Gravity Falls - Un parcial de misterios

Objetivos y recomendaciones

- Resolver los problemas planteados aprovechando los conceptos del paradigma funcional, en particular **deben haber usos interesantes de orden superior, aplicación parcial y composición**.
- Generar buenas abstracciones y evitar la repetición de lógica.
- **Sólo usar recursividad cuando se indique**, en pos de favorecer soluciones declarativas.
- Las dos partes del parcial son independientes entre sí, vale empezar por la parte que más les guste. Es recomendable **leer la parte completa antes de arrancar la solución**, ya que podría ayudar a entender el problema y evitar complicaciones al tomar decisiones inconvenientes de modelado.
- **¡No se traben!** Vale saltear un punto que les esté costando y usarlo en uno siguiente como si estuviera resuelto. Para la primera parte, que es más abierta, alcanza con explicitar el tipo de la función que saltean y dejarla sin definir. Siempre pueden volver sobre eso más adelante, luego de avanzar con lo demás.

Primera parte: Las Rarezas

En el pueblo de Gravity Falls, Oregon, siempre han pasado cosas muy raras. Esto llamó la atención de un investigador de identidad desconocida, quien dedicó muchos años de su vida a registrar estas peculiaridades en una serie de diarios.

Mucho después, Dipper Pines, un pre adolescente amante de los misterios y conspiraciones, encontró uno de estos diarios repleto de información sobre criaturas que el autor encontró en las cercanías del pueblo, varias de las cuales son muy peligrosas, y quiere usar esta información para poder defenderse.

Sabemos que existen distintas criaturas, de las cuales nos interesa poder determinar cuál es su nivel de peligrosidad y qué tiene que cumplir una persona para deshacerse de ellas. En principio queremos contemplar las siguientes criaturas:

- El **siempredetras**: la peligrosidad de esta criatura legendaria es 0, ya que no le hace nada a la persona que está acechando, es tan inofensivo que nunca nadie pudo afirmar que estaba siendo acechado. Sin embargo, no hay nada que se pueda hacer para que te deje en paz.
- Los **gnomos**: individualmente son inofensivos, pero se especializan en atacar en grupo. La peligrosidad es 2 elevado a la cantidad de gnomos agrupados. Una persona puede deshacerse de un grupo de gnomos si tiene un soplador de hojas entre sus ítems.
- Los **fantasmas**: se categorizan del 1 al 10 dependiendo de qué tan poderosos sean, y el nivel de peligrosidad es esa categoría multiplicada por 20. Cada fantasma tiene un asunto pendiente distinto, con lo cual se debe indicar para cada uno qué tiene que cumplir la persona para resolver su conflicto.

1. Modelar a las personas, de las cuales nos interesa la edad, cuáles son los ítems que tiene y la cantidad de experiencia que tiene; y a las criaturas teniendo en cuenta lo descrito anteriormente, y lo que queremos hacer en el punto siguiente.
2. Hacer que una persona se enfrente a una criatura, que implica que si esa persona puede deshacerse de ella gane tanta experiencia como la **peligrosidad** de la criatura, o que se escape (que le suma en 1 la experiencia, porque de lo visto se aprende) en caso de que no pueda deshacerse de ella.
3.
    a. Determinar cuánta experiencia es capaz de ganar una persona luego de enfrentar sucesivamente a un grupo de criaturas.
    b. Mostrar un ejemplo de consulta para el punto anterior incluyendo las siguientes criaturas: al siempredetras, a un grupo de 10 gnomos, un fantasma categoría 3 que requiere que la persona tenga menos de 13 años y un disfraz de oveja entre sus ítems para que se vaya y un fantasma categoría 1 que requiere que la persona tenga más de 10 de experiencia.

Segunda parte: Mensajes ocultos

Luego de inspeccionar mejor el diario, Dipper notó que algunas frases que parecían no tener sentido, en realidad eran mensajes para descifrar. Si bien podría intentar hacerlo a mano, sería muy conveniente automatizar esta tarea.

1) Definir recursivamente la función:
`zipWithIf :: (a -> b -> b) -> (b -> Bool) -> [a] -> [b] -> [b]`
que a partir de dos listas retorne una lista donde cada elemento:
- se corresponda con el elemento de la segunda lista, en caso de que el mismo no cumpla con la condición indicada
- en el caso contrario, debería usarse el resultado de aplicar la primer función con el par de elementos de dichas listas

Sólo debería avanzarse sobre los elementos de la primer lista cuando la condición se cumple.

```
> zipWithIf (*) even [10..50] [1..7]
[1,20,3,44,5,72,7] ← porque [1, 2*10, 3, 4*11, 5, 6*12, 7]
```

2) Notamos que la mayoría de los códigos del diario están escritos en código **César**, que es una simple sustitución de todas las letras por otras que se encuentran a la misma distancia en el abecedario. Por ejemplo, si para encriptar un mensaje se sustituyó la **a** por la **x**, la **b** por la **y**, la **c** por la **z**, la **d** por la **a**, la **e** por la **b**, etc.. Luego el texto "**`jrzel zrfaxal!`**" que fue encriptado de esa forma se desencriptaría como "**`mucho cuidado!`**".

a. Hacer una función **`abecedarioDesde :: Char -> [Char]`** que retorne las letras del abecedario empezando por la letra indicada. O sea, **`abecedarioDesde 'y'`** debería retornar **`'y':'z':['a' .. 'x']`**.
b. Hacer una función **`desencriptarLetra :: Char -> Char -> Char`** que a partir una letra clave (la que reemplazaría a la **a**) y la letra que queremos desencriptar, retorna la letra que se corresponde con esta última en el abecedario que empieza con la letra clave. Por ejemplo: **`desencriptarLetra 'x' 'b'`** retornaría **`'e'`**.
   Hint: se puede resolver este problema sin tener que hacer cuentas para calcular índices ;)
c. Definir la función **`cesar :: Char -> String -> String`** que recibe la letra clave y un texto encriptado y retorna todo el texto desencriptado, teniendo en cuenta que cualquier caracter del mensaje encriptado que no sea una letra (por ejemplo **`'!'`**) se mantiene igual. Usar **`zipWithIf`** para resolver este problema.
d. Realizar una consulta para obtener todas las posibles desencripciones (una por cada letra del abecedario) usando **`cesar`** para el texto "**`jrzel zrfaxal!`**".

3 - BONUS) Un problema que tiene el cifrado César para quienes quieren ocultar el mensaje es que es muy fácil de desencriptar, y por eso es que los mensajes más importantes del diario están encriptados con cifrado **Vigenére**, que se basa en la idea del código César, pero lo hace a partir de un texto clave en vez de una sola letra. Supongamos que la clave es "**`pdep`**" y el mensaje encriptado es "**`wrpp, irhd to qjcgs!`**".

Primero repetimos la clave para poder alinear cada letra del mensaje con una letra de la clave:

```
pdep  pdep pd eppde
wrpp, irhd to qjcgs!
```

Después desencriptamos cada letra de la misma forma que se hacía en el código César (si **p** es **a**, **w** es **h**...).

```
pdep  pdep pd eppde
wrpp, irhd to qjcgs!
-------------------
hola, todo el mundo!
```

Nuestro trabajo será definir la función **`vigenere`** para desencriptar un mensaje de este tipo a partir de la clave usada para encriptarlo, evitando repetir lógica con lo resuelto anteriormente (vale refactorizar lo anterior).

```
> vigenere "pdep" "wrpp, irhd to qjcgs!"
"hola, todo el mundo!"
```