

Compressie en decompressie van data: Huffman-algoritme

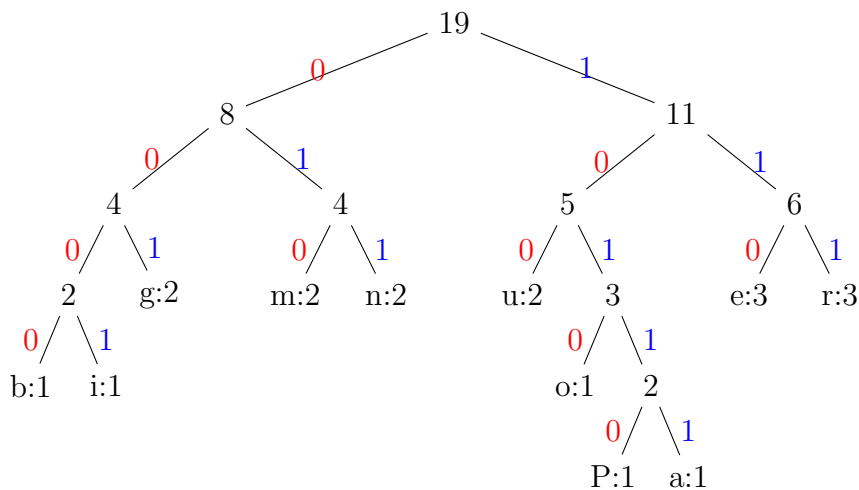
Opgave voor het tweede assessment

Werk de opgaven uit in groepen van drie studenten. Alle belangrijke code moet voorzien zijn van JUnit-testen. Geef commentaar in de broncode daar waar nodig! Boomklassen uit de javabibliotheek mogen niet worden gebruikt. Daarentegen mogen wel HashMap en/of ArrayList worden gebruikt. Documenteer in ieder geval de oplossingsstrategie in een verslag, eventueel voorzien van codefragmenten.

Opgave 1: Huffmanbomen

Tekstbestanden laten zich vaak goed comprimeren, omdat ze maar een klein deel van de beschikbare tekens bevatten. Een teken gebruikt meestal maar 1 Byte = 8 bits opslagruimte; 8 bit komt overeen met 8 binaire tekens (0 of 1). Zo kunnen dan in een byte karakters met positie nummers van 0 tot en met 255 coderen.

Om opslagruimte te besparen probeert men de veel voorkomende tekens niet met 8 bit, maar met minder bits te coderen. Als men dat toepast kan er veel ruimte bespaard worden. Een techniek om coderingen voor tekens te maken zijn zogenaamde Huffmanbomen. Die hebben de volgende structuur:



Figuur 1: Een Huffmanboom voor de volgende tekst: `Programmieruebung`

- Huffmanbomen zijn binaire bomen,
- De bladen in deze boom zijn knopen met twee attributen: symbool en frequentie. De andere knopen worden inwendige knopen genoemd. Die hebben maar een attribuut: de frequentie. De frequentie is de som van de frequenties van beide kindknopen. Laatstgenoemde zijn of van het type bladknoop of van het type inwendige knoop.
- De *tak*, die van een knoop naar linksonder gaat is met 0 gelabeld, de andere met 1.

Uw opgave:

Ontwerp een geeignende datatype voor de representatie van Huffmanbomen. Schrijf de programmacode, die een tekstbestand inleest en daarvoor een Huffmanboom opbouwt. Pas daartoe het volgende algoritme toe:

1. Lees het bestand volledig in en tel hoe vaak elk teken (symbool) voorkomt (bijv. 13 keer 'a', 14 keer 'b', ...). Er moeten minimaal twee verschillende tekens in het bestand voorkomen.
2. Maak bladknopen, voor elk teken een. Zet de attributen voor dat teken: symbool en frequentie.
3. Maak een lijst L met alle knopen. Een List uit de javabibliotheek mag worden toegepast.
4. Kies twee knopen in L met de kleinste frequenties. Hebben meerdere knopen dezelfde frequentie, kies er dan twee willekeurig uit. Verwijder die twee knopen uit L.
5. Maak een nieuwe inwendige knoop en maak de frequentie van die knoop gelijk aan de som van de twee bladknopen.
6. De twee bladknopen worden nu kindknopen van de nieuwe inwendige knoop. De bladknoop met de kleinste frequentie komt links, en de tak krijgt het label 0. De andere bladknoop komt rechts en de tak wordt gelabeld met 1. Bij gelijke frequenties in de bladknopen is de volgorde willekeurig.
7. Voeg de nieuwe inwendige knoop toe aan de lijst L.
8. Is er in L precies een knoop aanwezig, dan is die knoop de wortel van de Huffmanboom! Het algoritme stopt dan.
 - Volgt men de weg van de wortel van de Huffmanboom naar een bladknoop, dan is de opeenvolging van de labels op de takken de binaire code voor het symbool dat in de bladknoop staat.
 - Laat het programma een lijst van tekens printen met de bijbehorende codering. De volgorde van de tekens is niet voorgeschreven.
 - Maak deze lijst, door in het algoritme de boom zo te doorlopen, dat de volgorde van de labels op de takken wordt uitgegeven. (Tip: overleg of er nuttig gebruik gemaakt kan worden van een stack, doordat de inhoud van de stack het codewoord bevat.)

Voorbeeld:

- Invoerbestand text.txt: Programmieruebungen
- Uitvoer in tabel 1 (de bijbehorende boom is in figuur 1 gegeven.)

Opgave 2: Compressie

Breid het programma van opgave 1 uit, zodat het tekstbestand nogmaals wordt ingelezen, maar dat nu, nadat de boom gemaakt is, de gecomprimeerde versie, in 1-en en 0-en wordt uitgegeven. Laat het programma ook berekenen hoeveel bits voor en hoeveel bits na compressie nodig waren voor deze tekst. Ga er daarbij vanuit dat de tekens in de tekst ongecomprimeerd met 8 bit/teken gecodeerd werden.

	Frequentie		Codering
P:	1	P:	10110
a:	1	a:	10111
b:	1	b:	0000
e:	3	e:	110
g:	2	g:	001
i:	1	i:	0001
m:	2	m:	010
n:	2	n:	011
o:	1	o:	1010
r:	2	r:	111
u:	2	u:	100

Tabel 1: Frequentietabel en coderingsresultaat

Voorbeeld:

- Invoerbestand text.txt: Programmieruebungen
- Uitvoer:
1011011110100011111011101001000011101111001100000100011001110011
- Voor compressie: 152 Bits
- Na compressie: 64 Bits

In de bijlage bevindt zich een frequentietabel voor letters in duitstalige teksten. Aan de hand van deze tabel kan men ook een Huffmanboom maken en nogmaals dezelfde tekst coderen. Realiseer ook deze codering en vergelijk het resultaat met al in opgave 1 gemaakte codering.

Tip:

- Gebruik de functionaliteit uit de eerste opgave, implementeer eventueel een nieuwe interface.
- Als de uitwerking van opgave 1 niet volledig is, schrijf in dat geval een methode, die de boom niet vanuit een tekstbestand opbouwt, maar een of andere zinvolle boom bouwt.

Opgave 3: decompressie

De decompressie is in de praktijk wat moeilijker, omdat hiervoor natuurlijk de Huffmanboom nodig is. Deze boom kan men niet met behulp van het compressiealgoritme reconstrueren. Daarom is deze opgave ietwat onrealistisch vereenvoudigd:

Schrijf een volgend programmadeel voor de decompressie, dat een tekstbestand als invoer krijgt en daarna een Huffmanboom opbouwt. Vervolgens leest het programma een tweede tekst in die enkel uit 1-en en 0-en bestaat.

Voorbeeld:

- Invoerbestand text.txt: Programmieruebungen
- Invoer:
1011011110100011111011101001000011101111001100000100011001110011
- Uitvoer: Programmieruebungen

Tip:

- Gebruik de functionaliteit uit de eerste opgave, implementeer eventueel een nieuwe geschikte interface.

Bijlage

Frequentietabel van de letters in duitstalige.

Freuentie: aantal in 1 miljoen willekeurige symbolen.

a	43309	p	4992
b	15972	q	142
c	26733	r	68577
d	43854	s	53881
e	147004	t	47310
f	13598	u	31877
g	26672	v	7350
h	43554	w	14201
i	63770	x	129
j	1645	y	173
k	9558	z	14225
l	29312	ä	4907
m	21336	ö	2547
n	88351	ü	5799
o	17717	overige tekens	151505