CS_221/M36 Functional Programming I

Coursework 2

Question 1. Modular exponentiation is defined by

\[ \text{expmod}(b, e, m) = b^e \mod m \]

where \( b, e, m \) are integers with \( b > 0, e \geq 0 \) and \( m > 0 \).

Remark: Modular exponentiation plays an important role in cryptography.

Give an efficient recursive definition of \( \text{expmod} \) using the following idea of repeated squaring: Assume \( e \) is a positive even number. Set

\[ f = e/2 \]
\[ p = \text{expmod}(b, f, m) \]

Then one can express \( \text{expmod}(b, e, m) \) in terms of \( p^2 \) and \( m \) using the square function and \( \mod \).

In order to find the solution you may use the fact that \( \mod \) commutes with multiplication, that is,

\[ (x \ast y) \mod m = ((x \mod m) \ast (y \mod m)) \mod m \]

If \( e \) is an odd number, then one can, with a similar calculation, reduce the computation of \( \text{expmod}(b, e, m) \) to the computation of \( \text{expmod}(b, e - 1, m) \).

Use your fast implementation of \( \text{expmod} \) to compute the last 6 digits (in decimal notation) of \( x^e \) where \( x \) is your student number.

Compare your program with a naive implementation of \( \text{expmod} \). [20 marks]

Question 2. You are at city \( n \) where \( n \) is your student number. Your goal is to reach city 1. The rules for the journey are as follows:

If you are at city \( k \), then
- if \( k \) is even, go to city \( k/2 \),
- if \( k \) is odd, go to city \( 3 \ast k + 1 \).

Compute the list of all cities visited on your journey. What is the largest city (number) visited? [20 marks]

Remark: It is an open problem whether the journey terminates for every number. This problem is also known as the 3x + 1 Problem, or Collatz’ Problem.
Question 3. Newton’s Method:

- To approximate $\sqrt{x}$, start with 1 (or any other value) as first approximation.
- If $y$ is an approximation of $\sqrt{x}$, then $(y + x/y)/2$ is a better approximation.
- Stop if the approximation $y$ is good enough, say $|y^2 - x| < 0.00001$.

Hint: Use the predefined functional `until`. Beware of rounding errors!

Implement a variant of Newton’s Method where the stopping condition is given by an upper bound for the number of iterations, and the result is the list of all approximations computed. [20 marks]

Question 4. Simple exercises in processing lists:

(a) Define a function that computes, for a list of integers $[x_0, \ldots, x_{n-1}]$, the list of all pairs $(i, j)$ such that $0 \leq i < j < n$ and $x_i < x_j$.

(b) Define a function `tails` that computes, of a given list, the list of its iterated tails. For example, `tails` $[1, 2, 3]$ should yield $[[1, 2, 3], [2, 3], [3], []]$.

(c) Define a function that tests whether one string is an initial segment of another.

(d) Define a function that tests whether one string is a substring of another. For example, “llo” is a substring of “Hello World”, but not of “Haskell rocks”.

(e) Generalise the functions in (a), (c) and (d) so that they work for as large classes of types as possible.

[40 marks]

Remark: Question 4 (d) is known as the string matching problem which plays an important role in text processing. Efficient string matching is accomplished by the famous Knuth-Morris-Pratt Algorithm which runs in linear time. How fast is your string matching program?

Due date: Thursday, 17 November, 2005

Notes:

For the submission of this coursework similar conditions as for Coursework 1 apply. In particular, the subject of the email must have the format `cs221-05-cw2-surname-firstname` respectively `csm36-05-cw2-surname-firstname` (everything lower case). Please submit all answers in one file.