Homework assignment #2

Please submit a `.zip` archive containing a single file `homework2.hs` with your solutions to the problems below. Moodle blocks file submissions with `.hs` extension.

Good luck! 🍀

Acknowledgement: The problems of this homework assignment were suggested by Jason BILLARD and Cosmin RADOI.

Problem A: The longest palindrome 🤔😳🤓😎😳🤔

This exercise consist in writing a function that finds the longest subsequence of a string that matches a predicate. The goal is to understand why decomposing a complex function into simpler self-containing functions is necessary to obtain simple and readable code.

(1) *A palindrome is a word, phrase, or sequence that reads the same backwards as forwards, e.g. 'madam'. (Oxford dictionary)*

Write a polimorphic function `palindrome :: (Ord a) => [a] -> Bool` which verifies whether the input is a palindrome or not.

In [ ]:
```
palindrome :: (Ord a) => [a] -> Bool
palindrome = undefined

testA1 :: Bool
testA1 = palindrome "kayak"
testA2 :: Bool
testA2 = palindrome "hello"
```

(2) *In mathematics, a subsequence of a given sequence is a sequence that can be derived from the given sequence by deleting some or no elements without changing the order of the remaining elements. (Wikipedia)*

Write a function `subsequence :: [a] -> [[a]]` which computes all possible subsequences of a list. Duplicates are allowed.

In [ ]:
```
subsequence :: [a] -> [[a]]
subsequence = undefined
```

(3) Import the `Data.List` module and look at the documentation of the maximumBy :: (a -> a -> Ordering) -> [a] -> a function.

In [ ]:
```
import Data.List ( maximumBy, nub, sort )

testA3 :: Bool
testA3 = (sort . nub . subsequence) "123" == ["","1","12","123","13","2","23","3"]
testA4 :: Bool
testA4 = subsequence "Byte" ==
         ["","B","Be","Bt","Bte","By","Bye","Byt","Byte","e","t","te","y","ye","yt","yte"]
```

(4) Use the built-in compare function to create the `compareLength :: [a] -> [a] -> Ordering` function which compares the length of the input lists.

In [17]:
```
compareLength :: [a] -> [a] -> Ordering
compareLength = undefined

testA5 :: Bool
testA5 = compareLength "magnificent" "gud" == GT
```

(5) Finally, write a function `longestValidSubsequence :: ([a] -> Bool) -> [a] -> [a]`. This function takes as input a predicate and a generic list, and finds the longest subsequence that matches the predicate.

In [ ]:
```
longestValidSubsequence :: ([a] -> Bool) -> [a] -> [a]
longestValidSubsequence = undefined

testA6 :: Bool
testA6 = length (longestValidSubsequence palindrome "programmer debugging") == 6
testA7 :: Bool
testA7 = length (longestValidSubsequence palindrome "Why are holidays short") == 7
```

Problem B: Stairway to Heaven 🎸

You are standing at the bottom of a staircase that consists of `n` steps. Each step on this staircase is numbered sequentially from `1` to `n`. However, some of the steps are labeled as prime numbers.

Your goal is to determine how many prime-numbered steps you can reach from the bottom of the staircase, given a set of allowed step sizes. You can only move up with each movement, and you can start from the bottom again at any time.

For example, if `n=10`, and your allowed step sizes are `[2,3]`, you can reach the following prime-numbered steps: `2`, `3`, `5`, and `7`. So, the answer would be `4`. If `n=20` and the step size is `[3,4]` you would only be able to reach the steps numbered `3`, `7`, `11` (=3+*24), `13` (=*33+4), `17` (=*33+*42), `19` (=3+4*4). Thus, the answer in that case would be `6`.

Your task is to implement a function that given the total number of steps `n` and a list of allowed step sizes, returns the count of prime-numbered steps you can reach.

Feel free to create any number of helper functions you deem necessary in order to achieve your goal.

Input: An integer `n` (`1 <= n <= 1000`), representing the total number of steps in the staircase, and a list of integers, containing all allowed step sizes.

Output: The count of prime-numbered steps that can be reached from the bottom of the staircase.

HINT 💡: At any point, you can extend a list of reachable steps by moving up some allowed number of steps X, or never moving up X steps at once again.

In [15]:
```
totalSteps :: Int -> [Int] -> Int
totalSteps = undefined

testB1 :: Bool
testB1 = totalSteps 10 [3, 4] == 2
testB2 :: Bool
testB2 = totalSteps 10 [11] == 0
testB3 :: Bool
testB3 = totalSteps 500 [30, 51, 71] == 31
testB4 :: Bool
testB4 = totalSteps 500 [30, 51, 72] == 0
testB5 :: Bool
testB5 = totalSteps 1000 [2, 3] == 168
```