Tinpro01-8 Opdracht 3: Huffman compressie

W. Oele

20 mei 2022

Inleiding

In de vorige opdracht heb je kunnen zien dat het run-length algoritme niet altijd even efficiënt is. In deze opdracht ga je het Huffman algoritme programmeren. Enkele eigenschappen van dit algoritme:

- aanzienlijk efficiënter dan het run-length algoritme.
- heeft geen problemen met getallen.
- behaalt vaak een compressie factor die zeer dicht tegen het theoretische maximum aanzit.
- aanzienlijk moeilijker te programmeren dan het run-length algoritme.
- wordt gebruikt in het `.zip` bestandsformaat.

Uitleg

Om de werking van het Huffman algoritme te illustreren gaan we een kort stukje tekst comprimeren, te weten onderstaande string:

```
aaabbcbbcccddccbbcccdecccaaaaabbbb
```

Huffman compressie stap 1: letters tellen

In de gegeven string tellen we van elke letter hoe vaak deze voorkomt in de tekst. Zodoende krijgen we de onderstaande tabel:

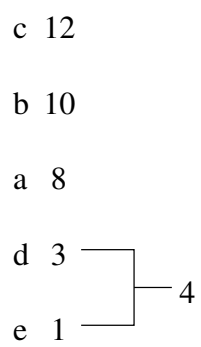
8	10	12	3	1
a	b	c	d	e

We sorteren deze tabel aflopend op het aantal keren dat de letter voorkomt in de tekst:

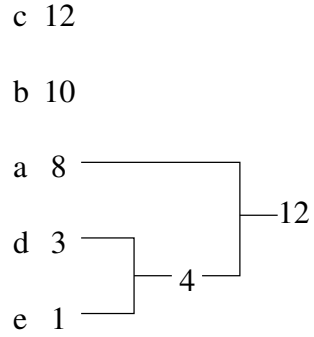
12	10	8	3	1
c	b	a	d	e

Huffman compressie stap 2: de coderingsboom bouwen

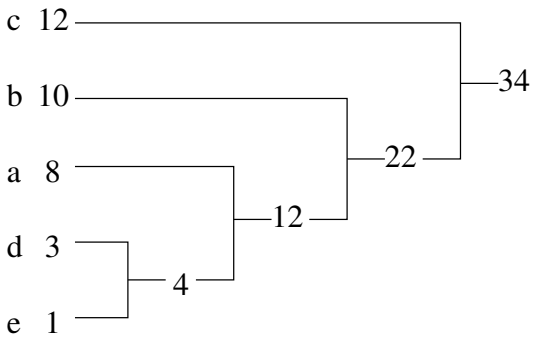
We kiezen uit de lijst met letters de twee letters die het *minst* in de tekst voorkomen en tellen hun getalswaarde bij elkaar op. Zo ontstaat:



We herhalen deze stap...

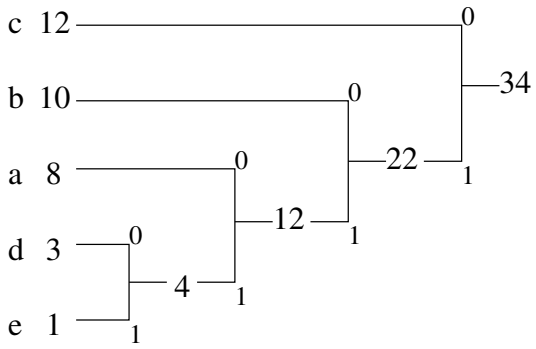


...totdat we alle letters gehad hebben en de hele boom is opgebouwd:



Opmerking: In bovenstaande stappen is te zien dat de boom netjes van beneden naar boven wordt opgebouwd. Dit is niet altijd het geval.

Nu lezen we de boom van rechts naar links en bij elke vertakking plaatsen we een 0 en een 1:



Huffman compressie stap 3: de Huffman codering opstellen

We kunnen een Huffman codering opbouwen door de boom van rechts naar links te lezen en elke 0 of 1 te noteren die we onderweg tegenkomen. Zo ontstaat:

0	10	110	1110	1111
c	b	a	d	e

Huffman compressie stap 4: de tekst comprimeren

We nemen de oorspronkelijke tekst erbij en vervangen elke letter die we tegenkomen door het bitpatroon in de tabel. Zo ontstaat:

`aaabbcbbcccddccbbcccdecccaaaaabbbb`

```
1101101101010010100001110111000101000011101111000110110110110110101010101
```

Wat zijn we nu opgeschoten? Welnu:

- Het aantal letters in de oorspronkelijke tekst bedraagt 34. Dat is $34 \times 8 = 272$ bits.
- Het aantal bits na Huffman compressie is 72.
- De compressiefactor is derhalve: $\frac{72}{272} \times 100 = 26\%$.

Huffman decompressie

Het decomprimeren van een gecomprimeerde/gecodeerde tekst bestaat uit het in omgekeerde volgorde doorlopen van bovenstaand proces:

- Lees een bit uit het gecomprimeerde bestand
- Neem de juiste afslag in de coderingsboom
- Herhaal bovenstaande stappen, totdat je in de coderingsboom bij een letter uitkomt.

Uit bovenstaande procedure kunnen wij beredeneren dat voor het decomprimeren van een tekst niet alleen de gecomprimeerde tekst nodig is, maar ook de coderingsboom, waarmee de oorspronkelijke tekst gecomprimeerd werd.

Opdracht 1: Huffman compressie

Schrijf een programma dat een eenvoudig tekstbestand comprimeert m.b.v. Huffman compressie. Voorwaarden:

- Noem het programma huffmancompress
- Het programma draait stand-alone, dus zonder interpreter.
- Het programma krijgt op de commandline drie parameters mee: De naam van het te comprimeren bestand, gevolgd door de naam van het bestand, waarin de gecomprimeerde gegevens moeten worden opgeslagen, gevolgd door de naam van het bestand, waarin de coderingsboom wordt opgeslagen.
- Gebruik voor het opslaan van de coderingsboom de serialisatie techniek zoals uitgelegd in de les.

- Het programma berekent de compressiefactor zoals boven beschreven en print deze op het scherm.
- Test je programma met een eenvoudig ASCII textbestand dat minstens 500 woorden bevat.
- Test je programma ook op een stuk ASCII art.

Een uitvoer van het programma ziet er, bijvoorbeeld, als volgt uit:

```
wessel@digitalsnail: ./huffmancompress text.txt compressed.txt tree.txt
length of text.txt: 34 characters, 272 bits.
length of compressed file compressed.txt: 72 bits.
factor: 72/272*100=26%
file compressed.txt written to disk...
file tree.txt written to disk...
done...
```

Tips, werkwijze en aanwijzingen

Letters tellen

Voor het tellen van de aantallen voorkomens van letters in een string kun je uiteraard je eigen functies schrijven, maar wie even verder denkt ziet meteen dat het werken met een hashmap wellicht handig is. In Haskell bestaat de library `Data.HashMap.Lazy`. Je kunt deze library direct importeren in je programma, maar dan zullen er conflicten ontstaan daar zowel in de `Prelude` als in `Data.HashMap.Lazy` functies als `map`, `foldr` etc. voorkomen. De oplossing bestaat uit het gebruiken van de volgende constructie:

```
import Data.HashMap.Lazy as L
import Prelude as P
```

In je programma kun je nu onderscheid maken tussen de functies die in beide libraries aanwezig zijn:

```
P.map ...
L.map ...
```

Opdracht 2: Huffman decompressie

Schrijf een programma dat een eenvoudig tekstbestand decomprimeert m.b.v. Huffman compressie. Voorwaarden:

- Noem het programma huffmandecompress
- Het programma draait stand-alone, dus zonder interpreter.
- Het programma krijgt op de commandline drie parameters mee: De naam van het te decomprimeren bestand, gevolgd door de naam van het bestand, waarin de gedecomprimeerde gegevens moeten worden opgeslagen, gevolgd door de naam van het bestand, waaruit de coderingsboom die voor het decomprimeren nodig is, wordt gelezen.
- Gebruik voor het lezen van de coderingsboom de serialisatie techniek zoals uitgelegd in de les.
- Test je programma met de gecomprimeerde bestanden en bijbehorende coderingsbomen uit de vorige opdracht.
- Controleer of de gedecomprimeerde tekst gelijk is aan de originele tekst.

Een uitvoer van het programma ziet er, bijvoorbeeld, als volgt uit:

```
wessel@digitalsnail: ./huffmandecompress compressed.txt decompressed.txt tree.txt
length of decompressed file: 34 characters, 272 bits.
file decompressed.txt written to disk...
done...
```