

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp

Exercises *Functional Programming* – Sheet 4

Solutions will be collected until Wednesday, May 25, 2005 in the exercise course.

Exercises can be solved both in English and in German.

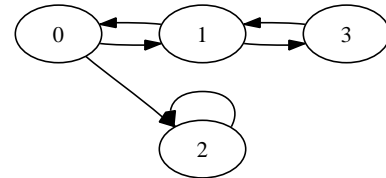
Exercise 1 (2+2 points)

To represent graphs we use finite lists of pairs of nodes. Each pair represents an edge between two nodes:

```
type Graph a = [(a,a)]
```

Consider the graph g over integers on the right-hand side. A representation of g as an object of type `Graph Int` in Haskell would be:

```
[(0,1), (1,3), (3,1), (0,2), (1,0), (2,2)]
```

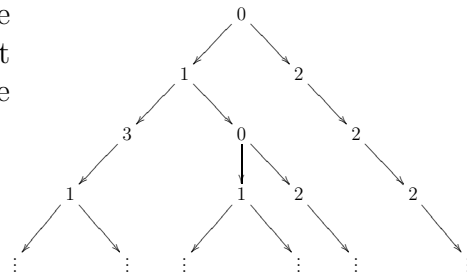


The following data structure represents trees where every node may have an arbitrary number of children:

```
data Tree a = Node a [Tree a]
```

Consider the tree t that is the result of *unfolding* the graph g starting from 0. A representation of the first four layers of the infinite tree t as an object of type `Tree Int` in Haskell could be:

```
Node 0 [Node 1 [Node 3 [Node 1 [..., ...]],
Node 0 [Node 1 [..., ...], Node 2 [...]],
Node 2 [Node 2 [Node 2 [...]]]
```



Implement the following functions in Haskell.

- The function `unfold` of type `Eq a => Graph a -> a -> Tree a` takes a graph g and a node x and returns the tree which results from unfolding g starting from x . Note that for a cyclic graph, the tree can be infinite.
- The function `hasPath` of type `hasPath :: Eq a => Graph a -> a -> a -> Int -> Bool` takes a graph g , two nodes x and y , and an integer n . It returns `True` if there is a path of exactly length n from x to y in g and `False` otherwise. You should make use of the function `unfold` from (a)!

For the graph g above, `hasPath g 0 0 n` is `True` only for $n \in \{0, 2, 4, \dots\}$ while `hasPath g 0 2 n` is `True` for all $n > 0$.

Exercise 2 (2+4 points)

The two functions `fibs1` and `fibs2` compute the infinite list of Fibonacci numbers starting from 0 and 1, i.e. the list `[0,1,1,2,3,5,8,13,21,...]`. The Fibonacci sequence fib_0, fib_1, \dots is defined by $fib_0 = 0$, $fib_1 = 1$, and $fib_{n+1} = fib_n + fib_{n-1}$.

```
fibs1 = map fib [0..] where
```

```
  fib 0 = 0
```

```
  fib 1 = 1
```

```
  fib (n+2) = (fib (n+1)) + (fib n)
```

```
fibs2 = map fib [0..] where
```

```
  fib n = fib' n 0 1 where
```

```
    fib' n a b | n == 0    = a
```

```
               | otherwise = fib' (n-1) (a+b) a
```

- (a) Analyze the complexity of evaluating the two expressions `take n fibs1` and `take n fibs2` depending on `n`. To this end, give lowest upper bounds on their complexity using the O -notation, i.e., give two functions f_1 and f_2 such that $O(f_1(n))$ and $O(f_2(n))$ are the respective lowest upper bounds.

Here, you should assume that addition has constant complexity, i.e., evaluating $n + m$ has complexity $O(1)$.

- (b) Implement a function `fibs3` which computes the same list as `fibs1` and `fibs2` by using a cyclic data structure. Analyze the complexity of evaluating `take n fib3` depending on `n` by giving a lowest upper bound as in (a).

Exercise 3 (4+6 points)

- (a) Implement a function `sumList :: IO ()` in Haskell. If `sumList` is executed, the user can type in a list of integers on the keyboard in standard Haskell notation. Afterwards, the sum of these integers is printed on the screen using the function `putStr`. For example, if the user enters the string `[1, -5, 70]`, then the number `66` is displayed on the screen.

In order to convert a `String` into an `[Int]` you may use the built-in function `read :: String -> [Int]`. For example, `read "[1, -5, 70]" :: [Int]` will read in the list of integers `[1, -5, 70]`.

- (b) Write a program in Haskell which tells the user to think of a secret integer number. Then the program asks the user for a lower and an upper bound of an interval containing the secret number. Afterwards, the computer tries to guess the secret number and asks the user if its guess is the right one or the secret number is smaller or bigger. The program repeats guessing until it finds the secret number. It should try to use as few guesses as possible.

Example sessions for the programs in (a) and (b) can be found at

<http://www-i2.informatik.rwth-aachen.de/lufgi2/fp05/Uebungen/guess.txt>