Discussion on modular exponentiation

V. Barbera

Abstract
This paper presents an extension of the left-to-right binary method to perform modular exponentiation 2^c (mod m) by representing the exponent not in binary notation but in base 2^b.

Modular exponentiation

Modular exponentiation[1] is the remainder $r=a^c (mod\, m)$ when an integer a is raised to the power c and divided by a positive integer m .

One method to perform modular exponentiation is left-to-right binary method[1]:

the exponent c must be converted to binary notation

$$c=\sum_{i=0}^{n-1} c_i 2^i$$

then

$$a^c=a^{\sum_{i=0}^{n-1} c_i 2^i}=\prod_{i=0}^{n-1} a^{c_i 2^i}$$

the remainder is:

$$r=\prod_{i=0}^{n-1} a^{c_i 2^i} (mod\, m)$$

Algorithm of conversion to binary notation:

Inputs An integer c

Outputs The vector C with the coefficients c_i of binary notation of c

1. $i \leftarrow 0$
2. While $c>0$ do
    1. $C[i] \leftarrow c\, mod\, 2$
    2. $c \leftarrow \lfloor c/2 \rfloor$
    3. $i \leftarrow i+1$
3. Output C

Algorithm left-to-right modular exponentiation:

Inputs An integer a , integer c , vector C with $C[i]=c_i$ the coefficients of binary notation of c , and a positive integer m

Outputs $r=a^c(mod\, m)$

1. $r \leftarrow 1$
2. for $i \leftarrow n-1=\lfloor \log_2 c \rfloor$ to 0 do
    1. $r \leftarrow r^2 mod\, m$
    2. if $C[i]\neq 0$ then $r \leftarrow (r \cdot a) mod\, m$
3. Output r

In the case $a=2$ we use a base 2^b notation for c

$$c=\sum_{i=0}^{n-1} c_i (2^b)^i$$

then

$$2^c=2^{\sum_{i=0}^{n-1} c_i (2^b)^i}=\prod_{i=0}^{n-1} 2^{c_i (2^b)^i}$$

the remainder is:

$$r=\prod_{i=0}^{n-1} 2^{c_i (2^b)^i}(mod\, m)=\prod_{i=0}^{n-1} (2^{c_i})^{2^{ib}}(mod\, m)$$

Algorithm of conversion to base 2^b notation:

Inputs An integer c

Outputs The vector C with the coefficients of base 2^b notation of c

1. $i \leftarrow 0$
2. While $c>0$ do
    1. $C[i] \leftarrow c\, mod\, 2^b$
    2. $c \leftarrow \lfloor c/2^b \rfloor$
    3. $i \leftarrow i+1$
3. Output C

Algorithm left-to-right modular exponentiation:

Inputs An positive integer b, vector C with $C[i]=c_i$ the coefficients of base 2^b notation of c, and a positive integer m

Outputs $r=2^c (mod\, m)$

1. $i \leftarrow n=length(C)$
2. $r \leftarrow 2^{C[i-1]} mod\, m$
3. while $i>1$ do
   1. $i \leftarrow i-1$
   2. for $j \leftarrow 1$ to b do
      1. $r \leftarrow r^2\, mod\, m$
   3. $r \leftarrow (r \cdot 2^{C[i-1]})\, mod\, m$
4. Output r

Indeed at the start

$$r=2^{c_{n-1}} mod\, m$$

after step 3.2

$$r=(2^{c_{n-1}})^{2^b} mod\, m$$

after step 3.3

$$r=2^{c_{n-2}} \cdot (2^{c_{n-1}})^{2^b} mod\, m$$

after step 3.2

$$r=(2^{c_{n-2}})^{2^b} \cdot (2^{c_{n-1}})^{2^{2 \cdot b}} mod\, m$$

after step 3.3

$$r=2^{c_{n-3}} \cdot (2^{c_{n-2}})^{2^b} \cdot (2^{c_{n-1}})^{2^{2 \cdot b}} mod\, m$$

…

after step 3

$$r=\prod_{i=0}^{n-1} (2^{c_i})^{2^{i \cdot b}} (mod\, m)$$

Note that if $b=1$ the algorithm is the same as the previous one.

Example of implementation in C++ with GMP library

```
#include <iostream>
#include <cmath>
#include <vector>
#include <cstdlib>
#include <gmp.h>
void mod2pow(std::vector<unsigned> &C, unsigned b, mpz_t &m, mpz_t &r){
    // get r = 2^c (mod m) with C vector coefficient of  base 2^b notation of c
    long long len_C = C.size();
    mpz_set_ui(r, 1);
    mpz_mul_2exp(r, r, C[len_C - 1]);
    if (len_C > 1){
        while (len_C > 1){
            len_C--;
            for (unsigned j = 1; j < b; j++) {
                mpz_mul(r, r, r);
                mpz_fdiv_r(r, r, m);
            }
            mpz_mul(r, r, r);
            mpz_mul_2exp(r, r, C[len_C - 1]);
            mpz_fdiv_r(r, r, m);
        }
    }
    else
        mpz_fdiv_r(r, r, m);
}
int main(){
    unsigned long long c = 3007;
    mpz_t r, m;
    mpz_init_set_ui(m, 101);
    mpz_init(r);
    const unsigned b = 6;
    std::cout << "2^" << c << " (mod " << mpz_get_str(NULL, 10, m) << ") = ";
    //conversion to 2^b notation
    std::vector<unsigned> C;
    if (c >= (1ull << b)) {
        unsigned base_m1 = (1 << b) - 1;
        while (c > 0) {
            C.push_back(c & base_m1);
            c >>= b;
        }
    }
    else
        C.push_back(c);
    mod2pow(C, b, m, r);
    std::cout << mpz_get_str(NULL, 10, r);
    mpz_clear(m);
    mpz_clear(r);
    return 0;
}
```

References

[1] https://en.wikipedia.org/wiki/Modular_exponentiation