99 questions/Solutions/35

From HaskellWiki

< 99 questions | Solutions

(**) Determine the prime factors of a given positive integer. Construct a flat list containing the prime factors in ascending order.

```
primeFactors :: Integer -> [Integer]
primeFactors a = let (f, f1) = factorPairOf a
                     f' = if prime f then [f] else primeFactors f
                     f1' = if prime f1 then [f1] else primeFactors f1
                 in f' ++ f1'
 where
 factorPairOf a = let f = head $ factors a
                  in (f, a `div` f)
 factors a    = filter (isFactor a) [2..a-1]
 isFactor a b = a `mod` b == 0
 prime a      = null $ factors a
```

Kind of ugly, but it works, though it may have bugs in corner cases. This uses the factor tree method of finding prime factors of a number. factorPairOf picks a factor and takes it and the factor you multiply it by and gives them to primeFactors. primeFactors checks to make sure the factors are prime. If not it prime factorizes them. In the end a list of prime factors is returned.

Another possibility is to observe that you need not ensure that potential divisors are primes, as long as you consider them in ascending order:

```
primeFactors n = primeFactors' n 2
  where
    primeFactors' 1 _ = []
    primeFactors' n f
      | n `mod` f == 0 = f : primeFactors' (n `div` f) f
      | otherwise      = primeFactors' n (f + 1)
```

Thus, we just loop through all possible factors and add them to the list if they divide the original number. As the primes get farther apart, though, this will do a lot of needless checks to see if composite numbers are prime factors. However we can stop as soon as the candidate factor exceeds the square root of n:

```
primeFactors n = primeFactors' n 2
  where
    primeFactors' n f
      | f*f > n        = [n]
      | n `mod` f == 0 = f : primeFactors' (n `div` f) f
      | otherwise      = primeFactors' n (f + 1)
```

You can avoid the needless work by just looping through the primes:

```haskell
primeFactors n = factor primes n
  where
    factor ps@(p:pt) n | p*p > n      = [n]
                       | rem n p == 0 = p : factor ps (quot n p)
                       | otherwise    =     factor pt n
    -- primes = 2 : filter (\n-> n==head(factor primes n)) [3,5..]
    -- primes = 2 : filter isPrime [3,5..]      -- isPrime of Q.31
    primes = primesTME
```

Using the proper tree-merging sieve of Eratosthenes version from a solution to question 31 (http://www.haskell.org/haskellwiki/99_questions/Solutions/31) instead of a trial division, speeds it up a lot (especially as memoized). Or you can find it on prime numbers haskellwiki page.

Here's a concise alternative:

```haskell
factor :: Integer -> [Integer]

factor 1 = []
factor n = let prime = head $ dropWhile ((/= 0) . mod n) [2 .. n]
           in (prime :) $ factor $ div n prime
```

And here's an improved version of the previous algorithm, which is slightly more verbose but only checks for primality up to (ceiling $ sqrt n); and is thus much faster.

```haskell
factor 1 = []
factor n = let divisors = dropWhile ((/= 0) . mod n) [2 .. ceiling $ sqrt $ fromIntegral n]
           in let prime = if null divisors then n else head divisors
              in (prime :) $ factor $ div n prime
```


Category:

- Programming exercise spoilers
