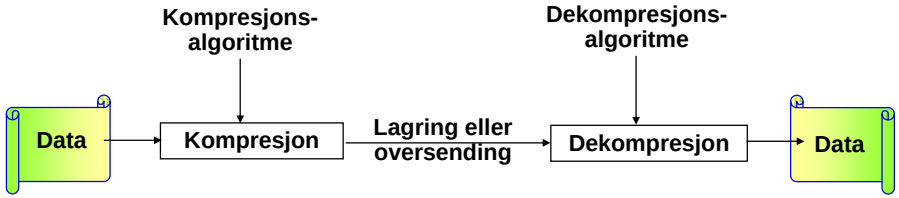
INF 1040
Kompresjon og koding

- Tema i dag :
  1. Noen begreper
  2. Redundans
  3. Differanse- og løpelengdetransformer
  4. Gray kode
  5. Entropi
  6. Shannon-Fano og Huffman koding
  7. Lempel-Ziv koding
  8. JPEG koding
- Pensumlitteratur: Læreboka, kapittel 18.
- Neste gang: Kryptering og steganografi.

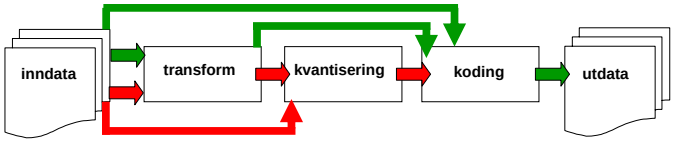
Noen begreper



- Kompresjon består i å pakke informasjonsinnholdet i dataene (tekst, bilde, lydsignaler etc.) på en så kompakt måte at redundant informasjon ikke lagres.
- Dataene komprimeres, deretter lagres de.
- Når de senere skal leses, må vi dekomprimere dem.
- Koding er en del av kompresjon, men vi koder for å lagre effektivt, ikke for å hemmeligholde eller skjule informasjon.

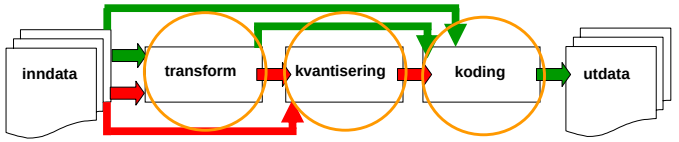
Kompresjon

- Kompresjon kan deles inn i tre steg:
  - Transform - representer dataene mer kompakt.
  - Kvantisering - avrunding av representasjonen.
  - Koding - produksjon og bruk av kodebok.



- Kompresjon kan gjøres
  - Eksakt / tapsfri ("loss-less") – følg de grønne pilene
    - Her kan vi rekonstruere den originale meldingen eksakt.
  - Ikke-tapsfri ("lossy") – følg de røde pilene
    - Her kan vi ikke rekonstruere meldingen eksakt.
    - Resultatet kan likevel være "godt nok".
- Det finnes en mengde ulike metoder for begge kategorier kompresjon.

De tre stegene i kompresjon



- Mange kompresjons-metoder er basert på å representere dataene på en annen måte, altså transformer av original-dataene.
  - Differansetransform
  - løpelengder/run-length,
- Hvis vi kvantiserer original - dataene, så kan ikke dette reverseres.
- Koding bygger ofte på sannsynlighetsfordelinger,
  - Estimert ved normaliserte histogrammer.
- Transformer og koding er alltid reversible.
- Kvantisering gir alltid et tap av presisjon.

Anvendelser

- Kompresjon og koding benyttes for å redusere antall biter som skal til for å beskrive bildet (eller en god approksimasjon til bildet).
- Anvendelser innen data-lagring og data-overføring
  - Televideo-konferanser
  - Fjernanalyse / meteorologi
  - Overvåking / fjernkontroll
  - Telemedisin / medisinske arkiver (PACS)
  - Dokumenthåndtering / FAX
  - Multimedia / nettverkskommunikasjon
  - Mobil kommunikasjon
  - MP3-spillere, DAB-radio, digitalkameraer, ...
  - ....
- Tidsforbruket ved off-line kompresjon er ikke særlig viktig.
- Dekompresjons-tiden er langt viktigere.
- Ved sanntids data-overføring er tidsforbruket kritisk.

Plass og tid

- Digitale data kan ta stor plass
  - Spesielt lyd, bilder og video
- Eksemper :
  1. Digitalt bilde: 512 · 512 · 8 biter · 3 farger = 6 291 456 biter
  2. Røntgenbilde: 7112 · 8636 · 12 biter pr. piksel = 737 030 784 biter
- Overføring av data tar tid:

Linje med 64 kbit/s:
1. ca. 1 min. 38 s.
2. ca. 3 timer 12 min.

Linje med 1 Mbit/s:
1. ca. 6 s.
2. ca. 12 min.

Overføringskapasitet og bps

- Filstørrelser er oftest gitt i binære enheter, gjerne i potenser av 1 024:
  - 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)
- Overføringshastigheter og linjekapasitet angis alltid i titallsystemet, oftest som antall biter per sekund:
  - 1 kb/s = 1000 b/s = 10^3 biter per sekund.
  - 1 Mb/s = 1000 kb/s = 10^6 biter per sekund.
  - 1 Gb/s = 1000 Mb/s = 10^9 biter per sekund.
- Kapasitet for noen typer linjer:
  - GSM-telefonlinje: 9.6 kb/s
  - Analogt modem: f.eks. 56 kb/s
  - ADSL (Asymmetric Digital Subscriber Line): 1- 2 Mb/s

Melding, data og informasjon

- Vi skiller mellom presentasjon, representasjon og informasjon:
- Melding: teksten, bildet eller lydsignalet som presenteres.
- Data: strømmen av biter som lagres eller sendes (representasjon).
- Informasjon: Et matematisk begrep som kvantifiserer mengden overraskelse/uventethet i en melding.
  - Et signal som varierer har mer informasjon enn et monotont signal.
  - I et bilde: kanter rundt objekter har høyest informasjonsinnhold,
    - spesielt kanter med mye krumning.

Redundans

- Vi kan bruke ulike mengder data til å lagre/overføre samme melding.
  - Et bilde av et 5-tall (50 · 50 piksler a 8 biter = 20 000 biter)
  - Teksten ”fem” i 8 biters ISO 8859-1 (24 biter)
  - ISO 8859-1 tegnet ”5” (8 biter)
  - Et binært heltall ”1 0 1” (3 biter)
- Begrepet redundans sier noe om hvor stor del av datamengden vi kan fjerne uten at vi mister informasjonen i dataene.
- Eks: vi skal lagre tallet 0, men hvordan lagres det faktisk:
  - Binært i en byte: 00000000 - 8 biter med 0
  - Standard 7 bits ASCII kode for 0
  - Vet vi at vi bare skal lagre tallet 0, trenger vi bare 1 bit.
  - Ved smart komprimering og dekomprimering kan vi fjerne redundante bits.

Kompresjon av bilder

- Hvorfor behandles kompresjon av bilder spesielt?
  - Det er generelt en mengde redundant informasjon i bilder
    - mellom piksler på samme linje
    - mellom ulike linjer i bildet
    - mellom bildene i en sekvens
  - Bilder inneholder gjerne objekter
    - Objekter kan ofte representeres mer kompakt.
  - Vi kan også ta i betraktning psykovisuelle effekter:
    - Vi lagrer ikke det vi likevel ikke kan oppfatte

Ulike typer redundans

- Psykovisuell/psykoakustisk redundans
  - Det finnes informasjon vi ikke kan høre eller se (se kapittel 10, 11 og 16).
- Interbilde redundans
  - Det er en viss likhet mellom bilder som kommer etter hverandre i en sekvens
  - Vi koder noen bilder i sekvensen, og deretter bare differanser (kapittel 16).
- Intersampel redundans
  - Nabo-symboler, - amplituder, -piksler ligner på hverandre eller er like.
  - Eks: 00001116611 kan ”run-length” kodes som (0,4),(1,3),(6,2),(1,2)
- Kodings-redundans
  - Gjennomsnittlig kodelengde minus et teoretisk minimum.
  - Eks: 5535552635535 - Huffman-koding bruker færrest biter på å kode 5-tallet, som forekommer ofte, og flere biter for 6-tallet. Hvor godt er dette?

Kompresjonsrate og redundans

- Kompresjonsrate (”Compression Ratio”):

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

i **er antall biter pr. sampel originalt,**
c **er antall biter pr. sampel i det komprimerte bildet.**

- Relativ Redundans :

$$RR = 1 - \frac{1}{CR} = 1 - \frac{c}{i}$$

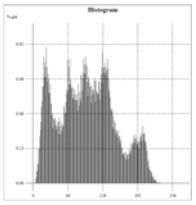
- ”Percentage Removed” :

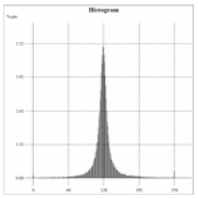
$$PR = 100\left(1 - \frac{c}{i}\right)\%$$

Differansetransform

- Gitt en linje i bildet med gråtoner
  $f_1,.....f_N$, $0 \le f_i \le 2^b-1$.
- Transformer (reversibelt) til

  $g_1 = f_1$, $g_2 = f_2-f_1$, $g_3 = f_3-f_2$, ... , $g_N = f_N-f_{N-1}$
- Vi trenger nå b+1 biter hvis vi skal tilordne
  like lange binære koder til alle mulig verdier.
- i differansehistogrammet
  vil de fleste verdiene samle seg rundt 0.
- Derfor er det ikke slik
  at en naturlig bit-koding
  av differansene er det optimale.



Litt mer om differansetransform

- Lag f.eks. en 16 ords naturlig kode
  - c_1=0000, c_2=0001,.... c_{16}=1111
  - tilordne de 14 kode-ordene i midten:
    c_2,.... c_{15} til differansene -7, -6, ..., -1, 0, 1, 2, ..., 5, 6
- Kodene c_1 og c_{16} kan brukes til å indikere om differansen
  $\Delta x < -7$ eller om $\Delta x \ge 7$ (to-sidet shift-kode)
  - $\Delta x = 21 \Rightarrow c_{16}c_{16}c_2$ $\Delta x = -22 \Rightarrow c_1c_1c_{15}$

…	-22	-21	…	-8	-7	…	0	…	6	7	…	20	21	…
…	$c_1c_1c_{15}$	c_1c_2	…	c_1c_{15}	c_2	…	c_9	…	c_{15}	$c_{16}c_2$	…	$c_{16}c_{15}$	$c_{16}c_{16}c_2$	…

- Her øker kodeordets lengde trinnvis med 4 biter.
  - Neppe helt optimalt i forhold til differansehistogrammet.
- Alternativt kan vi kode differansene ved hjelp av andre kodeteknikker.

Løpelengde-transform (run-length)

- Ofte inneholder bildet objekter med lignende gråtoner,
  f.eks. svarte bokstaver på hvit bakgrunn.
- Vi kan benytte oss av kompresjonsteknikker
  som tar hensyn til at nabopiksler på samme linje ofte er like.
- Løpelengde-transform er reversibel.
- Først et eksempel:

  333333555555555544777777 (24 byte)
- Når tallet 3 forekommer 6 ganger etter hverandre,
  trenger vi bare lagre tallparet (3,6).
- Tilsammen trenger vi her 4 tallpar, (3,6), (5,10), (4,2), (7,6)
  altså 8 tall til å lagre hele sekvensen 24 tall ovenfor.
- Hvor mange biter vi bruker pr. tall, avhenger av den videre kodingen.

Løpelengder i binære bilder

- I to-nivå bilder trenger vi bare å angi løpelengden for hvert ”run”, forutsatt at vi vet om linjen starter med et hvitt eller et svart run.
- Vi trenger kodeord for EOL og EOI.
- Run-lengde histogrammet er ofte ikke flatt.
  - Benytter da en kode som gir
    korte kode-ord til de hyppigste run-lengdene.
- En standard (CCITT) Huffman kode basert på dokument-statistikk brukes for dokument-overføring pr fax.
- Egen kodebok for svarte og hvite runs.
- Mer effektiv dersom kompleksiteten er lav.

Koding

- Et *alfabet* er mengden av alle mulige symboler (eks. gråtoner)
- Hvert symbol får et *kode-ord*, til sammen er de en *kode-bok*.
- Koding skal være reversibel
  - fra koden skal vi kunne rekonstruere det originale symbolet
- Unikt dekodbare koder har den egenskap at en mottatt sekvens av kode-ord kan dekodes på en og bare en måte.
- Instantant dekodbare koder kan dekodes uten skilletegn.

Naturlig binær-koding

- Naturlig binær-koding:
  - Alle kode-ord er like lange.
  - Kjenner vi noen eksempler på dette?
- Eks: vi har 8 mulige verdier

Symbol nr.	1	2	3	4	5	6	7	8
Symbol	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
Kode c_i	000	001	010	011	100	101	110	111

- Naturlig binærkoding er bare optimal hvis alle verdiene i sekvensen er like sannsynlige.

"Gray code"

Er den konvensjonelle binære representasjonen av gråtoner optimal?

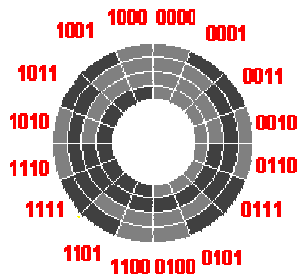
- La oss se på et ett-bånds gråtone-bilde med b biter
  - Kan sees som et b-bånds binært bilde
  - Vi sier at bildet har b bit-plan.
- Ønskelig med minst mulig kompleksitet i hvert bit-plan
  - Da blir løpelengde-transformasjonen mest effektiv.
- Konvensjonell binær representasjon gir høy bit-plan kompleksitet.
  - Hvis gråtoneverdien fluktuerer mellom 2^k-1 og 2^k vil k+1 biter skifte verdi: eksempel: 127 = 01111111 mens 128 = 10000000
- I "Gray Code" skifter alltid bare en bit når gråtonen endres med 1.

- Overgangen fra binær kode til "gray code" er en transformasjon, men både naturlig binær kode og "gray code" er selvsagt koder.

"Binary reflected gray code"

- 4 biter Gray kode og binær

Gray	Binær	Desimal
0000_g	0000_b	0_d
0001	0001	1
0011	0010	2
0010	0011	3
0110	0100	4
0111	0101	5
0101	0110	6
0100	0111	7
1100	1000	8
1101	1001	9
1111	1010	10
1110	1011	11
1010	1100	12
1011	1101	13
1001	1110	14
1000	1111	15



"Gray code shaft encoder"

Gir sikker avlesing av vinkel.

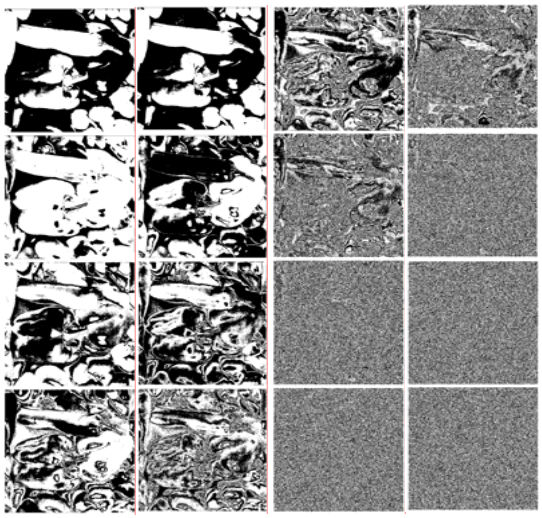
Emilie Baudot's telegrafskive 1878.

Koden patentert av Gray i 1953.

Brukes i styring av robot-armer etc.

Gray kode i gråtonebilder

- MSB er likt i de to representasjonene.
- Større homogene områder i hvert bitplan i Gray-kode enn i naturlig binærkode.
- Flere bitplan med støy i vanlig binærkode.
- Det er en gevinst i løpelengdekoding av bitplan i Gray kode.



Informasjonsteori og koding

- Kompresjon/koding bygger på informasjonsteori og sannsynligheter.
- Hvis et symbol forekommer ofte, bør vi lagre det med et lite antall biter for å bruke minst mulig lagerplass til sammen.
- Hvis et symbol forekommer sjeldent, kan vi tillate oss å bruke mange biter på å lagre det.
- Vi vil bruke et variabelt antall biter per symbol.
  - Dette skiller seg fra lagring med like mange biter per symbol.

Koder med variabel lengde

- For symboler med ulike sannsynligheter er koder med variabel lengde på kode-ordene bedre enn like lange koder
  - Hyppige symboler ⇨ kortere kode-ord.
  - Sjeldne symboler ⇨ lengere kode-ord.
  - Dette var forretnings-ideen til Samuel Morse

De vanligste symbolene i engelsk tekst er

e, t, a, n, o, i,...

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	-.--

Entropi – en liten forsmak

- Entropi er et matematisk mål på gjennomsnittlig informasjonsmengde i en sekvens av tegn eller tall.
- *Har vi en sekvens av N tegn som lagres med b biter per sampel, så kan vi si vi har et alfabet med 2^b mulige symboler.*
- Vi er interessert i gjennomsnittlig informasjon pr. symbol.
- Intuitivt vil en mindre sannsynlig hendelse gi mer informasjon enn en mer sannsynlig hendelse.

Histogrammet til en sekvens av symboler

- 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.

Gjennomsnittlig antall biter pr. symbol

- Vi konstruerer en rekke kodeord $c_1,....c_N$
  slik at symbol s_i kodes med kodeordet c_i.
- b_i er lengden (angitt i biter) av kodeordet c_i.
- Gjennomsnittlig antall biter pr. symbol for denne koden er:

$$R = b_1 p_1 + b_2 p_2 + ... + b_N p_N = \sum_{i=1}^{N} b_i p_i$$

- Entropien H gir oss en nedre grense
  for hvor mange biter vi gjennomsnittlig trenger per symbol
  (hvis vi bare koder ett symbol av gangen).

Informasjonsinnhold

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

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

- $\log_2(x)$ er 2-er logaritmen til x
  - Hvis $\log_2(x)=b$ så er $x=2^b$
  - Eks: $\log_2(64) = 6$ fordi $64 = 2^6 = 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$

    $\log_2(8) = 3$ fordi $8 = 2 \cdot 2 \cdot 2 = 2^3$
  - Har ikke $\log_2$ på kalkulatoren, hjelp! $\log_2(\text{tall}) = \log_{10}(\text{tall}) / \log_{10}(2)$
    (se nederst på side 2, Oblig 2 !)
- $\log_2(1/p(s_i))$ gir oss informasjonsinnholdet
  i den hendelsen det er at symbolet s_i forekommer en gang,
  uttrykt i biter.

Entropi

- Hvis vi tar gjennomsnittet over alle symbolene s_i i alfabetet,
  får vi gjennomsnittlig informasjon pr. symbol.
- Entropi H:

$$H = \sum_{i=0}^{2^b-1} p(s_i) I(s_i) = -\sum_{i=0}^{2^b-1} p(s_i) \log_2(p(s_i))$$

- Entropien setter en nedre grense for hvor kompakt
  sekvensen kan representeres
  - Dette gjelder hvis vi bare koder hvert symbol for seg.

Øvre og nedre grense for entropi

- Hvis alle symboler like sannsynlige => entropi lik antall biter.
  - Det er 2^b symboler
  - sannsynligheten for hvert av dem er $p(s_i) = 1/2^b$.
  - Da blir entropien

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

  - Husk at: Hvis det er 2^b symboler som alle er like sannsynlige, så kan vi ikke representere dem mer kompakt enn med b biter per symbol.
- 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.

$$H = -\log_2(1) = 0$$

To eksempler

- Et binært bilde med N · M piksler inneholder 1 bit per piksel.
- Det er N · M biter data i bildet, men hvor mye informasjon er det i bildet?
- Hvis det er like mange 0 som 1 i bildet, så er det like stor sannsynlighet for at neste piksel er 0 som 1. Informasjonsinnholdet i hver mulig hendelse er da like stort, og entropien til et nytt symbol er 1 bit.

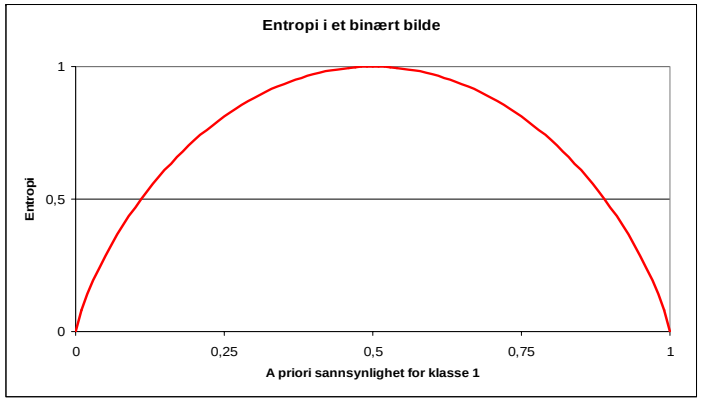
$$H = \frac{1}{2}\log_2(\frac{1}{1/2}) + \frac{1}{2}\log_2(\frac{1}{1/2}) = \frac{1}{2}\cdot 1 + \frac{1}{2}\cdot 1 = 1$$

- Hvis det er 3 ganger så mange 1 som 0 i bildet, så er det mindre overraskende å få en 1, og det skjer oftere. Entropien er da mindre:

$$H = \frac{1}{4}\log_2(\frac{1}{1/4}) + \frac{3}{4}\log_2(\frac{1}{3/4}) = \frac{1}{4}\cdot 2 + \frac{3}{4}\cdot 0.415 = 0.5 + 0.311 = 0.811$$

For de matte-interesserte !

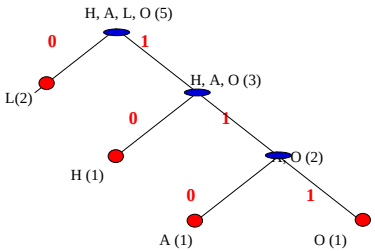
Entropi i binært bilde



- Vi vil alltid måtte bruke 1 bit per piksel i et binært bilde, selv om entropien godt kan bli nær null!
- Hvis det er like mange svarte og hvite piksler, er kodings-redundansen = 0

Shannon-Fano koding

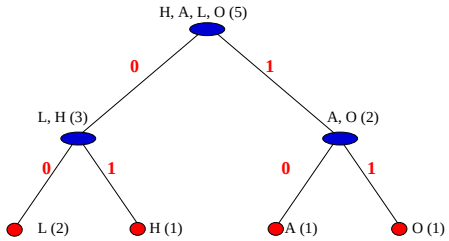
- En enkel metode:
  - Sorterer symbolene etter hyppighet.
  - Deler symbolene rekursivt i to "omtrent like store grupper".
    - Fortsett til hver gruppe er ett symbol.
  - Tilordner en bit til hver gren i treet
    - 1 til høyre, 0 til venstre.
  - Traverser treet "fra rot til blad"
    - Finner koden for hvert symbol.
- Eksempel: "HALLO" = 1011000111
- Koden er unikt dekodbar.



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 - II

- Oppdeling i ”omtrent like store grupper” kan gi flere forskjellige binære trær:
- Samme eksempel: ”HALLO”
- Selv om treet er annerledes, og kodeboken blir forskjellig, så er koden unikt dekodbar.



- ”HALLO” = 0110000011
- Vi har altså flere likeverdige løsninger.

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 optimal koding med variabel-lengde koder.
- Huffman-koding er basert på at vi kjenner hyppigheten for hvert symbol
  - Dette betyr at vi må lage et histogram.
  - Ofte beregner vi sannsynlighetene
    - Men det holder at vi kjenner hyppighetene.
- Huffman-koden er unikt dekodbar.

Framgangsmåte - Huffman-koding

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 i en gruppe, og sorter igjen etter sannsynlighet.
3. Gjenta 2 til det bare er to grupper igjen.
4. Gi kodene 0 og 1 til de to gruppene.
   - Kode 0 til den mest og 1 til den minst sannsynlige av de to
5. Traverser bakover, og legg til 0 og 1 i kodeordet for de to minst sannsynlige gruppene i hvert steg.

Eksempel - Huffman-koding

- Gitt 6 begivenheter A, B, C, D, E, F 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 minst sannsynlige, slå også sammen sannsynligheten deres

0.15

Finn så de to som nå er minst sannsynlige, og slå dem sammen på samme måte

0.25

0.4

0.6

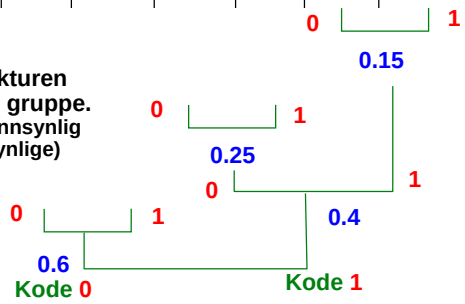
Fortsett til det er bare to igjen

Eksempel - Huffman-koding

- Gitt 6 begivenheter A, B, C, D, E, F med sannsynligheter

Begivenhet	A	B	C	D	E	F
Sannsynlighet	0.3	0.3	0.13	0.12	0.1	0.05

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



Kodeboken

- Dette gir følgende kodebok

Begivenhet	A	B	C	D	E	F
Kode	00	01	100	101	110	111

- Med sannsynlighetene 0.3 0.3 0.13 0.12 0.1 0.05
  blir gjennomsnittlig antall biter pr. symbol (R) for denne koden: (se foil 26)

$$R = b_1 p_1 + b_2 p_2 + \ldots + b_N p_N = \sum_{i=1}^{N} b_i p_i = 0.6 \cdot 2 + 0.4 \cdot 3 = 2.4$$

- Entropien H er her mindre enn R (se foil 28):

$$H = -\sum_{i=0}^{2^b - 1} p(s_i) \log_2 (p(s_i)) = 2.34$$

Om Huffman-koding

- Ingen kode-ord danner prefiks i en annen kode
  - Dette sikrer at en sekvens av kodeord kan dekodes entydig.
  - Man trenger IKKE ende-markører.
- Mottatt kode er instantant (øyeblikkelig) dekodbar.
  - Dette gjelder også Shannon-Fano og naturlig binærkoding.
- Hyppige symboler har kortere koder enn sjeldne symboler.
  - Dette gjelder også for Shannon-Fano.
  - De to minst sannsynlige symbolene har like lange koder.
    - Siste bit skiller dem fra hverandre.
- Merk at kodeboken må overføres!

Huffman-koding i praksis

- Den ideelle binære kode-ord lengden for symbol s_i er
  $b_i = - \log_2(p(s_i))$
- Siden bare heltalls ordlengder er mulig, er det bare hvis

$$p(s_i) = \frac{1}{2^k}$$

  for heltalls k at dette er tilfelle.

Eksempel: hvis vi har :

Symbol	s_1	s_2	s_3	s_4	s_5	s_6
Sannsynlighet	0.5	0.25	0.125	0.0625	0.03125	0.03125
Kode	0	10	110	1110	11110	11111

så blir den gjennomsnittlig ordlengden her R = 1.9375 = H.

Lempel-Ziv-koding

- Premierer mønstre i dataene.
- Ser på samforekomster av symboler.
- Bygger opp en symbolstreng-liste
  både under kompresjon og dekompresjon.
- Denne listen skal ikke lagres eller sendes,
  for mottakeren kan bygge opp listen
  ved hjelp av den symbolstrengen han mottar.
- Det eneste man trenger er et standard alfabet
  - (f.eks ASCII).

Eksempel på Lempel-Ziv

- Anta at alfabetet er a, b og c som tilordnes kodene 1, 2 og 3.
- La dataene være ababcbababaaaaabab (18 tegn)
  - sender: lager ny frase = sendt streng pluss neste usendte symbol
  - mottaker: lager ny frase = nest siste mottatte streng pluss første symbol i sist tilsendte streng

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=1,b=2,c=3			a=1, b=2, c=3
a	1	ab=4	1	a	
b	2	ba=5	2	b	ab=4
ab	4	abc=6	4	ab	ba=5
c	3	cb=7	3	c	abc=6
ba	5	bab=8	5	ba	cb=7
bab	8	baba=9	8		

- Nå har vi mottatt en kode (8) som ikke finnes i listen.
- Vi vet at kode 8 ble laget som "ba+?" , og så ble kode 8 sendt.
- Ergo må vi ha "?" = "b" => 8 = ba + b = bab.

Eksempel på Lempel-Ziv - II

Ser	Sender	Senders liste	Mottar	Tolker	Mottakers liste
		a=1,b=2,c=3			a=1, b=2, c=3
a	1	ab=4	1	a	
b	2	ba=5	2	b	ab=4
ab	4	abc=6	4	ab	ba=5
c	3	cb=7	3	c	abc=6
ba	5	bab=8	5	ba	cb=7
bab	8	baba=9	8	bab	bab=8
a	1	aa=10	1	a	baba=9
aa	10	aaa=11	10	aa	aa=10
aa	10	aab=12	10	aa	aaa=11
bab	8		8	bab	aab=12

- Kode 10 ble laget idet 1 = a ble sendt, som 10 "a+?". Så sendes "10".
- Da må vi ha 10="a+a" ="aa".
- Istedenfor 18 tegn er det sendt 10 koder.
- 5 av 12 produserte koder ble ikke brukt i dette eksemplet.

JPEG-koding (tapsfri)

- JPEG (Joint Photographic Expert Group) er et av de vanligste bildeformatene med kompresjon.
- JPEG-standarden har varianter både for tapsfri og ikke-tapsfri kompresjon.
- JPEG kan bruke enten Huffman-koding eller en variant av universell koding kalt aritmetisk koding.
- Prediktiv koding brukes for
  - å predikere at neste piksel på samme linje har lignende verdi som forrige piksel
  - å predikere at en piksel har lignende verdi som pikselen på linjen over
  - å predikere at neste piksel på linjen har lignende verdi som de tre nærmest pikslene
- Typen koding bestemmes fra bilde til bilde

Ikke-tapsfri (lossy) kompresjon

- For å få høye kompresjonsrater, er det ofte nødvendig med ikke-tapsfri kompresjon.
- Ulempen er at man ikke kan rekonstruere det originale bildet, fordi et informasjonstap har skjedd.
- Enkle metoder for ikke-tapsfri kompresjon er rekvantisering til færre antall gråtoner, eller resampling til dårligere romlig oppløsning.
- Andre enkle metoder er filtering der f.eks. 3 · 3 piksler erstatter med ett nytt piksel som er enten middelverdien eller medianverdien av de opprinnelige pikselverdiene.
- Husk: Vi klarer oss med lavere oppløsning i kromasi-komponentene.

"Lossy" JPEG-kompresjon av gråtonebilde

- Bildet deles opp i blokker på 8 · 8 piksler, og hver blokk kodes separat.
- Subtraher 128 fra alle pikselverdiene.
- Hver blokk transformeres med DCT (Diskret Cosinus Transform):

$$F(u,v) = \frac{2}{\sqrt{MN}} \sum_{x=0}^{N-1} \sum_{y=0}^{M-1} c(x)c(y) \cos\left[\frac{\pi u}{2N}(2x+1)\right] \cos\left[\frac{\pi v}{2M}(2y+1)\right] f(x,y), \qquad c(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{ellers} \end{cases}$$

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

DCT transform

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.05	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

DCT-transformen er IKKE pensum !

- Informasjonen i de 64 pikslene samles i en liten del av de 64 koeffisientene
  - Mest i øverste venstre hjørne

"Lossy" JPEG-kompresjon – 2

- Transformkoeffisientene skaleres med en vektmatrise og kvantiseres til heltall.

39.88	6.56	-2.24	1.22	-0.37	-1.08	0.79	1.13
-102.43	4.56	2.26	1.12	0.35	-0.63	-1.05	-0.48
37.77	1.31	1.77	0.25	-1.50	-2.21	-0.10	0.23
-5.67	2.24	-1.32	-0.81	1.41	0.22	-0.13	0.17
-3.37	-0.74	-1.75	0.77	-0.62	-2.65	-1.30	0.76
5.98	-0.13	-0.45	-0.77	1.99	-0.26	1.46	0.00
3.97	5.52	2.39	-0.55	-0.05	-0.84	-0.52	-0.13
-3.43	0.51	-1.07	0.87	0.96	0.09	0.33	0.01

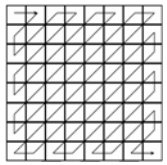
dividere med

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

avrundes til

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	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	0	0	0	0	0

- Sikk-sakk-scanning ordner koeffisientene i 1D-rekkefølge.
  - Koeffisientene vil da stort sett avta i verdi utover i rekka
  - Mange koeffisienter er rundet av til null.
  - Løpelengde-transform av koeffisientene.
  - Huffman-koding av løpelengdene.



- Huffman-koden og kodeboken sendes til mottaker eller til lager.
- Gjentas for alle blokker i alle kanaler.

JPEG dekompresjon av 8 · 8 blokk

- Huffman-koden for en blokk er reversibel og gir løpelengdene.
- Løpelengdetransformasjonen er reversibel, gir kvantiserte DCT-koeffisienter.
- Sikk-sakk transformen er reversibel, og gir en heltallsmatrise.
- Denne matrisen multipliseres med vektmatrisen.

2	1	0	0	0	0	0	0
-9	0	0	0	0	0	0	0
3	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	0	0	0	0	0

multipliseres med

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Dette gir

32	11	0	0	0	0	0	0
-108	0	0	0	0	0	0	0
42	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	0	0	0	0	0

- Dette er IKKE helt likt koeffisientene etter forlengs DCT-transformasjon.
- Men de store trekkene er bevart:
  - De største tallene ligger i øvre venstre hjørne
  - De fleste tallene i matrisen er lik 0.

JPEG dekompresjon – forts.

- Så gjør vi en invers DCT, og får en rekonstruert 8 · 8 piksels bildeblokk.

32	11	0	0	0	0	0	0
-108	0	0	0	0	0	0	0
42	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	0	0	0	0	0

invers DCT transform

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	139	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

- Differansene fra den originale blokken er små!

124	125	122	120	122	119	117	118
121	121	120	119	119	120	120	118
126	124	123	122	121	121	120	120
124	124	125	125	126	125	124	124
127	127	128	129	130	128	127	125
143	142	143	142	140	139	139	139
150	148	152	152	152	152	150	151
156	159	158	155	158	158	157	156

-

123	122	122	121	120	120	119	119
121	121	121	120	119	118	118	118
121	121	120	119	119	118	117	117
124	124	123	122	122	121	120	120
130	130	129	129	128	128	128	127
141	141	140	140	139	139	138	137
152	152	151	151	150	149	149	148
159	159	158	157	157	156	155	155

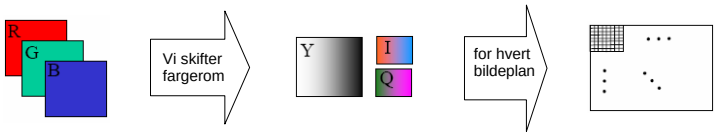
=

1	3	0	-1	-2	-1	-2	-1
0	0	-1	-1	0	2	2	0
5	3	3	3	3	3	3	3
0	0	2	3	4	4	4	4
-3	-3	-1	0	2	0	-1	-2
2	1	3	2	1	0	1	2
2	-4	1	1	2	3	1	3
-3	0	0	-2	-1	2	2	1

- Forskjellige vektmatriser for intensitet og kromatisitet.
- Kompresjon / dekompresjon kan gi blokk-effekter i bildene.

JPEG-kompresjon av fargebilde

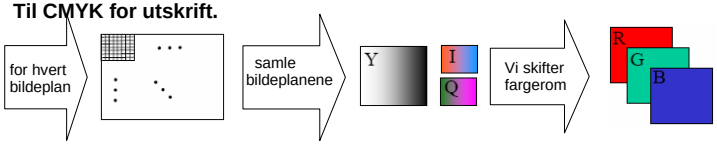
- Vi skifter fargerom slik at vi separerer lysintensitet fra kromasi.
- Resampler kromasi-komponentene (benytter 4:2:0).
  - Dermed fjernes perseptuell redundans, og vi sparer plass.
- Kanalene deles opp i blokker á 8 · 8 piksler, hver blokk kodes separat.



- Hver blokk transformeres med DCT (foil 46-47).
- Forskjellige vektmatriser for intensitet og kromatisitet.
  - ..... resten er som for gråtonebilder ...

JPEG dekompresjon av fargebilde

- Hver 8 · 8 blokk dekomprimeres (foil 48-49)
- Alle dekomprimerte 8 · 8-blokker i hvert bildeplan samles til et bildeplan.
- Bildeplanene samles til et YIQ fargebilde
- Vi skifter fargerom
  - fra YIQ til RGB for fremvisning,
  - Til CMYK for utskrift.



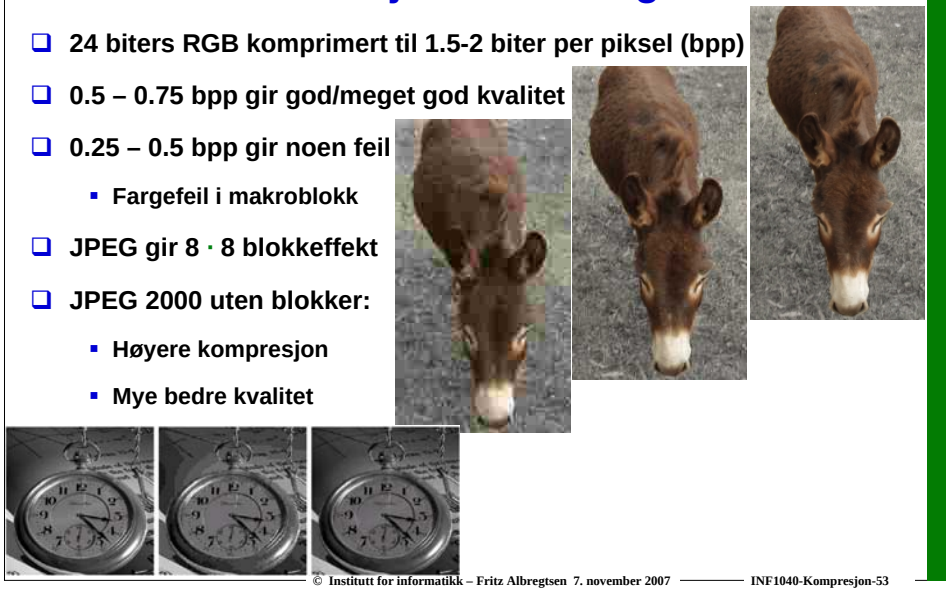
- Vi har redusert oppløsning i Y og Q, men full oppløsning i RGB:
  - Gir 8 · 8 blokkeffekt i intensitet
  - Gir 16 · 16 piksels blokkeffekt i fargene i RGB

Rekonstruksjons-feil i gråtonebilder

- DCT kan gi ”blokk-artefakter”, sløring og dobbelt-konturer.
- Avhengig av
  - vektmatrise
  - antall koeffisienter

Rekonstruksjons-feil i fargebilder

- 24 biters RGB komprimert til 1.5-2 biter per piksel (bpp)
- 0.5 – 0.75 bpp gir god/meget god kvalitet
- 0.25 – 0.5 bpp gir noen feil
  - Fargefeil i makroblokk
- JPEG gir 8 · 8 blokkeffekt
- JPEG 2000 uten blokker:
  - Høyere kompresjon
  - Mye bedre kvalitet



Oppsummering - kompresjon

- Hensikten med kompresjon er mer kompakt lagring eller rask oversending av informasjon.
- Kompresjon er basert på informasjonsteori.
- Antall biter pr. sampel er sentralt, og varierer med kompresjonsmetodene og dataene.
- Sentrale metoder:
  - Huffman-koding – lag sannsynlighetstabell, send kodebok
  - Lempel Ziv – utnytter mønstre – sender ikke kodebok
  - For bilder: løpelengde-koding, differansekoding, JPEG.
- Kompresjon og koding er helt sentralt i moderne teknologi !!!