Morse Code(The last assignment)

Lucas Larsson

Spring Term 2022

Introduction

Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called ”dots” and ”dashes”, or ”dits” and ”dahs”. Morse code is named after Samuel Morse, one of the inventors of the telegraph.Wikipedia (Note that i am using an extended version given by the professor)

This was not a poetically difficult assignment since i have already solved a similar problem with Huffman coding assignment just a week ago. The difference between them is that Morse code are standardized depending on the language it is used to communicate, which makes the assignment even easier. And that the characters are separated by a `space " "` character, which also make it easier.

Encoding

As mentioned in the instructions the Morse codes are given in the appendix, using it i create a encoding table that is later used to encode messages from text to Morse code.

First Step is the encoding table, The function `encode_table/0` returns an encoding table that is going to be used to look up the corresponding Morse code for the characters to be coded.

The table is long but a short snippet of it looks like below

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

iex(3)> Encode,encode_table()
%{105 => '..', 48 => '-----', 63 => '..--..'...
```

The `encode_table/0` method also uses the codes function, it is not shown here since i wrote it heavily influenced from the teacher code shown at the seminar. But to sum it up it uses the `morse/0` method that represents the Morse tree, and returns a tuple of {`ASCCI, Morse_code`}.

After the encoding table is ready we start building the encode method, the code is shown below.

```
def encode(text) do
    encode(text,encode_table(),[])
end

def encode([],_, coded) do Enum.reverse(coded) end
def encode([char|text],table, sofar) do
encode(text,table, encode_lookup(char, table) ++[32] ++ sofar)
end
```

What the above shown code does is basically, one calls the method with the text to be coded, and in it is turn it calls another clauses that calls another with clauses adding the necessary parameters, that is the text to be encoded, the table and how far we have came in the encoding to keep track.

I am aware that this is an over simplified explanation, but this is the last assignment, and i am not taking any chances.

After implementing all the above shown code, I encoded my name and the results are shown below.

```
iex(1)> Encode.encode('lucas larsson')
' .-.. ..- -.-. .- ... ..-- .-.. .- .-. ... ... --- -.'
```

I didn't say anything in particular about Figure 1, but it is just a visual representation over how the Tree of the Morse code looks like, note that the figure is not the exact tree used in this assignment but just an example.

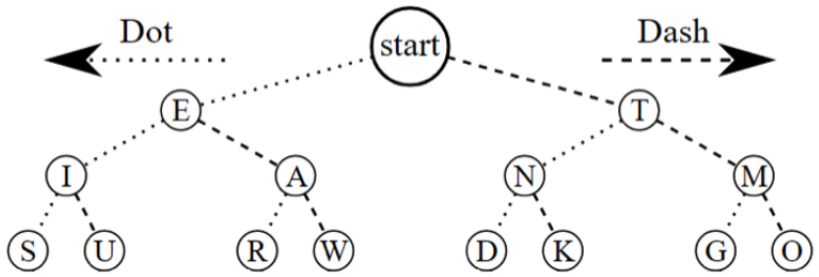

Figure 1: A graph representing a tree of Morse code.

Decoding

Decoding step: the code used is included below, and the requirements for it was for it to be memory efficient this is it "should not use stack space", the solution is easy our teacher has mentioned this many times, the solution is to make the decode function tail recursive. Just a quick summary: methods written using recursion usually require a lot of stack space, that is one call on the `call stack` for each recursive call, which can be very expensive in case of traversing/making operations on a long list using a recursive function. Here comes the beauty of the `tail-recursive` method that doesn't not require as much memory, the problem is that since it builds from the tail to the head it will build a list in reverse order, the solution is easy just using the `Enum.reverse/1` method. it takes a bit more time, but it is totally worth it considering the space it saves.

```
def decode(signal) do
    table = decode_table()
    decode(signal, table, [])
end

def decode([], _, acc) do Enum.reverse(acc) end
def decode(signal, table , acc) do
    {char, rest} = decode_char(signal,table)
    decode(rest, table,[char|acc])
end
```

When following the Link from above one would go to the YouTube page shown below.

The look up function is called decode_char and it works somewhat similar to the Huffman decode. Using pattern matching one traverse the tree to find the wanted character. Not much to say about it really, it is a genius solution that i got help with from the professor, basically the characters `?-, ?.  and ?` co-respond to characters `- , ., " "` which are used in the encoded sequence and is used to navigate the tree to the seeked character. And it does meet the requirements for the `O(m)` Complexity, where `m` is the length of the Sequence.

```
def decode_char(signal) do
  table = decode_table()
  decode_char(signal,table)
end
def decode_char([],{:node, char, _, _}) do {char,[]} end

def decode_char([?-|signal], {:node, _, long, _}) do
  decode_char(signal, long)
end
def decode_char([?.| signal], {:node, _,_, short}) do
  decode_char(signal,short)
end
```

```
def decode_char([?\s|signal], {:node,:na , _, _ }) do
{?*,signal}
end
def decode_char([?\s|signal], {:node,char , _, _ }) do
  {char,signal}
end
```

Last step was using the implemented decoder to decode the message from the instructions and below is the code for it and a snippet of the executed command.

```
def solution() do
    IO.puts(Decode.decode(base()))
    Decode.decode(rolled())
end

iex(1)> Morse.solution()
all your base are belong to us
'https://www.youtube.com/watch?v=d%51w4w9%57g%58c%51'
```

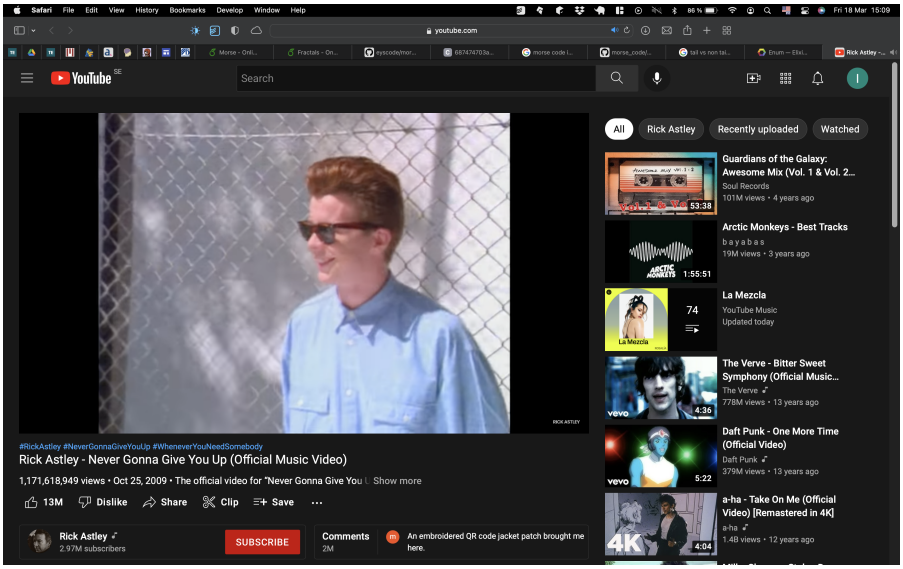

Figure 2: A screenshot showing the YouTube page.