TP Bestani Tornielli

este documento tiene todos los comentarios que nos ayudaron a entender las diferentes funciones, y justificaciones de simplificaciones de código que resultamos entender más que la manera larga de hacerlo.

Point.hs

```haskell
module Point ( Point, newP, difP)
       where

data Point = Poi Int Int deriving (Eq, Show)

newP :: Int -> Int -> Point
newP x y = Poi x y

difP :: Point -> Point -> Float -- distancia absoluta
-- difP (Poi x1 y1) (Poi x2 y2) = sqrt (fromIntegral (x1-x2)^2 + fromIntegral
(y1-y2)^2)
difP (Poi x1 y1) (Poi x2 y2) = sqrt $ fromIntegral (x1-x2)^2 + fromIntegral
(y1-y2)^2
```

City.hs

```haskell
module City ( City, newC, nameC, distanceC )
       where

import Point

data City = Cit String Point deriving (Eq, Show)

newC :: String -> Point -> City
newC name point = Cit name point

nameC :: City -> String
nameC (Cit name _) = name

distanceC :: City -> City -> Float
distanceC (Cit _ point1) (Cit _ point2) = difP point1 point2
```

Quality.hs

```haskell
module Quality ( Quality, newQ, capacityQ, delayQ )
       where

data Quality = Qua String Int Float deriving (Eq, Show)

newQ :: String -> Int -> Float -> Quality
{-- newQ tipo cap demora = Qua tipo cap demora --}

newQ = Qua
-- esto es lo mismo que la línea anterior, pero usando currying
-- newQ "fibra" 100 0.1
-- Qua "fibra" 100 0.1
-- newQ "cobre" 10 0.5
-- Qua "cobre" 10 0.5


capacityQ :: Quality -> Int -- cuantos túneles puede tolerar esta conexión
capacityQ (Qua _ cap _) = cap

delayQ :: Quality -> Float -- la demora por unidad de distancia que sucede en las
conexiones de este canal
delayQ (Qua _ _ demora) = demora
```

Link.hs

```haskell
module Link ( Link, newL, linksL, connectsL, capacityL, delayL )
       where

import Quality
import City


data Link = Lin City City Quality deriving (Eq, Show)

newL :: City -> City -> Quality -> Link -- genera un link entre dos ciudades
distintas
newL c1 c2 q | c1 == c2 = error "No se puede vincular una ciudad consigo misma" --
CHECKEO QUE NO PERMITE VINCULAR UNA CIUDAD CONSIGO MISMA
             | otherwise = Lin c1 c2 q

connectsL :: City -> Link -> Bool -- indica si esta ciudad es parte de este link
connectsL ciudad (Lin ciudad1 ciudad2 _) = ciudad == ciudad1 || ciudad == ciudad2

linksL :: City -> City -> Link -> Bool -- indica si estas dos ciudades distintas
estan conectadas mediante este link
linksL ciudad1 ciudad2 (Lin ciudad3 ciudad4 _) = (ciudad1 == ciudad3 && ciudad2 ==
ciudad4) || (ciudad1 == ciudad4 && ciudad2 == ciudad3)

capacityL :: Link -> Int
capacityL (Lin _ _ calidad) = capacityQ calidad

delayL :: Link -> Float -- la demora que sufre una conexion en este canal
delayL (Lin _ _ calidad) = delayQ calidad


-- Ejemplo de uso:

-- newL "Buenos Aires" "Rosario" (newQ "fibra" 100 0.1)
-- Lin "Buenos Aires" "Rosario" (Qua "fibra" 100 0.1)
```

```
-- Un Link es la conexión física que podemos establecer entre dos puntos, es
bidireccional
{-
{ Un Link es naturalmente bidireccional, Si las ciudades A y B están enlazadas por
un link
li, linksL A B li y linksL B A li es true }

{ Un Link no puede enlazar una ciudad consigo misma, es decir, linksL A A li es
false }

{ Un Link no puede enlazar dos veces la misma ciudad, es decir, si linksL A B li es
true, entonces linksL B A li es false }

{ Un Link no puede enlazar dos ciudades que ya están enlazadas por otro link, es
decir, si linksL A B li1 es true, entonces linksL A B li2 es false }
-}




{- Un Link tiene una capacidad, que es la cantidad de túneles que puede soportar. -
Un Link tiene una demora, que es la demora por unidad de distancia que sucede en
las conexiones de este link. -}

{-
{ esta demora es en unidades de tiempo }
{ La demora de un link es en tiempo, segundos, milisegundos, etc.
La demora de la calidad de un enlace es en velocidad, por ejemplo km/segundo.
No importan las unidades, sí la relación entre los valores de demora de los links y
la calidad de los enlaces.}
-}
```

Tunel.hs

```haskell
module Tunel ( Tunel, newT, connectsT, usesT, delayT )
       where

import Link
import City

data Tunel = Tun [Link] deriving (Eq, Show)

newT :: [Link] -> Tunel
newT = Tun

connectsT :: City -> City -> Tunel -> Bool -- inidca si este tunel conceta estas dos ciudades distintas
connectsT ciudad1 ciudad2 (Tun links) = any (linksL ciudad1 ciudad2) links -- any hace un or entre todos los elementos de la lista (devuelve true si alguno es true)

-- ojo con esta linea: (ver si es necesaria)
{- connectsT ciudad1 ciudad2 (Tun links) = any (linksL ciudad1 ciudad2) links -}

usesT :: Link -> Tunel -> Bool -- indica si este tunel atraviesa ese link
usesT link (Tun links) = elem link links
-- elem link links hace que devuelva true si el link que le pasamos esta en la lista de links del tunel
-- any(== link) hace que devuelva true si alguno de los links es igual al link que le pasamos

delayT :: Tunel -> Float -- la demora que sufre una conexion en este tunel
delayT (Tun links) = sum (map delayL links) -- map es una funcion que aplica una funcion a todos los elementos de una lista y devuelve una lista con los resultados


-- Ejemplo de uso:

-- newT [newL "Buenos Aires" "Rosario" (newQ "fibra" 100 0.1)]
-- Tun [Lin "Buenos Aires" "Rosario" (Qua "fibra" 100 0.1)]


-- Un Tunel es la conexión lógica que podemos establecer entre dos puntos, también es bidireccional

-- Un Tunel no puede conectar una ciudad consigo misma, es decir, connectsT A A tunel es false
```

```
-- dadas dos ciudades esta función da si si las ciudades son los extremos del túnel

-- Un túnel recorre una serie de uno o más links, esta función indica si el link
consultado es parte de esa serie

-- delayT: esta demora es en unidades de tiempo, cuanto demora la información en
recorrer el túnel.

-- (chat) La demora de un túnel es la suma de las demoras de los links que lo
componen. La demora de un link es la demora de la calidad de ese link. La demora de
la calidad de un enlace es en velocidad, por ejemplo km/segundo. No importan las
unidades, sí la relación entre los valores de demora de los links y la calidad de
los enlaces.
```

Region.hs

```
module Region ( Region, newR, foundR, linkR, tunelR, connectedR, linkedR, delayR,
availableCapacityForR )
      where

import City
import Link
import Tunel
import Quality

data Region = Reg [City] [Link] [Tunel]
newR :: Region
newR = Reg [] [] []

foundR :: Region -> City -> Region -- agrega una nueva ciudad a la región
foundR (Reg cities links tunnels) city = Reg (city:cities) links tunnels

linkR :: Region -> City -> City -> Quality -> Region -- enlaza dos ciudades de la
región con un enlace de la calidad indicada
linkR (Reg cities links tunnels) city1 city2 quality = Reg cities (newL city1 city2
quality:links) tunnels

tunelR :: Region -> [ City ] -> Region -- genera una comunicación entre dos
ciudades distintas de la región
{- no funciona
tunelR (Reg cities links tunnels) cities = Reg cities links (newT cities:tunnels)
where
```

```haskell
newT cities = Tun cities -}
tunelR (Reg cities links tunnels) [city1, city2]
| city1 /= city2 && elem city1 cities && elem city2 cities = Reg cities links
(newTunnel : tunnels)
| otherwise = Reg cities links tunnels
where
newTunnel = Tunel city1 city2

connectedR :: Region -> City -> City -> Bool -- indica si estas dos ciudades estan
conectadas por un tunel

connectedR (Reg cities links tunnels) city1 city2 = any (== [city1, city2]) (map
citiesT tunnels)
-- acá el any hace un or entre todos los elementos de la lista (devuelve true si
alguno es true). El map hace que se aplique citiesT a todos los elementos de la
lista tunnels y devuelve una lista con los resultados de aplicar citiesT a cada
elemento de la lista tunnels (en este caso, devuelve una lista de listas de
ciudades) y el any chequea si alguna de esas listas es igual a [city1, city2] (si
es igual, devuelve true)

-- otra forma: sin map
-- connectedR (Reg _ _ tunnels) ciudad1 ciudad2 = any (\(Tunel c1 c2) -> (c1 ==
ciudad1 && c2 == ciudad2) || (c1 == ciudad2 && c2 == ciudad1)) tunnels

linkedR :: Region -> City -> City -> Bool -- indica si estas dos ciudades estan
enlazadas
linkedR (Reg cities links tunnels) city1 city2 = any (== [city1, city2]) (map
citiesL links)

delayR :: Region -> City -> City -> Float -- dadas dos ciudades conectadas, indica
la demora
delayR (Reg cities links tunnels) city1 city2 = delayT (head (filter (== [city1,
city2]) (map citiesT tunnels))) -- filter devuelve una lista con los elementos que
cumplen la condición (en este caso, devuelve una lista con el tunel que conecta
city1 y city2) y head devuelve el primer elemento de la lista (en este caso,
devuelve el tunel que conecta city1 y city2)

availableCapacityForR :: Region -> City -> City -> Int -- indica la capacidad
disponible entre dos ciudades
availableCapacityForR (Reg cities links tunnels) city1 city2 = minimum (map
capacityL (filter (== [city1, city2]) (map citiesL links))) -- filter devuelve una
lista con los elementos que cumplen la condición (en este caso, devuelve una lista
con el link que conecta city1 y city2) y map capacityL devuelve una lista con las
capacidades de los links que conectan city1 y city2 y minimum devuelve el mínimo de
esa lista
```

Test.hs

```haskell
import Tunel
import Link
import Point
import Quality
import City
import Region

point1 = newP 0 0
point2 = newP 4 9
point3 = newP 10 4
point4 = newP 4 4

city1 = newC "GaiaLandia" point1
city2 = newC "NachoLandia" point2
city3 = newC "UdeSA City" point3
city4 = newC "Victoria" point4

metal = newQ "metal" 2 0.2

gaiaToVictoria = newL city1 city2 metal
victoriaToNacho = newL city2 city3 metal
tunel = newT [gaiaToVictoria, victoriaToNacho]

argentina = linkR ( foundR ( foundR (foundR newR city1) city2) city3) city1 city2
metal

test = [difP point1 point2 == 10.0,
       delayT tunel == 1.0 ,
       nameC city1 == "GaiaLandia",
       nameC city2 == "NachoLandia",
       distanceC city1 city2 == 12.0,
       capacityL gaiaToVictoria == 2,
       capacityQ metal == 2,
       delayQ metal == 0.2,
       delayL gaiaToVictoria == 0.2,
       connectsL city2 gaiaToVictoria,
       connectsT city1 city2 tunel,
       linksL city4 city2 victoriaToNacho,
       linksL city1 city4 gaiaToVictoria,
       usesT victoriaToNacho tunel,
       delayT tunel == 1.0,
       True
       ]
```