PRÁCTICA DE PROGRAMACIÓN FUNCIONAL
CURSO 2020/2021

PARTE 1 :

La fecha y hora límites para la entrega de la práctica es el **20 de mayo de 2021 a las 12:00h**. Dedicaremos la sesión de ese día a analizar las soluciones.

La práctica se hará preferiblemente en parejas, pero si alguien prefiere trabajar sólo podrá hacerlo siempre que no haya ningún alumno que desee trabajar en pareja y no encuentre compañero.

SÓLO ENTREGARÁ UNO DE LOS MIEMBROS DE LA PAREJA, DEBIENDO CONSTAR EL NOMBRE Y APELLIDOS DE AMBOS EN LA PRIMERA LÍNEA COMENTADA DEL FICHERO ENTREGADO.

La práctica se entregará como un fichero Haskell ejecutable (extensión .hs) a través de una tarea Moodle.

Para la realización de la **Parte 1** debe utilizarse el esqueleto que encontrareis en el fichero Puzle8_1.hs, en el que se definen los tipos a manejar y se sugiere la definición de varias funciones auxiliares que facilitan la de las funciones pedidas.

Para ejecutar las prácticas se debe utilizar el intérprete ghci del **Glasgow Haskell Compiler**

Se podrán utilizar **exclusivamente los recursos del lenguaje con los que hemos trabajado** - como librerías exclusivamente QuickCheck y Data.List.

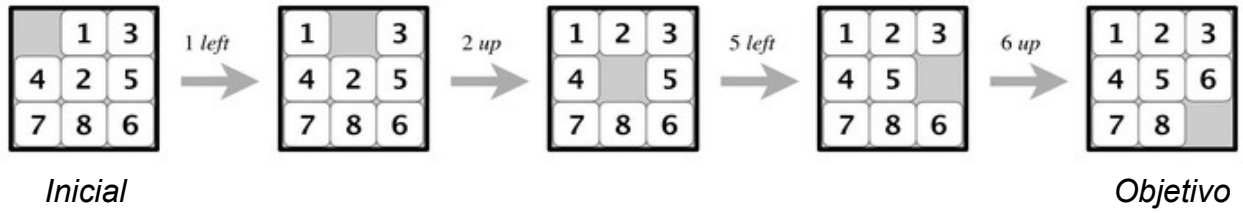
La nota máxima que será posible obtener en esta parte es de **7.5 puntos.**

EL PROBLEMA DEL 8-Puzle

El 8-Puzle es un juego que se desarrolla en un tablero de dimensión 3x3 cuyas casillas contienen los números del 1 al 8 y un blanco, ordenados de cualquier forma.

El objetivo es utilizar deslizamientos para reordenar las casillas hasta conseguir que los números aparezcan ordenados de izquierda a derecha y de arriba abajo, quedando el blanco en la posición 3x3.
Una casilla se puede mover exclusivamente hacia la posición del blanco si es contigua, bien en horizontal o en vertical.

Aquí podemos ver una secuencia de movimientos desde un tablero inicial hasta el tablero objetivo:



Inicial *Objetivo*

Fig.1

Existen tableros iniciales para los que el juego tiene solución, como se ve en la *Fig.1*, y tableros para los que no existe ninguna secuencia de movimientos que conduzca hasta el objetivo.

El número de tableros distintos es 9!, es decir, 362.880, de los cuales se sabe que la mitad de ellos tienen al menos una solución – pueden tener más de una - y la otra mitad no. El número promedio de movimientos de una solución es 22.

Uno de los ejercicios de la práctica consistirá en definir una función que dado un tablero nos diga si tiene solución.
Podemos ver el proceso de búsqueda de una solución como el *recorrido en profundidad* de un árbol que tiene como raíz el tablero de partida, en el que cada nodo conteniendo un tablero *t* tiene como nodos hijos los tableros a los que se puede llegar en un movimiento desde *t*.
Supongamos que la función recibe como argumento un tablero que no tiene solución; el promedio de movimientos posibles a partir de un tablero es de 2.6 (2 si el blanco está situado en cualquiera de las cuatro esquinas, 4 si el blanco está en el centro y 3 para las restantes

cuatro posiciones).
Si por ejemplo fijamos la limitación de explorar secuencias de movimientos a partir del tablero inicial que impliquen como máximo 22 movimientos, el número de tableros diferentes que se generarán en una búsqueda exhaustiva hasta concluir que o bien el tablero de partida no tiene solución o de tenerla requiere más de 22 movimientos, será de 2.6^22, cantidad que supera los 1.347 millones de movimientos.
Obviamente si en la búsqueda de solución evitamos revisitar tableros la cantidad se reduce drásticamente, pero aún así deberemos además imponer límites a la longitud o tiempo de ejecución de las búsquedas.

Existe un atajo pero no lo tomaremos: se sabe que un tablero tiene solución si y sólo si el número de inversiones del tablero inicial es par; dos casillas forman una inversión si los números que contienen están mal ordenados respecto a su posición en el tablero final.
La explicación de este resultado es que el tablero objetivo tiene 0 inversiones, y se puede comprobar que un movimiento horizontal no altera el número de inversiones y un movimiento vertical o bien lo mantiene inalterado o bien lo incrementa o decrementa en 2 unidades.

Tablero 1 *Tablero 2*

Fig. 2

En la Figura 2, el primer tablero tiene solución, pues el número de inversiones es 10, pero el segundo, con 11, no.

PRÁCTICA - PARTE 1 (fichero Puzle8_1.hs)

Hay 4 funciones puntuables:

1. Función que dado un tablero inicial dice si **tiene solución,** es decir, si existe una secuencia de 0 o más movimientos que permitan alcanzar el tablero objetivo (sin recurrir al resultado conocido sobre la paridad del número de inversiones).
Restringiremos las búsquedas a soluciones que impliquen como máximo 15 movimientos, por limitaciones en la capacidad de cómputo.

Pista: *resolver recursivamente el problema,* según el esquema a continuación.
a. Identificar el **caso base**: ¿de qué tablero podemos decir sin necesidad de llamadas recursivas que tiene una solución?
b. Si el *tablero* no representa un caso base, sabemos que tendrá solución si es posible encontrar un ***movimiento*** **que nos lleve a un tablero que sí la tiene**.
Será necesario evitar que la función cicle indefinidamente explorando un camino (secuencia de movimientos) en el que se revisiten tableros.
Se os proporcionará una lista de tableros con y sin solución y para puntuar es requisito imprescindible que vuestra función detecte correctamente los tableros que tienen y que no tienen solución. **Sólo puntúan los códigos que superen las pruebas** (es decir, que hagan lo que se pide en el enunciado). **(5 ptos.)**

2. **Generador de tableros con solución:** generador de tableros con al menos una solución en un número de pasos aleatorio menor que uno dado. Generar directamente las funciones, sin pasar por listas.
(0.5 ptos.)
3. Función ***prettyprint*** que imprime un tablero de la siguiente forma :

```
-------
|1|2|3|
|4|5|6|      <=== ejemplo: tablero objetivo
|7|8| |
-------
```

(0.5 ptos.)
(Obs: no se puntuarán definiciones con una línea de código para cada fila del tablero)

4. Función que dado un tablero me da **todas las soluciones** (nos limitaremos a soluciones consistentes en un máximo de 15 movimientos). **(1.5 ptos.)**

4-Alternativa Función que dado un tablero con solución me **da una solución** (limitarse a soluciones con longitud máxima 15 movimientos). Si se elige esta opción en lugar de la función que da todas las soluciones, la pregunta 4 puntúa la mitad. Puede en cualquier caso ser útil hacerla si no se sabe cómo atacar la función que devuelve todas las soluciones. **(0.75 ptos.)**

Para todas las funciones, sólo puntúan los códigos que calculen correctamente lo que pide el enunciado.

Observaciones sobre los tipos, las funciones y las pruebas

. El tipo **Tablero** se define como una **tupla** cuya primera componente es la **posición en que se encuentra el hueco** (es un dato que habrá que consultar frecuentemente) y una **función** que a cada **posición** del tablero le asigna un entero comprendido entre 1 y 8 envuelto en un **tipo Maybe**, de forma que el blanco del tablero se representa con **Nothing**.

. Un **movimiento** se representa con una tupla (x,y) que indica la posición en el tablero de la casilla que se ha desplazado a la posición del hueco.

. Una **solución** es una secuencia de movimientos. Para el ejemplo de la ***Figura 1***, la solución vendría dada por la lista de movimientos **[(1,2), (2,2), (2,3), (3,3)] .** Los movimientos se aplican en **orden de izquierda a derecha**.

. Se os proporciona una lista de **tableros para pruebas.** Los tableros se mostrarán - mediante la definición de **show** para tableros - como la lista de *Maybe Int* resultante de recorrer el tablero de izquierda a derecha y de arriba abajo:
> tableroFinal
[Just 1,Just 2,Just 3,Just 4,Just 5,Just 6,Just 7,Just 8,Nothing]

. Para trabajar con los tableros de prueba necesitaréis por tanto una función que dada una lista como la anterior devuelva el tablero correspondiente (función auxiliar ***list2tab***).

. Igualmente necesitaréis **definir una función show** para tableros si queréis visualizar los tableros resultantes de la evaluación de algunas de las funciones sugeridas.

. Ejemplo de aplicación de la función auxiliar que dados un movimiento y un tablero devuelve el tablero resultante de realizar el movimiento:
> **actualizaTab** (2,3) tableroFinal
[Just 1,Just 2,Just 3,Just 4,Just 5,Nothing,Just 7,Just 8,Just 6]

. Ejemplo de aplicación de la función auxiliar que dado un tablero devuelve la lista de movimientos posibles:
> **movPosibles** tableroFinal
[(2,3),(3,2)]

. Ejemplo de aplicación que dados un tablero y una lista de tableros ya visitados devuelva la lista de tableros no visitados previamente a los que se puede llegar en un movimiento:
> **nextTabs** tableroFinal []
[[Just 1,Just 2,Just 3,Just 4,Just 5,Nothing,Just 7,Just 8,Just 6],[Just 1,Just 2,Just 3,Just 4,Just 5,Just 6,Just 7,Nothing,Just 8]]

. Para **testar** la función **dameTodasSol** teneis que evaluar
> **check testTodasSol**
Si es vuestra función es correcta evaluará a **“Test superado”**.