
INF2310 – Digital bildebehandling

FORELESNING 12

KOMPRESJON OG KODING – I

Fritz Albregtsen

Tre steg i kompresjon

Redundanser

Koding og entropi

Shannon-Fano-koding

Huffman-koding

Aritmetisk koding

Kompendium: 18-18.3, 18.5-18.7.2, 18.7.4 og Appendiks B

Anvendelser

- Kompresjon benyttes for å **redusere antall biter** som brukes for å beskrive bildet (eller en god tilnærming av bildet).
- En mengde anvendelser innen datalagring og dataoverføring.
 - Televideokonferanser
 - Fjernanalyse / meteorologi
 - Overvåking / fjernkontroll
 - Telemedisin / medisinske arkiver (PACS)
 - Dokumenthåndtering / FAX
 - Multimedia / nettverkskommunikasjon
 - Mobil kommunikasjon
 - MP3-spillere, DAB-radio, digitalkameraer, ...
- **Tidsforbruket er viktig**, men det varierer om man ønsker å minimere kompresjonstiden eller dekompresjonstiden.
 - Det man gjør oftest ønsker man at tar kortest tid. Asymmetrisk kompresjon?
 - Begge tidene kan være omtrent like viktige. Symmetrisk kompresjon?

Eksempler: Plassbehov uten kompresjon

- Digitalt RGB-bilde:
 - $512 \times 512 \times 8 \text{ biter} \times 3 \text{ farger} = 6\,291\,456 \text{ biter} \approx 0,79 \text{ MB}$
 - $3264 \times 2448 \times 8 \text{ biter} \times 3 \text{ farger} = 191\,766\,528 \text{ biter} \approx 24 \text{ MB}$
- Røntgen-bilde:
 - $7\,112 \times 8\,636 \text{ piksler} \times 12 \text{ biter} = 737\,030\,784 \text{ biter} \approx 92 \text{ MB}$
- Radarbilde fra Radarsat-1-satellitten:
 - 400 MB, $300 \text{ km} \times 300 \text{ km}$, 16 biter per piksel.
 - For miljøovervåkning av Middelhavet:
28 slike bilder trengs for å dekke hele Middelhavet.
- 3D-seismikk-data fra $60\,000 \text{ km}^2$ i Nordsjøen.
 - $4 \text{ TB} = 4\,000 \text{ GB} = 4\,000\,000 \text{ MB}$

Plass og tid

- Digitale data kan ta stor plass.
 - Spesielt lyd, bilder og video.

- Eksempler:

1. Digitalt bilde:

$512 \times 512 \times 8 \text{ biter} \times 3 \text{ farger} = 6\,291\,456 \text{ biter}$

2. Røntgenbilde:

$7112 \times 8636 \times 12 \text{ biter} = 737\,030\,784 \text{ biter}$

- Overføring av data tar tid:

Linje med 64 kbit/sek:

1. ca. 1 min. 38 s.
2. ca. 3 timer 12 min.

Linje med 1 Mbit/sek:

1. ca. 6 s.
2. ca. 12 min.

Kapasiteten til enkelte linjer:

3G: Minst 200 kbps,
ofte noen Mbps

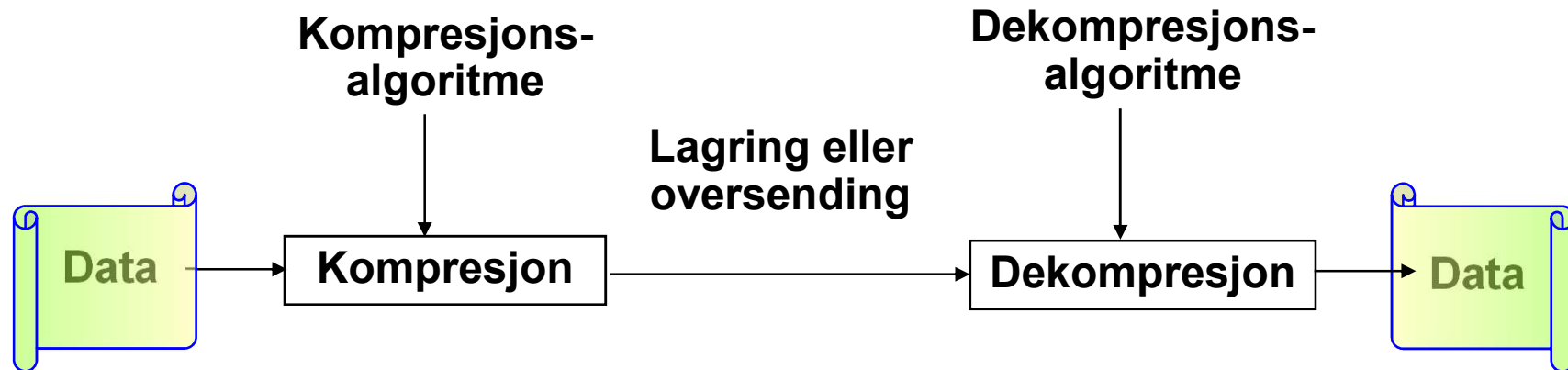
ADSL2+: Opptil 24 Mbps

VDSL2: Opptil 100 Mbps

SI-prefikser og binære prefikser

- Overføringshastigheter og linjekapasitet angis **alltid** med SI-prefikser, oftest som antall biter per sekund:
 - 1 kbps = 1000 bps = 10^3 biter per sekund
 - 1 Mbps = 1000 kbps = 10^6 biter per sekund
 - 1 Gbps = 1000 Mbps = 10^9 biter per sekund
 - 1 Tbps = 1000 Gbps = 10^{12} biter per sekund
- Filstørrelser er **oftest** gitt med binære prefikser:
 - Kibibyte (KiB = 2^{10} byte = 1 024 byte),
 - Mebibyte (MiB = 2^{20} byte = 1 048 576 byte),
 - Gibibyte (GiB = 2^{30} byte = 1 073 741 824 byte),
 - Tebibyte (TiB = 2^{40} byte = 1 099 511 627 776 byte)

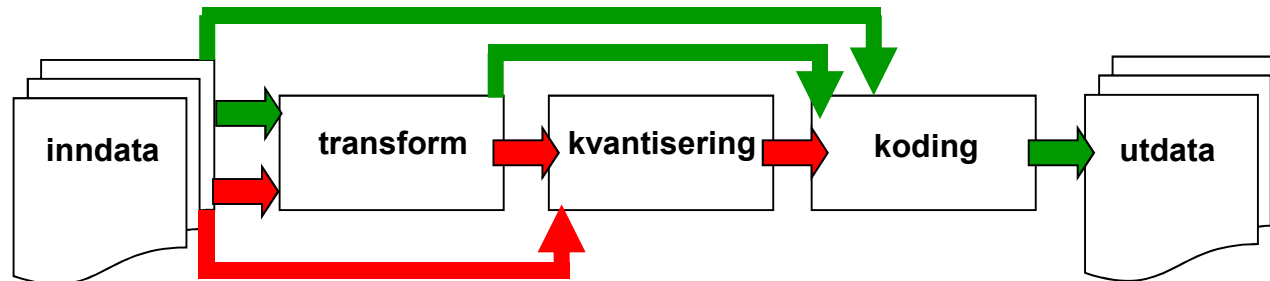
Kompresjon



- **Bildekompresjon** består i å representere informasjonen i bildet ved bruk av færre biter og ev. å ikke lagre redundant informasjon.
 - Bildet **komprimeres** og deretter lagres eller overføres dataene.
 - Når bildet senere skal brukes, så **dekomprimeres** dataene.
- Koding er en del av kompresjon, men målet med kodingen er å bruke færrest mulig biter, ikke å hemmeligholde eller skjule informasjon.

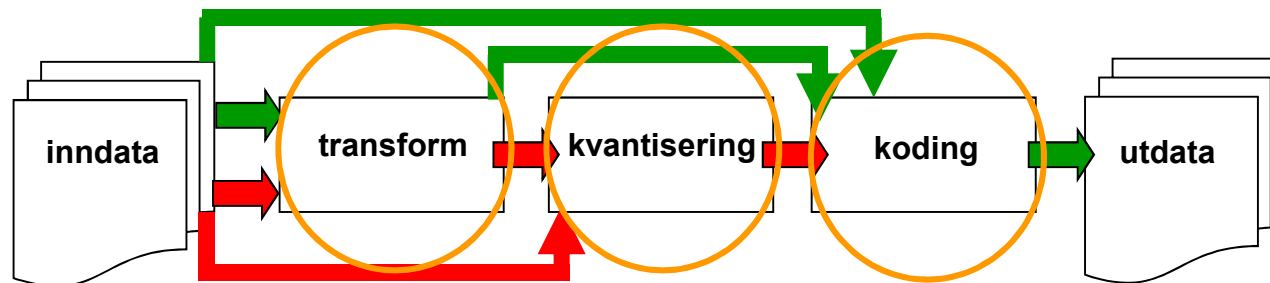
Kompresjon

- Kompresjon kan deles inn i tre steg:
 - **Transform** - representerer bildet mer kompakt.
 - **Kvantisering** - avrund representasjonen.
 - **Koding** - produser og bruk en kodebok.



- Kompresjon kan gjøres:
 - **Eksakt / tapsfri** (eng.: *lossless*) – følg de grønne pilene.
 - Kan da eksakt rekonstruere det originale bildet.
 - **Ikke-tapsfri** (eng.: *lossy*) – følg de røde pilene.
 - Kan da (generelt) ikke eksakt rekonstruere bildet.
 - Resultatet kan likevel være «godt nok».
 - Det finnes en mengde ulike metoder innenfor begge kategorier.

De tre stegene i kompresjon



- Mange kompresjonsmetoder er basert på å **representere** bildet på en annen måte, altså **transformer** av original-bildet.
 - Eks.: Differansetransform, løpelengde-transform.
- Hvis vi **kvantiserer** det (originale eller transformerte) bildet, så kan ikke dette reverseres \Rightarrow ikke-tapsfri kompresjon.
- Til slutt **koder** vi, dvs. transformerer melding til binærrepresentasjon.
 - Baserer seg ofte på normaliserte histogrammer.
- **Kodingene** vi bruker **er alltid reversible.**
- **Transformene** vi bruker **er alltid reversible.**
- **Kvantisering er ikke reversibelt!**


Melding, informasjon og data

- **Melding:** Teksten eller bildet som vi skal lagre eller sende.
- En melding inneholder en viss mengde informasjon.
- **Informasjon:** Et matematisk begrep som kvantifiserer hvor overraskende / uventet en melding er.
 - Et varierende bilde har mer informasjon enn et monotont bilde.
 - I bilder har kanter rundt objekter høyt informasjonsinnhold, spesielt kanter med mye (geometrisk) krumning.
- **Data:** En bitsekvens som representerer meldingen.

Redundans

- Vi kan bruke ulike mengder data på samme melding.
 - Anta meldingen er 13.
 - Med ISO 8859-1 trengs 16 biter; «13»; 00110001 00110011
 - Med 8-biters naturlig binærkoding trengs 8 biter; 00001101
 - Med 4-biters naturlig binærkoding trengs 4 biter; 1101
- **Redundans:** Det som kan «fjernes» fra dataene uten å miste (relevant) informasjonen.
 - Med «fjerne» menes her å redusere plassen dataene tar.
- I kompresjon ønsker vi å fjerne redundante biter.

Ulike typer redundans

- **Psykovisuell** redundans.  Mer generelt: **Irrelevant informasjon:** Unødvendig informasjon for anvendelsen, f.eks. for visuell betraktning av hele bildet.
 - Det finnes informasjon vi ikke kan se.
 - Eksempler på enkle muligheter for å redusere redundansen: Subsample eller redusere antall biter per piksel.
- **Interbilde**-redundans.
 - Likhet mellom nabobilder i en tidssekvens.
 - Kan reduseres ved å lagre noen bilder i tidssekvensen og differanser.
- **Intersampel**-redundans.
 - Likhet mellom nabopiksler.
 - Kan reduseres ved å løpelengde-transformere hver linje i bildet.
- **Kodings**-redundans.
 - Enkeltsymboler (enkeltpiksler) blir ikke lagret optimalt.
 - Gitt som gjennomsnittlig kodelengde minus et teoretisk minimum.
 - Velg en metode som er «grei» å bruke og gir liten kodingsredundans.

Kompresjonsrate og redundans

- **Kompresjonsraten:**

$$CR = \frac{b}{c}$$

der b er (det faste) antall biter per symbol i den ukomprimert datamengden, og c er gjennomsnittlig antall biter per symbol i den komprimerte datamengden.

- I et ukomprimert bilde blir alle pikslene lagret separat med naturlig binærkoding.

- **Relativ redundans:**

$$R = 1 - \frac{1}{CR} = 1 - \frac{c}{b}$$

- Også kalt plassbesparelse (eng. *space savings*).
- Ofte oppgitt i prosent.
- Prosentverdien kan kalles «percentage removed»:

$$PR = 100 R = 100 \left(1 - \frac{c}{b} \right)$$

Koding

- **Alfabet:** Mengden av alle mulige symboler (i bilder: alle mulige gråtoner).
- Ofte får hvert symbol et **kodeord**.
- **Kodebok:** Alle kodeordene og deres betydning.
- Kodingene vi bruker er **reversible**.
 - Denne egenskapen kalles for **unik dekodbarhet**; en kodeord-sekvens kan dekodes på én og bare én måte.
 - Vi kan betrakte ikke-reversible kodinger som en kombinasjon av en kvantifisering og en reversibel koding.
 - Hvis hvert symbol har et kodeord betyr dette at kodeordet skal entydig gi det originale symbolet.
- **Instantant dekodbare koder** kan dekodes uten skilletegn.

Naturlig binærkoding

- Alle kodeord er like lange.
- Symbolets kode er binærrepresentasjonen til symbolets (null-indekserte) indeks.
 - Man legger til 0-ere foran slik at koden får den ønskelige lengden.
- Eks: En 3-biters naturlig binærkode har 8 mulige verdier:

Symbolindeks	0	1	2	3	4	5	6	7
Symbol	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7
Kode c_i	000	001	010	011	100	101	110	111

- Naturlig binærkoding er bare «optimal» hvis alle verdiene i sekvensen er like sannsynlige.
 - Med «optimal» menes her at kompresjonen er «best» mulig.

Informasjonsteori og koding

- Koding bygger på sannsynligheter.
- Forekommer en pikselverdi oftere enn en annen, bør førstnevnte lagres med mindre antall biter for å bruke minst mulig biter på å lagre hele bildet.
- Det er altså plassbesvarende å bruke flere biter på symboler som forekommer sjeldent, fordi hyppige symboler da kan bruke færre biter.
- **Vi bør bruke et variabelt antall biter per symbol.**
- Vi skal nå først se på koding av enkeltpikslar.
- Bør allerede ha minimert annen redundans.
 - F.eks. gjennom transform-steget, som vi skal se på neste uke.

Koder med variabel lengde

- Når symbolene forekommer med ulik sannsynlighet er det bedre å bruke **kodeord med variabel lengde**.
 - Hyppige symboler \Rightarrow kortere kodeord.
 - Sjeldne symboler \Rightarrow lengre kodeord.
 - Dette var forretningsideen til Samuel Morse;

*De vanligste symbolene i engelsk tekst er:
e, t, a, o, i, n, ...*

Morse-kode: . og mellomrom varer 1 enhet, - varer 3 enheter

A	. -	F	.. - .	K	- . -	P	. - - .	U	.. -
B	- ...	G	-- .	L	. - ...	Q	-- . -	V	... -
C	- . - .	H	M	--	R	. - .	W	. - -
D	- ..	I	..	N	- .	S	...	X	- . . -
E	.	J	. - - -	O	- - -	T	-	Y	- . - -

Histogram og normalisert histogram

- Anta vi har en sekvens med N symboler.
- Tell opp antall ganger symbol s_i forekommer og la n_i være dette antallet.
 - Dette er det samme som histogrammet til sekvensen.
- Sannsynligheten til symbolene finnes da som:
 - $p_i = n_i / N$
 - Dette er det normaliserte histogrammet.
 - Hvis man antar at pikselverdiene er uavhengige realiseringer av en underliggende diskret variabel, så er p_i et estimat av sannsynligheten for at variabelen er symbol s_i .

Gjennomsnittlig antall biter per piksel

- Vi konstruerer en kodebok c_0, \dots, c_{G-1} slik at symbol s_i kodes med kodeordet c_i .
 - G er antall symboler i alfabetet.
- b_i er lengden av kodeordet c_i (angitt i biter).
- Gjennomsnittlig antall biter per symbol for koden er:

$$c = b_0 p_0 + b_1 p_1 + \dots + b_{G-1} p_{G-1} = \sum_{i=0}^{G-1} b_i p_i$$

der sannsynligheten til symbol s_i er p_i

Informasjonsinnhold

- Definer informasjonsinnholdet $I(s_i)$ i hendelsen s_i ved:

$$I(s_i) = \log_2 \frac{1}{p_i}$$

- $\log_2(x)$ er 2-er-logaritmen til x .
 - Hvis $\log_2(x) = b$ så er $x=2^b$
 - Eks: $\log_2(64)=6$ fordi $64=2^6 (=2*2*2*2*2*2)$
 $\log_2(8)=3$ fordi $8=2*2*2=2^3$
 - $\log_2(\text{tall}) = \log_{10}(\text{tall}) / \log_{10}(2)$ ($\log_{10}(2) \approx 0.301030$)
- $\log_2(1/p_i)$ gir oss informasjonsinnholdet i hendelsen: «symbolet s_i forekommer en gang», uttrykt i biter.

Entropi

- Gjennomsnittlig informasjonsinnhold i sekvensen, også kalt gjennomsnittlig informasjon per symbol, er da:

$$H = \sum_{i=0}^{G-1} p_i I(s_i) = - \sum_{i=0}^{G-1} p_i \log_2 p_i$$

Hvis $p(s_i)=0$ lar vi det tilhørende entropibidraget, $0 \log_2 0$, være 0.

- H er entropien til sekvensen av symbolene.
- **Entropien setter en nedre grense for hvor kompakt sekvensen kan representeres.**
 - Dette gjelder bare hvis vi koder hvert symbol for seg.

Øvre og nedre grense for entropi

- Hvis alle symboler like sannsynlige => entropi lik antall biter.

- Hvis det er $G=2^b$ symboler i alfabetet, og sannsynligheten for hvert av dem er $p_i = 1/2^b$, så er entropien:

$$H = -\sum_{i=0}^{G-1} \frac{1}{2^b} \log_2\left(\frac{1}{2^b}\right) = -\log_2\left(\frac{1}{2^b}\right) = b$$

- Altså: Hvis det er 2^b symboler som alle er like sannsynlige, så kan de ikke representeres mer kompakt enn med b biter per symbol - når vi koder symbolene enkeltvis.

- Hvis alle pikslene er like => entropi lik 0.

- Hvis bare ett symbol forekommer, er sannsynligheten for dette symbolet lik 1, og alle andre sannsynligheter er lik 0. Siden $\log_2(1) = 0$ vil entropien da bli lik 0.

Entropi i et binært bilde: To eksempler

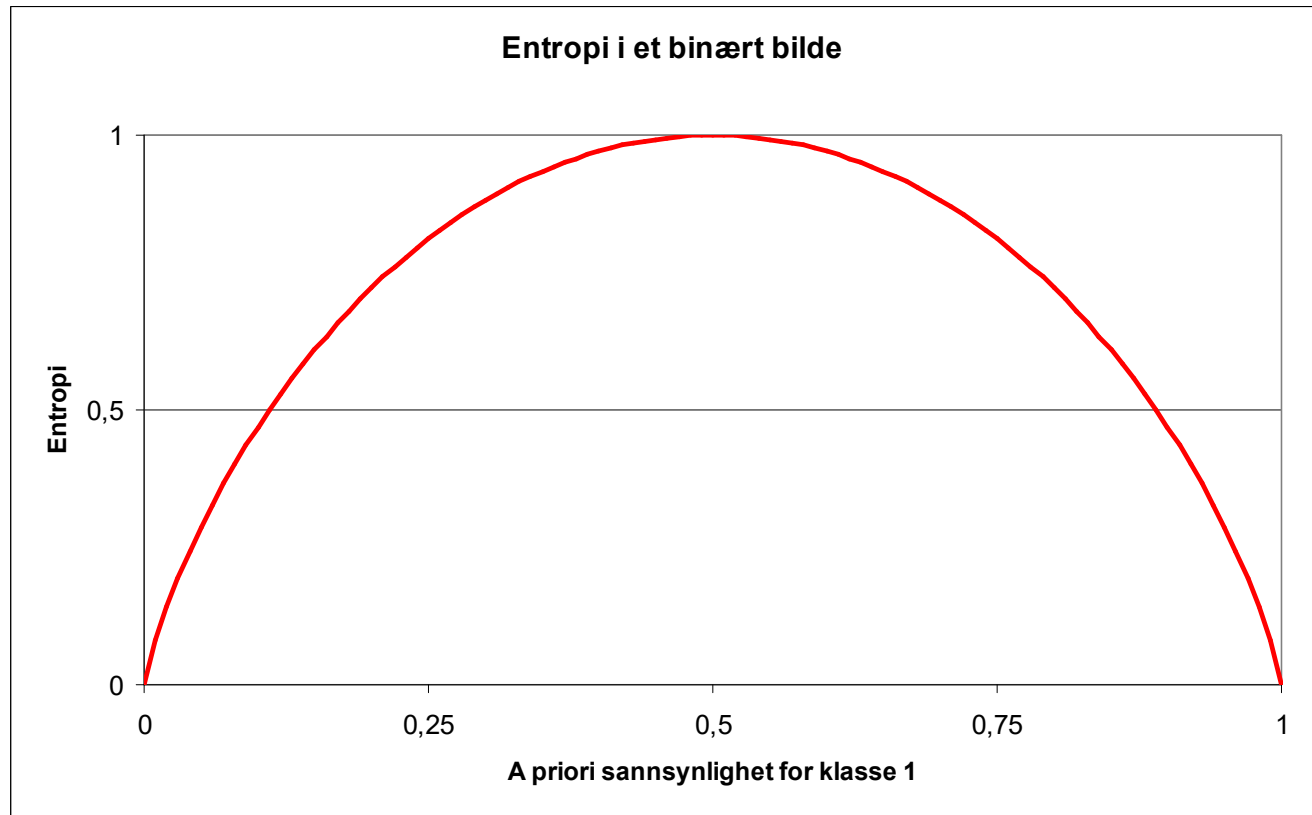
- Anta vi har et $M \times N$ binært bilde.
- Hvis vi skal lagre hver pikselverdi for seg selv, må vi alltid bruke MN biter, men hvor mye informasjon er det i bildet?
- **Like mange 0 som 1 i bildet** (og ingen interpiksel-redundans): Informasjonsinnholdet i hver mulig hendelse er da like stort, derfor er entropien 1 bit.

$$H = \frac{1}{2} \log_2\left(\frac{1}{1/2}\right) + \frac{1}{2} \log_2\left(\frac{1}{1/2}\right) = \frac{1}{2} * 1 + \frac{1}{2} * 1 = 1$$

- **3 ganger så mange 1 som 0 i bildet** (og ingen interpiksel-redundans): Mindre overraskende å få en 1 og det skjer oftere. Entropien er da mindre:

$$H = \frac{1}{4} \log_2\left(\frac{1}{1/4}\right) + \frac{3}{4} \log_2\left(\frac{1}{3/4}\right) = \frac{1}{4} * 2 + \frac{3}{4} * 0,415 = 0,5 + 0,311 = 0,811$$

Entropi i et binært bilde



- Når vi lagrer pikselverdi for pikselverdi vil vi alltid måtte bruke 1 bit per piksel i et binært bilde, selv når entropien er nær 0!
- Kodingsredundansen er null når det er like mange svarte og hvite piksler.

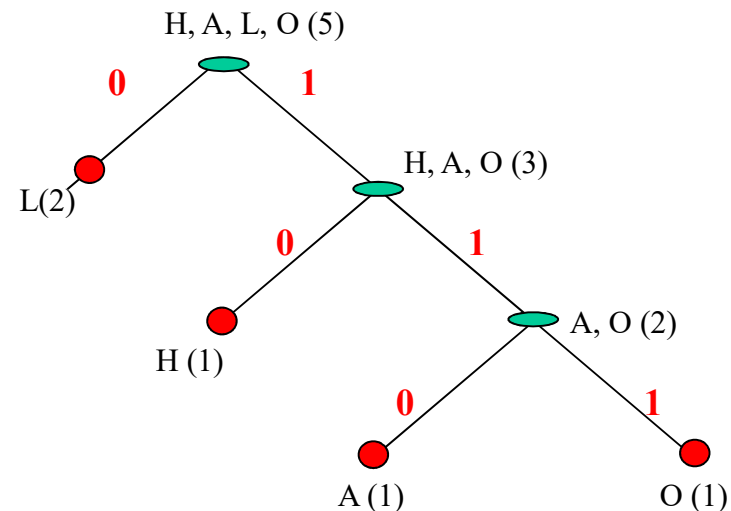
Shannon-Fano-koding

Eksempel: Koding av: HALLO

En enkel metode:

1. Sorter symbolene etter hyppighet, hyppigst til venstre.
2. Del symbolene rekursivt i to grupper som forekommer så like hyppig som mulig.
 - Oppdelingen skal skje ved at symbolene til venstre for en grense blir en subgruppe, mens resten blir en annen subgruppe.
 - Venstre gruppe tilordnes 0, høyre gruppe tilordnes 1.
 - Rekursjonen stopper når hver gruppe inneholder ett symbol.

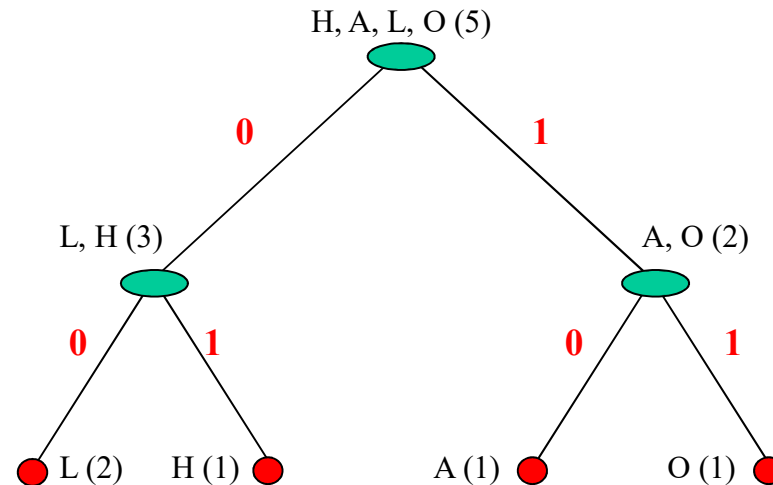
3. Traverser treet fra rot til hvert blad for å finne koden for hvert symbol.



Symbol	Ant.	Kodeord	Lengde	Antall biter
L	2	0	1	2
H	1	10	2	2
A	1	110	3	3
O	1	111	3	3
Totalt antall biter				10

Shannon-Fano-koding

- Oppdeling i to «så like store grupper som mulig» kan gi et annet binærtre for eksempel: HALLO
 - Selv om treet er annerledes, og kodeboken blir forskjellig, så er koden unikt dekodbar.
 - For dette eksempelet er de to løsningene er likeverdige, men slik er det ikke alltid.



- Generelt for Shannon-Fano-koding er gjennomsnittlig antall biter per symbol relatert til entropien:

$$H \leq c \leq H+1$$
- Øvre grense for kodingsredundans: **1 bit per symbol**

Symbol	Ant.	Kodeord	Lengde	Antall biter
L	2	00	2	4
H	1	01	2	2
A	1	10	2	2
O	1	11	2	2
Totalt antall biter				10

Huffman-koding

- Huffman-koding er en algoritme for variabel-lengde koding som er **optimal** under begrensningen at vi **koder symbol for symbol**.
 - Med *optimal* menes her minst mulig kodings-redundans.
- Antar at vi kjenner hyppigheten for hvert symbol.
 - Enten spesifisert som en modell.
 - Huffman-koden er da optimal hvis modellen stemmer.
 - Eller så kan vi bruke symbol-histogrammet til sekvensen.
 - Huffman-koden er da optimal for sekvensen.
 - Ofte bruker vi sannsynlighetene i stedet, men vi kunne likegodt benyttet hyppighetene.

Huffman-koding: Algoritmen

Gitt en sekvens med N symboler:

1. Sorter symbolene etter sannsynlighet, slik at de minst sannsynlige kommer sist.
2. Slå sammen de to minst sannsynlige symbolene til en gruppe, og sorter igjen etter sannsynlighet.
3. Gjenta 2 til det bare er to grupper igjen.
4. Gi koden 0 til den ene gruppen og koden 1 til den andre.
5. Traverser innover i begge gruppene og legg til 0 og 1 bakerst i kodeordet til hver av de to undergruppene.

Eksempel: Huffman-koding

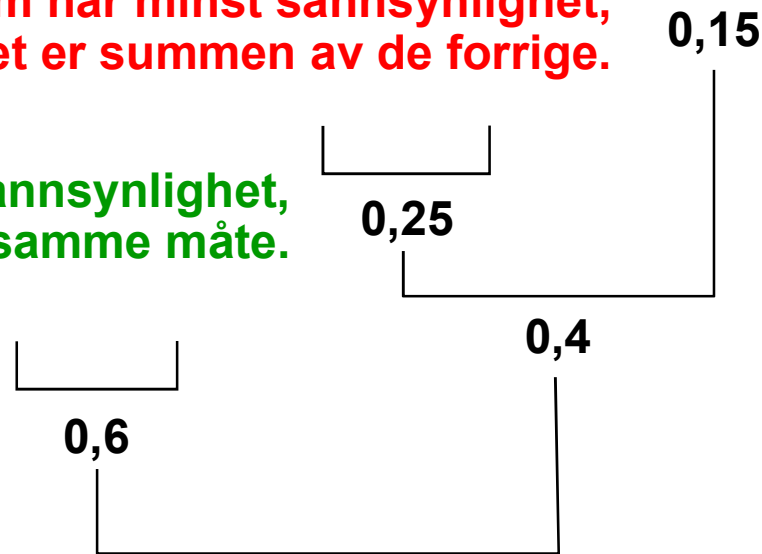
- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

Slå sammen de to gruppene som har minst sannsynlighet, Den nye gruppens sannsynlighet er summen av de forrige.

Finn de to som nå har minst sannsynlighet, og slå dem sammen på samme måte.

Fortsett til det er bare to igjen.



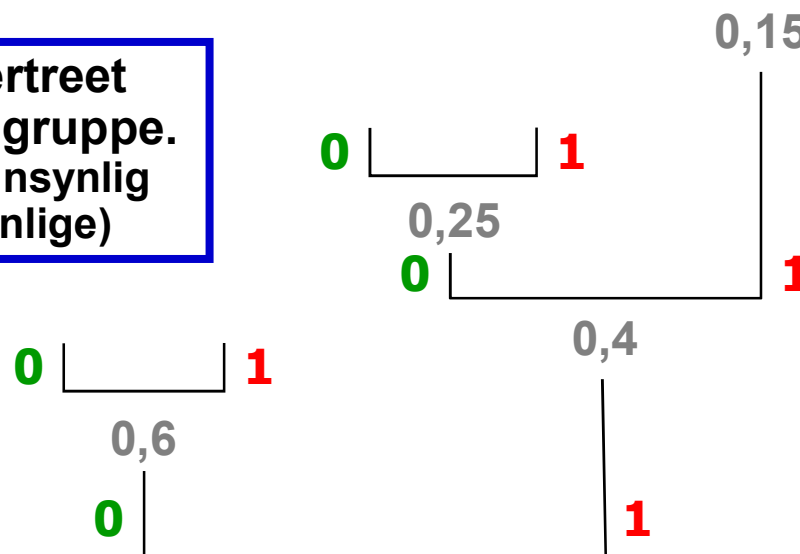
Eksempel: Huffman-koding

- La oss finne Huffman-koden til modellen som består av følgende seks begivenheter med sannsynligheter:

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05

0 1

Gå baklengs gjennom binærtreet og tilordne 0 eller 1 til hver gruppe. (F. eks. kode 0 til den mest sannsynlig og kode 1 til den minst sannsynlige)



Eksempel: Huffman-koding

- Vi får dermed følgende kodebok:

Begivenhet	A	B	C	D	E	F
Huffman-kodeord	00	01	100	101	110	111

- Siden sannsynlighetene er:

Sannsynlighet	0,3	0,3	0,13	0,12	0,1	0,05
---------------	-----	-----	------	------	-----	------

blir gjennomsnittlig antall biter per symbol:

$$c = b_0 p_0 + b_1 p_1 + \dots + b_{G-1} p_{G-1} = \sum_{i=0}^{G-1} b_i p_i = 0,6 * 2 + 0,4 * 3 = 2,4$$

- Entropien H er her litt mindre enn c :

$$H = -\sum_{i=0}^{G-1} p_i \log_2 p_i \approx 2,34$$

Huffman-koding: Kodingsredundans

- Shannon-Fano-koder har en øvre grense for kodingsredundans på 1 bit per symbol.
- Kodingsredundansen til Huffman-koder har samme øvre grense, men har også en tettere grense dersom p_{\max} , sannsynligheten til det hyppigste symbolet, ikke er veldig stor.

$$c - H \leq p_{\max} + \log_2 \left(\frac{2 \log_2 e}{e} \right) \approx p_{\max} + 0,086$$

- Kodingsredundansen til Huffman-koder blir større ettersom p_{\max} nærmer seg 1;
Selv om p_{\max} er mye større enn 0,5 så må vi bruke 1 bit på å kode det tilhørende symbolet!

Generelt om Shannon-Fano- og Huffman-koding

- **Ingen kodeord danner prefiks i en annen kode.**
 - Dette sikrer at en sekvens av kodeord kan dekodes entydig og at man IKKE trenger endemarkører / skilletegn.
 - **Mottatt kodesekvens er unikt og instantant dekodbar.**
 - Dette gjelder også naturlig binærkoding.
- **Hyppige symboler har kortere kodeord enn sjeldne symboler.**
- De to minst sannsynlige symbolene har like lange koder.
 - Siste bit skiller dem fra hverandre.
- **Merk: Kodeboken må overføres!**
 - Kodeboken til et b -biters bilde inneholder opptil **$G=2^b$ kodeord** og det lengste **kodeordet** kan ha **opptil $G-1$ biter**.

Eksempel: Huffman-koding

- La oss Huffman-kode alfabetet som består av de seks mest sannsynlige symbolene i engelsk tekst:

Symbol	^	e	t	a	o	i
Sannsynlighet	0,34	0,19	0,14	0,12	0,11	0,10
Huffman-kodeord	00	10	010	011	110	111
Kodeordlengde	2	2	3	3	3	3
Entropi-bidrag	0,529	0,455	0,397	0,367	0,350	0,332

- Det gjennomsnittlige antall biter per symbol, c , er gitt ved:

$$c = \sum_{i=0}^{G-1} b_i p_i = 2 \cdot 0,53 + 3 \cdot 0,47 = 2,47$$

- Entropien H er igjen litt mindre enn c : $H = -\sum_{i=0}^{G-1} p_i \log_2 p_i \approx 2,43$

- Kodingsredundansen $c-H$ er dermed: $c - H \approx 2,47 - 2,43 = 0,04$

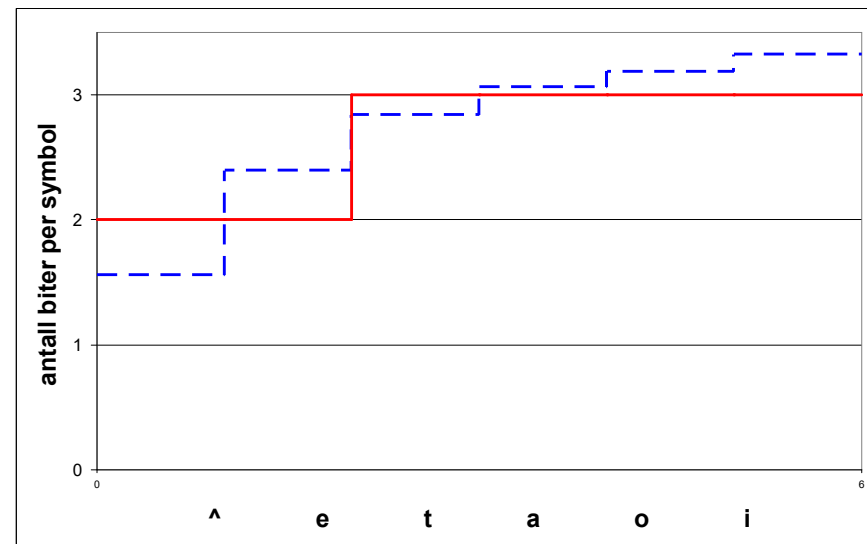
Ideell og faktisk kodeord-lengde

- Hvis gjennomsnittlig antall biter per symbol, c , skal være lik entropien, H , så må:

$$c = \sum_{i=0}^{G-1} b_i p_i = - \sum_{i=0}^{G-1} p_i \log_2 p_i = H$$

- Informasjonsinnholdet $I(s_i)$ i hendelsen s_i angir altså den **ideelle binære kodeordlengden** for symbol s_i : $b_i = I(s_i) = \log_2 \frac{1}{p_i}$

- Plotter den ideelle lengden på kodeordene (vist i blått) sammen med de faktiske kodeordlengden (vist i rødt) for forrige eksempel, får vi:



Når gir Huffman-koding ingen kodingsredundans?

- Den **ideelle binære kodeordlengden** for symbol s_i er:

$$b_i = -\log_2(p_i)$$

- Siden **bare heltalls kodeordlengder er mulig**, er det bare når $p_i = \frac{1}{2^k}$ for et heltall k som dette kan tilfredsstilles.

- Eksempel: Hvis meldingen har sannsynlighetene:

Symbol	s_0	s_1	s_2	s_3	s_4	s_5
Sannsynlighet	0,5	0,25	0,125	0,0625	0,03125	0,03125
Huffman-kodeord	0	10	110	1110	11110	11111

er gjennomsnittlig bitforbruk per symbol etter Huffman-koding:

$$c = 1,9375 = H$$

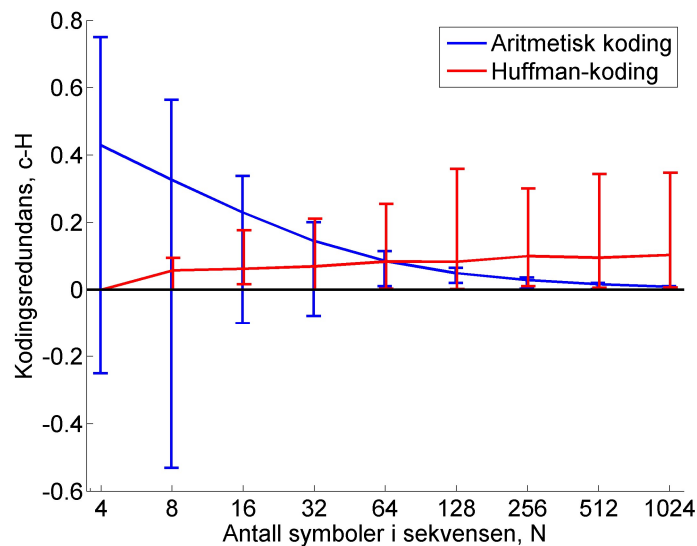
der H er entropien. Altså får vi **ingen kodingsredundans!**

Aritmetisk koding

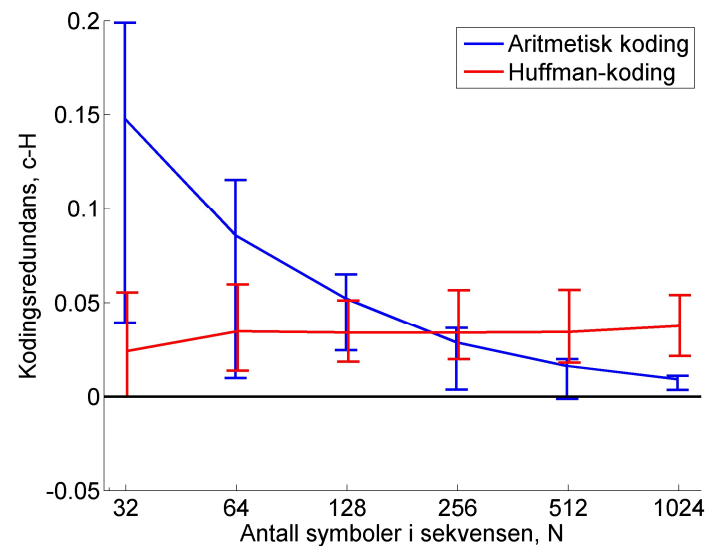
- Alternativ til Huffman-koding, som den deler flere likheter med:
 - Tapsfri kompresjonsmetode.
 - Entropikoder: Koder mer sannsynlige symboler mer kompakt.
 - Bruker sannsynlighetsmodell / histogram av symbolforekomster.
 - Huffman-koding: Sender / bruker kjent kodebok.
 - Aritmetisk koding: Sender / bruker kjent modell/histogram.
- Skiller seg fra Huffman-koding ved at **aritmetisk koding ikke lager kodeord for enkeltsymboler.**
 - I stedet kodes en sekvens av symboler som ett tall D ($0,0 \leq D < 1,0$).
 - Kan oppnå bedre kompresjon enn Huffman-koding.
 - Entropien setter ikke en nedre grense for bitforbruket.
 - Men setter faktisk fortsatt en nedre grense for gjennomsnittet!
- Resulterer i et **bitforbruk per symbol som er nær entropien.**

Aritmetisk koding vs Huffman-koding

1. Definér antall symboler i alfabetet, G , og antall symboler i sekvensen, N .
2. Gjenta 100 ganger:
 - a. Tilfeldig generer sannsynligheten for hvert symbol i alfabetet, p_i
 - b. Tilfeldig generer det oppgitte antall symboler iht. de genererte p_i
 - c. Hvis ikke alle symbolene i alfabetet forekommer i sekvensen, gå til 1.
 - d. Beregn kodingsredundansen etter aritmetisk koding og etter Huffman-koding.
3. Beregn maksimum, gjennomsnittlig og minimum kodingsredundans.



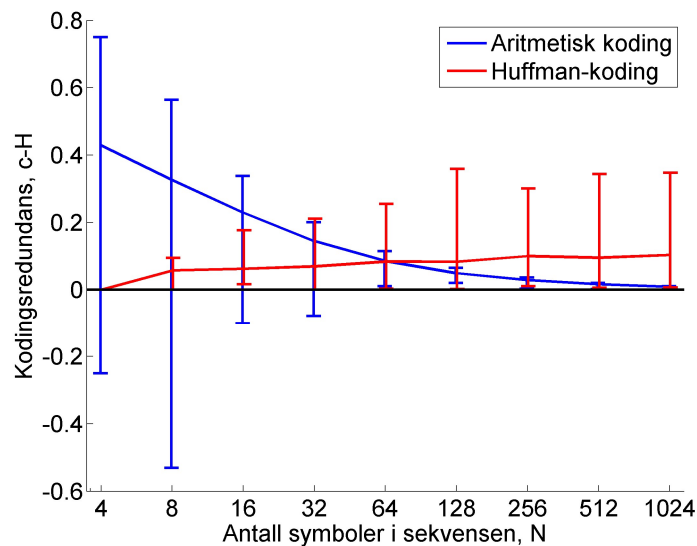
Med $G=4$ symboler i alfabetet.



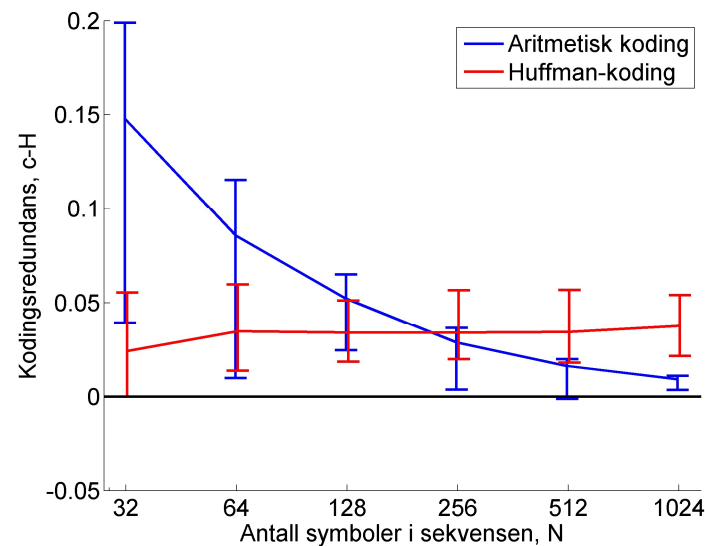
Med $G=20$ symboler i alfabetet.

Aritmetisk koding vs Huffman-koding

- Aritmetisk koding: Bedre kompresjon jo lenger symbolsekvensen er.
- Huffman-koding: Bedre kompresjon jo flere symboler i alfabetet.
- Aritmetisk koding komprimerer typisk litt bedre enn Huffman-koding, men er mer regnekrevende å utføre.
 - For vanlige bilder, dvs. med relativt få symboler i alfabetet og (potensielt sett) mange symboler i sekvensen.



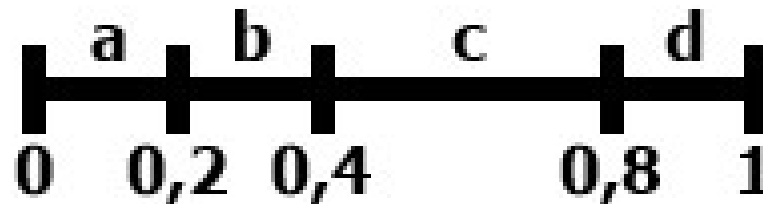
Med $G=4$ symboler i alfabetet.



Med $G=20$ symboler i alfabetet.

Aritmetisk koding: Grunntanke

- Symbolsannsynlighetene summerer seg til 1.
- Dermed definerer de en **oppdeling av intervallet [0; 1)**.
 - Hvert delintervall representerer ett symbol.



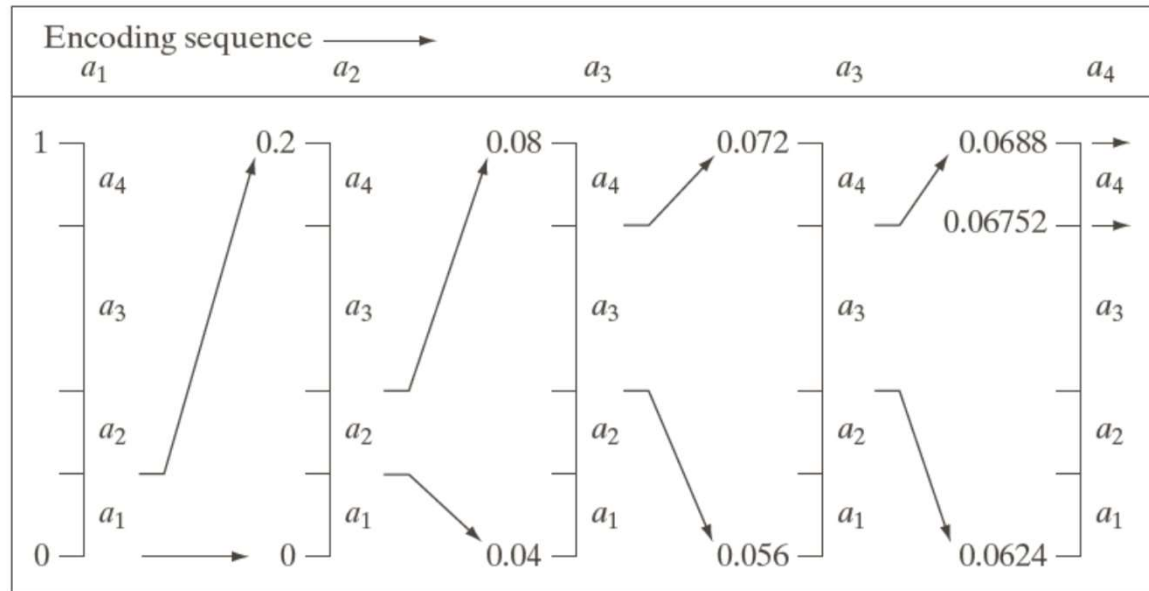
- Har vi to symboler etter hverandre, kan vi **oppdele intervallet som representerer det første symbolet**.
 - Hvert delintervall representerer symbolparet; det første symbolet etterfulgt av ett symbol.



- Tilsvarende for flere symboler etter hverandre.
- Resultat: Et halvåpent delintervall av [0; 1).
- Finner så en bitsekvens som representerer intervallet.

Eksempel: Aritmetisk koding

- Sannsynlighetsmodell: $P(a_1)=P(a_2)=P(a_4)=0,2$ og $P(a_3)=0,4$
- Melding/symbolsekvens: $a_1a_2a_3a_3a_4$



- a_1 ligger i intervallet $[0; 0,2)$
- a_1a_2 ligger i intervallet $[0,04; 0,08)$
- $a_1a_2a_3$ ligger i intervallet $[0,056; 0,072)$
- $a_1a_2a_3a_3$ ligger i intervallet $[0,0624; 0,0688)$
- $a_1a_2a_3a_3a_4$ ligger i intervallet $[0,06752; 0,0688)$

Aritmetisk koding: Algoritmen

1. La «current interval» = $[0; 1)$.
2. For hvert symbol a_i i sekvensen (fra venstre):
 - a) Del opp «current interval» i delintervaller, der størrelsen på hvert delintervall er proporsjonalt med sannsynligheten for det tilhørende symbolet.
 - Proporsjonalitetsfaktoren er størrelsen av «current interval».
 - b) Velg det delintervallet som svarer til a_i , og la «current interval» være dette delintervallet.
3. Representér «current interval» med en kortest mulig bitsekvens.

Aritmetrisk koding i praksis

- Alfabetet inneholder normalt **END-OF-DATA (EOD)**.
 - Dette symbolet må også få en sannsynlighet i modellen.
 - Alternativt kan lengden av symbolsekvensen defineres, enten predefinert eller spesifisert.
- I praksis oppdeles ikke «current interval»,
kun delintervallet til det trufne symbolet beregnes:
 - begynnelsen av nytt c.i. =
begynnelsen av gammelt c.i. +
(bredden av gammelt c.i. *
begynnelsen av symbolets intervall i modellen)
 - slutten av nytt c.i. =
begynnelsen av gammelt c.i. +
(bredden av gammelt c.i. *
slutten av symbolets intervall i modellen)
- Vi beregner altså kun ett intervall for hvert symbol.

AK: Kodingseksempel

- Modell: Alfabet [a; b; c] med sannsynligheter [0,6; 0,2; 0,2].
- Hvilket delintervall av [0; 1) vil entydig representere meldingen **acaba** ?
- **a** ligger i intervallet [0; 0,6).
 - «Current interval» har nå en bredde på 0.6.
- **ac** ligger i intervallet $[0+0,6*0,8; 0+0,6*1) = [0,48; 0,6)$.
 - Intervallbredden er nå 0,12 (= produktet $0,6*0,2$).
- **aca** ligger i intervallet $[0,48+0,12*0; 0,48+0,12*0,6) = [0,48; 0,552)$.
 - Intervallbredden er 0,072 (= produktet $0,6*0,2*0,6$).
- **acab** er i $[0,48+0,072*0,6; 0,48+0,072*0,8) = [0,5232; 0,5376)$.
 - Intervallbredden er 0,0144 (= produktet $0,6*0,2*0,6*0,2$).
- **acaba** er i $[0,5232+0,0144*0; 0,5232+0,0144*0,6) = [0,5232; 0,53184)$.
 - Intervallbredden er nå 0,00864 (= produktet $0,6*0,2*0,6*0,2*0,6$).
- Et tall i dette intervallet, f.eks. 0,53125, vil entydig representere **acaba**, forutsatt at mottakeren har den samme modellen og vet når å stoppe.

AK: Dekodingseksempel

- **Anta at vi skal dekode tallet 0,53125.**
- Samme modell: Alfabet [a, b, c] med sannsynligheter [0,6; 0,2; 0,2].
- La $[0; 1)$ være «current interval».
- Del intervallet iht. modellen; a er $[0; 0,6)$, b er $[0,6; 0,8)$ og c er $[0,8; 1)$.
- Tall 0,53125 ligger i delintervallet for a, $[0; 0,6)$: => **Første symbol: a**
- $[0; 0,6)$ er nå «current interval». Del dette opp i henhold til modellen:
 - delintervallet for a er $[0; 0,36)$ *bredden er 60% av $[0; 0,6)$*
 - delintervallet for b er $[0,36; 0,48)$ *bredden er 20% av $[0; 0,6)$*
 - delintervallet for c er $[0,48; 0,6)$ *bredden er 20% av $[0; 0,6)$*
- Tall 0,53125 ligger i delintervallet $[0,48; 0,6)$: => **Andre symbol: c**
- $[0,48; 0,6)$ er nå «current interval». Del dette opp i henhold til modellen:
 - delintervallet for a er $[0,48; 0,552)$
 - delintervallet for b er $[0,552; 0,576)$
 - delintervallet for c er $[0,576; 0,6)$
- Tall 0,53125 ligger i delintervallet $[0,48; 0,552)$: => **Tredje symbol: a**

AK: Dekodingseksempel

- **Anta at vi skal dekode tallet 0,53125.**
- Samme modell: Alfabet [a; b; c] med sannsynligheter [0,6; 0,2; 0,2].
- [0,48; 0,552] er nå «current interval». Del dette opp iht. modellen:
 - delintervallet for a er [0,48; 0,5232)
 - delintervallet for b er [0,5232; 0,5376)
 - delintervallet for c er [0,5376; 0,552)
- Tall 0,53125 ligger i delintervallet [0,5232; 0,5376): => **4. symbol: b**
- Vi deler opp intervallet [0,5232; 0,5376) for å finne neste symbol:
 - delintervallet for a er [0,5232; 0,53184)
 - delintervallet for b er [0,53184; 0,53472)
 - delintervallet for c er [0,53472; 0,5376)
- Tall 0,53125 ligger i delintervallet [0,5232; 0,53184): => **5. symbol: a**
- **Vi kunne fortsatt å "dekodet" symboler.**
- For å vite at vi skal stoppe her trenger vi:
 - Enten et EOD-symbol i modellen; stopp når vi dekker dette.
 - Eller vite hvor mange symboler vi skal dekode; stopp når vi har dekodet det antallet.

Desimaltall som bitsekvens

- Vi lagrer/sender ikke desimaltall, men en sekvens med biter.
- Spørsmålet er: Hvordan kan vi representere intervallet ved bruk av færre mulig biter?
- Først må vi å se hvordan vi representerer desimaltall binært:
- Et desimaltall D i intervallet $[0; 1)$ kan skrives som en veiet sum av negative toerpotenser:

$$D = d_1 2^{-1} + d_2 2^{-2} + d_3 2^{-3} + \dots + d_n 2^{-n} + \dots$$

der hver d_i er enten 0 eller 1.

- Rekken av koeffisienter $d_1 d_2 d_3 d_4 \dots$ er bitsekvensen som representerer desimaltallet D .
- Vi skriver at: $D = 0, d_1 d_2 d_3 d_4 \dots_2$ der $_2$ indikerer at tallet er skrevet i totallsystemet.

Desimaltall som bitsekvens

- **Situasjon:** Vi har et desimaltall i $[0; 1)$ som vi ønsker å skrive i totallsystemet.
- **Løsning:** Suksessiv multiplikasjon med 2:
 1. Multipliser begge sider av følgende likning med 2:
$$D = d_1 2^{-1} + d_2 2^{-2} + d_3 2^{-3} + \dots + d_n 2^{-n} + \dots$$
Heltallsdelen av resultatet er da lik c_1 fordi:
$$2D = d_1 + Q, \text{ der } Q = d_2 2^{-1} + d_3 2^{-2} + \dots + d_n 2^{-(n-1)} + \dots$$
Hvis resten Q er 0, så er vi ferdige.
 2. Multipliser resten Q med 2.
 - Heltallsdelen av produktet er neste bit og Q oppdateres til å være den nye resten.
 3. Hvis resten Q er 0, så er vi ferdige. Ellers går vi til 2.

Representasjon av intervall

- Spørsmålet var: Hvordan kan vi representere intervallet ved bruk av færre mulig biter?
- Eksempel: Intervallet er $[0,5232, 0,53184)$.
 - $0,53125_{10}$ er (som sagt) et desimaltall i dette intervallet, og er faktisk det desimaltallet i intervallet med kortest binær-representasjon.
 - Hva er bitsekvensen som representerer dette desimaltallet?
 - **$0,53125_{10} = 0,10001_2$** siden:

$2 * 0,53125 = 1,0625$	$\Rightarrow d_1 = \mathbf{1}$, rest = 0,0625
$2 * 0,0625 = 0,125$	$\Rightarrow d_2 = \mathbf{0}$, rest = 0,125
$2 * 0,125 = 0,25$	$\Rightarrow d_3 = \mathbf{0}$, rest = 0,25
$2 * 0,25 = 0,50$	$\Rightarrow d_4 = \mathbf{0}$, rest = 0,5
$2 * 0,5 = 1,0$	$\Rightarrow d_5 = \mathbf{1}$, rest = 0
 - Vi trenger altså bare 5 biter for å kode (et tall i) intervallet!
- Men hvordan kan vi finne hvilket desimaltall vi skal bruke slik at binær-representasjoner blir kortest mulig?

Eksempel: Representasjon av intervall

Finn kortest mulig $d=0,d_1d_2d_3\dots_2$ innenfor intervallet $[0,6; 0,7)$.

- Hvis $n \geq k$ så er:

$$2^{-k+1} = 2^{-(k-1)} > d_k 2^{-k} + \dots + d_n 2^{-n}$$

siden d_i er 0 eller 1.

- Derfor er:

$$D = 0, \mathbf{1} \dots_2 \quad \Rightarrow 0,5 \leq D < 1$$

$$D = 0, \mathbf{10} \dots_2 \quad \Rightarrow 0,5 \leq D < 0,75$$

$$D = 0, \mathbf{100} \dots_2 \quad \Rightarrow 0,5 \leq D < 0,625$$

$$D = 0, \mathbf{101} \dots_2 \quad \Rightarrow 0,625 \leq D < 0,75$$

- \Rightarrow Intervall kan kodes ved det binære kommatallet $0, \mathbf{101}_2$ (ekvivalent med $0,625_{10}$), altså med bare 3 biter.
 - Hvis vi vil kreve at øvre og nedre grense er innenfor intervallet; kan intervallet kodes ved $0,1010_2$ fordi:
 $D = 0, \mathbf{1010} \dots_2 \Rightarrow 0,625 \leq D < 0,6875$

AK: Problemer og løsninger

- Den stadige krympingen av «current interval» krever flyttall med **stadig økende presisjon** og eksakt aritmetikk.
 - Lengre symbolsekvenser krever bedre presisjon.
- Kompresjonsmetoden gir **ingen output** før hele sekvensen er behandlet.
- Løsning: **Send/lagre den mest signifikante biten** når entydig kjent, og doble lengden av «c.i.».
 - Bitsekvensen blir (i teorien) lik som før.
 - Kan i praksis øke presisjonen, men løser ikke problemet.
- Det finnes flere praktiske AK-implementasjoner.
 - Alle er ganske regnetunge.
 - Vi trenger uansett presis/nøyaktig flyttallsaritmetikk!
 - De aller fleste er belagt med patenter.

Oppsummering

- Vi komprimerer for å redusere antall bits.
- Ved kompresjon fjernes/reduseres redundans:
 - Psykvisuell -, interbilde-, intersampel-, koding-redundans.
 - Kun når vi fjerner irrelevant informasjon for anvendelsen (f.eks. psykvisuell redundans) er kompresjonen ikke-tapsfri.
- Kompresjon kan deles inn i tre steg:
 1. Transform.
 2. Kvantifisering. Hvis brukt, så blir kompresjonen ikke-tapsfri!
 3. Koding, f.eks. Shannon-Fano-, Huffman- eller aritmetisk koding.
 - Huffman-koding oppnår minst mulig kodingsredundans under antagelsen om at vi koder symbol for symbol, men aritmetisk koding kan ofte være litt bedre for vanlige bilder.