README.md

Sudoku Nonominó

Proyecto de programación Declarativa - C411 Hieu Do Ngoc

Detalles de implementación

La estructura del proyecto separa cada componente de la aplicación en diferentes módulos.

```
types.hs
matching.hs
solutions.hs
tests.hs
print.hs
main.hs
```

Para la solución del problema se implementó 2 estructuras que representan los nonominós y los sudokus.

```
data Nonomino = Nonomino [(Int, Int, Int)] deriving (Show, Eq)

data Sudoku = Sudoku [((Int, Int), Nonomino)] deriving (Show, Eq)
```

El nonomino esta representado por una lista de tuplas `(x, y, n)` que representa una casilla del nonomino. Donde `x, y` representan la posicion de la casilla con respecto a la esquina superior izquierda del nonomino, y `n` es el numero que contiene esa casilla.

La casilla de la esquina superior izquierda de un nonomino es la casilla de la 1ra fila que mas a la izquierda este, por ejemplo, en este nonomino:

la casilla que contiene al numero 2 es la superior izquierda, por tanto, la representacion de nonomino contendra esta tupla `(0, 0, 2)`.

En caso de que la casilla no contenga ningun numero, se representara con un `0`. Es decir, se representara con una tupla de la forma `(x, y, 0)`.

Se ha representado el nonomino con una lista de puntos, en vez de 9 tuplas de puntos, para poder hacer uso de la comprension de listas de `Haskell` y facilitar la resolucion del problema.

Un tablero de Sudoku se presenta como una lista de tuplas que contiene a un nonomino, y la posicion que tiene en el tablero `((i, j), Nonomino)`.

Para calcular la posicion que tiene una casilla en el tablero basta con calcular `(a + i, b + j)` donde `(a, b)` es la posicion de la casilla con respecto al nonomino (El nonomino esta ubicado en la posicion `(i, j)` del tablero).

Se ha implementado una funcion para hallar las permutaciones de una lista ya que se trabaja con el compilador `Hugs` y a diferencia de `GHC`, el modulo `Data.List` no contiene la funcion `permutations` para permutar listas.

```
-- Devuelve la lista de tuplas (x, xs) donde x es un elemento de la lista dada
-- y xs es el resto
selections :: [a] -> [(a, [a])]
selections [] = []
selections (x:xs) = (x,xs) : [(y,x:ys) | (y,ys) <- selections xs]


-- Devuelve la lista de todas las posibles permutaciones de 'list'
permutations :: [a] -> [[a]]
permutations [] = [[]]
permutations list = [x:ys | (x, xs) <- selections list, ys <- permutations xs]
```

Ademas, se ha asumido que la lista de nonominos que recibe los algoritmos tienen solucion, es decir, existe una forma de armarlos tal que formen un tablero de sudoku que sea resolvible.

Ideas para la solución del problema

Para la solución del sudoku, lo primero que se debe hacer es armar los nonominos de manera que formen un sudoku valido y despues se procede a resolverlo.

Armar un tablero valido de sudoku se hace probando cada forma de colocar los nonominos de manera ordenada, es decir, se prueba colocando los nonominos en un orden, por ejemplo: (2do, 4to, 5to, 3ro, 1ro, 6to, 7mo, 9no, 8vo), si colocar los nonominos en este orden no genera un tablero valido, entonces se busca otra forma de colocarlos que si genere un tablero valido.

Para colocar los nonominos se buscar la primeras casilla de izquierda a derecha, arriba a abajo que este desocupada. Colocar los nonominos de esta forma, garantiza que la casilla en que se ubique el nuevo nonomino coincida con la esquina superior izquierda del nonomino.

Un tablero valido de sudoku ya armado se representa con el tipo `Sudoku`:

```
Sudoku [
       ((0,0), Nonomino [(0,0,0),(0,1,0),(0,2,0),(0,3,1),(0,4,0),(1,1,0),(1,2,0),(1,3,0),(2,2,0)])
       ((0,5), Nonomino [(0,0,2),(0,1,0),(0,2,3),(0,3,0),(1,-1,0),(1,0,0),(1,1,0),(1,2,9),(1,3,0)])
       ((1,0), Nonomino [(0,0,6),(1,0,0),(1,1,0),(2,0,8),(2,1,0),(2,2,0),(2,3,0),(1,3,0),(3,2,4)])
       ((2,4), Nonomino [(0,0,0),(0,1,0),(0,2,0),(0,3,8),(0,4,0),(1,3,0),(1,4,0),(2,3,0),(2,4,0)])
       ((3,4), Nonomino [(0,0,7),(1,0,0),(1,-1,5),(2,-1,0),(2,-2,2),(3,-2,9),(4,-2,1),(5,-2,0),(5,-3,0)])
       ((3,5), Nonomino [(0,0,4),(0,1,0),(1,0,6),(1,1,0),(2,0,0),(2,1,0),(2,-1,0),(3,1,0),(4,1,0)])
       ((4,0), Nonomino [(0,0,0),(0,1,0),(1,0,0),(1,1,0),(2,0,3),(2,1,0),(3,0,4),(3,1,0),(4,0,0)])
       ((5,7), Nonomino [(0,0,0),(0,1,0),(1,0,0),(1,1,0),(2,0,0),(2,1,0),(3,0,7),(3,1,0),(3,-1,0)])
       ((6,3), Nonomino [(0,0,8),(0,1,0),(0,2,5),(1,0,7),(1,1,0),(1,2,9),(2,0,3),(2,1,6),(2,2,0)])
       ]
```

En el archivo `matching.hs` se encuentran los algoritmos para armar los sudokus.

Una vez encontrada un tablero valido se procede a resolverlo. La resolucion del sudoku se hace de una manera muy sencilla, utilizando Backtracking.

Para lograr una pequeña mejora en la eficiencia del algoritmo, se ordena primero las casillas por la cantidad de opciones de numeros que pueden ser escogidos, de menor a mayor.

Sin embargo, esto solo se hace al inicio del algoritmo ya que ordenar en cada llamado recursivo resultaria en un aumento considerable de la complejidad temporal del algoritmo.

Un sudoku resuelto se representa de la misma manera que uno no resuelto, aunque que el sudoku resuelto no tendra ningun nonomino que contenga un punto con el numero `0`.

En el archivo `solution.hs` se encuentran los algoritmos para resolver los sudokus.

Modo de uso

Para usar la aplicación debe cargar `main.hs` y llamar a la funcion `main`.

```
main =
  let
    sudoku = head $ buildSudokus testNonominos
    sol = solutionSudoku sudoku
  in
    do
      putStrLn " ++ SUDOKU ARMADO ++"
      putStrLn $ unlines $ sudokuToString sudoku
      putStrLn " ++ SUDOKU SOLUCIONADO ++"
      putStrLn $ unlines $ sudokuToString sol
```

La funcion `sudokuToString` toma una representacion de sudoku y la convierte en una lista de `strings` para facilitar su muestreo en la consola interactiva de `WinHugs`.

En el archivo `tests.hs` debe estar definido una lista de nonominos validos con el nombre `testNonominos`.

```
module Tests where
  import Types

  testNonominos = [
    Nonomino [(0,0, 0), (0,1, 0), (0,2, 0), (0,3, 1), (0,4, 0), (1,1, 0), (1,2, 0), (1,3, 0), (2,2, 0)],
    Nonomino [(0,0, 2), (0,1, 0), (0,2, 3), (0,3, 0), (1,-1, 0), (1,0, 0), (1,1, 0), (1,2, 9), (1,3, 0)],
    Nonomino [(0,0, 6), (1,0, 0), (1,1, 0), (2,0, 8), (2,1, 0), (2,2, 0), (2,3, 0), (1,3, 0), (3,2, 4)],
    Nonomino [(0,0, 0), (0,1, 0), (0,2, 0), (0,3, 8), (0,4, 0), (1,3, 0), (1,4, 0), (2,3, 0), (2,4, 0)],
    Nonomino [(0,0, 7), (1,0, 0), (1,-1, 5), (2,-1, 0), (2,-2, 2), (3,-2, 9), (4,-2, 1), (5,-2, 0), (5,-3, 0)],
    Nonomino [(0,0, 4), (0,1, 0), (1,0, 6), (1,1, 0), (2,0, 0), (2,1, 0), (2,-1, 0), (3,1, 0), (4,1, 0)],
    Nonomino [(0,0, 0), (0,1, 0), (1,0, 0), (1,1, 0), (2,0, 3), (2,1, 0), (3,0, 4), (3,1, 0), (4,0, 0)],
    Nonomino [(0,0, 0), (0,1, 0), (1,0, 0), (1,1, 0), (2,0, 0), (2,1, 0), (3,0, 7), (3,1, 0), (3,-1, 0)],
    Nonomino [(0,0, 8), (0,1, 0), (0,2, 5), (1,0, 7), (1,1, 0), (1,2, 9), (2,0, 3), (2,1, 6), (2,2, 0)]
    ]
```

Para el ejemplo anterior el algoritmo que arma los nonominos devolvera como solucion el siguente sudoku:

El mismo sudoku resuelto por el algoritmo: