Synopsis Lab 11

11.1 Tasks

1. Skim lecture #9. Test the following functions: `sieve`, `mymin`, `fibonacciGen`, `hammingGen`.

2. (ML and Haskell) Consider the function of Sudan in lab #6. Write a function `matrixSudan` with just one argument `n` which returns a list containing tuples of form `(m,x,y,s)` where each of `m`, `x` and `y` take every possible value in range `[0..n]` and `s` is the corresponding value of Sudan's function.

   E.g.:

   ```
   Main> matrixSudan 1
   [(0,0,0,0),(0,0,1,1),(0,1,0,1),(0,1,1,2),(1,0,0,0),(1,0,1,1),
   (1,1,0,1),(1,1,1,3)]
   ```

   Then write 2 more functions: `firstFloorSudan n`, which extracts all elements in `matrixSudan n` for which `m=0` and `secondFloorSudan n` which does the same thing as `firstFloorSudan n`, but for `m=1`.

   E.g.:

   ```
   Main> firstFloorSudan 3
   [(0,0,0,0),(0,0,1,1),(0,0,2,2),(0,0,3,3),(0,1,0,1),(0,1,1,2),
   (0,1,2,3),(0,1,3,4),(0,2,0,2),(0,2,1,3),(0,2,2,4),(0,2,3,5),
   (0,3,0,3),(0,3,1,4),(0,3,2,5),(0,3,3,6)]


   Main> secondFloorSudan 3
   [(1,0,0,0),(1,0,1,1),(1,0,2,4),(1,0,3,11),(1,1,0,1),(1,1,1,3),
   (1,1,2,8),(1,1,3,19),(1,2,0,2),(1,2,1,5),(1,2,2,12),(1,2,3,27),
   (1,3,0,3),(1,3,1,7),(1,3,2,16),(1,3,3,35)]
   ```

3. (Haskell) This problem asks you to write a function `appsqrt x` which successively approximates `sqrt(x)` up to a difference ϵ, using lazy evaluation. In order to achieve this, you will probably make use of 3 functions.

   First, write a function `iter` with 2 arguments: a function `f` and a variable `x` which returns a list containing the results of successively iterating `f` on `x`, i.e. `x,f(x),f(f(x)),...`
   E.g.:

   ```
   Main> take 4 (iter (*2) 1)
   [1,2,4,8]
   ```

After that, write a function `nextapp x y` which generates the next approximation of `sqrt(x)` according to the formula

$$y_{k+1} = (x/y_k + y_k)/2.0$$

Then write a function `absdif eps ys` which returns the first element in list `ys` for which the absolute value of the difference between it and its predecessor is less than or equal to the preset value `eps`.

E.g.

```
Main> absdif 2 [1,4,7,6,8,11]
6
```

4. (Haskell) This problem asks you to write a function `allfactorials` which generates the list containing the factorials of all natural numbers.

   E.g.

   ```
   Main> take 10 allfactorials
   [1,1,2,6,24,120,720,5040,40320,362880]
   ```

   The implementation should use lazy evaluation in a similar manner to the function `fibonacciGen`. In order to implement `allfactorials`, you will probably make use of 2 functions.

   First, write a function `numbersFrom n` which generates all numbers starting from `n`.

   E.g.:

   ```
   Main> take 5 (numbersFrom 2)
   [2,3,4,5,6]
   ```

   After that, write a function `multlist xs ys` which multiplies the corresponding elements in 2 lists and returns the list of results.
   E.g.:

   ```
   Main> multlist [1,2,3] [5,6,7]
   [5,12,21]
   ```