99 questions/Solutions/10

From HaskellWiki

< 99 questions | Solutions

(*) Run-length encoding of a list.

Use the result of problem P09 to implement the so-called run-length encoding data compression method. Consecutive duplicates of elements are encoded as lists (N E) where N is the number of duplicates of the element E.

```
encode xs = map (\x -> (length x,head x)) (group xs)
```

which can also be expressed as a list comprehension:

```
[(length x, head x) | x <- group xs]
```

Or writing it Pointfree (Note that the type signature is essential here to avoid hitting the Monomorphism Restriction):

```
encode :: Eq a => [a] -> [(Int, a)]
encode = map (\x -> (length x, head x)) . group
```

Or (ab)using the "&&&" arrow operator for tuples:

```
encode :: Eq a => [a] -> [(Int, a)]
encode xs = map (length &&& head) $ group xs
```

Or using the slightly more verbose (w.r.t. `(&&&)`) Applicative combinators:

```
encode :: Eq a => [a] -> [(Int, a)]
encode = map ((,) <$> length <*> head) . pack
```

Or with the help of foldr (*pack* is the resulting function from P09):

```
encode xs = (enc . pack) xs
        where enc = foldr (\x acc -> (length x, head x) : acc) []
```

Or using takeWhile and dropWhile:

```
encode [] = []
encode (x:xs) = (length $ x : takeWhile (==x) xs, x)
                 : encode (dropWhile (==x) xs)
```

Or without higher order functions:

```
encode []     = []
encode (x:xs) = encode' 1 x xs where
    encode' n x [] = [(n, x)]
    encode' n x (y:ys)
        | x == y    = encode' (n + 1) x ys
        | otherwise = (n, x) : encode' 1 y ys
```

Or we can make use of zip and group:

```
import List
encode :: Eq a => [a] -> [(Int, a)]
encode xs=zip (map length l) h where
    l = (group xs)
    h = map head l
```

Or if we ignore the rule that we should use the result of P09,

```
encode :: Eq a => [a] -> [(Int,a)]
encode xs = foldr f final xs Nothing
  where
    f x r (Just a@(i,q)) | x == q = r (Just (i+1,q))
                         | otherwise = a : r (Just (1, x))
    f x r Nothing = r (Just (1, x))

    final (Just a@(i,q)) = [a]
    final Nothing = []
```

which can become a good transformer for list fusion like so:

```
{-# INLINE encode #-}
encode :: Eq a => [a] -> [(Int,a)]
encode xs = build (\c n ->
  let
    f x r (Just a@(i,q)) | x == q = r (Just (i+1,q))
                         | otherwise = a `c` r (Just (1, x))
    f x r Nothing = r (Just (1, x))

    final (Just a@(i,q)) = a `c` n
    final Nothing = n

  in
    foldr f final xs Nothing)
```

Retrieved from "https://wiki.haskell.org/index.php?title=99_questions/Solutions/10&oldid=59174"

Category:

- Programming exercise spoilers

- This page was last modified on 28 December 2014, at 19:29.
-