CS3200: Programming Languages
Homework 8: more on Haskell

Required Problems

1. Data compression is very useful because it helps reduce resources usage, such as data storage space or transmission capacity. For this task, you can assume that the input strings consist only of letters of the English alphabet.

   (a) Run-length encoding (RLE) is a very simple form of data compression in which runs of data are stored as a single data value and a count, rather than as the original run. Define a function rle that applied RLE to a the given string.

   ```
   rle :: String -> String
   rle "aaabbbbbc" ⟹ "3a 5b 1c"
   rle "banana" ⟹ "1b 1a 1n 1a 1n 1a"
   ```

   (b) Define rleInverse that applies the inverse RLE operation (RLE decoding) on a given string.

2. Consider the following data declaration of a vector (as in a math vector in Euclidean space) in Haskell:

   ```
   type Vector = [Double]
   ```

   For the second part of this homework, create a file Vector.hs, which will hold a module for Vectors. You will be writing several functions to do mathematical calculations over a vector, as follows:

   (a) The function `scale :: Double -> Vector -> Vector` takes a Double y, and a vector, v, and returns a new Vector that is just like v, except that each coordinate is y times the corresponding coordinate in v.

   (b) The function `add :: Vector -> Vector -> Vector` takes two vectors and adds them together, so that each coordinate of the result is the sum of the corresponding coordinates of the argument Vectors. Your code should assume that the two vector arguments have the same length.

   (c) The function `dot :: Vector -> Vector -> Double` takes two vectors and computes their dot product (or inner product), which is the sum of the products of the corresponding elements. Your code should assume that the two vector arguments have the same length.

   (d) The function `norm :: Int -> Vector -> Double` which takes an integer p and a Vector $v = \{v_1, \ldots, v_n\}$ and computes the p^{th} norm of the vector, defined as:

   $$|v| = \left(\sum_i v_i^p \right)^{\frac{1}{p}}$$

   You can assume p is a positive integer.

(e) Finally, make a series of tests for your module. You may do this via a main and some sort of IO, or you may submit a sequence of commands you ran at the terminal to test your functions. (Note that I'm expecting actual testing! So don't just call each function once - that is not a proper set of unit tests to be sure it actually works.)

To get you started, you might begin your file as follows:

```
module Vectors where

-- Vectors are represented by finite lists of coordinate values.
type Vector = [Double]

scale :: Double -> Vector -> Vector
-- Add your function here

add :: Vector -> Vector -> Vector
-- Add your function here

dot :: Vector -> Vector -> Double
-- Add your function here

norm :: Int -> Vector -> Double
-- Add your function here
```

3. Extra credit: Define a function `subsequence` that takes two lists and returns the ascending list of indices at which the first list occurs as a subsequence of the second list. If there are multiple solutions, return the one with smallest sum of all indices.

```
subsequence ::  Eq a => [a] -> [a] -> [Int]
subsequence "abcde" "abcbcdef" ⟹ [0, 1, 2, 5, 6]
subsequence [9, 9, 7] [9, 7, 7, 9, 9, 7] ⟹ [0, 3, 5]
subsequence "abc" "caccdcbdca" ⟹ [1, 6, 8]
subsequence "312" "1212313" ⟹ error "subsequence does not exist"
```