Generating Random Factored Numbers, Easily

Adam Kalai*

Our goal is to generate a random "pre-factored" number, that is a uniformly random number between 1 and N, along with its prime factorization. Of course, one could simply pick a random number and then try to factor it, but there is no known polynomial-time factoring algorithm [3]. In his dissertation, Bach presents an efficient algorithm for generating such pre-factored numbers [1, 2]. Here, we present a significantly simpler algorithm and analysis for the same problem. Our algorithm is, however, a $\log(N)$ factor less efficient.

Algorithm:

Input: Integer $N > 0$.

Output: A uniformly random number $1 \leq m \leq N$.

1. Pick random seq. $N \geq s_1 \geq s_2 \geq \ldots \geq s_l = 1$ by choosing $s_1 \in \{1, 2, \ldots, N\}$ and $s_{i+1} \in \{1, 2, \ldots, s_i\}$.

2. Let r be the product of the **prime** s_i's.

3. If $r \leq N$, output r with probability r/N.

4. Otherwise, RESTART.

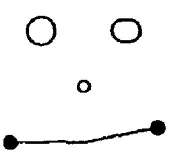
The key to understanding this algorithm is that each prime $p \leq N$ is included in the sequence independently with probability $1/p$. Intuitively, this is because p occurs iff it is chosen before $\{1, \ldots, p-1\}$, which happens with probability $1/p$. As a result, the probability of generating a factored number $r = p_1 p_2 \cdots p_k$ is proportional to $1/p_1 \cdots 1/p_k = 1/r$. Step 3[1] then makes each number equally likely with rejection sampling.

We could have equivalently, but more slowly, generated the sequence in Step 1 by first choosing the number of occurrences of N, and then generating such a sequence for $N-1$. This follows from the fact that, regardless of the number of occurrences of N, the first number in the sequence *less than* N is equally likely to be $\{1, \ldots, N-1\}$. Clearly N occurs at least once with probability $1/N$ and occurs exactly α times with probability $1/N^\alpha(1 - 1/N)$. It follows, by induction on N, that the probability of having α_j occurrences of j is $1/j^{\alpha_j}(1 - 1/j)$, and that occurrences of different j are independent.

The chance of having α_p occurrences of each prime $p \leq N$ and generating the factored number $r = \prod p^{\alpha_p}$ in Step 2 is, by independence,

$$Pr\left[r = \prod_{p \leq N} p^{\alpha_p}\right] = \prod_{p \leq N} (1/p^{\alpha_p})(1 - 1/p) = (1/r) \prod_{p \leq N} 1 - 1/p.$$

Thus the probability of generating an $r \leq N$ *and* outputting it in Step 3 is $(r/N)(1/r)\prod_{p \leq N} 1 - 1/p = (1/N)\prod 1 - 1/p$, which means that all $r \leq N$ are equally likely. So, with probability $\prod 1 - 1/p$ we output a uniformly random factored number, and otherwise we restart. Consequently, the expected number of restarts is $1/\prod 1 - 1/p = \theta(\log N)$, by Merten's theorem [3]. On a run, we test s for primality with probability $1/s$. Thus, we expect to execute $1 + 1/2 + \cdots + 1/N = \theta(\log N)$ primality tests, giving an expected $\theta(\log^2 N)$ primality tests before success. Bach's algorithm uses only an expected $O(\log N)$ tests. For either algorithm, primality tests can be implemented efficiently by a randomized algorithm [3], or as shown in the following diagram:



Acknowledgements. I would like to thank Manuel Blum, Michael Rabin, and Doug Rohde for helpful comments, and an IBM Distinguished Graduate Fellowship and NSF Postdoctoral Research Fellowship for funding.

References

[1] E. Bach, *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms,* MIT Press, Cambridge, 1985.

[2] E. Bach, *How to Generate Factored Random Numbers,* SIAM J. Computing, 17 (1988), pp. 179-193.

[3] E. Bach and J. Shallit, *Algorithmic Number Theory,* MIT Press, Cambridge, 1996.

*M.I.T. (akalai@mit.edu)

[1] After Step 2, we have the nice distribution over the infinite set of numbers whose factors are no larger than N, with probability of a particular r proportional to $1/r$.