Morse - Report

Adrian Hjert

15/March/2023

Introduction

The aim of this report is to explain the implementation of a morse code encoder and decoder similar to the Huffman coding algorithm that was explored in the previous assignment. The main task is of course to implement the encoding and decoding algorithms along with their respective helper functions. The code is then tested by decoding some secret messages given in the instructions and by encoding and decoding our names.

Methods

Setting Up

The tree that we are given requires some translation, so we put it in a map which we can then use to encode a message into morse. This is done in the following manner:

```
def encode_table() do    #tree to map
  Enum.reduce((codes(morse(), [])), %{}, fn({char, code}, acc)
      ->   Map.put(acc, char, code)
        end)
    end
```

We then implement a function that allows us to look up the characters in the map using the **Map.get/2** function. The lookup function will find the given key and return its corresponding morse code string, provided the key exists in the map, otherwise, the funciton will return **nil**. One additional setup function is implemented, namely **codes/2**, which is also a part of the Huffman code from before, albeit slightly different in this case. The function uses pattern matching and based on the results, builds a list of character-code pairs. The tree is traversed recursively depth first where *da* represents the left branches and *di* represents the right branches. Leafs containing relevant characters are added to the list. Aside from the base case where the first argument is **:nil** returning an empty list, the other cases are as follows:

```
def codes({:node, :na, da, di}, code) do
       codes(da, [?- | code]) ++ codes(di, [?. | code])
     end
     def codes({:node, char, da, di}, code) do
       [{char, code}] ++ codes(da, [?- | code])
       ++ codes(di, [?. | code])
     end
```

The first of these case depends on the **:na** atom which is when no character has the given morse code, so the function calles **codes** and concatenates the results. The latter of the cases has a **char** as its second element, wherein it concatenates a list of tuples containing the **char** and its corresponding morse code to the result of **codes** for *da* and *di*.

Encoding

Decoding

Results and Conclusions