Fast forwarding lrand48()

Dan Piponi

February 2024

A break from abstract nonsense to answer a question I've seen asked online a number of times. It requires nothing more than elementary modular arithmetic and it ends in some exercises.

Given a pseudo-random number generator, say BSD Unix `lrand48()`, is there a quick way to jump forward a billion numbers in the sequence, say, without having to work through all of the intermediate numbers? The method is no secret, but I couldn't find explicit code online so I thought I'd put some here. Literate Haskell of course.

```
{-# LANGUAGE ForeignFunctionInterface #-}
{-# OPTIONS_GHC -fno-warn-missing-methods #-}
```

On MacOSX, if you type 'man lrand48', you'll see the function `lrand48()` returns a sequence of 31 bit non-negative integers defined using the sequence $r_{n+1} = ar_n + c \bmod m$ where

```
a = 25214903917
c = 11
m = 2^48
```

The actual returned value is the floor of $r_n/2^{17}$ and $r_0 = 20017429951246$.

We can compute the nth element in the sequence the hard way by importing `lrand48` and looping n times:

```
foreign import ccall "lrand48" lrand48 :: IO Int

nthrand 1 = lrand48
```

```
nthrand n = lrand48 >> nthrand (n-1)
```

But there is a better way. If we iterate twice we get that $r_{n+2} = a(ar_n + c) + c \bmod m = a^2 r_n + ac + c \bmod m$. Note how two applications of the iteration give you back another iteration in the same form: a multiplication followed by an addition modulo m. We can abstract this a bit. Given two function $f(x) = ax + c \bmod m$ and $g(x) = a'x + c' \bmod m$ we get $g(f(x)) = (a' * a) * x + a' * c + c' \bmod m$. We can represent functions of this type using a simple Haskell type:

```
data Affine = Affine { multiply :: Integer, add :: Integer }
                     deriving (Show, Eq, Ord)
```

We can now write a function to compose these functions. I'm going to use the operator `*` to represent composition:

```
instance Num Affine where
   Affine a' c' * Affine a c = Affine (a'*a `mod` m)
                                      ((a'*c+c') `mod` m)
```

To skip forward n steps we just need to multiply n of these together, ie. raise `Affine a c` to the power of n using `^`. We then need to apply this function to r_0:

```
initial = Affine 0 20017429951246

nthrand' n = (add $ Affine a c ^ n * initial) `div` (2^17)
```

Now try firing up ghci and comparing the outputs of `nthrand 1000000` and `nthrand' 1000000`. Don't run `nthrand` more than once without resetting the seed, eg. by restarting ghci. (I know someone will post a reply below that it doesn't work...)

There are lots of papers on how to do this with other kinds of random number generator. My example is probably the easiest. The main application I can see is for jumping straight to that annoying regression test failure without going through all of the intermediates.

Exercises. 1. Read the corresponding man page for Linux. Port the above code to work there. 2. Can you split `lrand48()` into two? Ie. can you make two random generators that produce sequences s_i and t_i so that $s_0, t_0, s_1, t_1, ...$ form the sequence given by `lrand48()`. 3. I've neglected to

mention some special sauce in the code above. Why does it actually run so fast? (Clue: why did I use `Num`?)