Тема 7. Игра „Судоку“

Борис Иванов

Специалност: Компютърни науки

Общи бележки

- Документацията в този файл и в кода е във формат JavaDoc.
- Предложеното решение на задачата намира всички решения на дадено судоку и принтира върху стандартния изход само първото (ако има такова).
- Едно судоку се представя като списък от списъци от цели числа. Празната клетка се отбелязва с 0.
- Има възможност за решаване на судокуто от сорс кода или на судокуто от файла “sudoku.txt”. При стартиране, ако съществува “sudoku.txt”, потребителят трябва да въведе 0 – решаване на судокуто от сорс кода или 1 – решаване на судокуто от “sudoku.txt”. Ако не съществува файлът “sudoku.txt”, се решава директно судокуто от сорс кода.
- Според конкретната реализация файлът "sudoku.txt" допуска всякакви форматиращи символи. Единственото условие е да съдържа точно 81 цифри, а празните клетки в судокуто да се отбелязват с цифрата 0.

Документация

Основна функционалност

1.

* В цялата реализация, едно судоку се представя като списък от списъци от цели числа - [[Int]].
* За удобство е дефиниран типът Sudoku = [[Int]].

type Sudoku = [[Int]]

2.

* Намира елемент в grid по дадена позиция

 *
 * @param grid судоку, от което взимаме елемента
 * @param i номер на ред в grid
 * @param j номер на стълб в grid
 * @return елементът на позиция (i,j) в grid

at :: Sudoku-> Int -> Int -> Int
at grid i j

3.

 * Проверява дали можем да запишем дадено число в конкретния ред на судокуто, според правилата на играта
 *
 * @param grid конкретното судоку
 * @param i номер на ред в grid
 * @param val стойността, която проверяваме дали може да се запише в i-тия ред
 * @return true ако можем да запишем val на този ред, false в противен случай

checkRow :: Sudoku -> Int -> Int -> Bool
checkRow grid i val

4.

 * Проверява дали можем да запишем дадено число в конкретния стълб на судокуто, според правилата на играта
 *
 * @param grid конкретното судоку
 * @param j номер на стълб в grid
 * @param val стойността, която проверяваме дали може да се запише в j-тия стълб
 * @return true ако можем да запишем val в този стълб, false в противен случай

checkCol :: Sudoku -> Int -> Int -> Bool
checkCol grid j val

5.

 * Проверява дали може да се добави val в конкретното квадратче 3x3, което се определя от позицията (i,j), според правилата на играта. Квадратчето е това, в което се намира елементът на позиция (i,j)
 *
 * @param grid конкретното судоку
 * @param i номер на ред в grid
 * @param j номер на стълб в grid
 * @param val стойността, която проверяваме дали може да се запише в 3x3 квадратчето, определено от позицията (i,j)
 * @return true ако можем да запишем val в това квадратче, false в противен случай

checkSquare :: Sudoku -> Int -> Int -> Int -> Bool
checkSquare grid i j val

6.

* Връща ново судоку, но елементът на позиция (i,j) става равен на newVal
*
* @param i номер на ред в (x:xs)
* @param j номер на стълб в (x:xs)
* @param newVal новата стойност на елемента на позиция (i,j)
* @param (x:xs) входното судоку
* @return (x:xs), но елементът на позиция (i,j) става равен на newVal

update :: Int -> Int -> Int -> Sudoku -> Sudoku
update i j newVal (x:xs)

7.

* Намира празните клетки на конкретното судоку
*
* @param grid конкретното судоку
* @return списък от наредени двойки (i,j), където (i,j) е позиция на празна клетка в grid

emptyCells :: Sudoku -> [(Int,Int)]
emptyCells grid

8.

* От тук започва процесът по решаване на судокуто. Ако няма повече празни клетки, то grid е валидно решено судоку и връщаме списък с единствен елемент grid. В противен случай се извиква функцията possibleSolutions с аргументи grid и първата позиция на празна клетка в grid. По този начин, чрез possibleSolutions се генерират всички възможни запълвания и съответно, ако стигнем до невалидно судоку, не го добавяме към решението.
*
* @param grid входното судоку
* @return списък от всички решения на grid. В частност, ако grid няма решения - връща празен списък.

solve :: Sudoku -> [Sudoku]
solve grid

9.

* Запълва празната клетката (i,j) с всички възможни стойности, получени от possibleValues и продължава нататък със запълването като извиква процедурата solve за всяка възможна стойност. На дъното на рекурсията, ако има валидно судоку - това е едно решение. Ако за полученото на някоя стъпка судоку, possibleValues връща празен списък, то това е невалидно судоку, тъй като все още имаме празни позиции
*
* @param grid входното судоку
* @param (i,j) позиция на конкретната празна клетка в grid, която ще запълним
* @return списък от всички решения на grid, а ако няма решение на конкретното входно

судоку - празен списък

possibleSolutions :: Sudoku -> (Int, Int) -> [Sudoku]
possibleSolutions grid (i,j)

10.

```
 * Намира възможните стойности, които могат да се запишат на позиция (i,j)
 *
 * @param grid конкретното судоку
 * @param (i,j) позиция на празна клетка в grid
 * @return списък от числата, които могат да се запишат в тази клетка според правилата на
играта
```

possibleValues :: Sudoku -> (Int, Int) -> [Int]
possibleValues grid (i,j)

11.

```
 * Превръща цифра в String
 *
 * @param x цифра
 * @return String
 *
 * toStr 3 -> "3"
```

toStr :: Int -> String
toStr x

12.

```
 * Превръща дадено судоку в приятен за принтиране в конзолата вариант
 *
 * @param grid входното судоку
 * @param i ред в grid за рекурсивно итериране на grid. При първо повикване е = на 0
 * @param j стълб в grid за рекурсивно итериране на grid. При първо повикване е = на 0
 * @param result държи полученият до конкретната итерация резултат. При първо повикване е
""
 * @return String в приятен формат, готов за директно отпечатване по-късно в конзолата
```

sudokuToString :: Sudoku -> Int -> Int -> String -> String
sudokuToString grid i j result

13.

```
 * Взима всички решения на дадено судоку и принтира първото от тях
 *
 * @param solutions списък от решения на едно судоку
 * @return Ако има решения, отпечатва първото, ако няма - известява за това потребителя
```

printFirstSolution :: [Sudoku] -> IO ()
printFirstSolution solutions

Следват помощни функции за прочитане на судоку от файл

14.

```
 * Превръща даден символ в Int
 *
 * @param ch символ, който ще превърнем в Int
 * @return цяло число, съответстващо на ch, или -1, ако ch не е цифра

toInt :: Char -> Int
toInt ch
```

15.

```
 * Маха всички елементи, които не са цифрите [0,9] и превръща низа в списък от цели числа
 *
 * @param str низ, прочетен от файла "sudoku.txt"
 * @return списък от всички числа в судокуто

convertToValidList :: String -> [Int]
convertToValidList str
```

16.

```
 * Превръща низ във валидно Sudoku
 *
 * @param str низ, прочетен от файла "sudoku.txt"
 * @return валидно судоку

stringToSudoku :: String -> Sudoku
stringToSudoku str
```

17. main функция

Ако не съществува файлът "sudoku.txt", се отпечатва първото решение на судокуто от сорс кода.

Ако съществува файлът "sudoku.txt", при получен вход 0 се отпечатва първото решение на судокуто от сорс кода, а в противен случай се отпечатва първото решение на судокуто от "sudoku.txt".

Ако няма решение се известява потребителя във всички случаи.

Примерно съдържание на файла sudoku.txt

Пример 1: (директно copy-paste от судокуто в сорс кода)

```
[
    [5,3,0,    0,7,0,    0,0,0],
    [6,0,0,    1,9,5,    0,0,0],
    [0,9,8,    0,0,0,    0,6,0],

    [8,0,0,    0,6,0,    0,0,3],
    [4,0,0,    8,0,3,    0,0,1],
    [7,0,0,    0,2,0,    0,0,6],

    [0,6,0,    0,0,0,    2,8,0],
    [0,0,0,    4,1,9,    0,0,5],
    [0,0,0,    0,8,0,    0,7,9]]
```

Пример 2: (това е форматът, в който се принтира полученият резултат в конзолата)

```
5 3 0 |0 7 0 |0 0 0
6 0 0 |1 9 5 |0 0 0
0 9 8 |0 0 0 |0 6 0
------------------
8 0 0 |0 6 0 |0 0 3
4 0 0 |8 0 3 |0 0 1
7 0 0 |0 2 0 |0 0 6
------------------
0 6 0 |0 0 0 |2 8 0
0 0 0 |4 1 9 |0 0 5
0 0 0 |0 8 0 |0 7 9
```

Пример 3:

```
5 3 0 0 7 0 0 0 0
6 0 0 1 9 5 0 0 0
0 9 8 0 0 0 0 6 0
8 0 0 0 6 0 0 0 3
4 0 0 8 0 3 0 0 1
7 0 0 0 2 0 0 0 6
0 6 0 0 0 0 2 8 0
0 0 0 4 1 9 0 0 5
0 0 0 0 8 0 0 7 9
```

Пример 4:

```
5 3 0   0 7 0   0 0 0
6 0 0   1 9 5   0 0 0
0 9 8   0 0 0   0 6 0

8 0 0   0 6 0   0 0 3
4 0 0   8 0 3   0 0 1
```

```
7 0 0   0 2 0   0 0 6

0 6 0   0 0 0   2 8 0
0 0 0   4 1 9   0 0 5
0 0 0   0 8 0   0 7 9
```

Използвана литература

* https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
* http://www.geeksforgeeks.org/backtracking-set-7-suduku/
* https://spin.atomicobject.com/2012/06/18/solving-sudoku-in-c-with-recursive-backtracking/
* http://stackoverflow.com/
* http://learnyouahaskell.com/input-and-output
* https://en.wikibooks.org/wiki/Haskell/Simple_input_and_output