Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Programación Avanzada
Primer Cuatrimestre de 2020

Trabajo Práctico

Este trabajo práctico debe ser resuelto en grupos de máximo tres personas. El proyecto podrá ser entregado cualquier fecha anterior al **Viernes 26 de Junio utilizando Google Classroom.**.

El trabajo practico se trata de hacer un algoritmo para jugar el juego de las piedras. Este juego es muy sencillo y consiste de dos jugadores (Humano vs Computadora). El juego empieza con N piedras (con $N > 0$) y cada jugador puede tomar 1, 3 o 4 piedras en turnos. Cuando un jugador se queda sin piedras pierde, y por lo tanto gana el otro jugador.

La idea es hacer un programa en Haskell para que la Computadora juegue de la mejor forma posible. En el juego empieza a jugar el humano, y luego la computadora, y se repite por turnos. Se puede notar que hay algunas configuraciones iniciales que la computadora no tiene jugadas ganadoras, es decir, el humano le va a ganar si juega bien. De la misma forma, hay configuraciones en la cuales el humano no tiene jugadas ganadoras.

El código en el repositorio tiene varias funciones implementadas y comentadas. El trabajo práctico consiste en implementar las funciones que aparecen no implementadas en el archivo TP.hs.

El siguiente tipo define los jugadores del juego, `C` por Computadora y `H` por Humano

```
data Jugador = C | H deriving (Eq,Show)
```

Definimos los estados, un estado es un jugador mas las cantidad de piedras disponibles

```
type Estado = (Jugador, Int)
```

Definimos los posibles estados del juegos, el resultado del juego, puede ser que la computadora pierda, o gane.

```
data Resultado = CPerdio | CGano deriving (Eq,Ord,Show)
```

Definimos las posible jugadas, sacar 1 piedra, 3 piedras o 4 piedras.

```
jugadas = [1,3,4]
```

La función otro Jugador, dado un jugador, devuelve el otro jugador, por ejemplo: `otroJugador C = H`.

```
otroJugador :: Jugador -> Jugador
-- Debe ser implementada
```

La función `hacerJugada`, dada una jugada (cantidad de piedras que se retiran) y un estado, retorna el estado resultante, de deben controlar los casos de jugadas no posibles.

```
hacerJugada :: Int -> Estado -> Estado
-- Debe ser implementada
```

La función `evalEstado` toma un estado como parámetro, y dice si el estado es ganador o perdedor considerando las mejores jugadas del oponente. Por ejemplo, `evalEstado (H,2) = CGano`, porque `H` solo puede retirar 1 y luego la computadora retira 1 y gana.

```
evalEstado :: Estado -> Resultado
evalEstado  (j, k)  | (k == 0) = if j == C then CPerdio else CGano
                    |  k>0 && j == C   = foldl max CPerdio $ map evalEstado posibleJugs
                    |  k>0 && j == H   = foldl min CGano $ map evalEstado posibleJugs
                    |  otherwise = error "jugada no valida"
                   where posibleJugs = [(otroJugador j, k - i) | i<- jugadas, i<=k]
```

La función `mejorJug` calcula la mejor jugada para un estado dado para el jugador dado. Por ejemplo, si `mejorJug (H,3)=3`, ya que la mejor jugada para H cuando hay 3 piedras es retirar 3. Tener en cuenta que el tipo `Resultado` implementa la clase `Ord`, es decir, tenemos `CPerdio < CGano`. Entonces para el caso `mejorJug (C, k)` tenemos que devolver la jugada que nos da el resultado máximo con respecto a `<` (es decir, la mejor jugada para la computadora). En el caso `mejorJug (H, k)` tenemos que devolver la jugada que nos da el valor mínimo (es decir, consideramos la mejor jugada para `H`, que seria la peor para `C`).

```
mejorJug :: Estado -> Int
```

Las función `empezarJuego::Int -> IO()` (ya implementada) permite empezar un juego interactivamente (cuando todas las funciones están implementadas).

Finalmente se debe implementar una función `juegosGanadores:  Int -> [Int]`, que calcula todos los comienzos ganadores para la computadora hasta con `k` piedras. Por ejemplo, `juegosGanadores 10 = [2,7,9]`.

Para el trabajo práctico **se deben completar todas la funciones que no están implementadas en el código**. Con los comentarios pertinentes sobre el código.