99 questions/Solutions/39

From HaskellWiki

< 99 questions | Solutions

(*) A list of prime numbers.

Given a range of integers by its lower and upper limit, construct a list of all prime numbers in that range.

Contents

1 Solution 1.

```
primesR :: Integral a => a -> a -> [a]
primesR a b | even a = filter isPrime [a+1,a+3..b]
            | True   = filter isPrime [a,a+2..b]
```

If we are challenged to give all primes in the range between a and b we simply take all numbers from a up to b and filter all the primes through.

This is good for *very narrow ranges* as Q.31's `isPrime` tests numbers by *trial division* using (up to $\sqrt{b}$) a memoized primes list produced by sieve of Eratosthenes to which it refers internally. So it'll be slower, but immediate, testing the numbers one by one.

2 Solution 2.

For *very wide* ranges, specifically when $a < \sqrt{b}$, we're better off just using the primes sequence itself, without any post-processing:

```
primes :: Integral a => [a]
primes = primesTME               -- of Q.31
```

```
primesR :: Integral a => a -> a -> [a]
primesR a b = takeWhile (<= b) $ dropWhile (< a) primes
```

3 Solution 3.

Another way to compute the claimed list is done by using the *Sieve of Eratosthenes*.

```
primesR :: Integral a => a -> a -> [a]
primesR a b = takeWhile (<= b) $ dropWhile (< a) $ sieve [2..]
  where sieve (n:ns) = n:sieve [ m | m <- ns, m `mod` n /= 0 ]
```

The `sieve [2..]` function call generates a list of all (!) prime numbers using this algorithm and `primesR` filters the relevant range out. [But this way is very slow and I only presented it because I wanted to show how nicely the *Sieve of Eratosthenes* can be implemented in Haskell :)]

this is of course the famous case of (mislabeled) executable specification, with all the implied pitfalls of inefficiency when (ab)used as if it were an actual code.

4 Solution 4.

Use the *proper* Sieve of Eratosthenes from e.g. 31st question's solution (http://www.haskell.org/haskellwiki/99_questions/Solutions/31) (instead of the above sieve of Turner), adjusted to start its multiples production from the given starting point:

```
-- tree-merging Eratosthenes sieve, primesTME of Q.31,
--  adjusted to produce primes in a given range (inclusive)
primesR a b | b < a || b < 2 = []
            | otherwise      = takeWhile (<= b) $ primesFrom a

primesFrom a0 = (if a0 <= 2 then [2] else []) ++
                (gaps a $ mults $ span (< z) $ tail primesTME)
  where
    a = snap (max 3 a0) 3 2
    z = ceiling $ sqrt $ fromIntegral a + 1       -- p<z => p*p<=a
    snap v origin step = if r==0 then v else v+(step-r)
        where r = rem (v-origin) step   -- NB: origin <= v ; else use MOD

    mults (h,p':t) =                              -- p'>=z => p'*p'>a
      join union ( [[x,x+s..] | p <- h,           -- heads unordered
                         let s=2*p; x=snap a (p*p) s]
                   ++ [[p'*p',p'*p'+2*p'..]] )
      `union'` join union' [[p*p,p*p+2*p..] | p <- t]

    join  f (xs:t)    = f xs (join f (pairs f t))
    join  f []        = []
    pairs f (xs:ys:t) = f xs ys : pairs f t
    pairs f t         = t
    union' (x:xs) ys  = x : union xs ys          -- `union` of Q.31
    gaps k xs@(x:t) | k==x  = gaps (k+2) t
                    | True  = k : gaps (k+2) xs
```

It should be much better then taking a slice of a full sequential list of primes, as it won't try to generate any primes between the *square root of b* and *a*. To wit,

```
> primesR 10100 10200                                          -- Sol.4
[10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,10181,10193]
(4,776 reductions, 11,559 cells)

> takeWhile (<= 10200) $ dropWhile (< 10100) $ primesTME         -- Sol.2
[10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,10181,10193]
(140,313 reductions, 381,058 cells)

> takeWhile (<= 10200) $ dropWhile (< 10100) $ sieve [2..]       -- Sol.3
     where sieve (n:ns) = n:sieve [ m | m <- ns, m `mod` n /= 0 ]
[10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,10181,10193]
(54,893,566 reductions, 79,935,263 cells, 6 garbage collections)

> filter isPrime [10101,10103..10200]                            -- Sol.1
[10103,10111,10133,10139,10141,10151,10159,10163,10169,10177,10181,10193]
(12,927 reductions, 24,703 cells)                        -- isPrime: Q.31
```

(testing with Hugs of Nov 2002).

This solution is potentially much faster but not immediate. It has a certain preprocessing stage but then goes on fast to produce the whole range. To illustrate, it takes about 18 seconds on my oldish notebook for the 1st version to produce the 49 primes in 1000-wide range above 120200300100, with the first number produced almost immediately (~ 0.4 sec); but *this* version spews out all 49 primes at once after just under 1 sec.
