Metropolitan State University of Denver
Department of Mathematical and Computer Sciences

CS 3210 - Principles of Programming Languages (Spring 2020)

Programming Assignment 02

Deadline: April 5 (11:59pm)

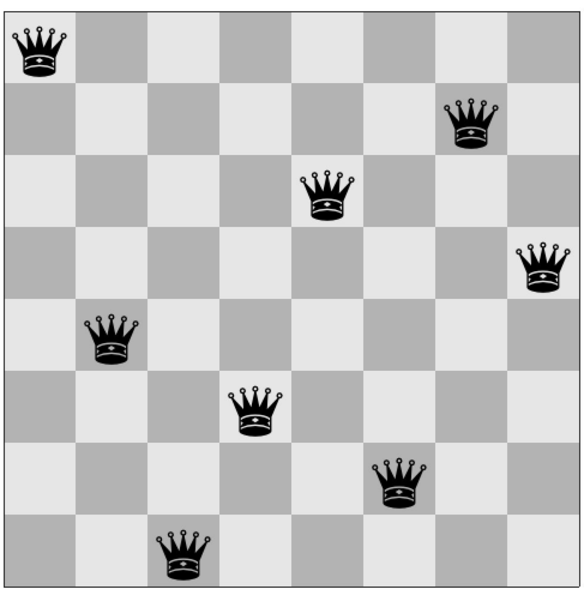
1. Introduction

The n-queens puzzle is a well known problem created by Max Bezzel, a German chess player, in 1848. To solve the puzzle one has to place `n` chess queens on an `n×n` chessboard so that no two queens threaten each other. In this assignment you are asked to write a Haskell program to find (at least) one solution to the puzzle.

In our particular Haskell implementation, the chess board is represented by `n`-lists of `n` elements of type `Char`. The following *type synonyms*[1] will be used in the solution:

```
type Seq[2]   = [Char]
type Board = [Seq]
```

At any time only 2 values can be assigned to a position on the board: `'-'` (to denote an empty location) or `'Q'` (to denote a location occupied by a queen). Below is a (solved) configuration of an `8x8` board and its correspondent representation.



Source: mathworks.com

```
[ "Q-------",
  "------Q-",
  "----Q---",
  "-------Q",
  "-Q------",
  "---Q----",
  "-----Q--",
  "--Q-----" ]
```

[1] *Type synonyms* in Haskell allow assigning names to pre-existent types.
[2] `Seq` (short for *sequence*) can be a row, a column, or a diagonal.

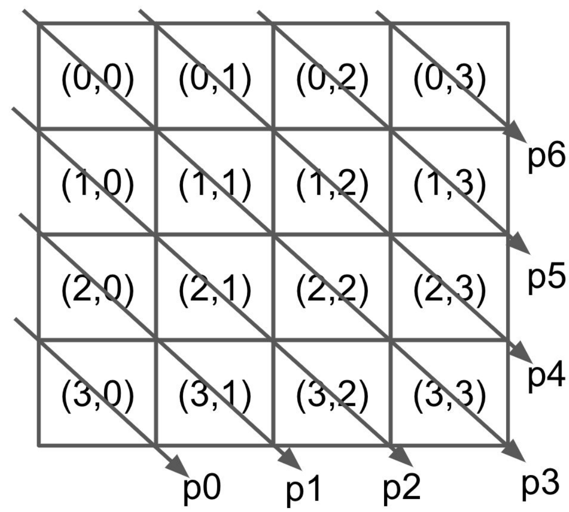
2. Diagonals

Part of the difficulty in writing a solution for the n-queens puzzle is to come up with a way to check if a queen is threatening another queen diagonally. This section explains the 2 functions that will help you solve this specific part of the problem. It uses a `4x4` board to illustrate how the functions work.

2.1. Main Diagonals

The figure below shows the main diagonals (left-right, top-bottom) of a `4x4` board.

(0,0)	(0,1)	(0,2)	(0,3)
(1,0)	(1,1)	(1,2)	(1,3)
(2,0)	(2,1)	(2,2)	(2,3)
(3,0)	(3,1)	(3,2)	(3,3)



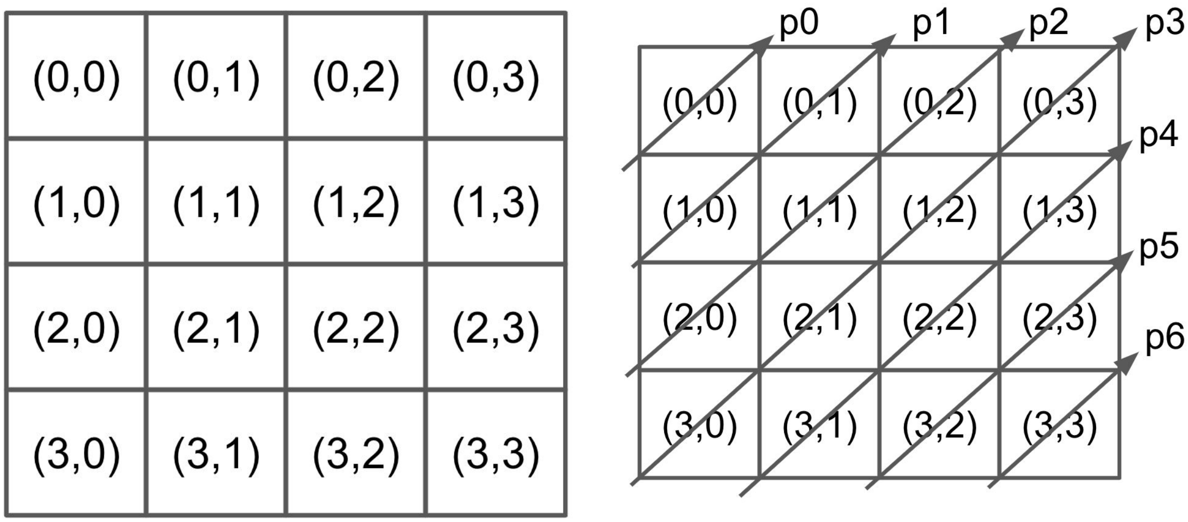

```
mainDiagIndices :: Board -> Int -> [ (Int, Int) ]
```

`mainDiagIndices` takes a `board` and a diagonal `index` and returns a sequence of tuples with the coordinates of the locations that are in the primary diagonal.

Example 1: `mainDiagIndices (setup 4) 0` results in `[(3,0)]`
Example 2: `mainDiagIndices (setup 4) 1` results in `[(2,0),(3,1)]`
Example 3: `mainDiagIndices (setup 4) 2` results in `[(1,0),(2,1),(3,2)]`
Example 4: `mainDiagIndices (setup 4) 3` results in `[(0,0),(1,1),(2,2),(3,3)]`
Example 5: `mainDiagIndices (setup 4) 4` results in `[(0,1),(1,2),(2,3)]`
Example 6: `mainDiagIndices (setup 4) 5` results in `[(0,2),(1,3)]`
Example 7: `mainDiagIndices (setup 4) 6` results in `[(0,3)]`

2.2. Secondary Diagonals

The figure below shows the secondary diagonals (left-right, bottom-top) of a `4x4` board.




```
secDiagIndices :: Board -> Int -> [ (Int, Int) ]
```

`secDiagIndices` takes a `board` and a diagonal `index` and returns a sequence of tuples with the coordinates of the locations that are in the secondary diagonal.

Example 1: `secDiagIndices (setup 4) 0` results in `[(0,0)]`
Example 2: `secDiagIndices (setup 4) 1` results in `[(1,0),(0,1)]`
Example 3: `secDiagIndices (setup 4) 2` results in `[(2,0),(1,1),(0,2)]`
Example 4: `secDiagIndices (setup 4) 3` results in `[(3,0),(2,1),(1,2),(0,3)]`
Example 5: `secDiagIndices (setup 4) 4` results in `[(3,1),(2,2),(1,3)]`
Example 6: `secDiagIndices (setup 4) 5` results in `[(3,2),(2,3)]`
Example 7: `secDiagIndices (setup 4) 6` results in `[(3,3)]`

3. TO DO

Your goal in this assignment is to correctly implement the functions described in this section. We strongly recommend testing each of the functions using the provided inputs/outputs.

`setup :: Int -> Board`

`setup` takes an integer `n >= 4` and creates an `nxn` board with all locations empty. If `n < 4` `setup` should return a `4x4` board.

Test 1: `setup 3` result in `["----","----","----","----"]`
Test 2: `setup 5` results in `["-----","-----","-----","-----","-----"]`

`rows :: Board -> Int`

`rows` takes a `board` and returns its number of rows.

Test 1: `rows ["----","----","----","----"]` results in `4`
Test 2: `rows ["-----","-----","-----","-----","-----"]` results in `5`

`cols :: Board -> Int`

`cols` takes a `board` and returns its number of columns if all rows have the same number of columns; it returns zero, otherwise.

Test 1: `cols ["----","----","----","----"]` results in `4`
Test 2: `cols ["-----","-----","-----","-----","-----"]` results in `5`
Test 3: `cols ["----","-----","-----","-----","-----"]` results in `0`

`size :: Board -> Int`

`size` takes a `board` and returns its size, which is the same as its number of rows (if it matches its number of columns), or zero, otherwise.

Test 1: `size ["----","----","----","----"]` results in `4`
Test 2: `size ["-----","-----","-----","-----","-----"]` results in `5`
Test 3: `size ["----","-----","-----","-----","-----"]` results in `0`

`queensSeq :: Seq -> Int`

`queensSeq` takes a `sequence` and returns the number of queens found in it.

Test 1: `queensSeq "----"` results in `0`
Test 2: `queensSeq "-Q--"` results in `1`
Test 3: `queensSeq "-Q-Q"` results in `2`

`queensBoard :: Board -> Int`

`queensBoard` takes a `board` and returns the number of queens found in it.

Test 1: `queensBoard ["----", "----", "----", "----"]` results in `0`
Test 2: `queensBoard ["Q---", "--Q-", "--Q-", "----"]` results in `3`

`seqValid :: Seq -> Bool`

`seqValid` takes a `sequence` and returns true/false depending whether the sequence no more than `1` queen.

Test 1: `seqValid "----"` results in `True`
Test 2: `seqValid "-Q--"` results in `True`
Test 3: `seqValid "-Q-Q"` results in `False`

`rowsValid :: Board -> Bool`

`rowsValid` takes a `board` and returns true/false depending whether ALL of its rows correspond to valid sequences.

Test 1: `rowsValid ["----","----","----","----"]` results in `True`
Test 2: `rowsValid ["-Q--","----","----","----"]` results in `True`
Test 3: `rowsValid ["-Q--","Q---","----","----"]` results in `True`
Test 4: `rowsValid ["-Q--","Q---","--Q-","----"]` results in `True`
Test 5: `rowsValid ["-Q--","Q---","--QQ","----"]` results in `False`

```
colsValid :: Board -> Bool
```

`colsValid` takes a `board` and returns true/false depending whether ALL of its columns correspond to valid sequences.

Test 1: `colsValid ["----","----","----","----"]` results in `True`
Test 2: `colsValid ["-Q--","----","----","----"]` results in `True`
Test 3: `colsValid ["-Q--","Q---","----","----"]` results in `True`
Test 4: `colsValid ["-Q--","Q---","--Q-","----"]` results in `True`
Test 5: `colsValid ["-Q--","Q---","--QQ","----"]` results in `True`
Test 6: `colsValid ["-Q--","Q---","--QQ","---Q"]` results in `False`

```
diagonals :: Board -> Int
```

`diagonals` takes a `board` and returns its number of primary (or secondary) diagonals. If a `board` has size `n`, its number of diagonals is given by the formula: `2 x n - 1`.

Test 1: `diagonals (setup 4)` results in `7`.
Test 2: `diagonals (setup 5)` results in `9`.

```
allMainDiagIndices :: Board -> [[ (Int, Int) ]]
```

`allMainDiagIndices` takes a `board` and returns a list of all primary diagonal indices.

Test 1: `allMainDiagIndices (setup 4)` results in:

```
[
[(3,0)],
[(2,0),(3,1)],
[(1,0),(2,1),(3,2)],
[(0,0),(1,1),(2,2),(3,3)],
[(0,1),(1,2),(2,3)],
[(0,2),(1,3)],
[(0,3)]
]
```

Test 2: `allMainDiagIndices (setup 5)` results in:

```
[
[(4,0)],
[(3,0),(4,1)],
[(2,0),(3,1),(4,2)],
```

```
[(1,0),(2,1),(3,2),(4,3)],
[(0,0),(1,1),(2,2),(3,3),(4,4)],
[(0,1),(1,2),(2,3),(3,4)],
[(0,2),(1,3),(2,4)],
[(0,3),(1,4)],
[(0,4)]
]
```

`mainDiag :: Board -> [Seq]`

`mainDiag` takes a `board` and returns a list of all primary diagonal elements.

Test 1: `mainDiag (setup 4)` results in:

```
["-","--","---","----","---","--","-"]
```

Test 2: `mainDiag (setup 5)` results in:

```
["-","--","---","----","-----","----","---","--","-"]
```

`allSecDiagIndices :: Board -> [[ (Int, Int) ]]`

`allSecDiagIndices` takes a `board` and returns a list of all secondary diagonal indices.

Test 1: `allSecDiagIndices (setup 4)` results in:

```
[
[(0,0)],
[(1,0),(0,1)],
[(2,0),(1,1),(0,2)],
[(3,0),(2,1),(1,2),(0,3)],
[(3,1),(2,2),(1,3)],
[(3,2),(2,3)],
[(3,3)]
]
```

Test 2: `allSecDiagIndices (setup 5)` results in:

```
[
[(0,0)],
[(1,0),(0,1)],
[(2,0),(1,1),(0,2)],
[(3,0),(2,1),(1,2),(0,3)],
[(4,0),(3,1),(2,2),(1,3),(0,4)],
```

```
[(4,1),(3,2),(2,3),(1,4)],
[(4,2),(3,3),(2,4)],
[(4,3),(3,4)],
[(4,4)]
]
```

```
secDiag :: Board -> [Seq]
```

`secDiag` takes a `board` and returns a list of all secondary diagonal elements.

Test 1: `secDiag (setup 4)` results in:
`["-","--","---","----","---","--","-"]`

Test 2: `secDiag (setup 5)` results in:
`["-","--","---","----","-----","----","---","--","-"]`

```
diagsValid :: Board -> Bool
```

`diagsValid` takes a `board` and returns true/false depending whether all of its primary and secondary diagonals are valid.

Test 1: `diagsValid ["Q---","--Q-","-Q--","---Q"]` results in `False` (primary and secondary diagonal fail).
Test 2: `diagsValid ["-Q--","---Q","Q---","-Q--"]` results in `False` (primary and secondary diagonal fail).
Test 3: `diagsValid ["Q----","--Q--","---Q-","-Q---", "---Q-"]` results in `False` (primary diagonal fail).
Test 4: `diagsValid ["---Q-","-----","-Q---","-----", "----Q"]` results in `False` (secondary diagonal fail).
Test 5: `diagsValid ["--Q-----","------Q-","-Q------","-------Q","-----Q--","---Q----","Q-------","----Q---"]` results in `True`.

```
valid :: Board -> Bool
```

`valid` takes a `board` and returns true/false depending whether the board configuration is valid (i.e., no queen is threatening another queen).

Test 1: `valid ["Q---","--QQ","----","----"]` results in `False` (`rowsValid` fail).
Test 2: `valid ["Q---","--Q-","--Q-","----"]` results in `False` (`colsValid` fail).
Test 3: `valid ["Q---","--Q-","-Q--","---Q"]` results in `False` (`diagsValid` fail).
Test 4: `valid ["--Q-----","------Q-","-Q------","-------Q","-----Q--","---Q----","Q-------","----Q---"]` results in `True`.

`solved :: Board -> Bool`

`solved` takes a `board` and returns true/false depending whether the board configuration is solved (i.e., the configuration is valid and also has the right amount of queens based on the board's size).

Test 1: `solved ["Q---","--Q-","----","----"]` results in `False` (not enough queens).
Test 2: `solved ["--Q-","Q---","---Q","-Q--"]` results in `True`

4. Solving the N-Queens Puzzle

This section describes the last three functions of this implementation aimed to find a solution to the problem.

`setQueenAt :: Board -> Int -> [Board]`

`setQueenAt` takes a `board` and returns a list of new board configurations, each with a queen added at all of the possible columns of a given `row` index.

Test 1: `setQueenAt ["Q---","--Q-","----","----"] 2` results in

```
[
["Q---","--Q-","Q---","----"],
["Q---","--Q-","-Q--","----"],
["Q---","--Q-","--Q-","----"],
["Q---","--Q-","---Q","----"]
]
```

Test 2: `setQueenAt ["Q---","--Q-","-Q--","----"] 3` results in

```
[
["Q---","--Q-","-Q--","Q---"],
["Q---","--Q-","-Q--","-Q--"],
```

```
["Q---","--Q-","-Q--","--Q-"],
["Q---","--Q-","-Q--","---Q"]
]
```

```
nextRow :: Board -> Int
```

`nextRow` takes a board and returns the first row index (from top to bottom) that does not have any queen.

Test 1: `nextRow ["Q---","--Q-","----","----"]` results in `2`.
Test 2: `nextRow ["Q---","--Q-","-Q--","----"]` results in `3`.

```
solve :: Board -> [Board]
```

`solve` takes a `board` and returns ALL of the board configurations that solve the n-queens puzzle.

Test 1: `solve (setup 4)` results in

```
[
["-Q--","---Q","Q---","--Q-"],
["--Q-","Q---","---Q","-Q--"]
]
```

Test 2: `solve (setup 5)` results in

```
[
["Q----","--Q--","----Q","-Q---","---Q-"],
["Q----","---Q-","-Q---","----Q","--Q--"],
["-Q---","---Q-","Q----","--Q--","----Q"],
["-Q---","----Q","--Q--","Q----","---Q-"],
["--Q--","Q----","---Q-","-Q---","----Q"],
["--Q--","----Q","-Q---","---Q-","Q----"],
["---Q-","Q----","--Q--","----Q","-Q---"],
["---Q-","-Q---","----Q","--Q--","Q----"],
["----Q","-Q---","---Q-","Q----","--Q--"],
["----Q","--Q--","Q----","---Q-","-Q---"]
]
```

5. Rubric

```
+5 setup
+5 rows
+5 cols
+5 size
+5 queensSeq
+5 queensBoard
+5 seqValid
+5 rowsValid
+5 colsValid
+5 diagonals
+7 allMainDiagIndices
+7 mainDiag
+7 allSecDiagIndices
+7 secDiag
+7 diagsValid
+5 valid
+5 solved
```

+5 `nqueens.hs` submitted through BB as an attachment (instead of a copy-and-paste)