99 questions/Solutions/27

From HaskellWiki

< 99 questions | Solutions

Group the elements of a set into disjoint subsets.

a) In how many ways can a group of 9 people work in 3 disjoint subgroups of 2, 3 and 4 persons? Write a function that generates all the possibilities and returns them in a list.

b) Generalize the above predicate in a way that we can specify a list of group sizes and the predicate will return a list of groups.

```
combination :: Int -> [a] -> [([a],[a])]
combination 0 xs     = [([],xs)]
combination n []     = []
combination n (x:xs) = ts ++ ds
  where
    ts = [ (x:ys,zs) | (ys,zs) <- combination (n-1) xs ]
    ds = [ (ys,x:zs) | (ys,zs) <- combination  n    xs ]

group :: [Int] -> [a] -> [[[a]]]
group [] _ = [[]]
group (n:ns) xs =
    [ g:gs | (g,rs) <- combination n xs
           ,  gs    <- group ns rs ]
```

First of all we acknowledge that we need something like `combination` from the above problem. Actually we need more than the elements we selected, we also need the elements we did not select. Therefore we cannot use the `tails` function because it throws too much information away. But in general this function works like the one above. In each step of the recursion we have to decide whether we want to take the first element of the list `(x:xs)` in the combination (we collect the possibilities for this choice in `ts`) or if we don't want it in the combination (`ds` collects the possibilities for this case). Now we need a function `group` that does the needed work. First we denote that if we don't want any group there is only one solution: a list of no groups. But if we want at least one group with n members we have to select n elements of `xs` into a group `g`

and the remaining elements into
`rs`
. Afterwards we group those remaining elements, get a list of groups
`gs`
and prepend
`g`
as the first group.

And a way for those who like it shorter (but less comprehensive):

```
group :: [Int] -> [a] -> [[[a]]]
group [] = const [[]]
group (n:ns) = concatMap (uncurry $ (. group ns) . map . (:)) . combination n
```

And for an intermediate length solution

```
group :: [Int] -> [a] -> [[[a]]]
group [] xs = [[]]
group (g:gs) xs = concatMap helper $ combination g xs
          where helper (as, bs) = map (as:) (group gs bs)
```


Category:

- Programming exercise spoilers
