Number-Theoretic Functions Which Are Equivalent to Number of Divisors

Jeffrey Shallit
Department of Computer Science
University of Chicago
Chicago, IL 60637

*Adi Shamir**
Applied Mathematics Department
The Weizmann Institute of Science
Rehovot 76100
Israel

Key Words and Phrases

number of divisors, Möbius function, prime factorization.

Abstract.

Let $d(n)$ denote the number of positive integral divisors of n. In this paper we show that the Möbius function, $\mu(N)$, can be computed by a single call to an oracle for $d(n)$. We also show that any function that depends solely on the exponents in the prime factorization of N can be computed by at most $\log_2 N$ calls to an oracle for $d(N)$.

* Presently visiting Department of Computer Science, University of Chicago.

The problem of computational equivalence between various number-theoretic problems has received considerable attention in the last few years (see [4] for a motivation from cryptography, and [1] for recent results concerning sums of divisors).

In this note, we prove that the problem of computing the number of divisors $d(N)$ of N is equivalent to the problem of computing the multiset

$$e(N) = \{e_1, e_2, \ldots, e_k\}$$

of exponents in the prime factorization of N:

$$N = {p_1}^{e_1} \cdots {p_k}^{e_k} \tag{1}$$

where the p_j are distinct primes and the e_j are positive integers.

Given $e(N)$, it is straightforward to compute $d(N)$ as

$$d(N) = (e_1 + 1)(e_2 + 1) \cdots (e_k + 1), \tag{2}$$

(e. g. [2]). The other direction is a bit harder, since $d(N)$ may be factorized in many ways and thus the e_i cannot be directly recovered from $d(N)$.

Before describing the general case, it is instructive to consider the problem of determining whether or not a number is squarefree; i. e. are all the e_i equal to one? A necessary condition for squarefreeness is that $d(N)$ be a power of 2, but this is not sufficient since, for example, p^3q^7 also satisifies this condition. To solve this problem, we compute $d(N^{q-1})$ instead of $d(N)$, where the prime q is approximately $\log_2 N$. Then N^{q-1} has approximately $(\log_2 N)^2$ bits and its computation can be done in polynomial time. We have:

Theorem 1.

Let q be a prime such that $q - 1 > \log_2 N$. Then N is squarefree iff $d(N^{q-1}) = q^k$ for some $k \geq 1$.

Proof.

If N is given by equation (1), then

$$d(N^{q-1}) = \prod_{i=1}^{k} (1 + (q-1)e_i).$$

When N is squarefree, all the e_i are 1 and therefore $d(N^{q-1}) = q^k$. Conversely, assume that $d(N^{q-1}) = q^j$ for some j. Then each term of the form $1 + (q-1)e_i$ must also be a power of q. However, since all the e_i are at most $\log_2 N$, we have $1 + (q-1)e_i < q^2$. The remaining possibility, that $(q-1)e_i + 1 = q$, implies $e_i = 1$ for all $1 \leq i \leq k$. ■

Corollary.

The Möbius function

$$\mu(N) = \begin{cases} 0, & \text{if } N \text{ is not squarefree;} \\ (-1)^k, & \text{if } N \text{ is squarefree and divisible by } k \text{ distinct primes.} \end{cases}$$

can be computed quickly with a single call to the $d(N)$ oracle.

Proof.

If N is squarefree, then the power of q that divides $d(N^{q-1})$ determines the value of k. ■

We now state the main result of this note.

Theorem 2.

The two problems

i) computing $d(N)$ and

ii) computing $e(N)$

are equivalent under a polynomial time deterministic Turing reduction.

Proof.

The reduction from $d(N)$ to $e(N)$ follows immediately from equation (2). We present a reduction in the other direction, which is a refinement of the proof of Theorem 1.

The main idea in this reduction is as follows: let $f(x)$ be the polynomial which has $e_1, e_2, \ldots, e_k$ as its zeroes, i. e.

$$f(X) = (X - e_1)(X - e_2) \cdots (X - e_k) = X^k - c_0 X^{k-1} + \cdots + (-1)^k c_{k-1}. \tag{3}$$

Suppose we could determine the coefficients $c_0, c_1, \ldots, c_{k-1}$; then by factoring f we could determine the e_j. Of course, we don't actually have to factor f since we know the roots are integers $\leq \log_2 N$, and thus we can find them quickly by exhaustive search.

Let q be a prime number. Then

$$\begin{aligned} d(N^q) &= \prod_{i=1}^{k} (q e_i + 1) \\ &= (-1)^k q^k f(-1/q) \\ &= c_{k-1} q^k + c_{k-2} q^{k-1} + \cdots + c_0 q + 1. \end{aligned} \tag{4}$$

If q is larger than each of the coefficients of $f(X)$, then we can consider $d(N^q)$ to be a number written in base q, and easily recover the coefficients $c_0, c_1, \ldots, c_{k-1}$.

The simplest way to read off the coefficients of $f(X)$ is to choose q larger than $\max_{1 \leq i \leq k-1} c_i$. Unfortunately, this naive approach does not give a polynomial-time algorithm, for it requires us to compute $d(N^q)$ with q roughly as big as N. If q is this big, we cannot compute N^q or even express it in time polynomial in $\log_2 N$.

Instead we evaluate equation (4) for many different *small* values of q, and then recover the coefficients c_j one by one, using the Chinese remainder theorem.

The algorithm presented below takes as input a positive integer N and an oracle for $d(n)$. It produces the multiset $S = \{e_1, e_2, \ldots, e_k\}$ of exponents in the prime factorization of N.

Algorithm A.

A1. [Initialize]. Set $S := \emptyset$, $B := (\log_2 N)(1 + \log_2 \log_2 N)$.

A2. [Choose set of primes P.]

  $P :=$ a set of primes q with $2B \leq q \leq 3.3B$, of cardinality
 not exceeding $\lceil \log_2 N \rceil$.

 for each $q \in P$ **do** compute and store $d(N^q)$;

A3. [Infer the coefficients of $f(x)$].

 Set $k := -1$;

 repeat begin

  $k := k + 1$;

  for each $q \in P$ **do** $d_k[q] := \frac{d(N^q) - 1 - \sum_{j=0}^{k-1} c_j q^{j+1}}{q^{k+1}}$;

  Compute c_k using the Chinese remainder theorem

  and the congruences $c_k \equiv d_k[q] \pmod{q}$

  end;

 until $c_k = 0$;

 define $f(x) := x^k - c_0 x^{k-1} + \cdots + (-1)^k c_{k-1}$;

A4. [Factor $f(x)$].

 for $i := 1$ **to** $\lfloor \log_2 N \rfloor$ **do**

  begin

   $b_i :=$ exponent of highest power of $x - i$ that divides $f(x)$;

   $S := S \cup \{b_i \text{ copies of } i\}$

  end

Lemma 3.

Algorithm A is correct and runs in time polynomial in $\log_2 N$*. It uses only* $\log_2 N$ *oracle calls.*

Proof.

Let p be a prime, let N be given by equation (1), and let $c_0, c_1, \ldots, c_{k-1}$ be as in equation (4) above. It is easy to see that

$$c_j < \binom{\log_2 N}{\frac{1}{2}\log_2 N} (\log_2 N)^{\log_2 N} < N^{1+\log_2 \log_2 N}.$$

First we show that the product over all primes in Q is sufficiently large to represent each coefficient of $f(X)$.

If Q has $\lceil \log_2 N \rceil$ elements, then it is clear that the product is sufficiently large, since each member of Q is larger than $2\log_2 N$.

Now suppose Q has fewer than $\lceil \log_2 N \rceil$ elements. We must show that

$$\prod_{\substack{2B < p \leq 3.3B \\ p \text{ prime}}} p > N^{1+\log_2 \log_2 N}.$$

It clearly suffices to show

$$\sum_{\substack{2B < p \leq 3.3B \\ p \text{ prime}}} \log_2 p > (\log_2 N)(1 + \log_2 \log_2 N). \tag{5}$$

We do this for all N "sufficiently large". By a theorem of Rosser and Schoenfeld [3] we have

$$.84x < \sum_{\substack{p \leq x \\ p \text{ prime}}} \log_e p < 1.01624x$$

for $x \geq 101$.

Hence we find

$$\sum_{\substack{2B < p \leq 3.3B \\ p \text{ prime}}} \log_2 p > (\log_2 e)(2.772B - 2.03248B) > B$$

and the truth of equation (5) easily follows from our choice of B in step A1.

Thus we see that in step A2 we use at most $\log_2 N$ calls to the oracle for $d(n)$. Now step A3 is completed correctly by equation (4) above. It is clear that the algorithm runs in polynomial time. This completes the proof of Lemma 3. ■

Thus we have completed the proof of Theorem 2. ■

Corollary.

Let N *be as in equation (1). Define*

$$\Omega(N) = e_1 + e_2 + \cdots + e_k.$$

Then $\Omega(N)$ *can be computed in one call to an oracle for* $d(N)$*.*

Proof.

Let q be a prime $> \log_2 N$. Then

$$\Omega(N) = \frac{d(N^q) - 1}{q} \bmod q. \quad ■$$

Corollary.

Let g, h *be integers with* $h \neq 0$*. Define*

$$r_{g,h}(N) = (ge_1 + h)(ge_2 + h) \cdots (ge_k + h)$$

Then the problem of computing $r_{g,h}(N)$ *is equivalent to the problem of computing* $e(N)$*.*

The proof is left to the reader.

Open Question: can $e(N)$ be computed from *one* call to the oracle for $d(n)$? Of course, we require that the argument to the oracle be of size polynomial in $\log_2 N$.

Acknowledgements.

We are pleased to acknowledge conversations with Eric Bach and Manuel Blum.

Thanks also go to the referee for suggesting several improvements in the exposition.

References

[1] Eric Bach, Gary Miller, and Jeffrey Shallit, Sums of divisors, perfect numbers, and factoring, *Proc. 16th Ann. ACM Symp. on Theory of Computing*, Association for Computing Machinery, New York, 1984, 183–190.

[2] William J. LeVeque, Fundamentals of Number Theory, Addison-Wesley, Reading, Massachusetts, 1977.

[3] J. Barkley Rosser and Lowell Schoenfeld, Approximate formulas for some functions of prime numbers, *Ill. Journ. Math.* **6** (1962) 64–94.

[4] R. L. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public key cryptosystems, *Comm. ACM* **21** (1978) 120–126.