The last assignment

ID1019 Programming II

Johan Montelius

Spring Term 2022

Introduction

Morse codes were used in the days of telegraphs and manual radio communication. It is similar in the idea to Huffman coding in that it uses fewer symbols for frequent occurring characters and more symbols for the infrequent ones. You task is to write an encoder and a decoder for Morse signals, encode your name and decode two secret messages.

The solution that you present should be your original code i.e. not copied from any source apart from the More code tree in appendix.

Morse codes

There are several standards for Morse codes and we will here use a slightly extended version since we also want to code some special character. The Morse code uses, as you probably know, long and short symbols to encode characters (often pronounced *da* and *di*). Since the idea is to encode frequent characters with few symbols you might think that is identical to Huffman codes but there is a difference. In Morse coding we have a special signal that tells us when one character ends and the next start. The pause between characters is necessary in order to decode a message.

As an exampe we can look at how "ai" is coded. The code for 'a' is *di-da*, 'i' is *di-di* but the code for 'l' is *di-da-di-di*. If we just had the sequence *di-da-di-di* we would not know if this was "ai" or "j"; we need a third signal, the pause, to tell the difference. A sequence di-da-pause-di-di-pause is then decoded as "ai".

How does this change the structure of our coding tree? In a Huffman tree we only have characters in the leafs and when we hit a leaf we know that we have a complete code for a character. In a Morse tree we can finish anywhere along the path to a leaf. We thus have characters in the nodes of the tree (not all nodes but almost).

Encoding

Your first task is to implement an encoder. The Morse codes are given in the appendix so the only thing that you need to do is to use this information to construct an encoding table that can then be used to encode a message. You should use the codes given since it includes some special characters not normally found in the Morse alphabet.

0.1 the encode table

If you look in the appendix the Morse codes are given represented as a tree where one branch represents a short signal '.' and one branch the long signal '-'. The character of the node is the ASCII value of a character or `:na` if no character has the corresponding code. Empty branches are represented by `nil`.

```
@tyespec node() :: {:node, char(), node(), node()} | :nil
```

The More trees starts like follows; a single long signal is the code for ASCII 116 which is a 't' and a single short is the code for 101 i.e. an 'e'.

```
{:node, :na,
  {:node, 116,  ..., ...},
  {:node, 101,  ..., ...}}
```

The tree is not the best structure to use when you encode a message so your first task is to transform this into a form that gives you a $O(lg(k))$ or $O(1)$ lookup operation (where k is the number of letters in the alphabet). Describe why you have chosen the representation that you have and its characteristics. Also present how you construct this table given the tree in the appendix.

the encoder

Once you have an encoding table you should implement an encoder that can encode a charlist (a list of ASCII values) as a Morse code represented as a charlist of short '.', long '-' and pause ' ' characters. These could in Elixir be written using there ASCII values (46,45,32) or the special syntax for ASCII-values (`?.` `?-` `?/s`).

The encoder should have a complexity of $O(n * m)$ where n is the length of the message and m is the length of the Morse codes. Since messages might be very long you should provide an implementation that does not use stack space proportional to the length of the message. This means that you need to provide a tail recursive solution. The Mores codes are so short so you could have a solution that uses stack space proportional to the code length.

Describe your encoder and why it meets the requirements and include the code in your report.

Show that your encoder works by encoding you name (and if you have non-ASCII characters in your name replace them with something close i.e. 'è' becomes 'e' etc)

Decoding

Your next task is to implement a decoder that can take a charlist of a Morse coded message and return the clear text message. You will of course first need a decoding table and the table that you constructed for the encoder might not be your best choice.

the decode data structure

The datastructure that you use should give you a lookup operation that has an $O(m)$ complexity i.e. it should be proportional to the length of the Morse code. This means that you can not use for example a list of all the codes and their corresponding characters since this would require and $O(k)$ operation to search through all the codes and an $O(m)$ operation to see if a code matches the code that you're looking for. The situation would improve somewhat if you choose to represent that table as a map structure but then the lookup operation would be $O(lg(k))$ which is good but does not take advantage of the fact that frequent occurring characters have short codes and this is something that you want to take advantage of (very much the way how Huffman decoding works).

So what is your choice of decoding data structure and what does your lookup function look like?

the decoder

Once you have the lookup operation you can implement the decoder. Since the Morse encoded messages of course could be very long your solution should not use stack space that is proportional to the length of the messages. Describe your decoder and why it meets the requirements, include the implementation in your report.

Decode the secret messages below. If you cut and paste the code, make sure that you don't have carriage-return etc in the string. The string should only contain the dash, dot and space characters (I've included an extra space in the end so all codes end with a pause).

```
'.- .-.. .-.. ..-- -.-- --- ..- .-. ..--
  -... .- ... . ..-- .- .-. . ..--
  -... . .-.. --- -. --. ..-- - --- ..-- ..- ... '
```

```
'.... - - .--. ... ---... .----- .----- .-- .--
  .-- .-.-.- -.-- --- ..- - ..- -... . .-.-.- -.-.
  --- -- .----- .-- .- - -.-. .... ..--.. ...- .----.
  -.. .--.-- ..... .---- .-- ....- .-- ----. .--.--
  ..... --... --. .--.-- ..... ---.. -.-. .--.--
  ..... .---- '
```

The Morse codes

```
def morse() do
  {:node, :na,
    {:node, 116,
      {:node, 109,
        {:node, 111,
          {:node, :na, {:node, 48, nil, nil}, {:node, 57, nil, nil}},
          {:node, :na, nil, {:node, 56, nil, {:node, 58, nil, nil}}}},
        {:node, 103,
          {:node, 113, nil, nil},
          {:node, 122,
            {:node, :na, {:node, 44, nil, nil}, nil},
            {:node, 55, nil, nil}}}},
      {:node, 110,
        {:node, 107, {:node, 121, nil, nil}, {:node, 99, nil, nil}},
        {:node, 100,
          {:node, 120, nil, nil},
          {:node, 98, nil, {:node, 54, {:node, 45, nil, nil}, nil}}}}},
    {:node, 101,
      {:node, 97,
        {:node, 119,
          {:node, 106,
            {:node, 49, {:node, 47, nil, nil}, {:node, 61, nil, nil}},
            nil},
          {:node, 112,
            {:node, :na, {:node, 37, nil, nil}, {:node, 64, nil, nil}},
            nil}},
        {:node, 114,
          {:node, :na, nil, {:node, :na, {:node, 46, nil, nil}, nil}},
          {:node, 108, nil, nil}}},
      {:node, 105,
        {:node, 117,
          {:node, 32,
            {:node, 50, nil, nil},
            {:node, :na, nil, {:node, 63, nil, nil}}},
          {:node, 102, nil, nil}},
        {:node, 115,
          {:node, 118, {:node, 51, nil, nil}, nil},
          {:node, 104, {:node, 52, nil, nil}, {:node, 53, nil, nil}}}}}}
end
```