Matrices Dispersas (Haskell)

Una matriz dispersa es una matriz en la que la mayoría de sus elementos son ceros. La representación habitual de matrices como arrays de arrays no es adecuada para este tipo de matrices ya que requiere memoria para almacenar cada uno de los elementos de la matriz, siendo gran parte de la información redundante, al ser la mayoría de los valores ceros.

Por ello, vamos a definir un tipo de dato `SparseMatrix` para representar este tipo de matrices de una forma más compacta. En primer lugar, definimos el tipo de datos `Index` que permite representar los índices (fila y columna) de un elemento no cero de la matriz:

```
data Index = Idx Int Int deriving (Eq, Ord, Show)
```

El convenio que seguiremos es que ambos índices toman valores de 0 en adelante (por ejemplo, la fila 0 es la primera fila de la matriz).

Un dato de tipo `SparseMatrix` representará una matriz dispersa de valores enteros mediante un diccionario/mapa cuyas claves serán valores de tipo `Index` y cuyos valores serán los elementos (de tipo `Int`) de la matriz. Así, habrá una asociación en el mapa entre el índice `Idx i j` y el valor v_{ij} por cada elemento v_{ij} que no sea **cero** de la matriz. Por ejemplo, la siguiente matriz dispersa:

```
0  0 10  0
20 0  0  0
0  0  0  0
```

quedaría representada con el siguiente mapa:

```
AVLDictionary(Idx 0 2 -> 10, Idx 1 0 -> 20)
```

Es decir, el elemento en la fila 0/columna 2 vale 10 y el elemento en la fila 1/columna 0 vale 20; todos los demás elementos que son cero no se incluyen en la asociación.

Nota. En todos los apartados que siguen, tu código debe comprobar las condiciones de error que pudieran darse y elevar las correspondientes excepciones. Se valorará, además de la corrección de tu solución, su eficiencia y claridad. Se valorará también la no repetición de código en la solución, es decir, se valorará favorablemente la reutilización de otras funciones ya definidas cuando sea conveniente.

(0.75 puntos) El tipo de dato `data SparseMatrix` permite representar matrices dispersas de valores enteros según se ha explicado.

```
data SparseMatrix = SM Int Int (D.Dictionary Index Int) deriving Show
```

Los valores del tipo de dato incluyen una componente entera (que almacena el número de filas de la matriz), otra componente entera (que almacena el número de columnas de la matriz) y una componente correspondiente al diccionario/mapa de índices en valores para los elementos **no ceros** de la matriz.
Define una función `sparseMatrix` que construya una matriz dispersa inicialmente nula (con todos sus elementos ceros, es decir, con el mapa vacío) dadas sus dimensiones como parámetros (número de filas y columnas).

(0.5 puntos) Define en el módulo una función privada `value` que tome como parámetros una `SparseMatrix` y un valor de tipo `Index`, y que devuelva el valor del elemento de la matriz en dicha fila y columna. Por ejemplo, para la matriz anterior, `value m (Idx 0 0)` devolvería 0 y `value m (Idx 0 2)` devolvería 10. En este apartado, puedes asumir que las coordenadas del índice proporcionado son válidas.

(0.5 puntos) Define en el módulo una función privada `update` que tome como parámetros una matriz, un valor de tipo `Index` y un valor, y que devuelva la matriz dispersa modificada resultante tras modificar el elemento correspondiente a las coordenadas indicadas con el valor proporcionado. Recuerda que el mapa solo debe almacenar asociaciones para elementos no ceros. En este apartado, puedes asumir que las coordenadas del índice proporcionado son válidas.

(0.5 puntos) Define en el módulo una función privada `index` que tome como parámetros una matriz y dos enteros, correspondientes a un número de fila y columna de la matriz, y que devuelva un objeto de tipo `Index` para dichos índices. En este apartado, las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(0.5 puntos) Define en el módulo una función pública `set` que tome como parámetros una matriz dispersa y tres enteros, correspondientes a un número de fila, un número de columna y un valor y que devuelva la matriz dispersa modificada, de forma que el elemento de la matriz en la fila y columna indicadas pase a valer el valor proporcionado como último argumento. En este apartado, asume que las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(0.5 puntos) Define en el módulo una función pública `get` que tome como parámetros una matriz dispersa y dos enteros, correspondientes a un número de fila y un número de columna de la matriz, y que devuelva el valor del elemento de la matriz en dicha fila y columna. En este apartado, asume que las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(1.25 puntos) Define en el módulo una función pública `add` que implemente **de forma eficiente** la suma de dos matrices dispersas que se pasan como argumento, devolviendo una nueva matriz dispersa con el resultado.

(1.25 puntos) Define en el módulo una función pública `transpose` que calcule **de forma eficiente** una nueva matriz dispersa, correspondiente a la traspuesta de su argumento.

(1.25 puntos) Define en el módulo una función `fromList` que tome como argumentos el número de filas y columnas y una lista de enteros y devuelva una matriz dispersa con el siguiente algoritmo (la lista debe tener una longitud múltiplo de 3. Si no es así, se lanzará una excepción): cada tres elementos de la lista se interpretan como fila, columna y valor y se insertan en una matriz dispersa inicialmente vacía devolviendo la matriz dispersa resultante. Estudia la complejidad de la operación `fromList` y pon tus conclusiones como comentario en esta función.

Matrices Dispersas (Java)

Una matriz dispersa es una matriz en la que la mayoría de sus elementos son ceros. La representación habitual de matrices como arrays de arrays no es adecuada para este tipo de matrices ya que requiere memoria para almacenar cada uno de los elementos de la matriz, siendo gran parte de la información redundante, al ser la mayoría de los valores ceros.

Por ello, vamos a definir una clase `SparseMatrix` para representar este tipo de matrices de una forma más compacta. En primer lugar, definimos la clase `Index` que permite representar los índices (fila y columna) de un elemento no cero de la matriz:

```
package sparseMatrix;
```

```
public class Index implements Comparable<Index> {
    private final int row;
    private final int column;

    public Index(int r, int c) {
        row = r;
        column = c;
    }

    public int getRow() {
        return row;
    }

    public int getColumn() {
        return column;
    }

    public int compareTo(Index ind) {
        int cmp = Integer.compare(row, ind.row);
        if (cmp == 0)
            cmp = Integer.compare(column, ind.column);
        return cmp;
    }
}
```

El convenio que seguiremos es que ambos índices toman valores de 0 en adelante (por ejemplo, la fila 0 es la primera fila de la matriz). Un indice es menor que otro si lo es su fila y en caso de igualdad, si lo es su columna.

Un objeto de la clase `SparseMatrix` representará una matriz dispersa de valores enteros mediante un atributo privado `nonZeros`, que será un diccionario/mapa cuyas claves serán objetos de tipo `Index` y cuyos valores serán elementos (de tipo entero) de la matriz. Así, habrá una asociación en el mapa entre el índice `Index(i, j)` y el valor v_{ij} por cada elemento v_{ij} **no cero** de la matriz. Por ejemplo, la siguiente matriz dispersa:

```
0  0 10  0
20 0  0  0
0  0  0  0
```

quedaría representada con un mapa con la siguiente información:

```
AVLDictionary(Index(0, 2) -> 10, Index(1, 0) -> 20)
```

Es decir, el elemento en la fila 0/columna 2 vale 10 y el elemento en la fila 1/columna 0 vale 20; todos los demás elementos se asumen cero.

Nota. En todos los apartados que siguen, tu código debe comprobar las condiciones de error que pudieran darse y elevar las correspondientes excepciones. Se valorará, además de la corrección de tu solución, su eficiencia y claridad. Se valorará también la no repetición de código en la solución, es decir, se valorará favorablemente la reutilización de otros métodos ya definidos cuando sea conveniente.

(0.5 puntos) La clase `SparseMatrix` permite representar matrices dispersas de valores enteros según se ha explicado. La clase incluye un atributo público constante `rows` (que almacena el número de filas de la matriz), un atributo público constante `columns` (que almacena el número de columnas de la matriz) y un atributo privado `nonZeros` (correspondiente al diccionario/mapa de índices en valores para los elementos **que no valgan cero** de la matriz). Define para la clase un constructor que permita crear una matriz inicialmente cero (con todos sus elementos ceros, es decir, con el mapa vacío), dadas sus dimensiones como parámetros (número de filas y columnas).

(0.5 puntos) Define en la clase `SparseMatrix` un método privado `value` que tome como parámetro un objeto de tipo `Index` y que devuelva el valor del elemento de la matriz en dicha fila y columna. Por ejemplo, para la matriz anterior, `value(new Index(0, 0))` devolvería 0 y `value(new Index(0, 2))` devolvería 10. En este apartado, puedes asumir que las coordenadas del índice proporcionado son válidas.

(0.5 puntos) Define en la clase `SparseMatrix` un método privado `update` que tome como parámetros un objeto de tipo `Index` y un valor, y que modifique la matriz de forma que el elemento en la fila y columna indicadas por el índice pase a valer el valor proporcionado como último argumento. Recuerda que el mapa solo debe almacenar asociaciones para elementos no ceros. En este apartado, puedes asumir que las coordenadas del índice proporcionado son válidas.

(0.5 puntos) Define en la clase `SparseMatrix` un método privado `index` que tome como parámetros dos enteros, correspondientes a un número de fila y columna de la matriz, y que devuelva un objeto de tipo `Index` para dichos índices. En este apartado, las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(0.5 puntos) Define en la clase `SparseMatrix` un método público `set` que tome como parámetros tres enteros, correspondientes a un número de fila, un número de columna y un valor y que **modifique** la matriz de forma que el elemento de la matriz en la fila y columna indicadas pase a valer el valor proporcionado como último argumento. En este apartado, asume que las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(0.5 puntos) Define en la clase `SparseMatrix` un método público `get` que tome como parámetros dos enteros, correspondientes a un número de fila y un número de columna de la matriz, y que devuelva el valor del elemento de la matriz en dicha fila y columna. En este apartado, asume que las coordenadas del índice proporcionado podrían ser inválidas, por lo que debes realizar las correspondientes comprobaciones.

(1 punto) Define en la clase `SparseMatrix` un método de clase público `add` que implemente **de forma eficiente** la suma de dos matrices dispersas que se pasan como argumento, devolviendo una nueva matriz dispersa con el resultado.

(1 punto) Define en la clase `SparseMatrix` un método público `transpose` que devuelva una nueva matriz dispersa, correspondiente a la traspuesta de `this` **calculada de forma eficiente**.

(1 punto) Define en la clase `SparseMatrix` un método público `toString`. El método deberá devolver una cadena de caracteres con todos los elementos de la matriz (ceros y no ceros), de forma que cada uno ocupe un ancho de 6 caracteres y de forma que los elemento correspondientes a distintas filas estén separados por un carácter de salto de línea.

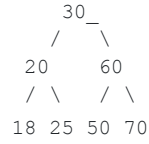
(1 punto) Define en la clase `SparseMatrix` dos métodos de clase `fromList` y `fromList2` que tomen como argumento una lista de enteros y creen y devuelvan una matriz dispersa con el siguiente algoritmo (la lista debe tener una longitud múltiplo de 3. Si no es así, se lanzará una excepción): cada tres elementos de la lista se interpretan como fila, columna y valor y se insertan en una matriz dispersa inicialmente vacía devolviendo la matriz dispersa resultante.
El método `fromList` implementará el algoritmo utilizando el método `get` que proporciona la lista. Para cada una de las implementaciones de la interfaz `List` que hay en el paquete `dataStructures.list`, estudia la complejidad de la operación `fromList` y pon tus conclusiones como comentario en este método.

El método `fromList2` implementará el algoritmo utilizando un iterador sobre la lista. Para cada una de las implementaciones de la interfaz `List` que hay en el paquete `dataStructures.list`, estudia la complejidad de la operación `fromList2` y pon tus conclusiones como comentario en este método.

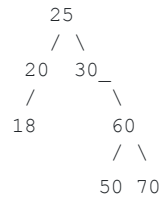
Ejercicio para evaluación solo por examen final (alumnos que renuncian a su evaluación continua)

En un árbol de búsqueda es posible ascender cualquiera de sus elementos hasta la raíz del árbol aplicando rotaciones simples izquierdas y derechas.

Por ejemplo, dado el árbol de búsqueda:



Podemos hacer ascender `25` hasta la raíz, obteniendo el árbol de búsqueda:



Haskell

(3 puntos) Completa la función `makeRoot x t` del módulo `Tree` que, dados un dato `x` y un árbol de búsqueda `t`, devuelve el árbol de búsqueda que resulta de ascender el dato `x` en `t` hasta la raíz. Si `x` no aparece en `t`, se devuelve el árbol original.

Java

(3 puntos) Completa el método `makeRoot(x)` de la clase `Tree` que modifica el árbol ascendiendo el dato `x` hasta su raíz. Si el dato `x` no aparece en el árbol, este no se modifica.