Desafío: Cuarentena Funcional

Programación Funcional - Curso 2020

En medio de la cuarentena, un estudiante de Progamación Funcional recibió el siguiente problema lógico en un grupo de WhatsApp:

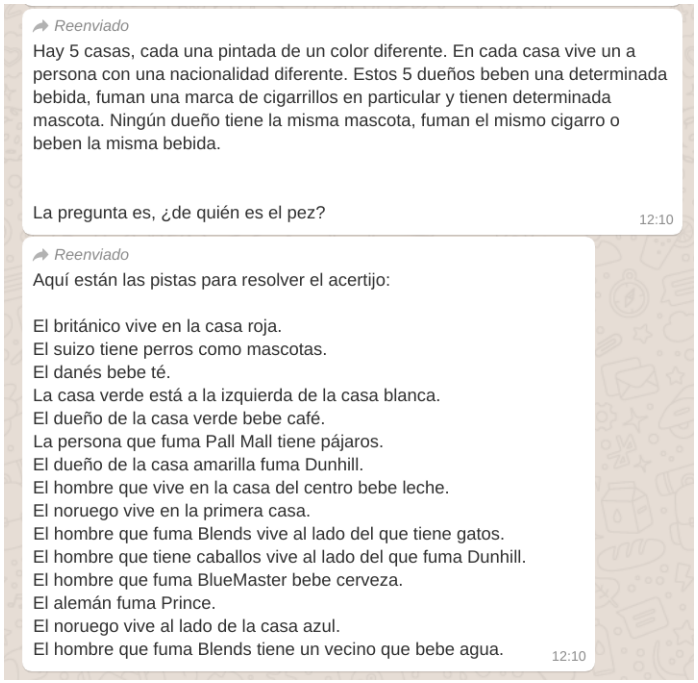

Figure 1: El problema

Como no tenía muchas ganas de pensar, y además se quería hacer el geek ante sus amigos, decidió implementar la solución en Haskell. Luego de un buen rato de pensar el problema[1], de diseñar la solución, repasar las slides y pelearse bastante con el compilador, llegó a un código que lo dejó muy contento, porque resuelve el problema de forma muy elegante y declarativa (ver Figura 2). Terrible fue su decepción, cuando puso a ejecutar el programa. Pasó un rato y no terminaba. Se aprontó el mate y no terminaba. Se puso a hacer el práctico 3 y no terminaba. Se miró todo un teórico en el chat y seguía sin terminar...

[1] Sí, tuvo que pensar.

Hasta que luego de un buen rato obtuvo la solución que buscaba, en un tiempo bastante peor del que se imaginaba:

```
real    319m34,886s
user    317m33,401s
sys     1m26,598s
```

Después de maldecir un rato, echarle las culpas al paradigma, al lenguaje, a los profesores, al coronavirus, y decirse cosas como "¡yo era feliz programando en Java!" o "¿por qué no habré hecho Programación Lógica?", decidió sentarse otro rato frente a la computadora y repensar la solución. Así fue que luego de un momento de iluminación obtuvo una solución que también es elegante, pero que cumple la tarea en un tiempo casi despreciable:

```
real    0m0,010s
user    0m0,010s
sys     0m0,000s
```

El desafío que les planteamos es competir con este estudiante y programar también una solución que sea: buena, bonita y barata. Es decir, que al menos sea fácil de leer y que no se coma todos los recursos de la máquina.

```
import Data.List
import Data.Maybe
data Color    = Roja | Verde | Blanca | Amarilla | Azul
  deriving (Show, Eq, Enum)
data Nacion   = Brit | Suizo | Danes | Noruego | Aleman
  deriving (Show, Eq, Enum)
data Bebe     = Te | Cafe | Leche | Cerveza | Agua
  deriving (Show, Eq, Enum)
data Fuma     = Pall | Dunhill | Blends | BlueMaster | Prince
  deriving (Show, Eq, Enum)
data Mascota = Perro | Pajaro | Gato | Caballo | Pez
  deriving (Show, Eq, Enum)
data Barrio   = Barrio [Color] [Nacion] [Bebe] [Fuma] [Mascota]
  deriving Show

numero e = (+1) . fromJust . elemIndex e

color    e (Barrio cs _ _ _ _)  = numero e cs
nacion   e (Barrio _ ns _ _ _)  = numero e ns
bebe     e (Barrio _ _ bs _ _)  = numero e bs
fuma     e (Barrio _ _ _ fs _)  = numero e fs
mascota e (Barrio _ _ _ _ ms) = numero e ms

casas = [Barrio cs ns bs fs ms
        | cs <- permutations . enumFrom $ Roja
        , ns <- permutations . enumFrom $ Brit
        , bs <- permutations . enumFrom $ Te
        , fs <- permutations . enumFrom $ Pall
        , ms <- permutations . enumFrom $ Perro]

sols = [b | b <- casas
        , nacion Brit    b == color    Roja      b
        , nacion Suizo   b == mascota  Perro     b
        , nacion Danes   b == bebe     Te        b
        , color  Verde   b == color    Blanca    b - 1
        , color  Verde   b == bebe     Cafe      b
        , fuma   Pall    b == mascota  Pajaro    b
        , color  Amarilla b == fuma    Dunhill b
        , bebe   Leche     b == 3
        , nacion Noruego  b == 1
        , next (fuma Blends b)  (mascota Gato b)
        , next (fuma Dunhill b) (mascota Caballo b)
        , fuma   BlueMaster b == bebe  Cerveza b
        , nacion Aleman      b == fuma Prince  b
        , next (nacion Noruego b) (color Azul b)
        , next (fuma Blends b)    (bebe Agua b)]

next x y = abs (x - y) == 1

main = print $ (naciones solucion) !! (mascota Pez solucion - 1)
where solucion = head sols
      naciones (Barrio _ ns _ _ _) = ns
```

Figure 2: Solución elegante e ineficiente