99 questions/Solutions/90

From HaskellWiki

< 99 questions | Solutions

This is a classical problem in computer science. The objective is to place eight queens on a chessboard so that no two queens are attacking each other; i.e., no two queens are in the same row, the same column, or on the same diagonal.

The simplest solution is a composition of separate functions to generate the list of candidates and to test each candidate:

```
queens :: Int -> [[Int]]
queens n = filter test (generate n)
    where generate 0      = [[]]
          generate k      = [q : qs | q <- [1..n], qs <- generate (k-1)]
          test []         = True
          test (q:qs)     = isSafe q qs && test qs
          isSafe   try qs = not (try `elem` qs || sameDiag try qs)
          sameDiag try qs = any (\(colDist,q) -> abs (try - q) == colDist) $ zip [1..] qs
```

By definition/data representation no two queens can occupy the same column.

`try `elem` alreadySet`
checks for a queen in the same row,
`abs (try - q) == col`
checks for a queen in the same diagonal.

This is easy to understand, but it's also quite slow, as it generates and tests N^N possible N-queen configurations.

The key to speeding it up is to fuse the composition
`filter test . generate`
into a semantically equivalent function
`queens'`
that does the tests as early as possible.

If a list already contains two queens in a line, there's no point in considering all the possible ways of adding more queens. Now that the recursive call incorporates testing, we avoid recomputing it by interchanging the two generators, and reverse each answer at the end to obtain the original order. This yields the following version, which is much faster:

```
queens :: Int -> [[Int]]
queens n = map reverse $ queens' n
    where queens' 0       = [[]]
          queens' k       = [q:qs | qs <- queens' (k-1), q <- [1..n], isSafe q qs]
          isSafe   try qs = not (try `elem` qs || sameDiag try qs)
```

```
        sameDiag try qs = any (\(colDist,q) -> abs (try - q) == colDist) $ zip [1..] qs
```

A solution using the
`Date.List`
's
`permutations`
is:

```
import Prelude.Unicode
import Data.List

queues :: Int → [[Int]]
queues n = filter (\x → test (zip [1..n] x)) candidates
    where candidates = permutations [1..n]
          unSafe (x₁, y₁) (x₂, y₂) = (y₁ ≡ y₂) ∨
                                     ((abs (x₁ - x₂)) ≡ (abs (y₁ - y₂)))
          test [] = True
          test (q:qs) = (not $ (any (\x → unSafe q x) qs)) ∧ (test qs)
```

If you approach this problem with an imperative mindset, you might be tempted to use an accumulating parameter for the list of candidates. This would make the function harder to understand, and would not help much (if at all): the important thing here is the breadth of the search tree, not its depth.
