Publ. Math. Debrecen
79/3-4 (2011), 469–478
DOI: 10.5486/PMD.2011.5164

A note on speeding up exponentiation by precomputation

By TAMÁS HERENDI (Debrecen)

Abstract. We analyse the lower bounds for the number of exponentiation necessary to compute the powers of a fixed member of an Abelian group and give a method for a reasonably fast powering by storing some previously computed powers.

1. Introduction

In many numerical application exponentiation appears as an essential subtask. In particular, in cryptography several practical methods are based on the computation of high powers of some element of a finite group. The most well known are the RSA, El Gamal and elliptic curve cryptosystems.

The aim of the present paper is to calculate some time and space bounds for the computation of powers, basically in finite structures, such as finite fields or elliptic curves.

The work was initiated by Attila Pethő and the results have direct applications in the research and developments done by his cryptography research team.

The subject is excessively studied by many authors. One can find surveys e.g. in [6] Ch. 4.6.3 and [7] Ch. 14.6.

Let $(A, \cdot)$ be a finite Abelian group, let a be a not necessarily generator element of A and let $x \in \mathbb{Z}^+$.

The task is to compute element a^x, which is defined by repeated multiplication of a by itself.

Mathematics Subject Classification: 11T71.
Key words and phrases: exponentiation, addition chains, precomputation.
Research supported by the TÁMOP 4.2.1./B-09/1/KONV-2010-0007 project and TARIPAR3 project grant Nr. TECH 08-A2/2-2008-0086.

Our main result is a lower bound on the size of the minimum set which can be a base of a k-step exponentiation.

Theorem 1. *Assume we know some power of a and store them in a set Q. If for any $y \leq x$ we can compute a^y with at most $k > 0$ group operation, then*

$$\operatorname{card}(Q) > \frac{\sqrt[k]{x}}{3k+1}.$$

2. Exponentiation without precomputation

Assume that

$$x = \sum_{i=0}^{k-1} 2^i \xi_i,$$

where $\xi_i \in \{0, 1\}$.

The well known fast exponentiation method based on the fact that $a^x = a_k$, where a_i $(i = 0 \dots k)$ is defined by the recurrence

$$a_0 = e \qquad a_{i+1} = a_i^2 \cdot a^{\xi_{k-i}},$$

where e is the unit element of the group A.

This property enables us to compute a^x at the cost of maximum $2k$ group operation.

A significant speed up can be achieved by sparse signed binary representation of x, where the digits are from the set $\{-1, 0, 1\}$. One can prove that there exists a not necessarily unique optimal representation of x, where the number of nonzero digits are minimal and not more than $\frac{1}{2}k$. This modification reduces the worst case complexity of the exponentiation to $\frac{3}{2}k$. However, this approach requires the computation of a^{-1}, which is not necessarily easy at all. (For the details and related results, see [3] and [4].)

Assume first that we want to have the most general exponentiation. This means, that we can use the group operation only and we can not compute and store anything in advance.

Let $M : \mathbb{Z}^+ \to \mathbb{N}$ the function expressing the necessary minimal number of group operations to compute a^x. Clearly M is well defined and $M(x) = M(x')$ where $x \equiv x' \mod m$. Here m is the order of a in A. By the time complexity of the fast exponentiation, $M(x) \leq 2\,[\log(x)]$, where $\log(\cdot)$ is the base 2 logarithm and $[\cdot]$ is the integer part function.

The function M is independent of A and actually depends only on m and not on the exact value of a. Since exponentiation have the property $a^{x_1+x_2} = a^{x_1} \cdot a^{x_2}$,

the computation of a^x can be transformed to an analogous computation in $\mathbb{Z}_m$. Here we have to compute x from 1 using additions only. One can represent the intermediate results in a list. This list is the well known addition chain of x. Clearly the addition chain for a given x is not unique and even more, the shortest one may not be unique. A detailed work on addition chains can be found in [8] and [5].

As a first approach on the bounds of M, we state the following:

Lemma 2. *Let* $S_n = \{x | x \in \mathbb{Z}, M(x) = n\}$. *Then* $\max(S_n) = 2^n$.

PROOF. We prove the lemma by induction. Let

$$T_n = \bigcup_{i=0}^{n} S_i.$$

a.) Clearly $S_0 = \{1\}$.

b.) Assume, that $\max(S_l) = 2^l$ for all $0 \leq l \leq k$, where $k \geq 0$ is a fixed integer. This yields, among others, that $M(2^k) = k$ and $\max(T_l) = 2^l$ for all $0 \leq l \leq k$.

If $x \in S_n$ then $M(2x) \leq n + 1$, whence

$$M\left(2^{k+1}\right) \leq k + 1. \tag{1}$$

Since $2^{k+1} > \max(T_l)$ thus (1) implies that $M(2^{k+1}) = k + 1$, whence

$$\max\{S_{k+1}\} \geq 2^{k+1}.$$

Assume now, that $\max\{S_{k+1}\} > 2^{k+1}$. Then $\exists x \in S_{k+1}$, such that $x > 2^{k+1}$. However, this means that $\exists x_1, x_2 \in T_k$ not necessarily different integers, such that $x = x_1 + x_2$. Then

$$2^{k+1} < x = x_1 + x_2 \leq 2 \cdot \max(T_k) = 2 \cdot 2^k,$$

which is a contradiction.

This implies that $\max\{S_{k+1}\} = 2^{k+1}$, whence by induction the lemma follows. □

Corollary 3. $\log(x) \leq M(x) \leq 2\,[\log(x)]$.

Remark 4. Corollary 3 means that basically there are no asymptotically better general exponentiation algorithm, than the well known fast exponentiation. However, there is a chance to increase its speed by a factor of 2.

Algorithm 5. *Let* $x \in \mathbb{N}$, $s = [\log(x)]+1$, $l = [\log(s)-2\log(\log(s))]$, $k = \lceil \frac{s}{l} \rceil$ *and*

$$x = \sum_{i=0}^{k-1} 2^{il} \xi_i \ ,$$

where $\xi_i \in \{0, 2^l - 1\}$. *Here* $\lceil \cdot \rceil$ *means the round up function.*

First compute a^i *for all* $i \in \{0, \ldots, 2^l - 1\}$. *Then define the recurrence*

$$a_0 = e \qquad a_{i+1} = a_i^{2^l} \cdot a^{\xi_{k-i}},$$

where e *is the unit element of the group* A.

Lemma 6. *The number of group operations necessary to compute* a^x *by using Algorithm 5 is at most*

$$\log x \left(1 + \frac{c}{\log \log x}\right)$$

with an absolute constant $c > 0$.

Proof. The number of necessary group operations are

$2^l - 2$ for computing a^i for all $i \in \{0, \ldots, 2^l - 1\}$,

$l(k-1)$ for computing $a_i^{2^l}$ for all $i \in \{2, \ldots, k\}$ and

$k - 1$ additional one for the multiplication by $a^{\xi_{l-i}}$.

All together

$$2^l + l(k-1) + k - 3 < 2^l + s + k < 2^{\log(s) - 2\log(\log(s))} + s + k = \frac{s}{\log^2(s)} + s + k \ .$$

Here for an arbitrary $\epsilon > 0$ there exists N, such that

$$k < \frac{s}{(1-\epsilon)\log(s)}$$

for all $x > N$, whence, for any $c > 0$, the time complexity of the algorithm is less than $\log(x) \cdot (1 + c\frac{1}{\log(\log(x))})$. □

Practically, if we assume, that x has magnitude 2^{1000}, the algorithm can compute a^x in around 1235 operations in the worst case, instead of the original 2000. During the computation, we have to store 62 powers of a ($k = 6$).

Remark 7. A slight improvement can be achieved, if we precompute a^i only for odd i and we don't use fix block size, but always extend to the next nonzero digit of the expansion of x.

Remark 8. As history shows, there are several way to improve the general algorithms assuming special conditions. Bos and Coster [2] with a detailed analysis of vector addition chains improved the above method to the average of 605 operations for 512-digit exponents in contrast to the average of 630 operations of the original method.

3. Speeding up by precomputation

In the remaining part of the paper, assume that the base a is fixed. Then some preliminary computations can be made to decrease the number of necessary operations after getting the input x. Suppose again, that we want to have general methods, thus no shortcuts and particular tricks, only the given group operation can be used.

Let R be the set of x's, such that a^x are known in advance and let

$$\mu = \sum_{x \in R} M(x) \ .$$

Then the computation of a^y for an arbitrary $y \in \mathbb{Z}^+$ requires at least $M(y) - \mu$ group operations. Unfortunately, this means that precomputation asymptotically does not improve anything. However we should not forget, that the group A is finite and thus the value of interesting x's are bounded. Taking this in account, the improvement can be quite considerable.

One can generalize the definition of addition chains to multiple base.

Again, let R be a finite set of integers. We say that the finite list L is an addition chain of base R, if either $L_k \in R$ or there exist $1 \leq i, j < k$ not necessarily different integers, such that $L_k = L_i + L_j$ for all the possible indices k. The length of L is the number of elements of L which are not in R.

Suppose that we want to compute the xth power of a and we want to use at most $k > 0$ group operation for our purposes. How many powers should be computed and stored in advance?

Let $R_k(x) \subseteq \mathbb{N}$, such that for every $y \leq x$ there exists an addition chain of length k based on $R_k(x)$ which contains y.

Clearly $R_k(x)$ is not unique, but there exists a minimal one, with the least elements – which is still not necessarily unique. Let $r_k(x)$ be the cardinality of this minimal base set.

Now we try to find a lower bound for $r_k(x)$.

Proof of Theorem 1. Without loss of generality, we may assume, that a generalized addition chain $L = [L_1, L_2, \ldots, L_{2k}]$ of length k is such, that $L_1, \ldots, L_k \in R_k(x)$. The remaining values in the list can be represented by pairs of integers, expressing which two of the previous members were added to obtain the actual one. If $L_i = L_j + L_m$, then the representing pair is (j, m). Here $j, m < i$ should hold.

Let $P = [P_1, \ldots, P_k]$ denote this list of pairs. Clearly, if $P_i = (j, m)$ then $1 \leq j$, $m < k + i$, for all $i = 1, \ldots, k$.

Let R be a set of integers of cardinality r. Similarly as in the proof of Lemma 1, denote by T the set of integers, which can be computed from the members of R using at most k additions and let $t = \mathrm{card}(T)$. Denote by l the number of different addition chains of length k with base R, by p the number of different list of pairs P corresponding to addition chains of length k.

Since every addition chain contains k computed values, thus $t < k \cdot l$.

The first half $[L_1, \dots, L_k]$ of an addition chain can be represented by a selection of k elements from R, while the second half $[L_{k+1}, \dots, L_{2k}]$ can be described by a list of pairs P. Hence $l < \binom{r}{k} \cdot p$.

For any pair P_i from P we can have $\binom{k+i}{2}$ different value, due to repetition in the pair. This implies

$$\begin{aligned} p &= \binom{k+1}{2} \cdot \binom{k+2}{2} \cdot \dots \cdot \binom{2k-1}{2} = \frac{1}{2^k} \frac{(2k-2)!}{(k-1)!} \frac{(2k-1)!}{k!} \\ &= \frac{1}{2^k} \frac{k}{2k-1} \left(\frac{(2k-1)!}{k!} \right)^2 . \end{aligned}$$

Substituting the above inequalities successively into the others, we get

$$\begin{aligned} t < k \cdot l &< k \cdot \binom{r}{k} \cdot p = k \cdot \binom{r}{k} \cdot \frac{1}{2^k} \frac{k}{2k-1} \left(\frac{(2k-1)!}{k!} \right)^2 \\ &= k \cdot \frac{1}{2^k} \frac{k}{2k-1} \frac{r!}{k!(r-k)!} \cdot \left(\frac{(2k-1)!}{k!} \right)^2 \\ &= \frac{k}{k} \frac{1}{2^k} \frac{k}{2k-1} \cdot \frac{r!}{(k-1)!(r-k)!} \cdot \frac{(2k-1)!}{k!} \frac{(2k-1)!}{k!} \\ &= \frac{1}{2^k} \frac{k}{2k-1} \cdot \frac{r!}{(r-k)!} \cdot \binom{2k-1}{k} \frac{(2k-1)!}{k!} \\ &< \frac{1}{2^k} \frac{k}{2k-1} \cdot \binom{2k-1}{k} \frac{(2k-1)!}{k!} \cdot r^k \\ &= \frac{1}{2^k} \frac{k}{2k-1} \cdot \binom{2k-1}{k} \frac{1}{2k} \prod_{i=k+1}^{2k} i \cdot r^k \\ &= \frac{1}{2^{k+1}} \frac{1}{2k-1} \cdot \binom{2k-1}{k} \prod_{i=k+1}^{2k} i \cdot r^k . \end{aligned}$$

Taking the kth root of both side we get

$$\sqrt[k]{t} < \frac{1}{2^{\frac{k+1}{k}}} \frac{1}{\sqrt[k]{2k-1}} \cdot \sqrt[k]{\binom{2k-1}{k}} \sqrt[k]{\prod_{i=k+1}^{2k} i} \cdot r.$$

Here

$$\frac{1}{2^{\frac{k+1}{k}}} < \frac{1}{2}, \qquad \frac{1}{\sqrt[k]{2k-1}} < 1,$$

by the Stirling formula

$$\sqrt[k]{\binom{2k-1}{k}} \simeq 4$$

(actually < 4 is true) and by the relation between geometric and arithmetic means

$$\sqrt[k]{\prod_{i=k+1}^{2k} i} < \frac{1}{k} \sum_{i=k+1}^{2k} i = \frac{3k+1}{2}.$$

Hence we get

$$\frac{\sqrt[k]{t}}{3k+1} < r.$$

Returning to our original problem, let $t_k(x)$ be the number of computable integers from the set $R_k(x)$ using at most k addition.

Clearly $x < t_k(x)$, whence

$$\frac{\sqrt[k]{x}}{3k+1} < r_k(x). \qquad \square$$

Since we have used very rough estimates at several points, a much better lower bound can be found.

Example 9. *A simple, but for small k, close to minimal choice for $R_k(x)$, if we assume, that during computations steps, we simply add together the members of $R_k(x)$. Then the base set can be chosen in the following way:*

Denote by $x_k = [\,\sqrt[k+1]{x}\,]$.

$$\begin{aligned} R_k(x) = \{&1, 2, \ldots, x_k - 1, \\ &1 \cdot x_k, 2 \cdot x_k, \ldots, (x_k - 1) \cdot x_k, \\ &1 \cdot x_k^2, 2 \cdot x_k^2, \ldots, (x_k - 1) \cdot x_k^2, \\ &\ldots \\ &1 \cdot x_k^k, 2 \cdot x_k^k, \ldots, (x_k - 1) \cdot x_k^k\}. \end{aligned}$$

Actually, $R_k(x)$ can be used as the base x_k representation of the numbers not exceeding x. Although $R_k(x)$ has cardinality $(k+1) \cdot \sqrt[k+1]{x}$, but it has the big advantage, that determining the necessary members we have to use for the computation is very fast.

Example 10. *One can improve the above choice of the base set by the following idea:*

Let $x_k = 4 \cdot \left[\frac{1}{4} \cdot \sqrt[k+1]{x} \right] + 2$.

Assume, that $y \leq x$. *Then using a modified digit expansion*

$$y = \sum_{i=0}^{k} \eta_i \cdot x_k^i + 2 \cdot \sum_{i=0}^{k} \xi_i \cdot x_k^i \ ,$$

where $\eta_i, \xi_i, \xi_k \in \{0, 1, 3, 5, \ldots, x_k - 1\}$ *for* $i = 0, \ldots, k-1$
and $\eta_k \in \{-1, 0, 1, 3, 5, \ldots, x_k - 1\}$, *furthermore* $\eta_i \cdot \xi_i = 0$ *for all* $i = 0, \ldots, k$.

For instance, let $x_k = 10$, $y = 41904$.

Since the last digit is even, thus $\eta_0 = 0$, $\xi_0 = 7$ *and we have a carry* $c_0 = -1$.

The next digit is 0, *with the carry we get* $\eta_1 = 9$, $\xi_1 = 0$ *and* $c_1 = -1$.

The next digit is 9, *with the carry we get* 8, *whence* $\eta_2 = 0$, $\xi_2 = 9$ *and* $c_2 = -1$.

The next digit is 1, *with the carry* -1 *we get* 0, *thus* $\eta_3 = 0$, $\xi_3 = 0$ *and* $c_3 = 0$.

Finally, the last digit of y *is* 4 *with* 0 *carry, whence* $\eta_4 = 0$, $\xi_4 = 7$ *and* $c_4 = -1$.

Since there is a -1 *carry remained, thus* $\xi_5 = -1$.

All together, $y = \overline{90} + 2 \cdot \overline{(-1)70907}$.

With this representation, one can reduce the necessary storage capacity at the cost of one extra operation for a squaring.

Thus the base set

$$\begin{aligned} R_k(x) = \{&1, 3, \ldots, x_k - 1, \\ &1 \cdot x_k, 3 \cdot x_k, \ldots, (x_k - 1) \cdot x_k, \\ &1 \cdot x_k^2, 3 \cdot x_k^2, \ldots, (x_k - 1) \cdot x_k^2, \\ &\ldots \\ &1 \cdot x_k^k, 3 \cdot x_k^k, \ldots, (x_k - 1) \cdot x_k^k, \\ &-1 \cdot x_k^{k+1}\}. \end{aligned}$$

Here $R_k(x)$ *has cardinality* $\frac{1}{2} \cdot (k+1) \cdot \sqrt[k+1]{x}$

Brickell, Gordon, McCurley and Wilson [1] has improved a version of the windowing powering method, by collecting the same coefficients and do the exponentiation with proper data flow.

They achieved an algorithm, which computes a^x with

$$k = (1 + o(1)) \frac{\log(x)}{\log(\log(x))}$$

group operations, while it uses

$$r = O\left(\frac{\log(x)}{\log(\log(x))}\right)$$

previously computed powers.

They give a sample and show that they can compute a^x if x is in the range $[1, 2^{512}]$ with 63 operations and $r = 16320$.

Using the idea of our Example 10 choosing the parameters correspondingly to their example, we get $x = 2^{512}$, $k = 63$, $x_k = 282$ and $r = 8883$. It means, that if we want to use the same range of exponents and the same amount of operations, our method need to precompute and store approximately the half of the powers of their method.

With another parameter choice, $x = 2^{512}$, $k = 55$, $x_k = 634$ and $r = 17435$. This yields, that with 7% more storage we can reduce the number of multiplications by 12%. Finally we present a table of the relations between the different parameters of our method:

	x		
k	2^{512}	2^{1024}	2^{2048}
32	$x_k = 65538$	–	–
	$r = 1048608$	–	–
48	$x_k = 1626$	$x_k = 2642246$	–
	$r = 39024$	$r = 63413904$	–
64	$x_k = 258$	$x_k = 65538$	–
	$r = 8256$	$r = 2097216$	–
96	$x_k = 42$	$x_k = 1626$	$x_k = 2642246$
	$r = 2016$	$r = 78048$	$r = 126827808$
128	$x_k = 18$	$x_k = 258$	$x_k = 65538$
	$r = 1152$	$r = 16512$	$r = 4194432$
192	$x_k = 10$	$x_k = 42$	$x_k = 1626$
	$r = 960$	$r = 4032$	$r = 156096$

References

[1] E. F. Brickell, D. M. Gordon, K. S. McCurley and D. B. Wilson, Fast exponentiation with precomputation: Algoritmhs and lower bounds, *Advances in Cryptology – Proceedings of Eurocrypt'92, Lecture Notes in Computer Science* **658** (1993), 200–207.

[2] J. Bos and M. Coster, Addition Chain Heuristics, *Advances in Cryptology - CRYPTO 89 Proceedings, Lecture Notes in Computer Science* **435** (1990), 400–407.

[3] J. Demetrovics, A. Pethő and L. Rónyai, On ±1-representations of integers, *Acta Cybernetica* **14** (1999), 27–36.

[4] C. Heuberger, Minimal expansions in redundant number systems and shortest paths in graphs, *Computing* **63** (1999), 341–349.

[5] G. A. Jensen and E. Wattel, Efficient calculation of powers in a semigroup, *Math. Centrum, Amsterdam, Afd. Zuivere Wisk.* **ZW 1968-001** (1968), 1–18.

[6] D. E. Knuth, The Art of Computer Programming, Vol. 2 Seminumerical Algorithms, *Addison-Wesley*, 1997.

[7] A. J. Menezes, P. C. van Oorschot and S. A. Vanstone, Handbook of Applied Cryptography, *CRC Press*, *Boca Raton, New York, London, Tokyo*, 1997.

[8] A. Scholz, On addition chains, *Jahresbericht der deutschen Mathematiker-Vereinigung, class II* **47** (1937), 41–42.

TAMÁS HERENDI
FACULTY OF INFORMATICS
UNIVERSITY OF DEBRECEN
H-4010 DEBRECEN, P.O. BOX 12
HUNGARY

E-mail: herendi@inf.unideb.hu

(Received February 28, 2011)