Juraj Vetrák
3AIN

Prémia Horská

Programovacie paradigmy
17. novembra 2019

```
:load Horska.hs
```

Hneď na začiatku som pochopil, že na to nemôžem ísť jednoduchým `list comprehension`, tak som skúšal rôzne taktiky budovania podzoznamov s násobkami `[2..9]` predošlých prvkov, potom sortovať a vybrať daný prvok. Nič nefungovalo, alebo ak áno, už pri hľadaní stého najmenšieho prvku som čakal do ďalšej doby ľadovej.

Začal som teda googliť, v zmysle 'dynamically build set haskell' a nejako som sa preklikal na tento link a začal skúmať, čo to tam riešia. Snažili sa vytvoriť množinu Hammingových čísel, ktorá obsahuje `1` a `2x`, `3x`, `5x`, kde `x` je Hammingovo číslo. vyskúšal som to a uvedomil som si, že je to úplne rovnaký princíp, ktorý potrebujem aj ja.

Kód je absolútne jednoduchý, teda som ho vedel hneď pochopiť a bez výčitiek implementovať, pričom je očividne aj efektívny. Kľúčom je funkcia `mergeUnique`, ktorá ako názov napovedá, spája dva zoznamy, pričom sa zároveň postará aj o duplicitné prvky (vynechá ich). Najprv som si myslel, že je súčasťou nejakého `Haskell` modulu, no trebalo ju doimplementovať (takto).

```
mergeUnique :: [Integer] -> [Integer] -> [Integer]
mergeUnique a  [] = a
mergeUnique [] b  = b
mergeUnique (a:as) (b:bs) = case compare a b of
                                EQ -> a: mergeUnique as bs
                                LT -> a: mergeUnique as (b:bs)
                                GT -> b: mergeUnique (a:as) bs
```

Vďaka tejto funkcii a `Haskell`ovej lenivosti si teraz vieme vytvoriť našu množinu takto zaujímavo, dáme mu akýsi návod, ako si vypočítať potrebný počet prvkov:

```
nums :: [Integer]
nums = 1 : mergeUnique (map (2*) nums)
            (mergeUnique (map (3*) nums)
                (mergeUnique (map (4*) nums)
                    (mergeUnique (map (5*) nums)
                        (mergeUnique (map (6*) nums)
                            (mergeUnique (map (7*) nums)
                                (mergeUnique (map (8*) nums) (map (9*) nums)))))))
```

Následne mu už len cez index povieme, ktorý prvok chceme získať:

```
horska :: Int -> Integer
horska n = nums !! (n-1)
```

Benchmarky sú nasledovné, porovnal som aj s benchmarkom v zadaní, tak je dôvod tomuto programu veriť.

```
*Horska> horska 1000
385875
(0.01 secs, 2,500,808 bytes)

*Horska> horska 1000
385875
(0.00 secs, 117,664 bytes)

*Horska> horska 10000
63221760000
(0.05 secs, 22,332,088 bytes)

*Horska> horska 100000
123093144973968750000
(0.48 secs, 230,867,416 bytes)

*Horska> horska 1000000
4157409948433216829957008507500000000
(3.91 secs, 2,574,677,928 bytes)

*Horska> horska 10000000
10377545069293932758463691948176758121650528085724609912441405440000
(39.43 secs, 26,789,343,112 bytes)

*Horska> horska 5
5
(0.00 secs, 113,984 bytes)

*Horska> horska 55
140
(0.01 secs, 222,096 bytes)

*Horska> horska 555
46875
(0.01 secs, 1,307,496 bytes)

*Horska> horska 5555
1475789056
(0.03 secs, 12,406,472 bytes)
```

```
*Horska> horska 55555
154414312500000000
(0.31 secs, 124,490,208 bytes)

*Horska> horska 555555
2866236871958278963200000000000
(2.31 secs, 1,418,698,456 bytes)

*Horska> horska 5555555
68625796024598400000000000000000000000000000000000000000000000
(21.65 secs, 14,653,399,784 bytes)
```