Problem 001 - Multiples of 3 or 5

Oliver Krischer

October 2021

Contents

1 Problem 001

If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. **Find the sum of all the multiples of 3 or 5 below 1000!**

1.1 Naive solution based on list comprehension

```
import Criterion.Main
sumMultiplesNaive :: Integer -> Integer
sumMultiplesNaive n =
    sum [x | x <- [1..n-1], x `rem` 3 == 0 || x `rem` 5 == 0]
```

The runtime complexity of this algorithm is linear to the input size n, thus $\mathcal{O}(n)$.

1.2 Improved solution using triangular numbers

The starting point for developing an efficient solution is the following idea: instead of checking if the target value is divisible by 3 and 5, we can check separately for division of 3 and 5 and add the results. But then we have to subtract the sum of numbers divisible by `15 (= 3 * 5)`, as we have counted them twice in the first step. When we define a function `sumDivisibleBy :: Int -> Int -> Int`, we can express the result like so:

```haskell
sumMultiplesOptim :: Integer -> Integer
sumMultiplesOptim  n = divBy3 + divBy5 - divBy15
  where divBy3  = sumDivisibleBy 3 n
        divBy5  = sumDivisibleBy 5 n
        divBy15 = sumDivisibleBy 15 n
```

If we apply our naive implementation on `sumDivisibleBy` for 3 and 5 we would then get:

$$3 + 6 + 9 + 12 + \dots + 999 = 3 * (1 + 2 + 3 + 4 + \dots + 333)$$
$$5 + 10 + 15 + \dots + 995 = 5 * (1 + 2 + 3 + \dots + 199)$$

Thus, we can apply the equation for *Triangular Numbers* (1)

$$T_n = \sum_{k=1}^{n} k = 1 + 2 + 3 + \dots + n = \frac{n * (n + 1)}{2}$$

on our function and we get:

```haskell
sumDivisibleBy :: Integer -> Integer -> Integer
sumDivisibleBy factor limit =
  let n = (limit - 1) `div` factor
  in factor * (n*(n+1)) `div` 2
```

Since `sumDivisibleBy` represents a closed formula, the runtime complexity of this algorithm is constant, thus $\mathcal{O}(1)$.

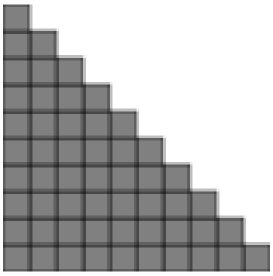
1.3 Triangular Numbers

1.3.1 Theorem

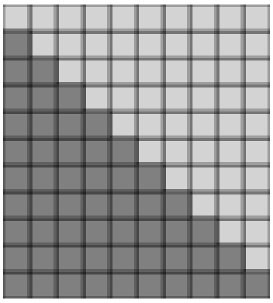
$$T_n = \sum_{k=1}^{n} k = \frac{n(n + 1)}{2} \tag{1}$$

1.3.2 Proof

Triangular numbers are formed by stacking rows of the first n integers, creating a *triangular* geometric pattern, e.g. for $n = 10$:



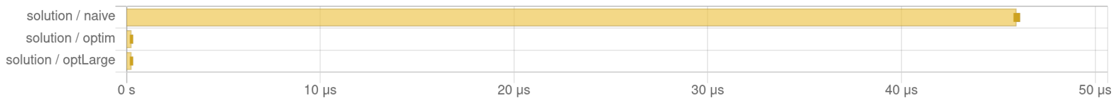
It's easy to see that the number of elements in such a *triangle* is the sum of the integers from 1 to n. If we now geometrically combine two copies of T_n, the resulting rectancle will have side lengths of n and $n + 1$:



Thus, the number of elements in this rectangle is $n(n + 1)$. Since such a rectangle has the double size of the underlying *triangular number*, the size of the *triangular number* is: $\frac{n(n+1)}{2}$ ■

1.4 Benchmarking the solutions

```
main = defaultMain [
  bgroup "solution" [ bench "naive"    $ whnf sumMultiplesNaive 1000
                    , bench "optim"    $ whnf sumMultiplesOptim 1000
                    , bench "optLarge" $ whnf sumMultiplesOptim 1000000
                    ]
  ]
```


solution / naive
solution / optim
solution / optLarge
0 s
10 μs
20 μs
30 μs
40 μs
50 μs