Morse

Programmering II

Axel Karlsson

VT22 18 mars 2022

Inledning

I denna rapport ska jag redovisa min lösning för den sista inlämningsuppgiften i Programmering II. En algoritm för att koda och avkoda morsekod skulle implementeras, med särskilda krav på dess komplexitet.

Kodning av text

I den första delen av uppgiften skulle en funktion som kodar en given sträng i morsekod implementeras. Först går jag igenom hur jag skapade en kodningstabell, och sedan hur jag använde den för att koda en text.

Kodningstabell

En kodningstabell som översätter ett tecken till morsekod skulle implementeras med kravet att komplexiteten för en uppslagning i tabellen var $O(log(k)$, där k är antalet tecken i tabellen. En kodningstabell i form av ett träd var givet, och målet var att översätta detta till en datastruktur som uppfyller kraven.
Jag valde att implementera tabellen som en tuppel. Varje index innehåller morsekoden för det tecken med motsvarande asciivärde (till exempel är morsekoden för a, `.-`, i index 97 i tuppeln).

```
# Det första anropet
# morse() är den givna avkodningstabellen
def encode_table_tup() do
  # En tuppel av 128 nollor
  empty_table = List.to_tuple(List.duplicate(0, 128))
  encode_table_tup(morse(), [], empty_table)
end
```

Funktionen går sedan genom varje nod i trädet. Stigen som har tagits sparas i `code` och den nya kodningstabellen sparas i `table`.

```elixir
# Basfall om ett löv nås
def encode_table_tup(:nil, _, table) do table end
# Om noden innehåller :na, lägg inte till något i tabellen
def encode_table_tup({:node, :na, long, short}, code, table) do
  table = encode_table_tup(long, code ++ '-', table)
  table = encode_table_tup(short, code ++ '.', table)
  table
end
# Lägg till morsekoden code i plats char i tabellen
def encode_table_tup({:node, char, long, short}, code, table) do
  table = insert_tup(table, char, Enum.reverse(code))
  table = encode_table_tup(long, code ++ '-', table)
  table = encode_table_tup(short, code ++ '.', table)
  table
end
```

Funktionen returnerar en tuppel av längd 128, där varje index innehåller koden för det motsvarande asciivärdet.

```elixir
# En del av avkodningstabellen
{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, '--..', 0, 0, 0,
 0, '--.--.', 0, 0, 0, 0, 0, 0, '--..--', '-....-', '-.-.-.',
 '-----.', '-----', '----.', ...}
```

Uppslagningar i tabllen görs med `encode_lookup_tup/2` och har komplexitet `O(1)`.

```elixir
def encode_lookup_tup(table, elem) do elem(table, elem) end
```

Kodaren

Nästa steg var att implementera en funktion som med hjälp av kodningstabellen kodar en sträng till morsekod. Kravet var en komplexitet på $O(m*n)$, där `n` är antal antal tecken i texten och `m` är antalet tecken i kodningstabellen. Funktionen skulle inte heller bygga på stacken.

Min implementation har komplexitet $O(n)$, då `n` uppslagningar med komplexitet $O(1)$ görs. Den är även svansrekursiv då alla uträkningar görs innan det rekursiva anropet.

```elixir
# Basfall: Alla tecken har kodats.
def encode_tup([], _, encoded) do Enum.reverse(encoded) end
def encode_tup([first | text], table, encoded) do
  # Uträkningar
  encoded = [32] ++ encode_lookup_tup(table, first) ++ encoded
  encode_tup(text, table, encoded) # Rekursivt anrop
end
```

Mitt namn i morsekod.

```elixir
'axel karlsson' == .- -..- . .-.. ..-- -.- .- .-. .-.. ... ... --- -.
```

Avkodning

I den sista delen av uppgiften skulle en funktion som avkodar morsekod till en sträng implementeras. Kravet var en komplexitet $O(m)$, där m är antalet tecken (punkter, bindesstreck och mellanrum) i koden.
Jag använde en liknande lösning som i Huffman uppgiften. Funktionen tar tecken ur koden, alltså punkter och bindesstreck, och använder dem för att gå genom det givna avkodningsträdet. När den stöter på ett mellanrum i koden kommer trädet (det andra argumentet) peka på en nod som innehåller den bokstav som avkodades. Den bokstaven läggs till i ackumulatorlistan.

```elixir
def decode(text) do decode(text, morse(), []) end # Lägg till argument
def decode([], _, acc) do Enum.reverse(acc) end # Alla tecken har avkodats
def decode([head | text], {:node, val, long, short}, acc) do
  case head do
  # Mellanrum
    32 ->
      acc = [val | acc] # Mellanrum, alltså slutet på ett tecken.
      decode(text, morse(), acc)
    ?- ->
      decode(text, long, acc) # Bindesstreck
    ?. ->
      decode(text, short, acc) # Punkt
  end
end
```

Funktionen har komplexitet $O(m)$ eftersom ett anrop görs för varje tecken i koden.

’.-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....-
.-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....-
.-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— -....-
.-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.-
-....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.-
-....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— -....- .-.-.-
.-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....-
.-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....-
..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.-
-....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.-
-....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....-
.-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....-
.-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....-
..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.-
-....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.-
-....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..—
-....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.- -....-
.-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....-
..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....-
.-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.-
-....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....-
.-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.-
.-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..—
.-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— -....- .-.-.-
.-.-.- .-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....-
..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.-
-....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.-
.-.-.- .-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..—
-....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.-
.-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.-
.-.-.- -....- ..— .-.-.- .-.-.- -....- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.-
-....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.-
-....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.-
-....- ..— .-.-.- .-.-.- -....- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— .-.-.- -....-
.-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.- -....-
.-.-.- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....-
..— .-.-.- .-.-.- -....- -....- ..— .-.-.- -....- .-.-.- -....- .-.-.- -....- ..— .-.-.- -....- .-.-.-
-....- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.- -....- ..— -....- .-.-.- .-.-.- .-.-.- .-.-.-
-....- ..— .-.-.- .-.-.- -....- -....- ..— ’