99 questions/Solutions/8

From HaskellWiki

< 99 questions | Solutions

(**) Eliminate consecutive duplicates of list elements.

```
compress :: Eq a => [a] -> [a]
compress = map head . group
```

We simply group equal values together (using Data.List.group), then take the head of each.

An alternative solution is

```
compress (x:ys@(y:_))
    | x == y    = compress ys
    | otherwise = x : compress ys
compress ys = ys
```

A variation of the above using
foldr
(note that GHC erases the
Maybe
s, producing efficient code):

```
compress xs = foldr f (const []) xs Nothing
  where
    f x r a@(Just q) | x == q = r a
    f x r _ = x : r (Just x)
```

Another possibility using foldr (this one is not so efficient, because it pushes the whole input onto the "stack" before doing anything else):

```
compress :: (Eq a) => [a] -> [a]
compress = foldr skipDups []
    where skipDups x [] = [x]
          skipDups x acc
                | x == head acc = acc
                | otherwise = x : acc
```

A similar solution without using
foldr
.

```
compress :: (Eq a) => [a] -> [a]
compress list = compress_acc list []
          where compress_acc [] acc = acc
                compress_acc [x] acc = (acc ++ [x])
                compress_acc (x:xs) acc
```

```
                | x == (head xs)  = compress_acc xs acc
                | otherwise       = compress_acc xs (acc ++ [x])
```

A very simple approach:

```
compress []     = []
compress (x:xs) = x : (compress $ dropWhile (== x) xs)
```

Another approach, using foldr

```
compress :: Eq a => [a] -> [a]
compress x = foldr (\a b -> if a == (head b) then b else a:b) [last x] x
```

Wrong solution using foldr

```
compress :: Eq a => [a] -> [a]
compress xs = foldr (\x acc -> if x `elem` acc then acc else x:acc) [] xs
-- Main> compress [1, 1, 1, 2, 2, 1, 1]
-- [2,1] - must be [1,2,1]
```

and using foldl

```
compress :: (Eq a) => [a] -> [a]
compress x = foldl (\a b -> if (last a) == b then a else a ++ [b]) [head x] x
compress' x = reverse $ foldl (\a b -> if (head a) == b then a else b:a) [head x] x
```

A crazy variation that acts as a good transformer for fold/build fusion

```
{-# INLINE compress #-}
compress :: Eq a => [a] -> [a]
compress xs = build (\c n ->
  let
    f x r a@(Just q) | x == q = r a
    f x r _ = x `c` r (Just x)
  in
    foldr f (const n) xs Nothing)
```


Category:

- Programming exercise spoilers
