Available online at www.sciencedirect.com
ScienceDirect
Information Processing Letters 106 (2008) 232–237
Information Processing Letters
www.elsevier.com/locate/ipl

Should one always use repeated squaring for modular exponentiation?

Shmuel T. Klein *

Department of Computer Science, Bar Ilan University, Ramat-Gan 52900, Israel

Received 28 July 2006; received in revised form 31 August 2007; accepted 25 November 2007

Available online 21 February 2008

Communicated by A. Moffat

Abstract

Modular exponentiation is a frequent task, in particular for many cryptographic applications. To accelerate modular exponentiation for very large integers one may use repeated squaring, which is based on representing the exponent in the standard binary numeration system. We show here that for certain applications, replacing the standard system by one based on Fibonacci numbers may yield a new line of time/space tradeoffs.

Keywords: Design of algorithms; Modular exponentiation; Fibonacci number system; Cryptography

1. Introduction

Modular exponentiation is defined as the task of raising a number a to a power m and considering the result modulo some integer N. This is a frequent and time consuming operation, and has many applications, in particular in cryptography. In a typical setting, a, m and N are large integers, say of the order of 2^{1024}, so it is not feasible to calculate $a^m \bmod N$ by using $m-1$ multiplications, each followed by a modulo operation. The standard solution to this problem is using *repeated squaring* and appears in many handbooks on algorithms, such as [4,2,8] to cite just a few.

To improve readability, we shall not always explicitly mention that the multiplications are to be taken modulo N, which is fixed throughout the paper. Note that instead of calculating

$$a^8 = \underbrace{a \times a \times a \times \cdots \times a}_{8 \text{ factors}},$$

the number of multiplications can be reduced by repeatedly squaring the results:

$$a^8 = \big((a^2)^2\big)^2.$$

If m is not a power of two, it can be expressed as a sum of such powers, giving, for example, $a^{12} = a^8 \times a^4$. For the general case, consider the standard binary representation of m as a sum of powers of 2, that is $m = \sum_{i=0}^{\lfloor \log_2 m \rfloor} b_i 2^i$, where each $b_i \in \{0, 1\}$. Then

$$a^m = a^{b_0} \times a^{2b_1} \times a^{4b_2} \times \cdots \times a^{2^i b_i} \times \cdots.$$

The procedure is thus as follows: prepare a list of basis items $a, a^2, \ldots, a^{2^i}, \ldots$ where each element is obtained

* Tel.: (972 3) 531 8865; fax: (972 3) 736 0498.
E-mail address: tomi@cs.biu.ac.il.

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2007.11.016

by taking its predecessor in the list, squaring it, and reducing the result modulo N; then take the subset of this list corresponding to the 1-bits in the binary representation of m and multiply the elements of this subset. Denoting the number of 1-bits in the binary representation of m by $h(m)$, the number of multiplications is thus $\lfloor \log_2 m \rfloor + h(m) - 1$.

This is not necessarily the minimum number of required multiplications. For example, for $m = 15$, $\lfloor \log_2 m \rfloor + h(m) - 1 = 6$, but a^{15} can be evaluated in 5 operations, calculating first $d = a^5 = (a^2)^2 \times a$ in 3 multiplications, and then $a^{15} = d^3 = d^2 \times d$ in two more multiplications; see Knuth [10, Section 4.6.3] for an investigation of the function $l(m)$, giving the smallest number of multiplications necessary to calculate a^m. We are, however, not interested in finding the minimum number for each given exponent m, but are rather looking for a general algorithm, giving good average performance when applied with a large number of possible values m. Repeated squaring is one such algorithm, and the point of this work is to show that for certain applications, a different general evaluation procedure might be preferable.

2. Alternatives to repeated squaring

2.1. Standard k-ary number system

As mentioned above, the standard evaluation algorithm is based on representing the exponent m in the standard binary number system as $m = \sum_{i=0}^{\lfloor \log_2 m \rfloor} b_i 2^i$, with $b_i \in \{0, 1\}$, yielding $\lfloor \log_2 m \rfloor + h(m) - 1$ multiplications. The first term can be reduced to $\lfloor \log_k m \rfloor$ for $k > 2$, if one uses the standard k-ary number system in which one can represent m as $m = \sum_{i=0}^{\lfloor \log_k m \rfloor} c_i k^i$, with $c_i \in \{0, 1, \ldots, k-1\}$. However, the basis elements $a^{k^{i+1}}$ cannot be anymore evaluated by a single multiplication from preceding basis elements:

$$a^{k^{i+1}} = a^{k^i \cdot k} = (a^{k^i})^k,$$

so for $k = 2$, only one multiplication is needed (this is squaring), but for $k = 3$, one needs two multiplication, for $k = 4$ also two multiplications are sufficient (squaring twice), etc.

Table 1 presents the relevant data for $k = 2, \ldots, 9$. The second column gives the number $l(k)$ of required operations for the calculation of each basis element. The number of necessary basis elements is the number of k-ary digits, which is $\log_k N$, where N is the modulus mentioned above, and this is normalized in the third column, which gives the number of basis elements in units of $\log_2 N$. The fourth column is the total number of multiplications needed for all the basis elements, again in units of $\log_2 N$.

After having evaluated the basis elements, the members of a selected subset of them have to be multiplied. If $m = \sum_{i=0}^{\lfloor \log_k m \rfloor} c_i k^i$, the basis element a^{k^i} is raised to power c_i in $l(c_i)$ multiplications if $c_i > 0$, and one more multiplication is needed per basis element with $c_i > 0$ to get the final product. Assuming that the exponent m is chosen at random, each of the digits $0, 1, \ldots, k-1$ appears in each position with probability $1/k$. The expected number of multiplications for a given digit is thus $\frac{1}{k} \sum_{i=1}^{k-1} (l(i) + 1)$, and these values appear in the fifth column of Table 1. The sixth column is then the average number of multiplications taking all the digits into account, and the seventh column, headed total # mult, is the total sum of operations for both the basis elements and the product of the elements of the subset, again in units of $\log_2 N$. One sees that though interestingly, the total number of multiplications is not monotonically increasing with the order k of the number system, the minimum is still reached for the binary case $k = 2$, so among these alternatives, binary squaring still seems to be the best choice.

There are nevertheless options to reduce the number of required multiplication [9]. The so-called *k-ary method* uses precomputed values $a^2, a^3, \ldots, a^{k-1}$ and Horner's rule as follows. Setting $r(k) = \lfloor \log_k m \rfloor$, the equation for the representation of m in basis k can be rewritten as

$$m = \sum_{i=0}^{r(k)} c_i k^i = c_0 + k\big(c_1 + k\big(c_2 + \cdots + k(c_{r(k)-1} + kc_{r(k)}) \cdots\big)\big).$$

This suggests that a^m can be evaluated iteratively as follows, working from the innermost parentheses outwards:

```
precompute A[j] ← a^j for j = 1, 2, ..., k − 1
R ← 1
for i ← r(k) to 0 by −1
    R ← R^k
    if c_i > 0 then R ← R × A[c_i]
```

The expected number of multiplications in each iteration for a randomly chosen m is $l(k) + (k-1)/k$, as $l(k)$ operations are necessary to raise R to the kth power and one more to multiply with $A[c_i]$, but c_i will be zero with probability $1/k$. The last column of Table 1 displays the total number of required multiplications in units of $\log_2 N$. As can be seen, it is possible to get better values than the 1.5 needed for $k = 2$, in particular for