b^r, $r \ge 0$. Thus $b_0 = \cdots = b_{r-1} = 0$. Set $u = x_r q' \pmod b$ and $x' = x + upb^r$. Counting modulo b^{r+1} shows that $x' \equiv x_r b^r + x_r pq' b^r \equiv x_r b^r + x_r(-1)b^r \equiv 0$. Hence b^{r+1} divides x', and $x' - x < pb^{r+1}$ is a multiple of pb^r. Proceeding inductively from $r = 0$ increases x in k steps by an integer vp, where $0 \le v < b^k$. The Montgomery reduction can be thus performed as follows:

```
INPUT: x = Σ_i x_i b^i ≤ pR, 0 ≤ i ≤ 2t − 1, 0 ≤ x_i < b.
OUTPUT: An integer y with 0 ≤ y < p and yR ≡ x mod R.
PARAMETERS: p, b, t, R, where R = b^t > p,
            q', where q'p ≡ −1 mod b and 0 < q' < p.
VARIABLES: i, u, 0 < u < p.
i=0;
while (i < t) do:
      u = x_i q' (mod b);
      x = x + pub^i;
      i = i + 1;
y = x/R;
if (y > p) then y = y − p;
return y.
```

Each multiplication in Montgomery arithmetics ends by the reduction. The efficiency may be raised by integrating both of these steps in an ensuing algorithm. The justification follows the description. Parameters and variable are the same as in the preceding algorithm.

```
INPUT: x = Σ_i x_i b^i < p, y = Σ_i y_i b^i < p.
OUTPUT: An integer z = Σ_i z_i b^i < p such that zR ≡ xy mod R.
```

```
i=0;
z=0;
while (i < t) do:
      u = (z_0 + x_i y_0) q' (mod b);
      z = (z + x_i y + p u)/b;
      i = i + 1;
if (z > p) then z = z - p;
return z.
```

To justify the division by b note that while counting modulo b

$$z + x_i y + pu \equiv z_0 + x_i y_0 + pu \equiv z_0 + x_i y_0 + pq'(z_0 + x_i y_0) \equiv 0$$

since $pq' \equiv -1 \bmod b$. To see that the procedure does what declared denote by $\bar{z}_i$ the value of z after the ith round. Thus $\bar{z}_t$ is equal to the output from the cycle. Put $\bar{x}_i = \sum_{j<i} x_j b^j$ and note that $\bar{x}_t = x$. The claim to verify is that there exists integer v_i such that

$$0 \le \bar{z}_i b^i - \bar{x}_i y = p v_i \text{ and } v_i < b^i.$$

In the first step $u = \bar{x}_1 y_0 q' \pmod b$ since $\bar{x}_1 = x_0$ and $\bar{z}_1 b - \bar{x}_1 y = pu$. The condition thus holds for $i = 1$. For the induction step first observe that $\bar{z}_{i+1} b^{i+1} = \bar{z}_i b^i + x_i y b^i + pub^i$ and $\bar{x}_{i+1} y = x_i y b^i + \bar{x} + iy$. By the induction assumption

$$\bar{z}_{i+1} b^{i+1} - \bar{x}_{i+1} y = \bar{z}_i b^i - \bar{x}_i y + pub^i = p(v_i + ub^i).$$

Therefore $v_{i+1} = v_i + ub^i < b^{i+1}$. By the final step, $0 \le zR - xy = pv_t < pR$. Thus $zR \equiv xy \bmod p$ and $zR < pR + p^2 < 2pR$.

Recall that in Montgomery arithmetic the procedure above is invoked with inputs $X \equiv xR \bmod p$ and $Y \equiv yR \bmod p$, and the output is equal to $Z \equiv xyR \bmod p$. In each step there are two multiplications of the form (long integer) $\times$ (computer word), and there is no multiplication of two long integers.

Note that whenever Montgomery arithmetic is applied, there is an initial cost of multiplying the inputs by R modulo p.

The final remark concerning the Montgomery arithmetic is about the computation of q'. The question thus is how to compute $p^{-1} \pmod b$ efficiently. In general the inverses may be computed by means of extended Euclidean algorithm. However, if $b = 2^w$, then there exists a more efficient procedure:

```
INPUT: An odd integer x, 0 < x < 2^w.
OUTPUT: Integer y such that yx ≡ 1 mod 2^w, 0 < y < 2^w.
PARAMETER: Integer w ≥ 1.
VARIABLES: Integers i, j, u.
y = 1;
i = 1;
while (i < w) do:
     j = i + 1;
     u = xy (mod 2^j);
     if (2^i < u) then y = y + 2^i;
     i = j;
return y.
```

To prove the correctness denote by y_i the value of y at the end of the ith round and set $y_0 = 1$. Thus $y = y_{w-1}$. The algorithm clearly implies that $y_i < 2^{i+1}$. It thus suffices to verify that $xy_i \equiv 1 \bmod 2^{i+1}$. For $i = 0$ this is true because x is odd.

Suppose that $i \geq 1$. By the induction assumption $xy_{i-1} \equiv 1 \bmod 2^i$. Hence $xy_{i-1} \pmod{2^{i+1}}$ is equal to 1 or to $1 + 2^i$. In the former case $y_i = y_{i-1}$. In the latter case $y_i = y_{i-1} + 2^i$ and $xy_i \equiv xy_{i-1} + 2^i \equiv 1 \bmod 2^{i+1}$.