99 questions/Solutions/26

From HaskellWiki

< 99 questions | Solutions

(**) Generate the combinations of K distinct objects chosen from the N elements of a list

In how many ways can a committee of 3 be chosen from a group of 12 people? We all know that there are C(12,3) = 220 possibilities (C(N,K) denotes the well-known binomial coefficients). For pure mathematicians, this result may be great. But we want to really generate all the possibilities in a list.

```haskell
-- Import the 'tails' function
--   > tails [0,1,2,3]
--   [[0,1,2,3],[1,2,3],[2,3],[3],[]]
import Data.List (tails)

-- The implementation first checks if there's no more elements to select,
-- if so, there's only one possible combination, the empty one,
-- otherwise we need to select 'n' elements. Since we don't want to
-- select an element twice, and we want to select elements in order, to
-- avoid combinations which only differ in ordering, we skip some
-- unspecified initial elements with 'tails', and select the next element,
-- also recursively selecting the next 'n-1' element from the rest of the
-- tail, finally consing them together

-- Using list comprehensions
combinations :: Int -> [a] -> [[a]]
combinations 0 _  = [ [] ]
combinations n xs = [ y:ys | y:xs' <- tails xs
                           , ys <- combinations (n-1) xs']

-- Alternate syntax, using 'do'-notation
combinations :: Int -> [a] -> [[a]]
combinations 0 _  = return []
combinations n xs = do y:xs' <- tails xs
                       ys <- combinations (n-1) xs'
                       return (y:ys)
```

Here is another recursive solution, using the Prelude instead of `tails`.

```haskell
combinations :: Int -> [a] -> [[a]]
combinations 0 _ = [[]]
combinations n xs = [ xs !! i : x | i <- [0..(length xs)-1]
                                  , x <- combinations (n-1) (drop (i+1) xs) ]
```

Another solution that's slightly longer, but also doesn't depend on `tails`.

```haskell
combinations :: Int -> [a] -> [[a]]
-- We don't actually need this base case; it's just here to
-- avoid the warning about non-exhaustive pattern matches
combinations _ [] = [[]]
-- Base case--choosing 0 elements from any list gives an empty list
combinations 0 _  = [[]]
```

```haskell
-- Get all combinations that start with x, recursively choosing (k-1) from the
-- remaining xs. After exhausting all the possibilities starting with x, if there
-- are at least k elements in the remaining xs, recursively get combinations of k
-- from the remaining xs.
combinations k (x:xs) = x_start ++ others
    where x_start = [ x : rest | rest <- combinations (k-1) xs ]
          others  = if k <= length xs then combinations k xs else []
```

Variant: This version avoids getting a result consisting of the empty choice if we ask for, say, 10 elements from the empty list. (Instead, an empty list, suggesting "no options available" is returned.)

```haskell
combinations :: Int -> [a] -> [[a]]
combinations 0 _ = [[]]
combinations _ [] = []
combinations n (x:xs) = (map (x:) (combinations (n-1) xs)) ++ (combinations n xs)
```

A solution using subsequences in Data.List

```haskell
combinations k ns = filter ((k==).length) (subsequences ns)
```

A funny solution using Applicative in Control.Applicative. Funny because it's the opposite of other solutions. The idea is to generate all possible combinations by allowing permutations and by allowing multiple instances of one value in the list. Then we filter the list to stick with the exercice.

```haskell
-- let's try it with combinations 2 [1,2,3]
combinations :: (Ord a) => Int -> [a] -> [[a]]
combinations n xs = compressed
    where
          -- create all combinations (multiple instances, permutations allowed)
          -- answer : [[1,1],[1,2],[1,3],[2,1],[2,2],[2,3],[3,1],[3,2],[3,3]]
          combinations' n _ | n <= 0 = [[]]
          combinations' 1 xs = map (:[]) xs
          combinations' n xs = (:) <$> xs <*> combinations (n-1) xs
          -- sort every sublist and the list itself after that
          -- [[1,1],[1,2],[1,2],[1,3],[1,3],[2,2],[2,3],[2,3],[3,3]]
          sorted = sort . map sort $ combinations' n xs
          -- for each sublist, eliminate duplicates (see Problem 8)
          -- [[1],[1,2],[1,2],[1,3],[1,3],[2],[2,3],[2,3],[3]]
          grouped = map (map head . group) sorted
          -- filter all sublist with length < n,
          -- this means that they had at least two times the same value in it
          -- [[1,2],[1,2],[1,3],[1,3],[2,3],[2,3]]
          filtered = filter (\xs -> length xs == n) grouped
          -- eliminate duplicates a second time, this time in the list itself
          -- [[1,2],[1,3],[2,3]]
          compressed = map head . group $ filtered
```

The long list of variables in the where clause are there for educational purpose.

Category:

- Programming exercise spoilers