Práctica sudoku

Arturo Precioso Garcelán

December 2022

1 Uso básico

Para utilizar sudoku.hs lo más cómodo es:

(1) Introducimos la función *iniciar_sudoku*

(2) Seguimos las indicaciones en pantalla para generar un archivo nombre.txt para jugar.

```
Main> iniciar_sudoku
Dame el nombre del nombre del sudoku: sudoku_1
Tamano del sudoku (formato (x,y)): (3,3)
0 0 0 | 4 0 0 | 0 0 5
0 0 0 | 6 7 0 | 2 0 0
7 6 5 | 0 8 0 | 0 0 9
- - - - - - - - - - -
0 0 7 | 0 0 8 | 0 0 0
0 0 8 | 9 0 0 | 0 4 0
2 0 0 | 0 0 0 | 1 0 0
- - - - - - - - - - -
0 3 2 | 0 9 0 | 0 0 6
9 0 0 | 0 0 0 | 0 1 0
0 0 6 | 3 0 2 | 7 0 0

jugar nombre para jugar
```

(3) Introducimos *jugar* "*nombre*" para jugar.

2 Función jugar

jugar te guía a la hora de jugar al sudoku. Eliges la posición que quieres cambiar:

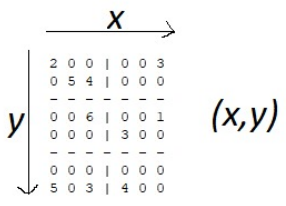

Figure 1: Cómo indicar la posición a cambiar, empezando en (1,1)

No te deja poner una solución incorrecta, puedes elegir que te de una pista de qué puede ir ahí o no en caso de que falles. Desde ese menú también se puede acceder a que te dé la solución de esa casilla o la solución de todo el sudoku. Si en cualquier momento frenamos la ejecución del bucle podemos volver a abrir nuestro fichero sudoku y seguir jugándolo.

```
Main> jugar "dos"
0 0 | 2 0
0 3 | 0 0
- - - - -
0 0 | 0 1
0 0 | 4 0

Posicion a cambiar (x,y): (1,1)
Por que numero lo quieres cambiar: 2
Ese numero no encaja ahi, si quieres una pista escribe s, sino escribe n: s
La lista de opciones es [1,4]
0 0 | 2 0
0 3 | 0 0
- - - - -
0 0 | 0 1
0 0 | 4 0
```

Figure 2: Pidiendo una pista

```
Main> jugar "dos"
0 0 | 2 0
0 3 | 0 0
- - - - -
0 0 | 0 1
0 0 | 4 0

Posicion a cambiar (x,y): (1,1)
Por que numero lo quieres cambiar: 3
Ese numero no encaja ahi, si quieres una pista escribe s, sino escribe n: solucion
La solucion es:
1 4 | 2 3
2 3 | 1 4
- - - - -
4 2 | 3 1
3 1 | 4 2
```

Figure 3: Pidiendo la solución

3 Definición de datos

```haskell
data Sudoku = Sudoku [[Celda]] (Int,Int)
   deriving(Eq, Read, Show)

data Celda = Def Int | May [Int]
   deriving(Eq, Read)

instance Show Celda where
   show (Def x) = show x
   show (May xs) = show 0

type Pos = (Int, Int)
```

El tipo Celda tiene dos opciones, $Def\ Int$ si sabemos que definitivamente ese número encaja en el sudoku o $May\ [Int]$ que es una celda vacía con una lista de los posibles candidatos a ser definitivos.

El tipo Sudoku es de la forma $Sudoku\ [[Celda]]\ (Int, Int))$, donde $[[Celda]]$ es la matriz del sudoku en sí y la tupla (Int, Int) indica las dimensiones de un subcuadrado del sudoku (por ejemplo, el sudoku clásico es un (3,3)).

Pos es útil para poder discernir entre la posición de una celda o las dimensiones de un sudoku, facilitando la lectura del código.

4 Listado de funciones

4.1 Funciones para el usuario

Principalmente son las dos ya mencionadas *iniciar_sudoku* y *jugar*, pero si se define un sudoku correctamente se pueden usar más para probar funciones individualmente.

4.2 Funciones útiles genéricas

En este apartado se incluyen funciones útiles genéricas a la hora de manejar listas y matrices, como trasponer, elementos comunes entre listas, drop n pero para la cola... Todas las funciones vienen con su respectiva explicación en el código

4.3 Funciones útiles para sudokus

Tenemos una serie de funciones que son útiles para trabajar con sudokus, como que nos de una fila, una columna, un rectángulo, cambiar el valor de una celda, generar un sudoku ordenado resuelto...

4.4 Aleatoriedad y generar nuestro sudoku para jugar

En este apartado hacemos uso de la biblioteca System.Random de Haskell. Para no importar la biblioteca System.Random.Shuffle se hace uso de una función llamada $fisherYates$ que baraja una lista. A partir de ahí muchas de nuestras funciones tendran de entrada un *StdGen* que será la semilla aleatoria que generamos en *iniciar_juego* y a partir de ahí estas funciones mismas "avanzaran" la semilla *rng* para generar la aleatoriedad que buscamos.

Para crear este *sudoku_jugable* seguimos los siguientes pasos:

(1) Usamos una serie de funciones para barajar nuesto sudoku ordenado.

(2) Una vez barajado, quitamos aleatoriamente casillas del sudoku.

(3) Cuando quitamos una casilla, vemos si el sudoku tiene solución. Sino quitamos otra casilla

4.5 Solucionar el sudoku

Para resolver el sudoku hacemos uso principalmente de dos funciones:

(1) *posible_num* mira si una casilla tiene solución por las reglas del sudoku. También reduce la lista de una casilla del tipo $May\ [Int]$ como 0 si puede.

(2) *redumay* mira si alguna casilla es la única capaz de albergar un número:

```
[May [1,2],Def 3,May[2,4],May[2,4]]
```

Figure 4: La casilla de la izquierda de la fila es la única que puede tener un 1, por tanto es definitivamente un 1

Se aplican ambas funciones a todo el sudoku con *solucionar* y se itera con *solucionar_iter* hasa que se da con una solución o no cambie el sudoku.

4.6 Enseñar el sudoku en pantalla

Esta serie de funciones ayudan a enseñar el sudoku en pantalla con sus respectivos separadores de bloques y poniendo los $May\ [Int]$ como 0 para indicar al usuario que son casillas vacías.

```
Main> iniciar_sudoku
Dame el nombre del nombre del sudoku: a
Tamano del sudoku (formato (x,y)): (5,5)
0 0 2 0 0 | 4 7 0 3 5 | 22 1 0 0 12 | 0 8 25 0 0 | 0 9 16 14 0
0 0 16 9 10 | 22 12 1 21 0 | 11 23 15 2 0 | 0 24 0 3 0 | 18 0 13 0 25
1 21 0 0 22 | 25 19 8 18 13 | 4 0 3 5 7 | 0 14 10 20 9 | 15 0 2 0 11
0 0 5 0 0 | 0 0 0 0 0 | 0 0 0 0 0 | 2 23 0 15 0 | 21 12 17 1 22
8 18 0 0 0 | 11 6 23 0 0 | 0 0 20 16 9 | 17 0 0 0 0 | 3 7 5 24 4
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0 1 0 17 18 | 0 13 6 8 0 | 20 0 0 10 0 | 22 0 21 14 16 | 23 2 0 0 3
0 8 11 0 0 | 0 0 0 23 4 | 21 0 0 22 0 | 25 19 18 0 17 | 0 0 0 0 0
9 0 0 0 20 | 0 16 12 0 22 | 0 6 8 0 13 | 0 7 0 23 0 | 0 0 25 0 18
12 0 0 0 21 | 18 17 0 1 25 | 0 7 0 4 2 | 10 0 20 0 5 | 8 13 11 6 0
7 23 0 0 3 | 20 5 0 24 10 | 18 19 1 0 17 | 11 6 15 0 13 | 14 16 0 12 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0 10 0 14 0 | 0 0 18 0 19 | 2 0 11 7 23 | 9 0 0 4 24 | 25 8 6 15 13
0 4 0 24 5 | 16 14 21 0 12 | 0 0 25 0 0 | 0 3 2 11 23 | 0 0 0 18 0
18 22 19 0 17 | 13 8 0 25 0 | 5 20 4 9 0 | 0 0 16 0 14 | 11 0 7 0 2
15 0 6 8 13 | 2 0 3 0 7 | 16 0 0 12 14 | 19 18 17 22 0 | 0 0 0 20 5
0 0 7 0 0 | 0 24 20 4 9 | 17 0 22 19 1 | 0 15 0 0 0 | 0 14 0 0 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
0 6 3 0 23 | 24 4 5 7 0 | 0 0 12 18 22 | 15 13 8 0 25 | 9 0 0 16 0
0 7 20 4 0 | 0 0 0 9 0 | 8 0 19 0 25 | 3 2 23 0 11 | 12 0 18 17 1
13 0 15 0 0 | 0 0 0 6 0 | 14 16 9 21 0 | 18 17 1 0 22 | 7 4 20 5 24
0 0 0 10 0 | 1 22 0 0 18 | 23 2 6 0 11 | 0 0 0 0 4 | 0 0 0 0 0
0 12 18 22 0 | 0 0 0 0 0 | 0 5 7 20 0 | 21 16 14 0 0 | 0 11 3 0 0
- - - - - - - - - - - - - - - - - - - - - - - - - - - - -
10 5 14 0 9 | 0 21 0 16 1 | 6 11 0 0 15 | 24 0 0 0 0 | 0 0 0 0 19
11 0 0 15 6 | 7 3 0 2 0 | 12 0 16 0 0 | 8 0 19 0 18 | 0 20 14 10 9
0 0 1 21 12 | 0 0 0 17 8 | 0 0 2 0 3 | 14 0 0 5 20 | 0 15 0 11 0
4 2 24 0 7 | 9 20 10 5 14 | 0 0 17 0 0 | 0 11 6 0 15 | 16 21 0 22 12
0 17 8 18 0 | 6 0 11 0 23 | 0 10 5 0 20 | 0 22 0 0 0 | 0 3 24 4 0

jugar nombre para jugar

Main> |
```

Figure 5: Tarda mucho en generar sudokus de dimensiones altas y como tienen números de dos digitos se ven raros, pero son jugables