lab session 2a: Functional Programming

Introduction

The second lab session of the course *Functional Programming* consists of 2 parts. This documents is part 2a, next week part 2b will be published. The reason for this split is that in part 2b we will need material that will be taught in the lecture of next week.

Part 2a consist of 4 programming exercises. Each exercise is worth 10 points, and you'll get 10 points for free. The remaining 50 points can be obtained with part 2b. You only need to hand in your code in Justitia: no report is required. The 10 points of each exercise are split into 5 points for 'correctness' (meaning, accepted by Justitia), and 5 points for (a Haskellish) style of programming and efficiency (assessment is done by the TAs). The main theme of this lab session is the use of infinite lists and higher-order functions. Therefore, the advise is to use these whenever possible. For some of the exercises,solutions can be found on the internet. Be warned, that copying those is considered plagiarism (we check programming forums)! Moreover, many of these published solutions do not use higher-order functions (effectively/at all), or are inefficient.

Exercise 1: Composites and their factorizations

In the lecture it was shown that the infinte list of prime numbers can be lazily computed with the following Haskell version of the sieve of Eratosthenes:

```
primes :: [Integer]
primes  =  sieve [2..]
  where
    sieve :: [Integer] -> [Integer]
    sieve (p:xs)  =  p : sieve [x | x <- xs, x `mod` p /= 0]
```

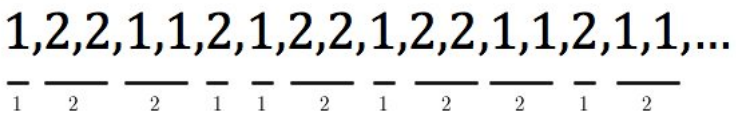
Using this code, write a Haskell definition of the infinite list `composites::[(Integer,[Integer])]`. Each element of this list is a pair, of which the first element is a composite number, and the second element is its prime factorization (an ascending list of prime factors). For example, `take 5 composites` must return `[(4,[2,2]),(6,[2,3]),(8,[2,2,2]),(9,[3,3]),(10,[2,5])]`.

On Nestor, a wrapper function (including sieve code) is available which must be used for the submission of your solution to Justitia.

Exercise 2: Run-length sequence

Consider the following sequence: $1, 2, 2, 1, 1, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 1, 2, 1, 2, 2, 1, 1, ...$
The sequence only contains 1s and 2s. They appear either in a run of one, or in a run of two. If we start at the beginning of the sequence, then the lengths of the runs recreate the original sequence: there is **1** one, followed by **2** twos, followed by **2** ones, followed by **1** two, followed by **1** one, etc. If

you put the bold-face runlengths in a sequence, then we arrive at the original sequence.



Write a Haskell definition of the list `selfrle::[Int]` that returns this infinite list. For example, `take 29 selfrle` must yield `[1,2,2,1,1,2,1,2,2,1,2,2,1,1,2,1,1,2,2,1,2,1,1,2,1,2,2,1,1]`. Your solution must make use of a Haskellish style of constructing lazy infinite list. On Nestor, a wrapper function is available which must be used for the submission of your solution to Justitia.

Exercise 3: Fibonacci and Catalan numbers

Of course, you know the recurrence that defines the *Fibonacci sequence*:

$$F_0 = 0, \quad F_1 = 1, \quad F_n = F_{n-1} + F_{n-2}$$

The first 10 terms of the Fibonacci sequence are: $0, 1, 1, 2, 3, 5, 8, 13, 21, 34$.
Another famous recurrence generates the *Catalan sequence*:

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}$$

The first 10 terms of the Catalan sequence are: $1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862$.

Write a Haskell definition of the infinite list `fibcat::[Integer]` that returns the infinite ascending list of positive integers that are either a Fibonacci number or a Catalan number (possibly both). For example, `take 11 fibcat` should yield the answer `[0,1,2,3,5,8,13,14,21,34,42]`. Your solution must make use of a Haskellish style of constructing lazy infinite list. On Nestor, a wrapper function is available which must be used for the submission of your solution to Justitia.

Exercise 4: Another prime sieve

Apart from the sieve of Eratosthenes, there are exist several other sieves that produce a list of prime numbers. One of these sieves generates the list of odd primes upto some upperbound $2n+1$ as follows:

- Start with a list of the integers from 1 to n.
- From this list, remove all integers of the form $i + j + 2ij$ where $1 \leq i \leq j$.
- The remaining numbers are doubled and incremented by one. The result is the list of all the odd prime numbers less than $2n + 2$.

Write a Haskell definition of the lazy infinite list `primes::[Integer]`, which is the list of all primes that is generated with an adapted unbounded version of this sieve. Of course, `take 5 primes` must produce `[2,3,5,7,11]`.

On Nestor, a wrapper function is available which must be used for the submission of your solution to Justitia.