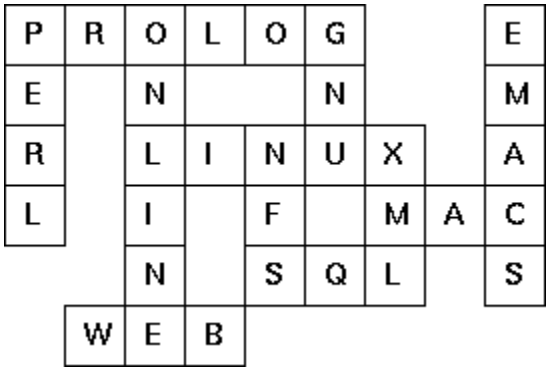
99 questions/Solutions/99

From HaskellWiki

< 99 questions | Solutions

Given an empty (or almost empty) framework of a crossword puzzle and a set of words. The problem is to place the words into the framework.



The particular crossword puzzle is specified in a text file which first lists the words (one word per line) in an arbitrary order. Then, after an empty line, the crossword framework is defined. In this framework specification, an empty character location is represented by a dot (.). In order to make the solution easier, character locations can also contain predefined character values. The puzzle above is defined in the file p99a.dat (http://www.hta-bi.bfh.ch/~hew/informatik3/prolog/p-99/p99a.dat) , other examples are p99b.dat (http://www.hta-bi.bfh.ch/~hew/informatik3/prolog/p-99/p99b.dat) and p99d.dat (http://www.hta-bi.bfh.ch/~hew/informatik3/prolog/p-99/p99d.dat) . There is also an example of a puzzle (p99c.dat (http://www.hta-bi.bfh.ch/~hew/informatik3/prolog/p-99/p99c.dat)) which does not have a solution.

Words are strings (character lists) of at least two characters. A horizontal or vertical sequence of character places in the crossword puzzle framework is called a site. Our problem is to find a compatible way of placing words onto sites.

Hints: (1) The problem is not easy. You will need some time to thoroughly understand it. So, don't give up too early! And remember that the objective is a clean solution, not just a quick-and-dirty hack!

(2) Reading the data file is a tricky problem for which a solution is provided in the file p99-readfile.pl (http://www.hta-bi.bfh.ch/~hew/informatik3/prolog/p-99/p99-readfile.pl) . See the predicate read_lines/2.

(3) For efficiency reasons it is important, at least for larger puzzles, to sort the

words and the sites in a particular order. For this part of the problem, the solution of P28 may be very helpful.

Solution:

```
-- import Control.Monad
-- import Data.List
import Data.Ord (comparing)
import Data.Function (on)

type Coord     = (Int,Int)
type Word      = String
data Site      = Site {siteCoords :: [Coord], siteLen :: Int} deriving (Show,Eq)
data Crossword = Crossword {cwWords :: [Word], cwSites :: [Site]}  deriving (Show,Eq)

equaling = ((==) `on`)

-- convert the text lines from the file to the "Site" datatype,
--   which contain the adjacent coordinates of the site and its length
toSites :: [String] -> [Site]
toSites lines = find (index_it lines) ++ find (transpose . index_it $ lines)
    where find       = map makePos . concatMap extractor
          extractor  = filter ((>1) . length) . map (filter ((=='.').snd)) . groupBy (equalin
          index_it   = zipWith (\row -> zip [(col,row) | col <- [1..]]) [1..]
          makePos xs = Site {siteCoords = map fst xs, siteLen = length xs}

-- test whether there exist no two different letters at the same coordinate
noCollision :: [(String, Site)] -> Bool
noCollision xs = all allEqual groupedByCoord
    where groupedByCoord = map (map snd) . groupBy (equaling fst) . sortBy (comparing fst) .
          allEqual []     = True
          allEqual (x:xs) = all (x==) xs

-- merge a word and a site by assigning each letter to its respective coordinate
together :: (Word, Site) -> [(Coord, Char)]
together (w,s) = zip (siteCoords s) w

-- returns all solutions for the crossword as lists of occupied coordinates and their respect
solve :: Crossword -> [[(Coord, Char)]]
solve cw = map (concatMap together) solution
    where solution = solve' (cwWords cw) (cwSites cw)

solve' :: [Word] -> [Site] -> [[(Word, Site)]]
solve' _     []     = [[]]
solve' words (s:ss) = if null possWords
                      then error ("too few words of length " ++ show (siteLen s))
                      else do try <- possWords
                              let restWords = Data.List.delete try words
                              more <- solve' restWords ss
                              let attempt = (try,s):more
                              Control.Monad.guard $ noCollision attempt
                              return attempt
    where possWords = filter (\w -> siteLen s == length w) words

-- read the content of a file into the "Crossword" datatype
readCrossword :: String -> Crossword
readCrossword = (\(ws,ss) -> Crossword ws (toSites (drop 1 ss))) . break (""==) . lines
```

This is a simplistic solution with no consideration for speed. Especially sites and

words aren't ordered as propesed in (3) of the problem. Words of the correct length are naively tried for all blanks (without heuristics) and the possible solutions are then backtracked.

To test for collisions, all (Word, Site) pairs are merged to result in a list of (Coord, Char) elements which represent all letters placed so far. If all (two) characters of the same coordinate are identical, there exist no collisions between words.