Morse Coding

Axel Månson Lokrantz

Spring Term 2023

Introduction

Morse code is a method of encoding text as sequences of signals, called dashes and dots. The task for this assignment is to write an encoder and a decoder for Morse signals. The program was written in Elixir together with Daniel Dahlberg at KTH.

Github link

A link to the program in its entirety can be seen here.

Encoding

There are three basic signals in Morse coding, a dot (short) a dash (long) and a pause represented as a white space to mark where a character ends. Branches to the left are dots, and branches to the right are dashes. There are a few key differences between a Morse tree and a Huffman tree. In a Huffman tree all characters are placed in the leafs of the tree, so whenever we hit a leaf we know we have found what we are looking for. A Morse tree however, can finish anywhere along the path. In this assignment the Morse codes were given in a tree representation, meaning all that was left was to create a table to encode our message. The given tree is however not the best structure for encoding therefor we created a separate table through our `traverse` method which transforms the tree into a map. A signal is represented as a key and a character as its corresponding value as seen below. The table becomes a more dense representation of the Morse tree since we get rid of all the empty nodes in the tree.

```
"k" => ["-", ".", "-"],
"7" => ["-", "-", ".", ".", "."],
"w" => [".", "-", "-"],
...
```

In elixir maps are not implemented as plain hash tables. Instead, they are implemented as hash tries or hash trees. When a map gets more than

32 entries (46 in our case), it converts to a hash tree, which has a lookup of $O(log(n))$.

```
def encode(str) do
   table = traverse()
   Enum.map(charlist(str), fn char ->
     Enum.find(table, fn {key, _} ->
       char == key end) |> elem(1) |> insert_last(" ") end)
       |> List.to_string()
 end
```

The following output was given when encoding my name:
`".- -..- .  .-..  ..-- .-..  --- -.- .-.  .- -.  - --.."`

Decoding

Next, a decoder was implemented which takes a sequence of of Morse signals, a Morse tree and return a text message. Morse code was designed so that the most frequently used letters have the shortest codes, much like frequency of characters in Huffman tree. Typically, the length of the code increases as frequency decreases. Therefor, searching through the tree when decoding shorter messages is preferred over searching through a table with the complexity of $O(log(n))$.

```
def decode(seq \\ sample(), morse_tree \\ morse_tree())
 def decode([], _) do [] end
 def decode(seq, morse_tree) do
   {char, rest} = lookup(seq, morse_tree)
   [char | decode(rest, morse_tree)]
 end
```

The lookup functions interprets each sign recursively and returns a character whenever it receives a white space which is represented as 32 in ASCII.

```
def lookup([sign | rest], {:node, val, left, right}) do
  case sign do
    46 -> lookup(rest, right)
    45 -> lookup(rest, left)
    32 -> {val, rest}
  end
end
```

When decoding the sequences in the assignment we got the following messages: `"all your base are belong to us"` and `"https://www.youtube.com/watch?v=d%51w4w9%57g%58c%51"`