

Programación Declarativa 2020-2
Facultad de Ciencias UNAM
Tarea 4: I want to play a game



Favio E. Miranda Perea

Javier Enríquez Mendoza

Fecha de entrega: 3 de abril de 2020

Criptoaritmos

Los criptoaritmos son operaciones de cálculo en las cuales se han sustituido las cifras por letras u otros símbolos de manera que se propone encontrar que valor corresponde a cada letra, teniendo en cuenta, claro, que una misma letra **no** puede representar dos valores numéricos diferentes. El ejemplo clásico que fue publicada en julio de 1925 en Strand Magazine por Henry Dudeney es:

$$SEND + MORE = MONEY$$

Una posible solución es: $O = 0$, $M = 1$, $Y = 2$, $E = 5$, $N = 6$, $D = 7$, $R = 8$ y $S = 9$, pues se cumple la ecuación:

$$9567 + 1085 = 10652$$

Las reglas son las siguientes:

- Los números están en base diez.
- Cada letra o símbolo representa un único número.
- El primer dígito de un número no puede ser cero.
- Las operaciones válidas son: suma, resta y multiplicación.

Define una función `criptoaritmo` que regrese todas las posibles soluciones del criptoaritmo. La firma de la función debe ser la siguiente:

```
criptoaritmos :: String -> [[(Char, Int)]]
```

La llamada con el ejemplo anterior se vería así:

```
λ > criptoaritmos "SEND + MORE = MONEY"
```

DuckTales ¹

El juego se lleva a cabo en un tablero rectangular de tamaño arbitrario. En cada celda del tablero hay un pato que puede ser bueno o malo. Cada pato es vecino de todos los patos que son adyacentes a él incluyendo los diagonales, puede pensarse que el tablero esta rodeado por una frontera de patos malos por lo que todos los patos dentro del tablero tienen exactamente ocho vecinos.

En el juego hay dos únicas reglas:

- Un pato bueno se mantiene bueno si es adyacente a exactamente dos o tres patos también buenos, en otro caso se transforma en un pato malo.
- Si un pato malo es adyacente a exactamente tres patos buenos entonces se convierte en bueno, en otro caso sigue siendo malo.

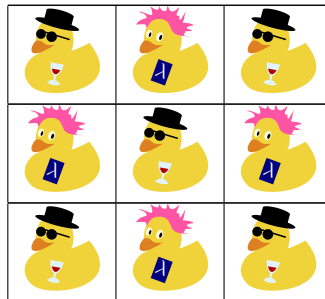
El juego solo va a depender del estado inicial del tablero, cada turno se conoce como una *generación* y aplicar alguna regla es una *evolución* del tablero. La generación inicial es la cero y no hay un orden específico para aplicar las reglas de evolución.



Definir las siguiente funciones

1. `genZero :: String -> Generation` construye una generación inicial a partir de una cadena de entrada con el siguiente formato

- El carácter B representa un pato malo.
- El carácter G representa un pato bueno
- El carácter _ representa el final de una fila del tablero (no es necesario ponerlo al final del tablero)

Por ejemplo la cadena "BGB_GBG_BGB" representa el tablero:



en donde claramente  es el pato bueno y  es el malo

2. `evolution :: Generation -> Generation` que calcula la siguiente generación evolucionando el tablero.
3. `generations :: Generation -> [Generation]` que recibe una generación inicial y calcula el ciclo de evoluciones de esa generación hasta que ya no haya cambios respecto a la generación anterior.

La definición del tipo `Generation` es a conveniencia, siempre y cuando modele una generación del juego.

¹Basado en el juego de la vida de Conway

Los Cantaros

Se tiene dos cantaros con capacidad de X y Y litros respectivamente y un río con flujo infinito en el cual se puede llenar los cantaros y vaciar el contenido de estos (los cantaros no tienen marcas de medición).

El problema consiste en averiguar como se puede lograr tener Z litros de agua en alguno de los cantaros empezando con ambos vacíos.

Solo hay 3 acciones permitidas:

- Llenar completamente un cántaro en el río.
- Llenar un cántaro con el otro.
- Vaciar completamente un cántaro en el río.

Definir una función

```
cantaros :: Int -> Int -> Int -> [Action]
```

En donde los parámetros de tipo `Int` corresponden a X , Y y Z respectivamente y la salida es una lista de acciones en el orden necesario para dejar Z litros de agua en alguno de los cantaros. El tipo `Action` puede definirse a conveniencia.

Especificaciones

Extra *hasta 1 punto*: Resolver alguno de los ejercicios utilizando el estilo de programación lógico en el lenguajes Prolog.

- Para resolver los ejercicios se puede utilizar cualquier lenguaje de programación declarativo o que tenga características declarativas (usando unicamente esta parte del lenguaje). Justificando al principio de cada archivo por qué se eligió este lenguaje.
- Incluir un archivo `readme.txt` con las instrucciones de compilado y ejecución de los programas.
- Todos los programas deben ser lo mas robustos posibles.
- La tarea puede entregarse de forma individual o en parejas.
- Se tomará en cuenta tanto el estilo de programación como la complejidad de las soluciones para la calificación.
- La tarea se entrega a través de Slack:
 - Todos los archivos deben tener al principio como comentario los nombres de los alumnos. Se debe tener un archivo por sección del PDF.
 - Si la tarea se realizo en parejas crear un canal privado con el ayudante y los miembros del equipo para la entrega.
 - Si se realizo de forma individual enviarla en un mensaje directo al ayudante.
 - **No** deben comprimir los archivos, en caso de tener mas de uno enviarlos por separado.