Morse Encoding

Programming II - Elixir Version

Johan Montelius

Spring Term 2024

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 a decoder for Morse signals and decode two secret messages.

1 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 (often pronounced *di* and *da*) to encode characters. You might therefore 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.

The code for 'a' is *di-da*, 'i' is *di-di* and '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 decoding 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 character and can start from the root again. In a Morse tree we can finish anywhere along the path to a leaf. We thus have characters in every node of the tree (apart from the root).

2 The decoder

The Morse codes that we will use are given in the decoding tree in the Appendix. As you see we have represented the table as a tree were each node is on the form:

```
{:node, character, long, short}
```

An empty tree is represented by the value `:nil` and the root holds dummy value instead of a character; if everything works fine we will never have to see the nil nor the dummy value.

To decode a message you only have to choose the *long branch* when you hear a long signal and a *short branch* when you hear a short signal. When you hear the pause you have decoded a character and can start from the root again.

Implement a function `decode(signal, tree)` that takes a *signal* and the decoding tree and returns a decoded message. The signal is in the form of a string with dots and dashes `'.- .  .-..  .-..  --'` (i.e. a charlist with ASCII characters 45, 46 and 32 (dash, dot and space), don't use the integers in yoru code but the elixir way of writing ASCII codes i.e. `?.`, `?-` and `?
s`)

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.

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

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

If you by accident have two spaces between to characters this might be decoded as `:na` which then will generate a string that is not a proper string. To solve this problem you might add a rule for the case where you're in the root of the tree (the character is `:na`) and you hear a space. Then you simply do nothing and just start from the root again. The rule can be made general so that any unknown character will simply make the decoder start from the root.

3 The encoder

To encode messages we simply need the table that gives us codes for each character. You could of course use the tree directly to find the code of a

character but this requires searching through the whole tree for each caracter. Why not traverse the tree once and build a map structure that maps characters to codes. The codes are best represented as charlist.

First implement a fucntion that takes the morse tree and returns an encoding table as a map. Is then a simple task to implement a function that takes a message (encoded as a charlist) and encodes each character of the message. When you construct the encoded message make sure that you include a pause between each code sequence. The message 'sos' is thus encoded `'...  --  ...  '` with spaces between the sequences.

4 The Morse codes

```
def tree 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
```