ENUMERATING AND COUNTING SMOOTH INTEGERS

Daniel J. Bernstein

950518 (draft T)

Abstract. We present a linear-time algorithm to list the y-smooth integers up to x, and an even faster algorithm to count the y-smooth integers up to x. We also show how all multiplications can be replaced by an equal number of additions.

1. Introduction

An integer is y-**rough** if it has a prime factor larger than y; otherwise it is y-**smooth**. Let $P(x,y)$ be the set of y-smooth integers between 1 and x inclusive, and let $\Psi(x,y) = \#P(x,y)$ be the number of such integers.

In section 2 we present a straightforward algorithm that, with fewer than $2\Psi(x,y)$ multiplications, lists the elements of $P(x,y)$. In section 3 we present a faster algorithm to compute $\Psi(x,y)$ without enumerating $P(x,y)$. In section 4 we show how to adapt these algorithms to use addition instead of multiplication.

For convenience we rely on the following nontraditional statement of unique factorization. Consider the set S of integers p^{2^k}, where p is a prime number and k is a nonnegative integer; the first few elements of S are $\{2,3,4,5,7,9,11,13,16\}$. For any finite subset T of S, the **product of** T—i.e., the product of the elements of T—is a positive integer. Unique factorization now says that every positive integer is the product of a unique finite subset of S.

Similarly, if $S = \{p^{2^k} : p \le y\}$, the y-smooth integers are exactly the products of finite subsets of S.

In general we will consider any set S of positive integers such that distinct finite subsets of S have distinct products. We write $P(x,S)$ for the set of products, no larger than x, of finite subsets of S; and we write $\Psi(x,S) = \#P(x,S)$ for the number of such products.

Now the algorithm in section 2 enumerates $P(x,S)$ for any S, and the algorithm in section 3 computes $\Psi(x,S)$. The P algorithm is in fact a critical component of the Ψ algorithm.

Our algorithms could be applied in much greater generality. We use just two facts about positive integers: (1) if $p > x$ then $ps > x$; (2) if $ps > x$ and $s' > s$ then $ps' > x$.

See [1] for more information about Ψ.

1991 *Mathematics Subject Classification*. Primary 11Y16.

This paper was included in the author's thesis at the University of California at Berkeley. The author was supported in part by a National Science Foundation Graduate Fellowship.

Typeset by $\mathcal{A}\mathcal{M}\mathcal{S}$-TEX

2. Enumerating smooth integers

Fix S, and fix an integer $x \geq 1$. One can enumerate $P(x, S)$ by starting from a single integer, 1, and multiplying by elements of S every which way, tossing out results larger than x:

Algorithm 1. We compute $P(x, S)$.
1. Set $P \leftarrow \{1\}$.
2. For each $s \in S$:
3. Set $Q \leftarrow \{\}$.
4. For each $p \in P$:
5. If $ps \leq x$: Add ps to Q.
6. Set $P \leftarrow P \cup Q$.
7. Stop. The answer is P.

To save time we consider the elements of S in order. Once we find $ps > x$, we need not multiply p by any later elements of S, since $s' > s$ implies $ps' > ps > x$. In this case we say that p is **dead** and we move it to a **dead pile**, D:

Algorithm 2. We compute $P(x, S)$, given S in order.
1. Set $P \leftarrow \{1\}$, $D \leftarrow \{\}$.
2. For each $s \in S$, in increasing order:
3. (Now P is nonempty.) Set $Q \leftarrow \{\}$.
4. For each $p \in P$:
5. If $ps \leq x$: Add ps to Q.
6. Otherwise: Remove p from P. Add p to D.
7. Set $P \leftarrow P \cup Q$.
8. If P is empty: Stop. The answer is D.
9. Stop. The answer is $P \cup D$.

Lemma 2.1. *Let S be a set of positive integers such that distinct finite subsets of S have distinct products. Then Algorithm 2 enumerates $P(x, S)$ with fewer than $2\Psi(x, S)$ multiplications.*

Since P is nonempty in step 3, we always run through the inner loop of Algorithm 2 at least once for every iteration of the outer loop. Hence Algorithm 2 takes time linear in $\Psi(x, S)$. (It is easy to compute $P \cup Q$ and $P \cup D$ if we represent P and Q and D as linked lists.)

Proof. Note that, by hypothesis on S, each integer is added to Q at most once and to D at most once.

Write m for the exact number of multiplications so far. Then $m = 2\#D + \#P - 1$ when we reach step 3 or step 8, and $m = 2\#D + \#(P \cup Q) - 1$ when we reach step 5 or step 7. Indeed, the first time we reach step 3, we have $D = \{\}$ and $P = \{1\}$, so $2\#D + \#P - 1 = 0$; and $m = 0$. In steps 5 and 6, we either add a new element to Q or we move an element from P to D. Either way we increase $2\#D + \#(P \cup Q) - 1$ by 1; and we also increase m by 1. At the beginning of step 7, we have $m = 2\#D + \#(P \cup Q) - 1$. We replace P by $P \cup Q$, so $m = 2\#D + \#P - 1$ at the beginning of step 8.

Finally, when we stop in step 8 or step 9, $m < 2\#D + \#P \leq 2\#(D \cup P) = 2\Psi(x, S)$. □

It will be convenient in the next section to have the output of Algorithm 2 in order. This is not a problem, since one can sort in time linear in the number of output bits.

3. Counting smooth integers

To find $\Psi(x, S)$ one can compute $P(x, S)$ by Algorithm 2 above. We do better by splitting S into two pieces, T and U. Then each element of $P(x, S)$ is the product of an element of $P(x, T)$ and an element of $P(x, U)$.

Algorithm 3. We compute $\Psi(x, S)$. In advance select a subset $T \subseteq S$ and put $U = S - T$.

1. Compute the elements $p_1 > p_2 > \cdots > p_m$ of $P(x, T)$, by Algorithm 2.
2. Compute the elements $q_1 < q_2 < \cdots < q_n$ of $P(x, U)$, by Algorithm 2.
3. Set $j \leftarrow 1$, $\Psi \leftarrow 0$.
4. For $i = 1, 2, \ldots, m$:
5. If $j \le n$ and $p_i q_j \le x$: Increase j by 1 and repeat this step.
6. Set $\Psi \leftarrow \Psi + j - 1$.
7. Stop. The answer is Ψ.

Lemma 3.1. *At the beginning of step 6 of Algorithm 3, $p_i q_k \le x$ if and only if $k < j$, for $1 \le k \le n$.*

So Algorithm 3 walks along the curve of approximate solutions (i, j) to $p_i q_j = x$.

Proof. Say $p_i q_k \le x$. Since we passed step 5 we have either $j > n$ or $p_i q_j > x$. If $j > n$ then $j > k$. If $p_i q_j > x$ then $p_i q_j > p_i q_k$ so $j > k$.

Conversely, say $k < j$. How did j increase past k? We must have found $p_h q_k \le x$ for some $h \le i$. But then $p_i \le p_h$ so $p_i q_k \le x$. □

Lemma 3.2. *Let S be a set of positive integers such that distinct finite subsets of S have distinct products. Then Algorithm 3 computes $\Psi(x, S)$ with fewer than $3(\Psi(x, T) + \Psi(x, U))$ multiplications.*

In general $\Psi(x, T)\Psi(x, U) \ge \Psi(x, S)$ so $\Psi(x, T) + \Psi(x, U) \ge 2\sqrt{\Psi(x, S)}$. On the other hand $\Psi(x, T) + \Psi(x, U) \le 2\Psi(x, S)$. So the bound in Lemma 3.2 is $6\Psi(x, S)^\alpha$ for some α between 1/2 and 1.

Proof. First we show that Algorithm 3 works. By Lemma 3.1,

$$\#\{k : p_i q_k \le x\} = \#\{k : k < j\} = j - 1.$$

By summing $j - 1$ for all i, we count the number of (i, k) such that $p_i q_k \le x$, i.e., the number of products $p_i q_k$ no larger than x, i.e., $\Psi(x, S)$.

In step 1 of Algorithm 3 we use fewer than $2\Psi(x, T)$ multiplications, by Lemma 2.1. In step 2 we use fewer than $2\Psi(x, U)$ multiplications. In step 5 we use at most $m + n = \Psi(x, T) + \Psi(x, U)$ multiplications, because the quantity $i + j$ starts at 2, never exceeds $m + n + 1$, and increases on each trip through step 5. □

How do we choose T and U? It seems reasonable to toss elements of S alternately into T and U. If we are counting smooth numbers this means that the first few elements of T are $\{2, 4, 7, 11, 16\}$ and the first few elements of U are $\{3, 5, 9, 13, 17\}$. Perhaps this is close to optimal; it should be possible to use the structure of $P(x, S)$ to find a realistic lower bound on $\Psi(x, T) + \Psi(x, U)$.

4. Avoiding multiplications

In computing Ψ we multiply positive integers and check whether the products exceed x. We can survive without multiplication; the idea is to represent each positive integer by an integer approximation to its logarithm. Here are the details.

Select b such that $2^b \geq x+1$, and select $Z \geq 2^{b+2}b+2$. Let p be a positive integer; we say that r **represents** p if $|r - Z\log p| \leq \lg p$. Here $\lg p = \log p/\log 2$.

For any positive integer p there is an integer r that represents p. For $p=1$ we take $r=0$. For $p\geq 2$ we select an integer r within 1 of $Z\log p$. (We may construct r from a precomputed table of $\log(2^k/(2^k-1))$, by writing p as an approximate product of terms of the form $2^k/(2^k-1)$. See [2, exercise 1.2.2–25].)

Lemma 4.1. *If r represents p and r' represents p' then $r+r'$ represents pp'.*

Proof. $|r+r'-Z\log pp'| \leq |r-Z\log p| + |r'-Z\log p'| \leq \lg p + \lg p' = \lg pp'$. □

Lemma 4.2. *Let s represent x, and let r represent p. Then $p \leq x$ if and only if $r < s+2b$.*

Proof. If $p \leq x$ then

$$r-s = r - Z\log p + Z\log p - s \leq r - Z\log p + Z\log x - s \leq \lg p + \lg x \leq 2\lg x < 2b$$

so $r < s+2b$. If $p \geq x+1$ then

$$\log p - \log x \geq \log\left(1+\frac{1}{x}\right) \geq \log\left(1+\frac{1}{2^b-1}\right) = -\log\left(1-\frac{1}{2^b}\right) > \frac{1}{2^b}$$

so

$$\begin{aligned} r-s+2b > r-s+2\lg x &\geq (Z\log p - \lg p) - (Z\log x + \lg x) + 2\lg x \\ &= \left(Z - \frac{1}{\log 2}\right)(\log p - \log x) > \left(Z-\frac{1}{\log 2}\right)\frac{1}{2^b} > \frac{Z-2}{2^b} \geq 4b \end{aligned}$$

so $r > s+2b$. □

We have thus replaced multiplication and comparison against x with addition and comparison against $s+2b$. One final trick: We can store *differences* of adjacent logarithms in the arrays of Algorithms 2 and 3. These differences are (usually) relatively small, so we save some space and time.

References

1. E. R. Canfield, Pal Erdős, Carl Pomerance, *On a problem of Oppenheim concerning "factorisatio numerorum"*, Journal of Number Theory **17** (1983), 1–28.
2. Donald E. Knuth, *The Art of Computer Programming, volume 1: Fundamental Algorithms*, 2nd edition, Addison-Wesley, Reading, Massachusetts, 1973.

5 Brewster Lane, Bellport, NY 11713