From: “Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms”, by Eric Bach, MIT Press, 1984

Chapter 2

THE GENERATION OF RANDOM FACTORIZATIONS

2.0 Introduction

This chapter discusses a new application for prime testing: the generation of "pre-factored" random numbers. To fix ideas, consider the following situation: Let N be a fixed positive number, and suppose that we want an integer x uniformly distributed on the interval $N/2 < x \leq N$; but instead of the usual binary notation, we want to output the prime factors of x.

This can be done by assembling "random" primes, but it is not clear with what distribution the primes should be selected, nor how to generate primes with a given distribution. Most of this chapter will deal with these two questions. However, the resulting method is easily sketched:

The algorithm selects a prime-power factor q of x whose length is nearly uniform between 0 and $\log N$, then recursively selects the factors of a number y between $N/2q$ and N/q and sets $x = y \cdot q$. This picks x with a known bias; to correct this, it flips a (very unfair) coin to decide whether to output x or repeat the whole process.

I will show that the resulting distribution is uniform, and that this is a fast algorithm; it requires $O(\log N)$ primality tests on the average. This can be put in another form, using the results of chapter 1. If the ERH is true, then expected time to generate a factored random number of length k is a bounded by a polynomial in k.

The method also behaves well if it uses only a probabilistic prime test that can err on composite input. In this case, the distribution of correctly factored numbers is still uniform, and the possibility of producing an incompletely factored output can in practice be disregarded.

The rest of this chapter is organized as follows. Section 2.1 presents a heuristic derivation of the algorithm. Section 2.2 discusses some ideas of probability theory; they are used in the algorithm, described in sections 2.3 and 2.4. The last three sections discuss the running time of the method, based on an estimate of the average number of prime tests found in section 2.5. The last two sections bound the expected number of single-precision operations, assuming perfect and imperfect prime testing, respectively.

2.1 A Method that Almost Works

Later I will present a detailed algorithm; to understand it, it is best to first think heuristically and ignore certain difficulties.

First, what is a "random factor" of a number? Consider the following picture: for each number of length $\log N$, write down its prime factorization. If the factorizations are arranged one per line, and given in binary notation, the picture will look something like this:

```
                        .
                        .
                        .
10001...0001  101010..011  1000...01 101..101
10001...0001  101010..011  1000...01 101..111
10001...0001  101010..011  1000...01 111..101
10001...0001  101010..011  1000...01 111..111
                        .
                        .
                        .
```

Imagine throwing a dart at this matrix, and picking p if the dart lands in the binary representation of p. Then p occurs in about $1/p$ of the numbers, and ignoring repeated factors, the dart will land on p in each of them about $\log p/\log N$ of the time.

This suggests selecting the first factor p with probability about $\log p/p\log N$, and using approximations given by the prime number theorem, this has the effect[3] of making the length of p uniformly distributed in $(0, \log N)$.

Thus, to choose x uniformly with $N/2 < x \leq N$, one might proceed as follows: Select a length λ uniformly from $(0, \log N)$ and pick the largest prime p with $\log p \leq \lambda$. Then recursively select the rest of x, call it y, from $(N/2p, N/p]$, and announce that x is p times the prime factorization of y.

Blithely assuming that the distribution of y is uniform, the probability of selecting x is about

$$\sum_{p \mid x} \frac{\log p}{p \log N} \cdot \frac{1}{N/p - N/2p}.$$

This is $2/N$, the correct probability for a uniform distribution, times a bias factor of

$$\frac{1}{\log N} \sum_{p \mid x} \log p.$$

This bias should be close to 1, and it is, provided that x doesn't have too many repeated prime factors.

Thus one suspects that this method is almost right, but a closer look at the algorithm reveals the complications listed below.

1) Merely picking the biggest prime less than some given value won't do; for one thing, the first member of a twin prime pair will be chosen less frequently than

[3] If $p < N$ is chosen with probability $\log p/p\log N$, then $\log p/\log N$ converges in distribution as $N \to \infty$ to a uniform $(0,1)$ random variable; see [37].

the second. The resulting method must be insensitive to these local irregularities.

2) The bias factor is quite small for certain x, say powers of 2. This problem does not go away unless prime power factors are also chosen in the first step.

3) At the end of the algorithm, x will have been chosen with a certain bias, but the recursion will not work unless all x's are equally likely. The odds must be changed somehow to make the eventual output uniform.

4) It was claimed that y, the rest of x, could be selected from $(N/2p, N/p]$ with probability $2p/N$. However, it is by no means certain, and in general not true, that there are $N/2p$ integers in this range.

Dealing with these problems requires some machinery that will be developed in the next three sections.

2.2 Doctoring the Odds

This section tells how to use one distribution to simulate another, using only a little information about the odds, a source of uniform $(0,1)$ random numbers, and some extra time.

Let $\{x_1, \ldots, x_n\}$ be a finite set. Say that X has a finite distribution with odds $(p_1, \ldots, p_n)$ if $X = x_i$ with probability $p_i / \Sigma p_j$. The odds of a distribution are only defined up to a multiplicative constant; this conforms to ordinary usage, in which odds of 2:1 and 10:5 are regarded as identical.

To see how to turn one distribution into another, consider an example. Suppose one has a coin that is biased in favor of heads with odds of 2:1, and wishes to make it fair. This can be done by the following trick. Flip the coin. If it comes up tails, say "tails"; if it comes up heads, say "heads" with probability 1/2, and with probability 1/2 repeat the process.

The stopping time can be analyzed by the following "renewal" argument. The process must flip the biased coin once no matter what happens, and after this first step, it has one chance in three of being born again. Thus the expected stopping time $E(T)$ must satisfy $E(T) = 1 + E(T)/3$, so $E(T) = 3/2$. More generally, $T = k$ with probability $(2/3)\cdot(1/3)^{k-1}$; this is a geometric distribution with expected value 3/2. At each reincarnation, the process has no memory of its past, so the stopping time and the ultimate result are independent.

Clearly this example is silly, as it requires a fair coin to produce the effect of one, but it points out some important features of the method.

a) Decisions are only made locally; after getting, say, heads, a decision can be made without knowing the other possible outcomes or even their total number.

b) Only the odds matter; knowing only the relative probability of each outcome is sufficient for undoing the bias.

The general version of this is called the "acceptance-rejection" method ([38]), and works as follows: we are given odds $(p_1, \ldots, p_n)$ but want odds $(q_1, \ldots, q_n)$. Assuming that $q_i \le p_i$, the recipe is: select X from the original distribution; if $X = x_i$, then output x_i with probability q_i/p_i, and repeat with probability $1 - q_i/p_i$.

THEOREM D. The above process yields a finite distribution with *odds* $(q_1, \ldots, q_n)$. Moreover, the stopping time is distributed geometrically with expected value $\Sigma p_i \,/\, \Sigma q_i$; it and the eventual output are independent.

PROOF: Generalize the earlier discussion.

This idea is at heart of the method, in two ways:

To select a factor with approximately uniform length, the algorithm chooses prime powers q with probability proportional to a certain Δ_q. To do this, it first picks integers q in the following way: 2 and 3 each appear with (relative) probability 1/2, 4, 5, 6, and 7 each appear with probability 1/4, and so on. Since $\Delta_q < 1/q$, acceptance-rejection is used twice: first to produce the distribution Δ_q, and then to throw away q's that are not prime powers.

At the top level, the algorithm generates x, $N/2 < x \le N$, with probability proportional to $\log x$. It accepts x with probability $\log N/2/\log x$, producing a uniform distribution.

2.3 A Factor Generation Procedure

This section presents and describes a factor selection process.

For real numbers a and b, let $\#(a,b]$ denote the number of integers x, $a < x \le b$. If $\lfloor x \rfloor$ denotes the greatest integer $\le x$, then $\#(a/2,a] = \lfloor (a+1)/2 \rfloor$; this implies the frequently-used estimate

$$(a-1)/2 \le \#(a/2,a] \le (a+1)/2. \tag{11}$$

For prime powers $q = p^{\alpha}$, and integers N, let

$$\Delta_N(q) = \frac{\log p}{\log N} \cdot \frac{\#(N/2q, N/q]}{N}. \tag{12}$$

In terms of the above notation, the method is the following.

PROCESS F: Factor generation.

```
(*) Select a random integer $j$ with $1 \le j \le \log_2 N$.
    Let $q = 2^j + r$, where $r$, $0 \le r < 2^j$, is chosen at random.
    Choose $\lambda$ from the $U(0,1)$ distribution.
    If $q$ is a prime power, $q \le N$, and $\lambda < \Delta_N(q) 2^{[\log_2 q]}$, output $q$.
    If not, go back to (*).
```

THEOREM E. Process F almost surely halts; the number of times (*) is reached has a geometric distribution whose expected value is $O(\log N)$. It outputs a prime power $q = p^\alpha$, $2 \le q \le N$, with probability proportional to $\Delta_N(q)$. The running time and the output value are independent.

PROOF: The first two steps select q with relative probability $2^{-[\log_2 q]}$, and since $2^{[\log_2 q]} \Delta_N(q) \le \frac{N+q}{2N} \le 1$ for $q \le N$, q is output with the stated probability. For the running time estimate, it will suffice to show that $\sum_{q \le N} \Delta_N(q)$ is roughly a constant. To do this I need two consequences of the prime number theorem (see, e.g. [37] p. 65): $\sum_{p \le N} \log p \sim N$ and $\sum_{p \le N} \log p / p \sim \log N$. Then

$$\sum_{q \le N} \Delta_N(q) \ \ge \ \sum_{p \le N} \Delta_N(p) \ \ge \ \sum_{p \le N} \frac{\log p}{2p \log N} - \sum_{p \le N} \frac{\log p}{2N \log N} \ \sim \ \frac{1}{2}.$$

The independence statement is a consequence of theorem D.

It was stated earlier that q's length is roughly uniformly distributed. This intuition can be refined into the following precise statement: $N \to \infty$, $\log q / \log N$ converges in distribution to a uniform $(0,1)$ random variable. This implies the following: if

$$F_N(x) = Pr[q \le x],$$

then $F_N(x)$, $E \log(N/q)$, and $E \log^2(N/q)$ are close to $\log x / \log N$, $1/2 \log N$, and $1/3 (\log N)^2$, respectively. The next three lemmas give upper bounds corresponding