Estructuras de Datos – UNQ – 2do semestre de 2010

PARCIAL 1 – *Solitario Spider*

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todo lo visto en la práctica y en la teórica, aclarando la referencia.*
- *No se olvide de poner nombre, nro. de legajo, nro. de hoja y cantidad total de hojas en cada hoja.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, en explicar lo que se propone antes de escribir código, en probar su código con ejemplos, etc.*

El Solitario Spider se juega con dos mazos de cartas francesas de 52 cartas. Al comenzar un nuevo juego, se reparten diez *columnas* de cartas (4 de 6 cartas, y 6 de 5 cartas), cada una con una única carta descubierta en la base. Las cartas restantes se colocan en cinco *pilas* de diez cartas, que se utilizan al repartir una nueva carta en cada columna. El objetivo del Solitario Spider es quitar todas las cartas de las diez columnas en el menor número de movimientos. Para quitar cartas de las diez columnas, las cartas pueden moverse de una columna a otra hasta que se obtenga una secuencia de cartas del mismo palo (*cartas alineadas*), ordenadas del Rey (K) al As (*palo alineado*), en cuyo caso serán removidas.

Las cartas pueden moverse según las siguientes reglas:

- Puede moverse la carta de la base de una columna a una columna vacía.
- Puede moverse la carta de la base de una columna a otra columna cuya carta en la base tenga el número siguiente más alto que la carta que se mueve, independientemente del palo o color.
- Puede moverse una secuencia de cartas todas del mismo palo y ordenadas por número como si fueran una sola carta. En ese caso, se cuenta la carta más alta como la que se mueve.
- Puede repartirse una de las pilas, colocando una carta en cada columna, si es que aún queda alguna pila sin repartir.

Si luego de cualquiera de los movimientos se obtiene un palo alineado, se remueven esas cartas de la columna automáticamente. El juego termina cuando todas las cartas se pudieron quitar de las columnas, o cuando no quedan más movimientos posibles.

La puntuación en el Solitario Spider se realiza de la siguiente manera. En cada juego, se comienza con 500 puntos. La puntuación puede aumentar o disminuir en función de las siguientes reglas:

- Cada vez que se mueva una carta (o secuencia de cartas), se perderá un punto.
- Cada vez que se obtenga un palo alineado y sea removido, se obtendrán 100 puntos.

El objetivo de esta evaluación es modelar el juego de Solitario Spider. Para ello, se define un tipo abstracto de datos `Spider`, que captura el estado del juego en un momento dado, y expresa las acciones posibles que se pueden realizar sobre el mismo. En la definición de este tipo abstracto se utilizan diversos tipos auxiliares.

En primer lugar, el tipo abstracto `Carta`, junto con los tipos algebraicos `Palo` y `Numero`.

```
data Palo = Diamante | Pique | Trebol | Corazon
data Numero = As | N2 | N3 | N4
            | N5 | N6 | N7 | N8
            | N9 | N10 | J | Q | K
```

Las operaciones del tipo `Carta` son

- `palo :: Carta -> Palo`

  que retorna el palo de una carta dada.

- `nro :: Carta -> Numero`

  que retorna el número de una carta dada.

- `siguienteNro :: Carta -> Carta -> Bool`

  que dadas cartas `c1` y `c2` establece si `c2` es la carta con el siguiente número de `c1`.

- `mismoPalo :: Carta -> Carta -> Bool`

  que establece si dos cartas dadas tienen el mismo palo.

- `listaMazo :: [Carta]`

  que retorna una lista de cartas con todas las cartas de un mazo.

- `tomarCartaN :: [Carta] -> Int -> (Carta, [Carta])`

  que dada una lista de cartas y un número `k`, separa la carta en la posición `k` de la lista. Esta operación es parcial si la lista no tiene suficientes elementos.

A continuación, el tipo abstracto `Mazo`, con las siguientes operaciones

- `darUna :: Mazo -> (Carta, Mazo)`

  que si hay cartas en el mazo, devuelve la primera de ellas y el resto del mazo.

- `hayCartas :: Mazo -> Bool`

  que establece si hay cartas en el mazo.

- `mezclarCartas :: Int -> [Carta] -> Mazo`

  que dado un número semilla `seed` y una lista de cartas, arma un mazo mezclando las cartas (seudoaleatoriamente según `seed`).

- `mazo2list :: Mazo -> [Carta]`

  que dado un mazo, retorna una lista con sus cartas.

Finalmente, el tipo abstracto `Spider`, junto con el tipo algebraico `Move`.

```
data Move = Repartir | MoverNDeA Int Int Int
```

Las operaciones del tipo `Spider` son

- `puntaje :: Spider -> Int`

  que retorna el puntaje de un juego de Spider en curso.

- `ganado :: Spider -> Bool`

  que dice si el juego fue ganado o no.

- `iniciarJuego :: Int -> Spider`

  que dado un número `n`, crea el Spider número `n`, repartiendo las cartas según las reglas del juego (usa `n` como semilla para mezclar las cartas).

- `moverDeA :: Int -> Int -> Int -> Spider -> Spider`

  tal que `moverDeA origen destino k sp`, mueve `k` cartas de la columna de origen a la de destino, siempre que haya al menos `k` cartas destapadas en la columna origen y el movimiento sea posible según las reglas del juego. Si no es posible realizarlo, devuelve el mismo juego.

- `repartir :: Spider -> Spider`

  si en el juego dado aún quedan pilas para repartir, reparte las cartas de una pila segun las reglas del juego, o deja el juego igual si no se puede repartir.

- `jugadasPosibles :: Spider -> [Move]`

  da todas las posibles movidas de un Spider.

Ejercicio 1 *Escribir las siguientes funciones:*

a) `mover :: Spider -> Move -> Spider`

que dado un juego y un movimiento, retorna el juego resultante de realizar dicho movimiento.

b) `jugar :: Spider -> [Move] -> Spider`

que dado un juego y una lista de movimientos, retorna el juego resultante de aplicar cada uno de los movimientos en orden sobre el juego inicial.

Un *árbol de juego* es un árbol general que tiene un juego en cada nodo, y de tal manera que los hijos son los posibles juegos a los que se llega desde la raíz con un único movimiento.

Ejercicio 2 *Escribir las siguientes funciones:*

a) `siguientes :: Spider -> [Move] -> [Spider]`

que dado un juego `sp` *y una lista de movidas, retorna una lista de juegos, cada uno de los cuales resulta de realizar* ***sólo una*** *de las movidas sobre el juego* `sp`*.*

b) `arbolSpider :: Int -> Spider -> GenTree Spider`

que dado un entero `h` *y un juego* `sp`*, retorna el* árbol de juego *con raíz* `sp` *de profundidad a lo sumo* `h`*.*

Ejercicio 3 *Implementar el tipo abstracto* `Spider`*,* sin implementar *la operación de* `jugadasPosibles`*. Tener en cuenta el proveer la adecuada barrera de abstracción.*

Ayuda: *Utilizar tantas funciones y operaciones auxiliares como sea necesario en cada una de las operaciones. Tener en cuenta utilizar adecuadamente los tipos abstractos* `Carta` *y* `Mazo`*. Puede resultar conveniente utilizar la siguiente guía:*

a) *Definir un tipo para las columnas que distinga entre cartas tapadas y destapadas. Definir como auxiliares operaciones* `emptyCol`*,* `isEmptyCol`*,* `destapadas`*,* `tapadas`*,* `agregarTapada`*,* `agregarDestapada` *y* `agregarDestapadas`*. Definir un tipo para las pilas.*

b) *Definir un tipo algebraico para representar a un* `Spider`*.*
Definir operaciones auxiliares de acceso `puntaje`*,* `descartadas`*,* `columnas` *y* `pilas`*.*

c) *Definir una función auxiliar* `movidaValida :: Spider -> Move -> Bool`*.*
Definir operaciones auxiliares `estanAlineadas :: [Carta] -> Bool` *y* `esPaloAlineado :: [Carta] -> Bool`*.*

d) *Definir las operaciones solicitadas del tipo abstracto.*
Para `iniciarJuego`*, considerar definir operaciones auxiliares*

- `darColumna :: Int -> Col -> Mazo -> (Col,Mazo)` *y*
- `darPila :: Int -> Pila -> Mazo -> (Pila, Mazo)`*.*

Para `moverDeA`*, considerar definir las operaciones auxiliares*

- `agregarYverificar :: [Carta] -> Col -> Col` *y*
- `tomarDestapadas :: Int -> Col -> ([Carta], Col)`*.*