Träd och koder

Anders Björner

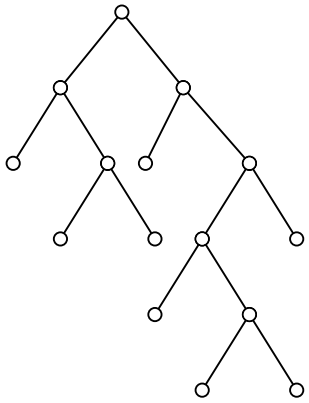
KTH

1. Inledning. Det är i flera sammanhang viktigt att representera information digitalt (d.v.s omvandla till sviter av nollor och ettor). Beroende på vilka villkor som olika situationer ställer leder detta till ett matematiskt studium av olika former av *koder*. Bortsett från det uppenbara kravet att informationen skall kunna återvinnas (avkodas) kan man vara intresserad av exempelvis motståndskraft mot störningar (felrättande koder) eller sekretess (kryptografi).

Här skall vi hitta en lösning på problemet hur språk, där bokstäverna förekommer med olika frekvenser, på effektivaste sätt kan kodas digitalt. Till hjälp använder vi enkla kombinatoriska egenskaper hos så kallade *binära träd*.

2. Binära träd. Låt oss först förklara detta begrepp intuitivt så som de flesta tänker sig det. Ett binärt träd T är då något som kan se ut så här:

(2.1)



De små runda ringarna kallas *noder*, och den nod som sitter längst upp är trädets *rot*. (Som synes växer träd i matematik och datalogi

i regel i motsatt riktning mot naturens träd – det har blivit en konvention.) För varje nod v gäller endera av följande två fall:

(1) v har inga barn (d.v.s. inga kanter leder neråt från v), v kallas då ett *löv*.

(2) v har exakt två barn, v kallas då en *inre nod*.

I fallet (2) skiljer man på *vänsterbarn och högerbarn*, så att exempelvis de två träden

(2.2)

anses olika som binära träd (ty i det ena är rotens högerbarn ett löv men ej i det andra).

Binära träd brukar ofta formellt definieras på följande rekursiva sätt:

(a) en mängd med ett element är ett binärt träd,

(b) ett binärt träd består av ett element (kallat *rot*) och ett ordnat par av binära träd (kallade *vänsterdelträd* och *högerdelträd*).

ÖVNING 1. Tänk igenom denna formella definition så att du blir övertygad om att den beskriver samma matematiska objekt som de binära träd vi mer intuitivt definierat ovan.

Antalet noder i ett binärt träd måste vara udda. (Varför?) Hur många binära träd med $2k+1$ noder finns det? Svaret för små värden på k ges i följande tabell:

n	1	3	5	7	9	11	13
antal binära träd med n noder	1	1	2	5		42	132

Övning 2. Rita alla binära träd med 9 noder och upptäck på så sätt det värde som saknas i tabellen.

Övning 3. Låt t_k vara antalet binära träd med $2k+1$ noder. Finn en rekursionsformel för talsviten $t_0, t_1, t_2, \ldots$, d.v.s en formel som uttrycker t_k som funktion av $t_0, t_1, \ldots, t_{k-1}$ för alla $k \geq 1$.

Övning 4. Komplettera tabellen till alla udda $n \leq 15$ med hjälp av rekursionsformeln för t_k.

Man kan visa att antalet binära träd med $2k+1$ noder ges av följande exakta formel

$$t_k = \frac{(2k)!}{(k+1)!k!}, \tag{2.3}$$

där $k! = k \cdot (k-1) \cdot (k-2) \cdot \ldots \cdot 2 \cdot 1$, och $0! = 1$.

Övning 5. Komplettera tabellen till alla udda $n \leq 25$ med hjälp av (2.3).

Vi betraktar nu något visst binärt träd och låter n beteckna antalet noder, ℓ antalet löv, i antalet inre noder och e antalet kanter. Exempelvis, för träden i (2.2) gäller $n = 5, \ell = 3, i = 2, e = 4$, och för trädet i (2.1) finner vi $n = 15, \ell = 8, i = 7, e = 14$.

Övning 6. Beskriv i formler samband mellan

(a) n och e,

(b) n och i,

(c) n, ℓ och i.

Dra ur de tre formler du funnit slutsatsen att om endast *ett* av talen n, ℓ, i och e, är känt så kan övriga bestämmas därur.

3. Viktade binära träd. Låt T vara ett binärt träd. Varje nod v befinner sig på ett visst *djup* $d(v)$ lika med avståndet (antalet

kanter) från roten. Om vi listar lövens djup i växande ordning för träden i figur (2.2) får vi sviten $1, 2, 2$, och för figur (2.1) får vi $2, 2, 3, 3, 3, 4, 5, 5$.

ÖVNING 7.(A) Låt $d_1, d_2, \ldots, d_\ell$ vara lövens djup för ett binärt träd. Visa att då gäller

$$\sum_{i=1}^{\ell} \frac{1}{2^{d_i}} = 1. \tag{3.1}$$

(B) Låt $d_1, d_2, \ldots, d_\ell$ vara en godtycklig svit av ickenegativa heltal för vilka ekvation (3.1) gäller. Visa att det finns ett binärt träd vars löv sitter på djup $d_i, i = 1, 2, \ldots, \ell$.

I många datalogiska sammanhang studeras sökstrukturer där information finns lagrad i löven hos ett binärt träd. För att snabbt komma åt denna information är det av intresse att veta vilka träd med l löv som minimerar summan (L = mängden av löv)

$$\sum_{v \in L} d(v). \tag{3.2}$$

ÖVNING 8. Visa att bland alla binära träd med ℓ löv så minimeras summan (3.2) av de träd vars alla löv har djup b eller $b + 1$, där b är heltalet bestämt av $b \leq \log_2 \ell < b + 1$. (Visa också att det alltid finns sådana binära träd, de kallas *balanserade*.)

ÖVNING 9. Låt T vara ett godtyckligt binärt träd med n noder. Visa

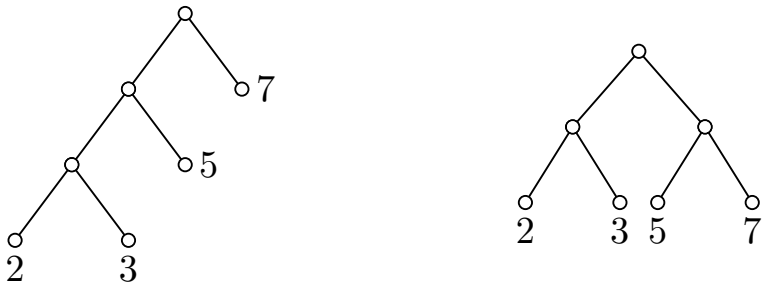
$$\sum_{v \in T} d(v) = 1 - n + 2 \sum_{v \in L} d(v). \tag{3.3}$$

(Således minimeras även totalsumman av *alla* noders djup av de balanserade träden.)

Vi har sett att de balanserade träden minimerar summan (3.2). De är därför de bästa träden för att lagra information som behöver åtkommas ungefär lika ofta. Men i praktiken är detta sällan fallet. Den information som finns lagrad i ett visst löv kanske efterfrågas mer än all annan information. Det är då intuitivt klart att detta löv bör sitta nära roten och på mindre djup än de andra löven. Detta leder till följande matematiska problemställning.

Först definierar vi begreppet *viktat binärt* träd. Detta är ett binärt träd T där varje löv v är märkt med ett reellt tal $w(v)$. Till exempel:

(3.4)



För varje viktat binärt träd bildar vi summan

$$w(T) = \sum_{v \in L} w(v) \cdot d(v), \tag{3.5}$$

kallad trädets *vikt*. Exempelvis, $w(T) = 2 \cdot 3 + 3 \cdot 3 + 5 \cdot 2 + 7 \cdot 1 = 32$ respektive $w(T) = (2 + 3 + 5 + 7) \cdot 2 = 34$ för träden i figur (3.4).

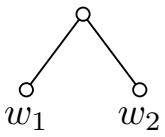
Problemet är nu att för givna vikter hitta ett binärt träd $T_\circ$ som bland alla viktade träd T minimerar $w(T)$. Mer precist, antag givna ℓ tal $w_1, w_2, \ldots, w_\ell$. Vi vill konstruera ett viktat binärt träd $T_\circ$ vars löv är märkta med $w_1, w_2, \ldots, w_\ell$, och sådant att $w(T_\circ) \leq w(T)$ för alla andra tänkbara sådana träd T. Vi kallar ett sådant viktat binärt träd $T_\circ$ *optimalt*.

Sambandet med det tidigare resonemanget om informationslagring är följande. Antag att ℓ *objekt* (t.ex. telefonnummer) skall lagras i löven till ett binärt träd. Varje objekt efterfrågas med en viss känd frekvens $w_i, i = 1, 2, \ldots, \ell$. När ett objekt skall sökas börjar sökningen alltid vid roten och fortskrider genom successiva höger/vänster-val neråt till det löv där det sökta objektet är lagrat. Antalet steg i sökningen (och därmed proportionellt även åtgången tid och kostnad) är lika med lövets djup. Problemet att lagra de ℓ objekten på effektivaste sätt (d. v. s. så att den totala tiden och kostnaden för sökning minimeras) är då uppenbart ett specialfall av vårt matematiska problem.

Innan vi beskriver problemets lösning är det värt att kommentera specialfallet $w_1 = w_2 = \cdots = w_\ell = 1$. I det fallet löste vi problemet redan i övning 8. Märk att de balanserade träden inte är optimala i allmänhet: det högra trädet i figur (3.4) är balanserat men har högre vikt än det vänstra obalanserade trädet (vilket som vi strax skall se är optimalt).

Följande eleganta algoritmiska lösning på problemet gavs av D. Huffman 1952. För $\ell = 2$ och givna vikter w_1 och w_2 är följande träd optimalt (detta är uppenbart):

(3.6)



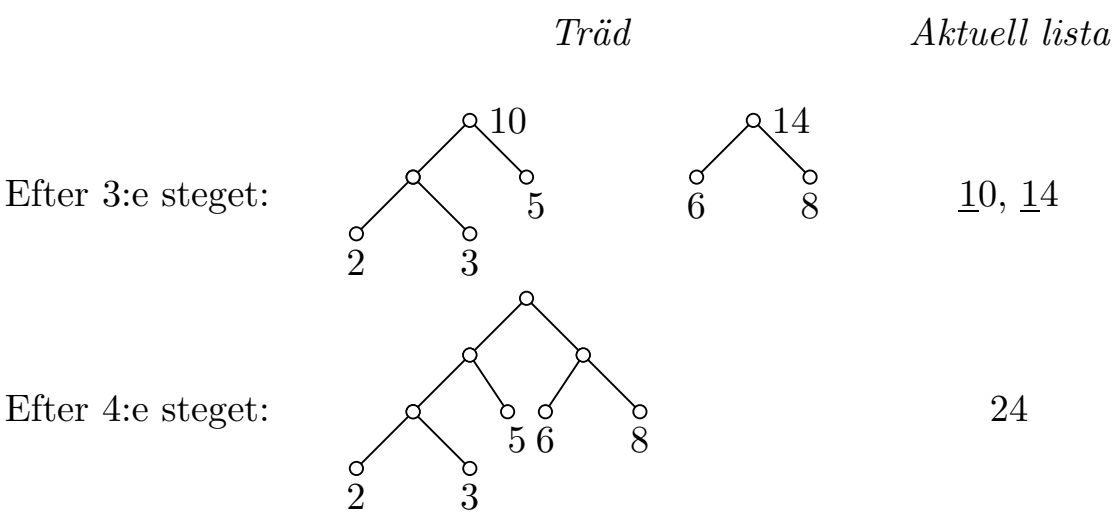
För $\ell \geq 3$ och givna vikter $w_1, w_2, \ldots, w_\ell$ väljer vi ut de två minsta, säg w_1 och w_2. Rekursivt kan vi anta att vi har en metod att konstruera ett optimalt träd T'_o med $\ell - 1$ löv och vikterna $w_1 + w_2, w_3, \ldots, w_\ell$. Ersätt det löv i T'_o som har vikten $w_1 + w_2$ med ett delträd som i (3.6). Det träd T_o som då erhålls kan visas vara optimalt för $w_1, w_2, \ldots, w_\ell$.

Den rekursiva formuleringen av Huffmans algoritm är lämpad för att bevisa dess korrekthet. Mer om detta strax. För praktiskt bruk är följande omformulering av algoritmen mer användbar. (Tänk igenom att de två versionerna verkligen uttrycker samma sak!)

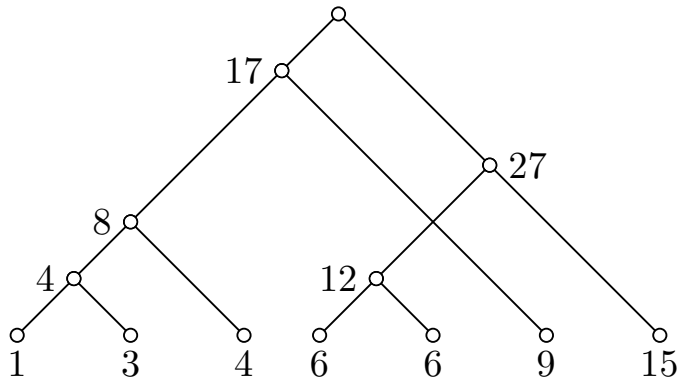
Huffmans Algoritm: Givet tal $w_1, w_2, \ldots, w_\ell$, som vi uppfattar som märkta noder. Upprepa steget: För de två minsta talen u_1 och u_2 i aktuell lista (d.v.s. $u_1 \leq u_2 \leq u_3 \leq \cdots \leq u_k, k \leq \ell$) skapa en *förälder*-nod märkt $u_1 + u_2$ som har u_1 och u_2 som barn. Aktuell lista är till att börja med $w_1, w_2, \ldots, w_\ell$ och senare ersätts i varje steg de två barnen av sin förälder i den aktuella listan. När den aktuella listan har bara ett element stannar algoritmen, och den har då producerat ett optimalt binärt träd.

Vi illustrerar algoritmen med följande exempel. Antag de givna vikterna är 2, 3, 5, 6, 8:

	Träd	*Aktuell lista*
Utgångsläge:	2 3 5 6 8	2, 3, 5, 6, 8
Efter 1:a steget:	5 (2, 3) 5 6 8	5, 5, 6, 8
Efter 2:a steget:	10 (5 (2, 3)) 6 8	10, 6, 8



Med denna algoritm bestämmer man förvånansvärt snabbt optimala viktade binära träd även för längre talsviter. Som ett ytterligare exempel utgår vi från talsviten 1, 3, 4, 6, 6, 9, 15:



ÖVNING 10. Bestäm ett optimalt viktat binärt träd för talsviten

(a) 1, 2, 2, 3, 3, 3, 4, 4, 4, 4.

(b) 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41.

ÖVNING 11. Säg att 7 telefonnummer, som vi kallar A, B, C, D, E, F och G, skall lagras i löven till ett binärt sökträd. Dessa telefonnummer efterfrågas med följande relativa frekvenser: A 9%, B 12%,

C 3%, D 41%, E 6%, F 22%, och G 7%. Hur skulle du välja ett lämpligt träd för detta ändamål?

Övning 12. Bevisa att Huffmans algoritm är korrekt, d. v. s. att det träd som algoritmen producerar verkligen är optimalt.

[*Ledning*: Resonemanget kan föras längs följande linjer. Antag att $w_1 w_2 \leq \cdots \leq w_\ell$.

(a) Antag att i ett optimalt träd lövet med vikt w_i sitter på djup $d_i, 1 \leq i \leq \ell$. Visa att om $w_i < w_j$ så gäller $d_i \geq d_j$.

(b) Dra ur (a) slutsatsen att det finns ett optimalt träd $T_\circ$ där w_1- och w_2-noderna har maximalt djup och är bröder (har samma förälder).

(c) Låt $T_\circ$ vara ett optimalt träd som i (b) och låt T_H vara ett träd som producerats enligt Huffmans algoritm. Jämför $w(T_\circ)$ och $w(T_H)$ och utnyttja att vi rekursivt kan anta att Huffmans algoritm är korrekt för de $\ell - 1$ vikterna $w_1 + w_2, w_3, \ldots, w_\ell$.]

4. Prefix-koder. Antag att vi har ett ändligt alfabet A och ett *språk* som består av vissa strängar av symboler ur A. Det kan exempelvis vara ett alfabet till ett naturligt språk som det svenska eller engelska, där symbolsträngarna är naturliga ord. Eller $A = \{0, 1, \ldots, 9\}$ och *orden* vissa naturliga tal skrivna i decimalsystemet (d.v.s. i bas 10).

Problemet att koda naturliga språks bokstäver dök upp i telegrafins barndom. Den välbekanta *Morse-koden* ersätter varje bokstav med en kombination av korta och långa tonstötar, vilka vi kan uppfatta som nollor och ettor. Exempelvis:

Bokstav	*Morsekod*	*Binär form*
a	$\cdot\,-$	01
e	$\cdot$	0
i	$\cdot\cdot$	00
m	$-\,-$	11
o	$-\,-\,-$	111
s	$\cdot\,\cdot\,\cdot$	000
t	$-$	1
z	$-\,-\,\cdot\cdot$	1100

Denna kod är konstruerad med hänsyn till att olika bokstäver förekommer med olika frekvenser: vanliga bokstäver som *e* och *t* har kortare kodord än mindre frekventa bokstäver. Morsekoden är dock inte helt binär. För att kunna avkoda måste man veta var ett kodord slutar och nästa börjar. Detta sker genom ett kort uppehåll i överföringen. Denna paus har alltså också innebörd, och Morsekoden är därför *ternär*, d.v.s. den använder 3 symboler: kort, lång och paus. Tag t.ex. den välkända nödanropssignalen *S O S*, som i Morsekod lyder "$\cdots, ---, \cdots$". Om pauserna tas bort blir lydelsen "$\cdots ------ \cdots$" vilket kan avkodas inte bara som *S O S* utan även som *iamei*, *eeetze*, m.m.

Eftersom pauser är så vanligt förekommande vore det en uppenbar effektivitetsvinst att slippa koda dem. Detta kan ske om det är möjligt att ändå otvetydigt avgöra var ett kodord slutar och nästa börjar. Det enklaste sättet att uppnå detta är att välja alla kodord av samma längd. Eftersom det finns 2^k binära ord av längd k, kan ett alfabet med n symboler, där $n \leq 2^k$, alltid kodas med en så kallad *blockkod* av längd k. Exempelvis kan vi för $A = \{a, b, c, d, e\}$ välja följande kod av längd 3:

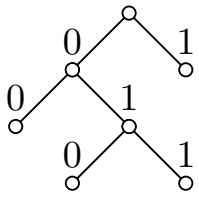
$$
\begin{array}{lcl}
a & \longleftrightarrow & 000 \\
b & \longleftrightarrow & 001 \\
c & \longleftrightarrow & 010 \\
d & \longleftrightarrow & 011 \\
e & \longleftrightarrow & 100 \\
f & \longleftrightarrow & 101
\end{array}
\tag{4.1}
$$

För att avkoda ett budskap som ”010000101101100” delar vi upp det i grupper om 3 och avläser *caffe*.

Nackdelen med blockkoder ur effektivitetssynpunkt är att de inte förmår ta hänsyn till bokstävernas relativa frekvenser: vanliga bokstäver tar onödigt mycket plats och ovanliga tar oskäligt liten plats. Kan Morsekodens frekvensanpassning och blockkodens paus-frihet kombineras? Svaret är *ja*, och har man studerat de binära trädens matematik är det inte svårt att se hur.

En ändlig samling K av binära ord (strängar av nollor och ettor) kallas en *prefix-kod* om det aldrig inträffar att ett ord i K utgör begynnelseavsnitt (prefix) i ett annat ord i K. Exempelvis, {00, 01, 1} är en prefix-kod men inte {00, 10, 1} eftersom 1 är ett prefix i 10. Det är uppenbart att det exakta villkoret för att en kod skall kunna avkodas utan någon särskild markering av var ett ord slutar och nästa börjar (paus-frihet) är att kodorden bildar en prefix-kod.

Prefix-koder kan på ett enkelt sätt erhållas ur binära träd. För varje löv v följer man stigen från roten till v och betecknar steg till vänster med 0 och steg till höger med 1. Exempelvis avläser vi från trädet



prefix-koden {00, 010, 011, 1}, och från träden i figur (2.2) får vi koderna {00, 01, 1} och {0, 10, 11}.

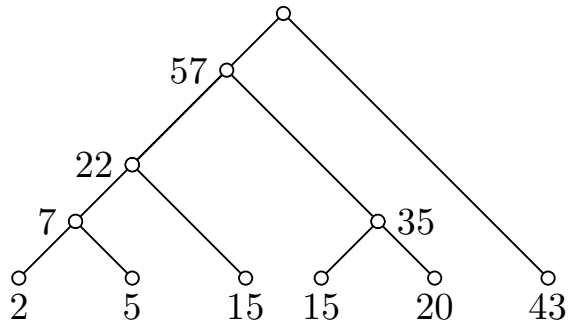
ÖVNING 13. (A) Ange den prefix-kod som bestäms av trädet i figur (2.1).
(B) Finn det binära träd som ger upphov till prefixkoden {00, 01, 1000, 10010, 10011, 101, 11}.
(C) Finns det något binärt träd som ger upphov till koden {00, 011, 10, 11} ? Är detta en prefix-kod?
ÖVNING 14. (A) Visa att en prefix-kod som kommer från ett binärt träd har egenskapen att varje oändlig binär talföljd $a_1a_2a_3\ldots$, $a_i \in \{0,1\}$, entydigt kan uppdelas i kodord.
(B) Omvänt, visa att om en prefix-kod har egenskapen i (a) så kommer den från ett binärt träd.

Vi kommer nu till lösningen på det problem som ställdes i inledningen: Hur skall man på effektivaste sätt digitalt koda ett språk vars bokstäver förekommer med vissa kända frekvenser? För att slippa koda pauser mellan kodorden bör vi välja en prefix-kod, och man kan visa (detta är inte svårt) att bland prefix-koderna är de som kommer från binära träd i varje situation minst lika effektiva som de övriga. I en kod som kommer från ett binärt träd svarar kodorden mot trädets löv, och ett kodords längd (antal binära siffror) är lika med lövets djup. För ett alfabet $A = \{x_1, x_2, \ldots, x_n\}$ med givna bokstavsfrekvenser $w(x_i), 1 \leq i \leq n$, gäller det alltså att finna ett binärt träd vars löv svarar mot A (exakt ett löv för varje bokstav) och så att summan

$$w(T) = \sum_{i=1}^{n} w(x_i) \cdot d(x_i)$$

är så liten som möjligt. Men detta är ju exakt det problem som Huffmans algoritm löser!

Vi illustrerar metoden med följande exempel. Antag att vi skall koda en text i alfabetet $\{a, b, c, d, e, f\}$ där de individuella bokstäverna förekommer med följande relativa frekvenser: $a - 0,43; b - 0,20; c - 0,15; d - 0,15; e - 0,05$ och $f - 0,02$. Vi använder Huffmans algoritm för att finna ett optimalt binärt träd och vikterna $43, 20, 15, 15, 5$ och 2 :



Från detta avläser vi den optimala prefix-koden:

$$
\begin{array}{lcl}
a & \longleftrightarrow & 1 \\
b & \longleftrightarrow & 011 \\
c & \longleftrightarrow & 010 \\
d & \longleftrightarrow & 001 \\
e & \longleftrightarrow & 0001 \\
f & \longleftrightarrow & 0000
\end{array}
\tag{4.2}
$$

Med koden (4.2) kommer den genomsnittliga kodordslängden för varje bokstav att bli

$$0,43 + 3(0,20 + 0,15 + 0,15) + 4(0,05 + 0,02) = 2,21.$$

Detta skall jämföras med kodordslängden $3,00$ för blockkoden (4.1), en effektivitetsvinst med 26%.

ÖVNING 15. De svenska bokstäverna förekommer i löpande tidningstext med de relativa frekvenserna (i procent)

a	9,26	k	3,24	u	1,75
b	1,30	l	5,25	v	2,28
c	1,23	m	3,47	w	0,06
d	4,43	n	8,71	x	0,10
e	9,89	o	4,00	y	0,65
f	2,01	p	1,67	z	0,03
g	3,16	q	0,01	å	1,58
h	1,99	r	8,34	ä	2,02
i	5,67	s	6,46	ö	1,47
j	0,62	t	8,55		

(KÄLLA: Nusvensk frekvensordbok av S. Allén m.fl., Almkvist och Wiksell, Stockholm, 1970, sid 1053.) Konstruera en optimal prefixkod för svenska språket baserad på dessa data. Bestäm hur mycket effektivare denna kod är än den bästa blockkoden.

(Kommentar: Summan av ovanstående frekvenser är 99,2%. Återstående 0,8% av tidningstexten fördelade sig på 0,58% siffror (symbolerna $0, 1, \ldots, 9$) och 0,22% övriga symboler. Vi bortser i denna övning från dessa, liksom från det faktum att om löpande text (och ej bara enstaka ord) skall kodas så måste alfabetet utökas med en symbol för mellanrum som blir mycket frekvent.)

Litteratur

Knuth, D., The art of computer programming, *Vol. 1: Fundamental Algorithms (2nd Ed.).* Addison - Wesley, Reading, Ma., 1973.