99 questions/Solutions/28

From HaskellWiki

< 99 questions | Solutions

Sorting a list of lists according to length of sublists

a) We suppose that a list contains elements that are lists themselves. The objective is to sort the elements of this list according to their length. E.g. short lists first, longer lists later, or vice versa.

Solution:

```
import List
import Data.Ord (comparing)

lsort :: [[a]] -> [[a]]
lsort = sortBy (comparing length)
```

This function also works for empty list. Import `List` to use `sortBy`. If you wanted to solve it without the `comparing` function, you could do:

```
import List

lsort :: [[a]] -> [[a]]
lsort = sortBy (\xs ys -> compare (length xs) (length ys))
```

Or using `on`

```
lsort' = sortBy (compare `on` length)
```

Another Solution (which faster than other ones)

"sortOn f is equivalent to sortBy . comparing f, but has the performance advantage of only evaluating f once for each element in the input list."

```
lsort = sortOn length
```

b) Again, we suppose that a list contains elements that are lists themselves. But this time the objective is to sort the elements of this list according to their **length frequency**; i.e., in the default, where sorting is done ascendingly, lists with rare lengths are placed first, others with a more frequent length come later.

In the example for this problem, sub-lists of length `N` appear in the same order they were in the original list; here, "ijkl" comes before "o" in the original list and in the resulting output:

```
> lfsort ["abc", "de", "fgh", "de", "ijkl", "mn", "o"]
["ijkl","o","abc","fgh","de","de","mn"]
```

If the input were to have another list of length 5 at the end, one might presume that the output would look like this:

```
> lfsort ["abc", "de", "fgh", "de", "ijkl", "mn", "o", "abcde"]
["ijkl","o","abcde","abc","fgh","de","de","mn"]
```

This solution satisfies the description of the problem (that is, lists appear in order of length frequency), although it does not give the same result as the example:

```
lfsort :: [[a]] -> [[a]]
lfsort lists = concat groups
    where groups = lsort $ groupBy equalLength $ lsort lists
          equalLength xs ys = length xs == length ys
```

Since this solution first applies `lsort`, the resulting output will have sub-lists appearing in ascending order of length, rather than in the same order they appeared in the original list. Sample output:

```
> lfsort ["abc", "de", "fgh", "de", "ijkl", "mn", "o"]
["o","ijkl","abc","fgh","de","de","mn"]

> lfsort ["abc", "de", "fgh", "de", "ijkl", "mn", "o", "abcde"]
["o","ijkl","abcde","abc","fgh","de","de","mn"]
```

A more succinct version of the above solution using
`on`
:

```
import Data.Function

lfsort :: [[a]] -> [[a]]
lfsort = concat . lsort . groupBy ((==) `on` length) . lsort
```

Different solution. Quite inefficient, but does give the same output as the example:

```
import List;

frequency len l = length (filter (\x -> length x == len) l)

lfsort :: [[a]] -> [[a]]
lfsort l = sortBy (\xs ys -> compare (frequency (length xs) l) (frequency (length ys) l)) l
```

Another solution

```
lfsort = map snd . concat . sortOn length . groupBy ((==) `on` fst) . sortOn fst .
 map (\x -> (length x, x))
```

Another solution that gives the same output but works more like the first solution. It also precalculates all the lengths then uses sortWith for efficiency:

```haskell
import Control.Arrow ((>>>),(&&&),second)
import GHC.Exts (sortWith)

lfsort :: [[a]] -> [[a]]
lfsort = zip [1..] >>> map (second (length &&& id)) >>> sortWith (snd>>>fst)
           >>> cntDupLength undefined [] >>> sortWith (snd>>>fst)
           >>> sortWith fst >>> map (\(_,(_,(_,a))) -> a)
  where cntDupLength :: Int -> [(Int,(Int,a))] -> [(Int,(Int,(Int,a)))]
        cntDupLength _ lls [] = map ((,) (length lls)) $ reverse lls
        cntDupLength _ [] (x@(_,(l,_)):xs) = cntDupLength l [x] xs
        cntDupLength l lls ys@(x@(_,(l1,_)):xs)
           | l == l1   = cntDupLength l (x:lls) xs
           | otherwise = (map ((,) (length lls)) $ reverse lls) ++ cntDupLength undefined []
```
Retrieved from "https://wiki.haskell.org/index.php?title=99_questions/Solutions/28&oldid=60011"

Category:

- Programming exercise spoilers
