Assessed exercise 1

Lossless text compression and decompression based on Huffman coding

1st stage, lab component

Functional Programming

October 19, 2016

Abstract

You have learned how to manipulate lists and trees in functional programming, and have exercised this. Now we will assess your understanding of list and tree manipulation. You will implement a form of lossless text compression and decompression, based on Huffman coding. This is assessment rather than learning, and hence you are supposed to produce and submit your own work.

1 Timeline for the exercise

1. Familiarize yourself with the task - lab 19th Oct .
2. Get started with the task - lab 19th Oct.
3. Get a formal specification of the task - Friday 21st Oct lecture.
4. Solve 1st part of the task and submit - by Tuesday 25th Oct.

2 Familiarize yourself with the task

1. The usual way to encode a text file is to work with characters encoded by some fixed-length coding by sequences of bits, such as ASCII:

   `http://www.asciitable.com/`

   or unicode, among other options.
2. A counter-example is Morse code, which uses variable length encoding, as you have seen in your unassessed assignments.
3. If we use variable length encoding wisely, we may significantly reduce the length of the encoded text.
4. This is how compression programs such as zip, gzip, etc. work.

5. Your task in this two-part exercise is to compress and decompress text based on this idea, in a precise form due to David A. Huffman (1952).

A starting reference is

`https://en.wikipedia.org/wiki/Huffman_coding`

Relevant lecture code on Canvas, which you are allowed to adapt for your solution,

prefixfreebittree.hs

The ideas behind Huffman coding are these:

1. We give shorter bit-sequences to letters that occur more often.
2. We have a table associating characters to bit-sequences.
3. We organize the table in a tree to facilitate encoding and decoding.

3 How do we do this? Getting started

The main objective of the lab session is to familiarize yourself with the technique and your task, and produce sketches of Haskell code (function types, data definitions, tentative algorithms). Then, in the lecture, based on your work, I will give precise forms for the tasks, after summarizing the techniques.

1. The first thing we want to do, according to the above technique, is to compute the frequencies of characters in a String: We need a function of type

   ```
   string -> [(Char,Int)]
   ```

   that counts how many times each character occurs in a string. For example, for "ababccebcc" we should get `[('a',2),('b',3),('c',4),('e',1)]` or the same list of pairs in a different order. (And it wouldn't be useful to e.g. add the pair `('z',0)` to this list, or any pair with zero frequency.) Can you write such a function?

2. Then we should order this by frequency, so that characters that occur more often appear first. Can you write such a Haskell function?

3. Then we should build a code-tree according to the character frequency so that characters that occur more often get shorter codes. But before that, we should think what kind of trees we need. Can you write such a Haskell definition of a type of trees suitable for this task, using a `data` definition?

4. Then, assuming that we have a code tree, we should be able to, given a bit sequence, turn it into a list of characters. This is decompression. Can you write such a Haskell definition of decompression?

5. But a more complicated task, for next week, is given a frequency list, build a code-tree. I will explain this in a lecture. But you may wish to think about this. And then given a code tree and a text, produce the encoded, compressed text.

6. These are the ideas I want you to think about in the lab, with as much Haskell code as you can produce, before I give you the formal specification of the task in the Friday lecture.