Introduction to Programming: Assignment 2

Due: March 17, 2021. 11.55 pm

Important Instructions: Submit your solution in a single file named **`loginid.2.hs`** on Moodle. For example, if I were to submit a solution, the file would be called **`spsuresh.2.hs`**. You may define auxiliary functions in the same file, but the solutions should have the function names specified by the problems.

---

1. Knights are chess pieces whose moves are commonly characterized as two and a half squares – they move two squares in one direction and one square in an orthogonal direction. On a chessboard, rows are called *ranks* and columns are called *files*. The square on the x^{th} rank and the y^{th} file is represented by the pair (x, y). The set of squares on an 8×8 chessboard is thus given by

   ```
   squares = [(x,y) | x <- [0..7], y <- [0..7]]
   ```

   Define a function **`knightMove :: (Int, Int) -> Int -> [(Int, Int)]`** with the behavior that **`knightMove (x, y) n`** gives the list of squares where the knight could be in after a total of **`n`** moves, starting from the square **`(x,y)`**. Make sure that your list has no duplicates, and is lexicographically sorted.

   Sample cases:

   ```
   knightMove (0,0) 0 = [(0,0)]
   knightMove (0,0) 1 = [(1,2),(2,1)]
   knightMove (0,0) 3 = [(0,1),(0,3),(0,5),(1,0),(1,2),(1,4),(1,6),
                         (2,1),(2,3),(2,5),(3,0),(3,2),(3,4),(3,6),
                         (4,1),(4,3),(4,5),(5,0),(5,2),(5,4),(6,1),(6,3)]
   knightMove (2,2) 2 = [(0,2),(0,6),(1,1),(1,3),(1,5),
                         (2,0),(2,2),(2,4),(2,6),(3,1),(3,3),(3,5),
                         (4,2),(4,6),(5,1),(5,3),(5,5),(6,0),(6,2),(6,4)]
   ```

2. This problem is related to rational numbers and their continued fraction representation.

   Rational numbers are represented using the data type **`Rational`** in Haskell. The ratio p/q is represented using the **`%`** operator defined in **`Data.Ratio`**. Acquaint yourself with the other functions defined in **`Data.Ratio`**, like **`numerator`** and **`denominator`**, and the function **`fromIntegral`**. (Note that **`numerator rat`** can be positive or negative, but **`denominator rat`** is always positive.)

A finite continued fraction is any expression of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots \cfrac{1}{a_n}}}$$

where a_0 is an integer and each a_i is a positive integer, for $i \geqslant 1$. This is succinctly represented as the list $[a_0; a_1, a_2, \ldots, a_n]$. (Note the semicolon after the first entry.) A finite continued fraction can be calculated to a rational of the form $\frac{p}{q}$. On the other hand, every rational number can be expressed as a finite continued fraction. For example, the rational number $\frac{42}{31}$ can be rendered as a continued fraction using the following steps:

$$\begin{aligned}
\frac{42}{31} &= 1 + \frac{11}{31} \\
&= 1 + \cfrac{1}{\cfrac{31}{11}} \\
&= 1 + \cfrac{1}{2 + \cfrac{9}{11}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{\cfrac{11}{9}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \frac{2}{9}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{\frac{9}{2}}}} \\
&= 1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{4 + \frac{1}{2}}}}
\end{aligned}$$

Thus one continued fraction corresponding to $\frac{42}{31}$ is $[1; 2, 1, 4, 2]$. A continued fraction corresponding to $-\frac{26}{21}$ is $[-2; 1, 3, 5]$. The continued fraction representation is not unique. For instance, both $[0; 1, 1, 1, 1]$ and $[0; 1, 1, 2]$ represent $\frac{3}{5}$.

Define a function `computeRat :: [Integer] -> Rational` that takes a *nonempty* list of integers, such that all but the first element is positive, and returns the rational number corresponding to it.

Define a function **`cf :: Rational -> [Integer]`** that takes a rational number as input and returns a continued fraction corresponding to it.

Sample cases: (Since there are multiple answers possible for **`cf`**, the cases below are only indicative. We will check the correctness of your solution by actually computing the inverse and checking.)

```
cf (26%21)          = [1,4,5]
cf (-26%21)         = [-2,1,3,5]
cf (-42%31)         = [-2,1,1,1,4,2]
computeRat [1,4,5]          = 26%21
computeRat [-1,1,1,1,1]     = (-2)%5
```

3. Just like finite continued fractions represent rationals, infinite continued fractions of the form

$$a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cdots}}$$

correspond to irrational numbers. For example, the **golden ratio** $\phi = \dfrac{1+\sqrt{5}}{2}$ can be written as[1]

$$1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \ldots}}}$$

This is represented more succinctly as $[1; 1, 1, 1, 1, \ldots]$. If we truncate this list at some finite point, we get a finite rational approximation for ϕ.

Assume the following Haskell definitions:

```
phi :: Double
phi = (1 + sqrt 5) / 2

computeFrac :: Rational -> Double
computeFrac x = fromIntegral (numerator x)/fromIntegral (denominator x)
```

Write a function **`approxGR :: Double -> Rational`** which returns a close rational approximation for the golden ratio, i.e. it accepts an input **`epsilon`** and returns some **`r :: Rational`** such that **`abs (phi - computeFrac r) < epsilon`**.

Note: Do not use the **`approxRatio`** function from **`Data.Ratio`**.

[1] This can be verified by denoting the continued fraction as x and observing that $x = 1 + \frac{1}{x}$, i.e. $x^2 = x + 1$, and solving for x (and considering the positive solution).

Sample cases: (Since there are multiple answers possible, the cases below are only indicative. We will check the correctness of your solution by actually calculating if the error is less than **epsilon**.)

```
approxGR 0.0001          = 144 % 89
approxGR 0.0000000000001 = 3524578 % 2178309
```

4. The edit distance is the minimum number of (delete/insert/modify) operations required to edit one given string into another given string. For instance, here is one way of going from **smiitten** to **sitting**.

```
smitten
s itting
-d---m-i
```

The third string denotes the operations performed at each position, with **-**, **m**, **i** and **d** standing for *do nothing*, *modify*, *insert* and *delete*, respectively. As we see, the number of operations is 3.

Here is another way of going from **smiitten** to **sitting** (of cost 12).

```
s      mitten
sitting
-iiiiiiddddddd
```

The task is to find the minimum number of operations required to go from one string to another. Define a function

```
editDistance :: String -> String -> Int
```

such that **editDistance as bs** returns the number of operations required to go from **as** to **bs**. A naive recursive program is easy to obtain, but its performance will be exponential in the worst case, because of repeated recursive calls with same arguments. You should program using lazy dynamic programming to perform the task in polynomial time.

Sample cases:

```
editDistance "smitten" "sitting"                 = 3
editDistance "nematode knowledge" "empty bottle" = 12
```

5. Write an action **processText :: IO ()** with the following behaviour. It reads lines from the standard input. If the first word of the line is **":r"** (stands for "repeat"), then the rest of the line is output twice. Otherwise the input line is output once. Furthermore, we want each output line to start with a line number followed by a period and a space.

Suppose the input lines are the following:

```
Hello!
:rMy name is Suresh
:r and I am new to
: r Haskell.
:r      And how about you?
:r
Oh! You are a pro?

Okay!
```

Then the output lines are as follows (pay attention to the spaces and empty lines!):

```
1. Hello!
2. :rMy name is Suresh
3.  and I am new to
4.  and I am new to
5. : r Haskell.
6.       And how about you?
7.       And how about you?
8.
9.
10. Oh! You are a pro?
11.
12. Okay!
```

6. Write an action **`runningTotals :: IO ()`** that reads a number on each line, and after reading each line, prints the running total of all the numbers read so far. Here is a sample run, where the user input is shown indented, and the output is unindented.

```
    2
2
    3
5
    22
27
    -5
22
    16
38
    0
38
```