Pollard's Factoring Algorithm
Exposition by William Gasarch

1 Introduction

There is a trivial algorithm that factors N in time $O(N^{1/2})$. We will present Pollard's algorithm for factoring which is believed to have complexity $O(N^{1/4})$ though this has not been proven. It works well in practice.

We take *factoring* to mean just finding a non-trivial factor. In practice we would use such an algorithm recursively.

2 We Seek x, y such that $x \equiv y \pmod p$

We want to factor N. Let p be the smallest prime factor of N. Note that $p \le N^{1/2}$. We do not know p. Lets say we somehow find x, y such that $x \equiv y \pmod p$. Then $GCD(x-y, N)$ will likely yield a nontrivial factor of N. We look at several approaches to finding such an x, y that do not work before presenting the approach that does work.

3 Some Probability

We first need some probability.

There are n boxes. I am going to put k balls in them at random. What is the probability that there is some box with at least two balls in it?

We first find the probability that no box has two balls.

The number of ways to put balls into boxes is n^k.

The number of ways to put balls into boxes so that no box has two balls is $n \times (n-1) \times \cdots \times (n-k+1)$.

Hence the probability that no two balls go in the same box is

$$\frac{n \times (n-1) \times \cdots \times (n-k+1)}{n^k} = \frac{n}{n}\frac{n-1}{n}\frac{n-2}{n} \cdots \frac{n-k+1}{n} = (1-\frac{1}{n})(1-\frac{2}{n}) \cdots (1-\frac{k-1}{n})$$

We ignore all terms that are $\le \frac{c}{n^2}$ for any c.

Hence we have:

$$(1-\frac{1}{n})(1-\frac{2}{n}) \cdots (1-\frac{k-1}{n}) \sim (1 - \frac{(1+2+\cdots+(k-1)}{n}$$

We approximate $1+2+\cdots(k-1) = \frac{k(k-1)}{2}$ by $\frac{k^2}{2}$. Hence the probability that no box has two balls is approximately

$$(1 - \frac{k^2}{2n}).$$

Hence the probability that some box has at least two balls is approximately

$$(1 - (1 - \frac{k^2}{2n}) = \frac{k^2}{2n}.$$

We want a value of k so that this probability is over $\frac{1}{2}$.

$$\frac{k^2}{2n} > \frac{1}{2}$$

$$\frac{k^2}{n} > 1$$

$$k \ge \sqrt{n}$$

We want to say *Great, we'll take* $k = \sqrt{n}$*.* But we did all of those approximations. We summarize what we did, and what is known, in the following Lemma.

Lemma 3.1

1. *There exists numbers* c *and* n_0 *such that for all* $n \ge n_0$*, if* $k = c\sqrt{k}$*, if* k *balls are randomly put into* n *boxes then the probability that some box has two balls is* $\ge \frac{1}{2}$*. The larger* n_0 *is the smaller* c *has to be. The following values work:* $n_0 = 43$*,* $c = \sqrt{2\ln(2)} \sim 1.16$.
2. *There exists numbers* c *and* n_0 *such that for all* $n \ge n_0$*, if* $k = c\sqrt{k}$*, if* k *balls are randomly put into* n *boxes then the probability that some box has two balls is* $\ge \frac{99}{100}$*. The larger* n_0 *is the smaller* c *has to be. I don't know the value of* n_0 *and* c *but they are reasonable. I would guess* $n_0 = 100$ *and* $c = 5$ *suffice.*

4 Use Randomization!

Given N we generate a sequence of random numbers $x_1, x_2, \ldots \in [0, N-1]$. Thought experiment: look at

$$x_1 \bmod p, x_2 \bmod p, \ldots.$$

This is a sequence of random elements in $[0, p-1]$. By Lemma 3.1.2 with probability 0.99 there exists $i, j \le cp^{1/2} \le cN^{1/4}$ such that $x_i \pmod p = x_j \pmod p$, or $x_i \equiv x_j \pmod p$. For the rest of this exposition we will ignore the c and just use $p^{1/2}$ and $N^{1/4}$.

So we could have an algorithm that generates this sequence and looks for repeats. NO WE CAN"T- we don't know p. But we can pretend that $x_i \equiv x_j \pmod p$ and try $GCD(x_i - x_j, N)$. Which x_i, x_j do we do this for? ALL of them which is why this algorithm is too slow. Even so, here is the algorithm.

```
x_1 = RAND(0,N-1)
i=2
FOUND = FALSE
while NOT FOUND
   {
     x_i := RAND(0,N-1)
     for j=1 to i-1
   {
     d=GCD(x_i-x_j,N)
     if (d NE 1) and (d NE N) then FOUND=TRUE
   }
     i=i+1
   }
output(d)
```

Assume If $x_i \equiv x_j \pmod{p}$ and $x_i \neq x_j$. Then $x_i - x_j \equiv 0 \pmod{p}$. Hence p divides $d = GCD(x_i - x_j, N)$. Therefore $d \neq 1$. Since $x_i, x_j \in [0, N-1]$, $d \neq N$. Hence if $x_i \equiv x_j \pmod{p}$ then the algorithm will terminate.

Look at the sequence $x_1 \bmod p$, $x_2 \bmod p$, …. By the birthday paradox this sequence will almost surely have a repeat before $O(p^{1/2})$ iterations. Hence the run time is almost surely bounded by

$$\sum_{i=1}^{p^{1/2}} \sum_{j=1}^{i-1} \log N \leq \log N \sum_{i=1}^{p^{1/2}} i = O(p) = O(N^{1/2}).$$

That's not better than the trivial algorithm. Oh well.

Also, the algorithm is a space hog.

5 Don't Use Randomization

The reason the last algorithm was a space hog is that it generated random numbers and had to store all of them. Instead we use a deterministic sequence that looks random.

The sequence that begins with a random x_1 and c, and then does $x_i := x_{i-1}^2 + c \pmod{N}$ appears random. This has not been proven (I am not even sure how you would state it); however, it does seem to have the property of repeating within $O(p^{1/2})$ steps.

With this in mind we can write the algorithm which is no longer a space hog but still takes too much time.

```
x_1 = RAND(0,N-1)
c = RAND(0,N-1)
i=2
FOUND = FALSE
while NOT FOUND
   {
     x_i := x_{i-1}^2 + c mod N
     for j=1 to i-1
      {
        compute x_j
        d=GCD(x_i-x_j,N)
        if (d NE 1) and (d NE N) then FOUND=TRUE
      }
        i=i+1
   }
output(d)
```

6 Using Cycle Detection

We plan to generate $x_1, x_2, \ldots$ deterministically. We need to find x_i, x_j such that $x_i \equiv x_j \pmod p$ without storing too much or spending too much time.

We prove a lemma due to Floyd that is interesting in its own right.

Lemma 6.1 *Let $z_1, z_2, z_3, \ldots$ be an infinite sequence. Let m be such that there is some $i \le m$ such that the sequence $z_i, z_{i+1}, \ldots$ is periodic with period $\rho \le m$. Then there exists $a \le 2m$ such that $z_a = z_{2a}$.*

Proof:

Let a be such that $(a-1)\rho \le i < a\rho$. Note that the sequence is $a\rho$-periodic.

Since the sequence is $a\rho$-periodic after z_i we have that, for all $\Delta \ge 0$, $z_{i+\Delta} = z_{i+a\rho+\Delta}$. Plug in $\Delta = a\rho - i$ (note that $a\rho - i \ge 0$ by the case that we are in) to obtain. $z_{a\rho} = z_{2a\rho}$.

How big is $a\rho$? We know that
$a\rho/2 \le (a-1)\rho \le i \le m$, so $a\rho \le 2m$. ∎

We will form two sequences. One will be $x_1, x_2, \ldots$. The other will be $x_2, x_4, \ldots$. Given c we let f_c be the function $f_c(x) \equiv x^2 + c \pmod p$.

```
x = RAND(0,N-1)
c = RAND(0,N-1)
y = f_c(x)
FOUND = FALSE
while NOT FOUND
   {
     x := f_c(x)
     y := f_c(f_c(y))
     d=GCD(x-y,N)
     if (d NE 1) and (d NE N) then FOUND=TRUE
   }
output(d)
```

Consider the sequence $x_1 = x$, $x_i = f_c(x_{i-1})$. Note that the x-sequence is $x_1, x_2, x_3, \ldots$ while the y-sequence is $x_2, x_4, \ldots$. We assume that the sequence has the same properties as a random sequence. Let $z_i = x_i \pmod{p}$. This is also random. By the Birthday paradox it is highly likely that there is a repeat before $O(p^{1/2})$ iterations. By Lemma 6.1 there exists $a \le p^{1/2}$ such that $z_a = z_{2a}$. When this occurs we have $x - y \equiv 0 \pmod{p}$, and hence $d \ne 1$ and $d \ne N$.

With high prob this algorithm takes $O(p^{1/2}) = O(N^{1/4})$ iterations. Each iteration only takes $\log N$ steps. Hence the algorithm takes $O(N^{1/4} \log N)$ steps.