Recuperatorio del segundo trabajo práctico
El juego de Pop-it

Taller de Álgebra 1

Primer cuatrimestre de 2021

El Pop-it es un juego de dos jugadores, que juegan alternadamente. Al comienzo del juego hay varios montoncitos de piedras sobre una mesa. Una jugada consiste en elegir alguno de los montoncitos y retirar algunas piedras de este montoncito (hay que retirar por lo menos una y también es válido retirar todas las piedras del montoncito). Gana el jugador que se lleva la última piedra. En otras palabras, pierde el jugador que no puede realizar una jugada porque al llegar su turno no queda ninguna piedra sobre la mesa.

Notemos que no hay empate y que el juego siempre termina, ya que en algún momento se terminan las piedras.

Si nunca jugaron al Pop-it, sería recomendable que lo hagan con algún compañere, amigue o familiar. Se puede comenzar con 4 pilas con 1, 3, 5 y 7 piedras respectivamente como se ve en la Figura 1. Otra posible posición inicial es con 4 pilas de 4 piedras cada una.

Figura 1: Fotograma de una reconocida película francesa de la década del '60, donde puede verse al protagonista jugando al Pop-it. Se las recomendamos, no está en Netflix.

El objetivo principal de este TP es escribir funciones que permitan jugar al Pop-it de forma óptima.

Posiciones ganadoras

Decimos que una posición es *ganadora* si el jugador al que le toca jugar puede asegurarse ganar el juego, sin importar que tan bien juegue su rival.

En un juego de dos jugadores como el Pop-it es posible caracterizar las posiciones ganadoras de forma recursiva:

→ Una posición es ganadora si existe una jugada tal que la posición obtenida al realizar dicha jugada no es ganadora.

Notar que no necesitamos un caso base: con esta definición de posición ganadora, se sigue que la posición vacía no es ganadora, pues el conjunto de jugadas válidas partiendo de la posición vacía es vacío. La implementación del Ejercicio 3 debe basarse en esta recurrencia.

Representando las posiciones y jugadas en Haskell

Vamos a representar una posición del Pop-it con el siguiente tipo:

```
type Posicion = [Int]
```

Los números de la lista deben ser enteros positivos y representan los tamaños de los montoncitos de piedras.

Observemos que una posición puede representarse de varias formas. Por ejemplo, la posición en la que hay un montoncito con 5 piedras y dos montoncitos con 2 piedras la podemos representar de tres formas equivalentes `[5,2,2]`, `[2,5,2]` y `[2,2,5]`. Sin embargo, `[2,2,5,0]` no es una forma válida de representar esta posición (no escribimos los montoncitos que quedan sin piedras). Notemos que el juego termina cuando se llega a la posición vacía `[]`.

Por otro lado, vamos a representar una jugada con el siguiente tipo

```
type Jugada = (Int,Int)
```

El primer elemento del par indica la posición de la pila y el segundo indica cuántas piedras se quitan de esa pila en esa jugada. Como es usual en la materia, numeramos las pilas empezando en 1.

Ejercicios

Ejercicio 1

Escribir la función

```
jugar :: Posicion -> Jugada -> Posicion
```

que recibe una posición `p`, una jugada válida `j` y devuelve la posición obtenida al realizar dicha jugada.

Ejercicio 2

Escribir la función

```
posiblesJugadas :: Posicion -> [Jugada]
```

que recibe una posición `p` y devuelve el conjunto de jugadas válidas a partir de `p`.

Ejercicio 3

Escribir la función:

```
esPosicionGanadora :: Posicion -> Bool
```

que decide si una posición `p` es ganadora.

La función `esPosicionGanadora` solo nos dirá si una posición es ganadora o no, pero es posible que queramos conocer *cómo* jugar a partir de esa posición para ganar el juego.

Ejercicio 4

Escribir la función

```
jugadaGanadora :: Posicion -> Jugada
```

que recibe una posición ganadora `p` y devuelve una jugada que dejaría al rival en una posición no ganadora.

En general, puede haber muchas jugadas ganadoras.

Ejercicio 5

Escribir la función

```
numeroDeJugadasGanadoras :: Posicion -> Int
```

que recibe una posición `p` (no necesariamente ganadora) y devuelve la cantidad de jugadas ganadoras partiendo de `p`.

Recuperatorio

El `xor` (también llamado "o exclusivo") es una operación entre dos booleanos que devuelve un booleano. Si pensamos a los dígitos `0` y `1` como `False` y `True` respectivamente, entonces nos queda: `0 xor 0 = 0`, `0 xor 1 = 1`, `1 xor 0 = 1`, `1 xor 1 = 0`.

Dados dos números naturales `n` y `m`, podemos definir `m xor n` como otro natural cuyo i-ésimo dígito (en base dos) es el xor del i-ésimo dígito de `m` (en base dos) y el i-ésimo dígito de `n` (en base dos). Si `m` y `n` tienen distinta cantidad de dígitos (en base dos) completamos con suficientes 0 a izquierda de modo que tengan la misma cantidad.

Por ejemplo, `13 xor 6 = 11` ya que $13 = 1101_2$, $6 = 110_2$ y $11 = 1011_2$.

Ejercicio 6: adicional recuperatorio

Escribir la función

```
xor:: Int -> Int -> Int
```

que devuelve el `xor` de dos números naturales `m` y `n`.

El `xor` es una operación asociativa, es decir que `a xor (b xor c) = (a xor b) xor c`, por lo que podemos olvidarnos de los paréntesis y escribir directamente `a xor b xor c`. Por lo tanto, podemos considerar el xor de los elementos de una lista de naturales.

Ejercicio 7: adicional recuperatorio

Escribir la función

```
xorLista:: [Int] -> Int
```

que devuelve el `xor` de todos los elementos de una lista de números naturales.

El juego de Pop-it admite una fórmula cerrada. Decimos que es *cerrada* en el sentido de que se calcula fácilmente el desarrollo del juego sin tener que explorar todo el árbol de jugadas de manera recursiva como se pide en el Ejercicio 3.

Teorema (fórmula cerrada del juego de Pop-it):

Una posición `p` es ganadora si y solamente si el `xor` bit a bit de los tamaños de los montoncitos es distinto de 0.

Para resolver el siguiente ejercicio es válido asumir este teorema (sin preocuparse por su demostración).

Ejercicio 8: adicional recuperatorio

Escribir la función

```
esPosicionGanadoraViaFormulaCerrada :: Posicion -> Bool
```

que decide si una posición `p` es ganadora utilizando la fórmula cerrada del juego de Pop-it.

Casos de prueba

En la siguiente tabla, la evaluación de las expresiones de la columna izquierda debe devolver los valores de la columna derecha. Para la función `jugadaGanadora` se listan todos los posibles resultados correctos.

Expresión	Resultado esperado
`posiblesJugadas []`	`[]`
`posiblesJugadas [1]`	`[(1,1)]`
`posiblesJugadas [1,2,2]`	`[(1,1),(2,1),(2,2),(3,1),(3,2)]`
`esPosicionGanadora []`	`False`
`esPosicionGanadora [1]`	`True`
`esPosicionGanadora [1,2]`	`True`
`esPosicionGanadora [1,2,3]`	`False`
`esPosicionGanadora [1,2,3,4]`	`True`
`esPosicionGanadora [1,1]`	`False`
`esPosicionGanadora [1,2,2]`	`True`
`jugar [3,3,3] (1,3)`	`[3,3]`
`jugar [3,3,3] (2,1)`	`[3,2,3]`
`jugar [5,4,3,2,1] (2,3)`	`[5,1,3,2,1]`
`jugadaGanadora [1,2,3,4]`	`(4,4)`
`jugadaGanadora [1,1,1]`	`(1,1) ó (2,1) ó (3,1)`
`numeroDeJugadasGanadoras []`	`0`
`numeroDeJugadasGanadoras [1]`	`1`
`numeroDeJugadasGanadoras [1,2,3,4]`	`1`
`numeroDeJugadasGanadoras [1,1,1]`	`3`
`xor 13 6`	`11`
`xor 1234 8463`	`9693`

```
xorLista [13,6,11]                          0
xorLista [1,2,3,23,585]                     606
esPosicionGanadoraViaFormulaCerrada []      False
esPosicionGanadoraViaFormulaCerrada [1]     True
esPosicionGanadoraViaFormulaCerrada [1,2]   True
esPosicionGanadoraViaFormulaCerrada [1,2,3] False
esPosicionGanadoraViaFormulaCerrada [1,2,3,4] True
esPosicionGanadoraViaFormulaCerrada [1,1]   False
esPosicionGanadoraViaFormulaCerrada [1,2,2] True
esPosicionGanadoraViaFormulaCerrada [13,6,11] False
esPosicionGanadoraViaFormulaCerrada [89,991] True
```

Condiciones de entrega

El trabajo práctico se debe realizar de manera individual. Está prohibido subir el código que implementaron a repositorios públicos, así como también usar total o parcialmente soluciones implementadas por sus compañeres. La entrega consiste en un único archivo `.hs` con las funciones de los ejercicios implementadas, junto con todas las funciones auxiliares que sean necesarias para ejecutarlas. Las funciones deben respetar la signatura (nombres y parámetros) especificados en cada ejercicio, dado que serán testeadas automáticamente. Para esto es importante no incluir una declaración de módulo al comienzo del archivo. El archivo que entregan tiene que poder compilarse sólo, y sin llamar a otro módulo.

El nombre del archivo entregado debe tener la forma "`turno-nombreapellido.hs`", donde `turno` debe ser "`TM`", "`TT`" o "`TN`" dependiendo del turno en el que estén cursando:

- `TM`: Miércoles 9 hs.
- `TT`: Viernes 14 hs.
- `TN`: Miércoles 17:30 hs.

Por ejemplo, un estudiante llamado Juan Pérez que esté cursando en el turno de los viernes debe entregar un archivo llamado "`TT-JuanPerez.hs`". La entrega se debe llevar a cabo a través del campus virtual, subiendo el archivo con el código con el mecanismo disponible dentro del espacio del taller en el campus virtual.

Para la resolución del trabajo práctico se deben utilizar exclusivamente los conceptos vistos en el Taller. Se evaluará la corrección de las funciones implementadas, la declaratividad y claridad del código, y que las funciones auxiliares (si las hay) tengan nombres apropiados.

Si tienen dudas o consultas respecto del trabajo práctico, pueden enviar un mail a la

lista de docentes `algebra1-doc (arroba) dc.uba.ar`. No hacer consultas a través del campus virtual ni mandando mails a la lista de alumnos.

Además del código, cada alumno deberá rendir un **coloquio individual**, en el que conversará sobre el trabajo que realizó con el docente que le corrigió y tendrá que responder una serie de preguntas al respecto.

Fecha de entrega: hasta el viernes 2 de julio a las 23:59 hs.