

Kompresjon og koding I

Kompendium: 18.1-18.5, 18.7.2, 18.7.4

Aasan og Tan

November 7, 2023

Table of contents

1 Kompresjon og Redundans

2 Enkle Transformasjoner

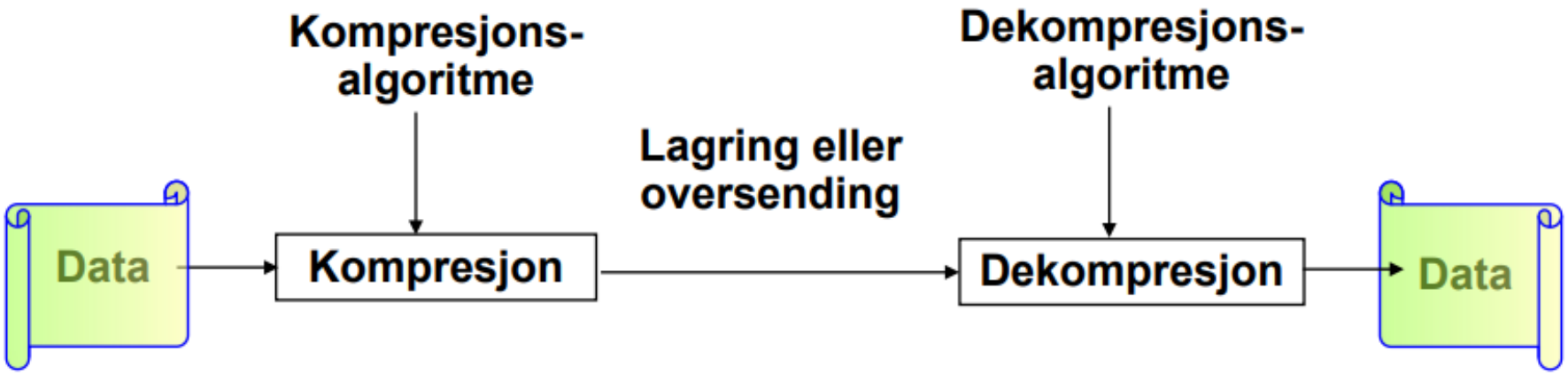
3 Koding og entropi

4 Bitkoding

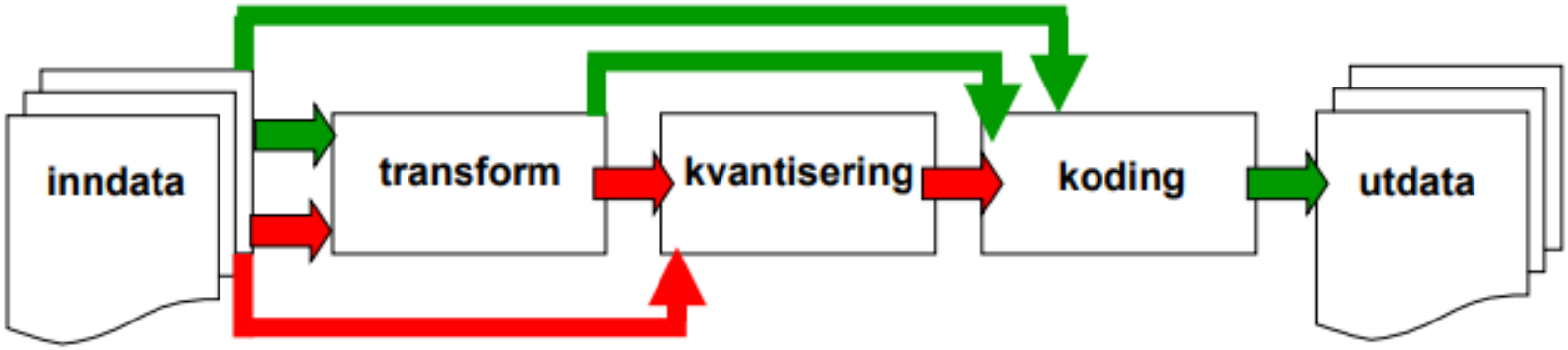
Kompresjon og Redundans

Hva er kompresjon?

- Kompresjon brukes for å redusere antall biter som brukes for å beskrive et bilde.
- Pakke data på en kompakt måte
- Overflødig informasjon lagres ikke
- Bruker koding for å lagre/sende data med færrest mulig biter



Kompresjonsprosessen



- Transform: Mer kompakt representasjon av bilde
- Kvantiserting: Avrunding av representasjonen
- Koding: Transformer til binærrepresentasjon

Tapsfri: Kan eksakt rekonstruere originalbildet.

- Bruker reversible transformer og kodinger

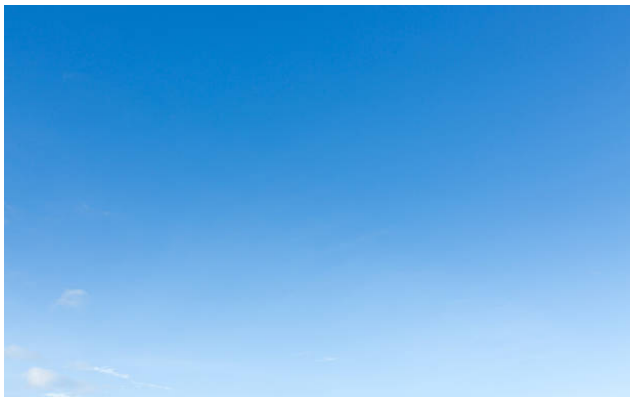
Ikke-tapsfri: Kan generelt ikke eksakt rekonstruere bildet.

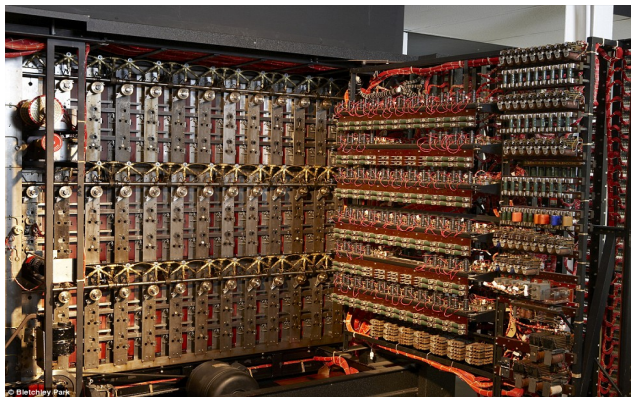
- Kan bruke steg som ikke er reversible, f.eks. kvantisering

Redundans I

En melding kan være tekst, bilde, lydsignal, m.m.

- Melding: Det vi skal lagre/sende.
- Datamengde vs. informasjon
  - Datamengde: Strømmen av biter i meldingen
  - Informasjon: Kvantifiserer mengden av “overraskelse” eller “uforventethet” i meldingen
- Hvilket bilde har mer informasjon?





- Mye variasjon $\rightarrow$ mye informasjon
- Kanter har høyt informasjonsinnhold

Redundans II

Redundans: Hvor mye av datamengden som kan fjernes uten at man mister relevant informasjon.

- **Interpiksel:** Redundans i naboskapstrukturer i et signal.
- **Koding:** Redundans i representasjon av enkeltverdier.
- **Psykovisuell:** Informasjon vi ikke legger merke til.
- **Interbilde:** Redundans mellom relaterte bilder, f.eks. i video.

Fjerne redundant data $\rightarrow$ Mindre datamengde som må lagres.
Vi utnytter dette i kompresjon.

Kompresjonsrate og relativ redundans

Kompresjonsrate:

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

Her er b antall biter per symbol/pixel i den ukomprimerte datamangden, og c er gjennomsnittelig antall biter per symbol/sampel i den komprimerte datamengden.

Relativ redundans:

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

Også kalt plassbesparelse (*space savings*). Ofte gitt i prosent (*percentage removed*):

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

Recap: Naturlig binærkoding

Generell n-biters binærkoding:

- Det er $k = 2^n$ symboler $s_0, s_1, ..., s_{k-1}$
- Hvert symbol har en unik kode c_i som består av n biter
- Alle koder er like lange

Eksempel

Eksempel: 3-biters naturlig binærkoding

Symbolindeks	0	1	2	3	4	5	6	7
Symbol s_i	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

Her er det $2^3 = 8$ mulige symboler.

Enkle Transformasjoner

Løpelengdetransform I

Hovedidé: *Utnytte lokalt homogene områder med liten eller ingen variasjon ved å se på løp med like pikselverdier gitt en forhåndsdefinert skanningsrekkefølge på bilde.*

Med vektorisert skan $f(1,1), f(1,2), \ldots, f(2,1), \ldots, f(h,w)$:

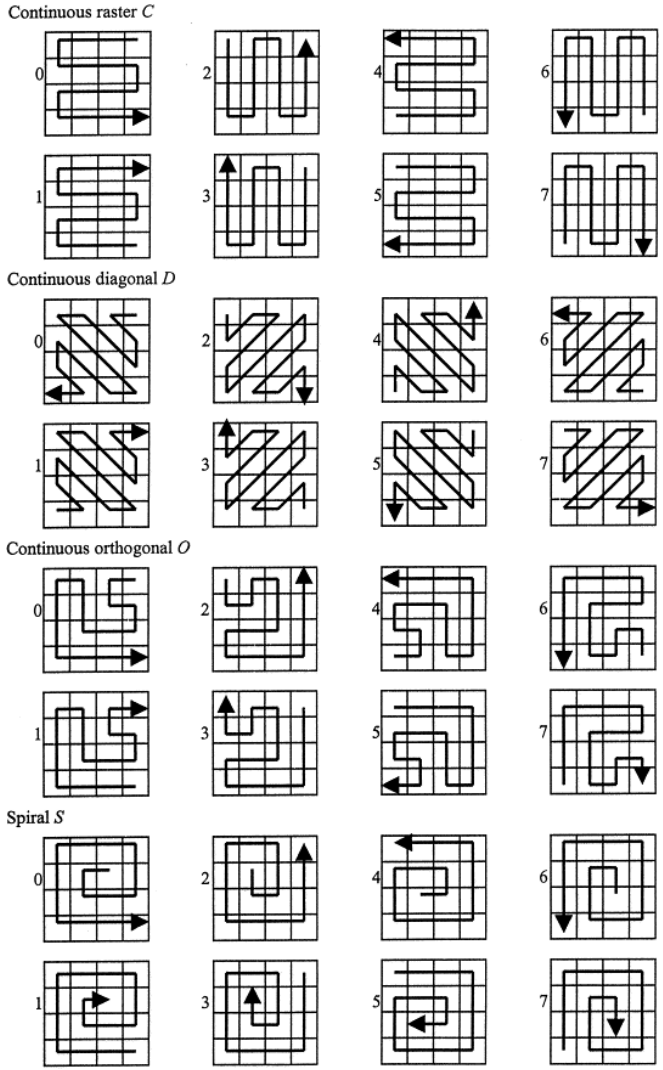
$$\mathrm{RLE}\left(\mathrm{vec}\begin{bmatrix}0 & 2 & 2 & 2\\ 2 & 2 & 1 & 1\\ 1 & 1 & 2 & 2\\ 2 & 2 & 1 & 1\end{bmatrix}\right) = \mathrm{RLE}\left([0,2,2,2,2,2,1,\ldots,0]\right)$$

$$= \begin{bmatrix}1 & 5 & 4 & 4 & 2\\ 0 & 2 & 1 & 2 & 1\end{bmatrix}$$

der hvor første rad angir løpslengde, og andre rad angir løpsverdier. Kalles *run-length encoding* (RLE) på engelsk.

Hva slags bilder er dette nyttig for?

Løpelendgetransform II



Figur: Eksempler på kontinuerlige skannerekkefølger.

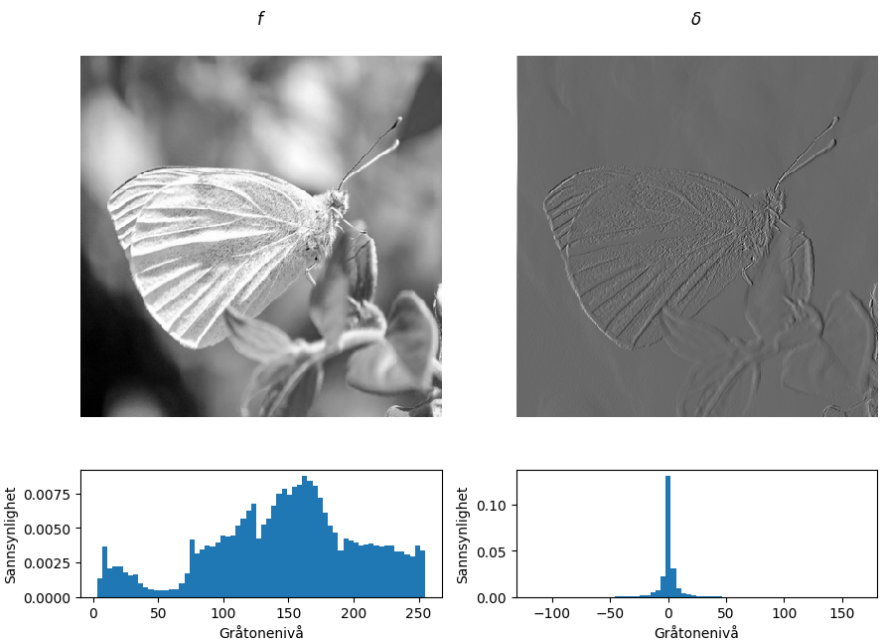
Løpelengdetransform III

- Uten kvantisering / Tapsfri:
  - Fungerer kun på bilder med store områder uten variasjon.
  - Kan være effektiv for bilde med lav bitdybde, indekserte bilder, eller segmenteringsmasker.
  - Kan være mer effektiv under forskjellige transformer.
  - Benyttes gjerne i kombinasjon med andre teknikker, ofte Huffman-koding eller aritmetisk koding.
- Med kvantisering / Ikke-tapsfri.
  - Kvantisering: reduserer antall nivåer og gjør RLE mer effektivt.
  - Kombineres ofte med transformasjoner som gjør bildet *glissent*.
    - Et glissent (sparse) bilde har brorparten av sine verdier nær null.
  - Typiske transformasjoner:
    - Fourier transform (DFT).
    - Cosinus transform (DCT).

Merk: For binære bilder trenger vi ikke en rad for verdier, men må tillate løp med null-lengde for tilfeller der bildet starter med 1.

Differansetransform I

Hovedidé: *Utnytte kontinuitet; nabopiksler har ofte like verdier.*
Radvis: $\delta(1, y) = f(1, y), \;\; \delta(x + 1, y) = f(x + 1, y) - f(x, y)$.



For relativt kontinuerlige bilder f blir δ glissen, og transformasjonen tilsvarer éndimensjonal gradientoperator.

Fungerer dette som et høypass eller lavpassfilter?

Differansetransform II

Differansetransform uten kvantisering er *tapsfri*.

Radvis invers: $f(1, y) = \delta(1, y), \;\; f(x+1, y) = \delta(x+1, y) + f(x, y)$
Alternativt: $f(x, y) = \sum_{j=1}^{x} \delta(j, y)$.

Er inverstransformen et høypass eller lavpassfilter?

Merker oss at for $f(x, y) \in [a, b]$ for alle $1 \leq x \leq w, 1 \leq y \leq h$, så har vi at $\delta(x, y) \in [a - b, b - a]$. For binærkoding med bitdybde $2^d - 1$ gir dette $\delta(x, y) \in [-2^d + 1, 2^d - 1]$. *Vi trenger derfor én ekstra bit for å representere* δ.

- Differansetransform kan med hell kombineres med kvantisering og løpelengdetransformasjon.
- Settes ofte i sammenheng med *prediktiv koding*.

Vi kan også ta transformen med andre skannerekkefølger, kolonnevis over bildet, eller eventuelt i begge prinsipielle retninger.

Bitplan og Gråkodetransform I

Når et gråtonebilde dekomponeres i bitplan, så gir naturlig binærkoding binære bilder med høy kompleksitet. Hvorfor?

$f \wedge 10000000$



$f \wedge 01000000$



$f \wedge 00100000$



$f \wedge 00010000$



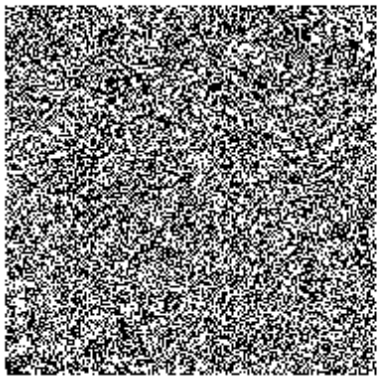
$f \wedge 00001000$



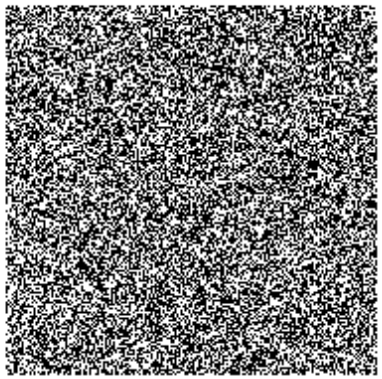
$f \wedge 00000100$



$f \wedge 00000010$



$f \wedge 00000001$



Bitplan og Gråkodetransform II

Svar: I naturlig binærkoding kan bitrepresentasjonen for gråtoner "hoppe"mellom nabonivåer.

Eksempel: 127 + 1 = 01111111 + 00000001 = 10000000.

Idé: Velger istedet koding som skifter med maks 1 bit når gråtone endres med ett nivå.

Gitt en bitrepresentasjon $b_{n-1}b_{n-2}\ldots b_0$ konstruerer vi gråkodetransformen gitt ved

$$g_{n-1} = b_{n-1}; g_i = b_i \oplus b_{i+1} \text{ for } i = 1, \ldots, n-2,$$

hvor $\oplus$ betegner bitvis XOR operator. Dette korrelerer bitplanrepresentasjoner av bilder, og gir bedre løpelengdekompresjon.

Bitplan og Gråkodetransform III

Eksempel

Gråkode	Naturlig binærkode	Desimaltall
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

Kodeordene er like lange i gråkode som i naturlig binærkode...

...men i gråkode skifter alltid bare én bit når gråtonen endres med 1.

Bitplan og Gråtonetransform IV

Bitplan etter gråtonetransform:

$G(f) \wedge 10000000$



$G(f) \wedge 01000000$



$G(f) \wedge 00100000$



$G(f) \wedge 00010000$



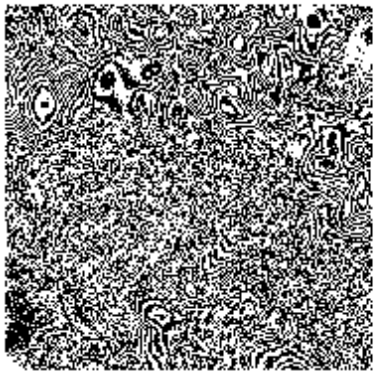
$G(f) \wedge 00001000$



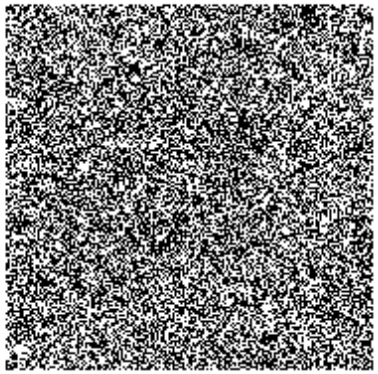
$G(f) \wedge 00000100$



$G(f) \wedge 00000010$



$G(f) \wedge 00000001$



Koding og entropi

Koding og Entropi I

Eksempel: Vi skal kode en datamengde, og observerer at *noen symboler forekommer ofte* mens *andre symboler forekommer veldig sjeldent*.

$\Longrightarrow$

Idé: Representerer symbolene som forekommer ofte med et lite antall biter, og de som forekommer sjeldent med et større antall biter.

Variabelt antall bitposisjoner per symbol $\rightarrow$ *Kan spare plass fordi hyppige symboler bruker færre biter.*

Eksempel: Morsekode lar de hyppigste bokstavene i engelsk tekst ha de korteste kodene. Se f.eks E, I, T, A, O og N.

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	- . - -

. og mellomrom varer 1 enhet, - varer 3 enheter

Koding og Entropi II

Anta vi har

- et *alfabet* med G symboler $s_0, s_1, ..., s_{G-1}$, og
- en *kodebok* $c_0, c_1, ..., c_{G-1}$ slik at symbolet s_i kodes med kodeordet c_i

Informasjonsteori bygger på sannsynlighen for at ulike begivenheter skal forekomme.

- $p(s_i) \in [0, 1]$ er sannsynligheten for at et symbol s_i forekommer.
- Gitt en sekvens med N symboler kan vi estimere

$$p(s_i) \approx \frac{n_i}{N} = p_i$$

  hvor n_i er antall ganger s_i forekommer i sekvensen.
- Estimatene $p_0, p_1, ..., p_{N-1}$ er det normaliserte histogrammet til sekvensen.

Koding og Entropi III

Gitt en sekvens med N symboler og det normaliserte histogrammet $p_i = n_i/N$ har vi følgende definisjoner

Definisjon

Gjennomsnittelig antall biter per piksel:

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

b_i er lengden på kodeordet c_i angitt i biter

Definisjon

Informasjonsinnholdet i hendelsen s_i:

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

Informasjons-innholdet er uttrykt i biter, derfor bruker vi $\log_2$

Entropi

Det gjennomsnittelige informasjonsinnholdet er kjent som entropi.

Definisjon

Entropien til sekvensen er gitt ved

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

- Også kjent som *Shannon entropi*
- Entropi kan tolkes som mengden av "overraskelse" eller "usikkerhet"
- Entropi er også relatert til "Mutual Information (MI)" som måler den gjensidige avhengigheten mellom to sekvenser. Les mer om dette på https://en.wikipedia.org/wiki/Mutual_information (ikke pensum).

Øvre og nedre grense på entropi

- Hvis alle symboler er like sannsynlige → Entropi lik antall biter

  Hvis det er $G = 2^b$ symboler of sannsynligheten for hver er $p_i = 1/2^b$, så er

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

  Hvis det er 2^b symboler som alle er like sannsynlige, så kan de ikke representeres mer kompakt enn med b biter per symbol.

- Hvis alle symboler er like → Entropi lik 0

  I dette tilfellet er det kun et symbol s som dukker opp i sekvensen, med sannsynlighet $p = 1$. Da er entropien

$$H = -p \log_2(p) = 0,$$

  siden $\log_2(1) = 0$.

Eksempel: Entropi i et binært bilde I

Anta vi har et *binært bilde* med $M \times N$ piksler.
Lagrer vi hver pikselverdi trenger vi MN biter. Men hvor mye informasjon er det egentlig i bildet? La oss se på to eksempler.

Eksempel

Anta det er like mange 0 som 1 i bildet.* Da er 0 og 1 like sannsynlig i bildet, så $p_0 = 1/2$ og $p_1 = 1/2$ og entropien er $H = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$.

Eksempel

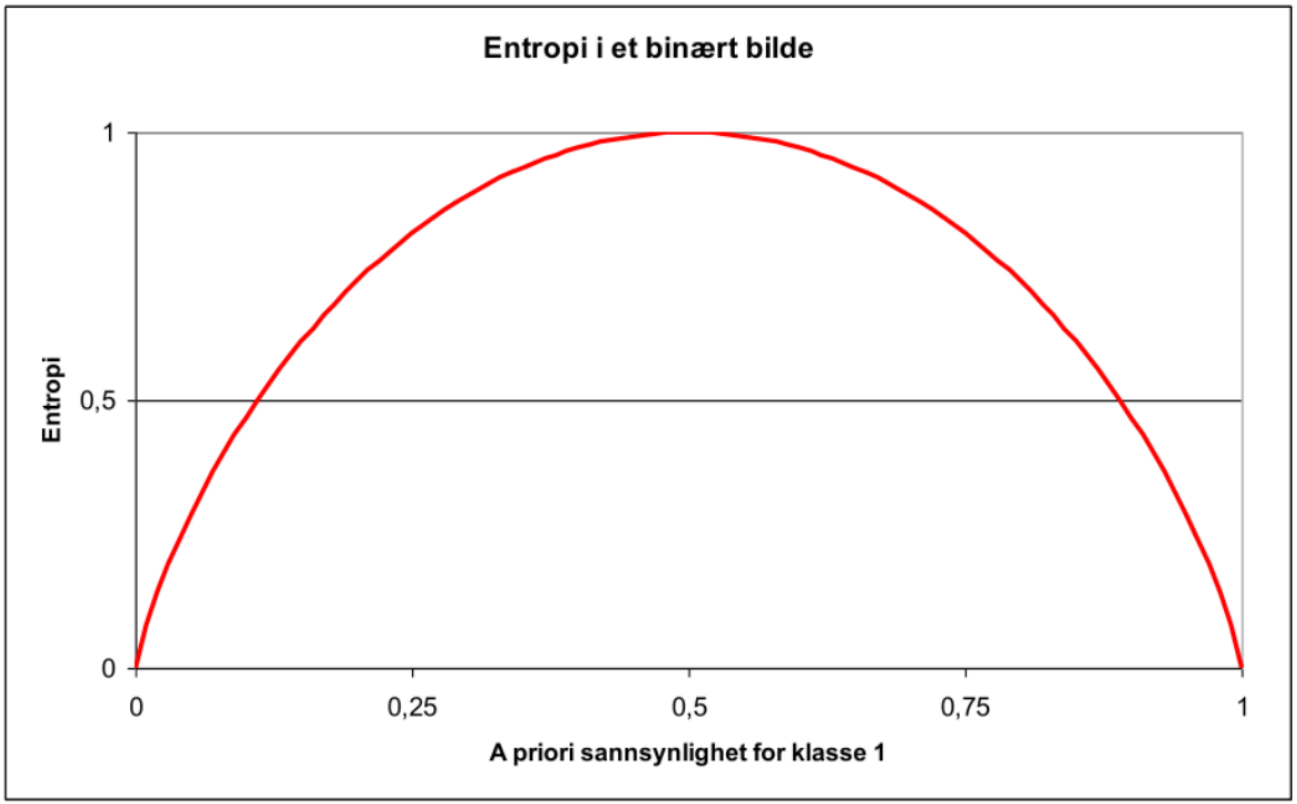
Anta det er 3 ganger så mange 1 som 0 i bildet.* Da er $p_0 = 1/4$ og $p_1 = 3/4$ så det er mindre "overraskende" å få en 1 siden det skjer oftere. Entropien er da litt mindre:

$$H = -\frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) = \frac{2}{4} + \frac{3}{4}\log_2\left(\frac{4}{3}\right) \approx 0,811$$

*og ingen interpiksel-redundans.

Eksempel: Entropi i et binært bilde II

Mer generelt vil entropien i et binært bilde følge denne kurven:



Bitkoding

Huffman Koding I

- Vi er gitt en sekvens $X = (x_1, \ldots, x_m)$.
- Sekvens inneholder et sett med mulige symboler $S = (s_1, \ldots, s_n)$.
- Vi ønsker å enkode hvert symbol med en optimal bitkode med variabel lengde.

Idé: ta antall symbol i X og kode frekvente symboler med lavt antall biter.

Eksempel

Sekvens: $X = (r, a, d, a, r)$.
Kode: 011010
Dekodet: $\hat{X} = (r, a, a, r, a, r)$?

Problem: Koden er ikke entydig!

Symbol	Antall	Kode
r	2	0
a	2	1
d	1	10

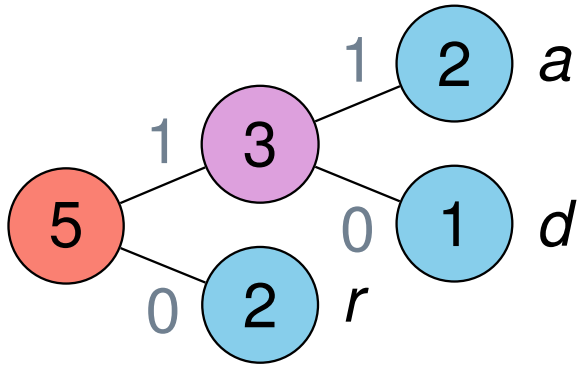
Kan fikse koden for X ved å enkode antall biter som i RLE, men dette krever en **prefiks** og utvider kodelengden.

Huffman Koding II

Ønsker: **prefiksfri** bitkoding som er **entydig**, har **variabel lengde**, og gir **lavest forventet kodelengde**.

Idé: bruke et binært tre der hver splitt i treet gis binær verdi.

- Hver bladnode angis ett symbol og en vekt lik dets frekvens i X.
- Hver node angis en vekt lik summen av røttene til dets subtre.
- Koden for hvert symbol angis av kantene langs stien til rotnoden.



Figur: Huffman tre for $X = [r, a, d, a, r]$.

Kode: $r = 0, a = 11, d = 10 \rightarrow 01110110$.

Huffman Koding III

La X med symboler $S = (s_1, \ldots, s_n)$ være gitt med frekvens $f(s_i)$.

Et Huffman tre kan konstrueres ved følgende algoritme:

1. Konstruer enkeltnode Huffman-trær med hvert av de n symbolene s_i med tilhørende vekt $f(s_i)$.
2. Repeter følgende til vi ender opp med ett enkelt tre:
   - Velg to trær T_0 og T_1 med minimal vekt.
   - Erstatt T_0 og T_1 med et nytt tre som har T_0 som venstre subtre og T_1 som høyre subtre.

Resultat: Et gyldig Huffman tre med entydig prefiksfri koding.

Merk: Selv om kodingen til Huffman treet er entydig er ikke **treet** unikt! Flere gyldige Huffman koder kan konstrueres gitt samme sekvens.

Enkodingen må derfor også lagres!

Huffman koding IV

Husk at H er entropi og c er gjennomsnittlig kodelengde i biter

- Generelt for Huffman koding har vi at $H \leq c \leq H + 1$
- Kodingsredundansen kan regnes ved å ta gjennomsnittlig kodelengde minus entropi
- Kodingsredundansen til Huffman-koder har altså en øvre grense $c - H \leq 1$
- Med andre ord er kodingsredundansen maks 1 bit per symbol.
- Ideelt sett ønsker vi $c - H = 0 \Rightarrow c = H$.
  For å oppnå dette må

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

  Dette blir oppfylt når $b_i = \log_2(1/p_i) = I(s_i)$.
- Informasjonsinnholdet $I(s_i)$ angir altså den ideelle binære kodelengden for symbolet s_i.

Aritmetisk Koding I

Ved nærmere kikk på optimalitetskriteriet for Huffman får vi

$$b_i = -\log_2(p_i)$$
$$2^{-b_i} = p_i.$$

Sannsynlighetene for hvert symbol må sammenfalle med en inverspotens av 2. Hvor realistisk er dette kriteriet for generelle sekvenser?

Dette er en restriksjon som kommer fra representasjon av koder med **heltallslengder**.

Er det mulig å istedet representere koder med **desimaltall**?

Aritmetisk Koding II

Idé: Representer koder med desimaltall!

- Lager ikke kodeord for enkeltsymboler og unngår begrensninger fra heltallslengder.
- Koder hele sekvensen som et enkelt tall i intervallet $[0, 1)$.
- Gir oss en mer granulær og korrekt representasjon av symbolsannsynligheter.
- Koder genereres slik at de er nærmere tilknyttet entropi enn kildesekvensen X.

Resultat: bitforbruk per symbol som er bedre tilrettelagt optimal entropi.

Aritmetisk Koding III

1. Finn sannsynlighetene for hvert symbol gitt sekvensen X.
   - Resulterer i sannsynlighetene p_i for $i = 1, \ldots, |S|$.
2. Finner så kumulative $P_i = \sum_{j=1}^{i} p_j$, og lar $P_0 = 0$.
3. Initialiser delmengden $I^{(0)} = [L^{(0)}, U^{(0)}) = [0, 1)$.
4. For hvert indeks $k = 1, \ldots, m$ der $m = |X|$:
   - Bestem i slik at $s_i = x_k$.
   - Oppdater delintervall i $I^{(k)}$ gitt P_i ved:
     - $L^{(k)} = L^{(k-1)} + (U^{(k-1)} - L^{(k-1)})P_{i-1}$.
     - $U^{(k)} = L^{(k-1)} + (U^{(k-1)} - L^{(k-1)})P_i$.
     - $I^{(k)} = [L^{(k)}, U^{(k)})$.
5. Til slutt; finn lavest mulig bitrepresentasjon for $I^{(m)}$.

I praksis: Må inkludere et *end-of-file* symbol i sekvensen som gis sannsynlighet $p_{i+1} = 1/(m+1)$.

I tillegg må vi passe på å gjøre steg 5 korrekt.

Aritmetisk Koding IV

Eksempel

Sekvens: $X = (r, a, d, a, r)$.

$r \in [0, .\overline{3}) \Rightarrow I^{(1)} = [0, .\overline{3})$.
$a \in [.\overline{3}, .\overline{6}) \Rightarrow I^{(2)} = [.\overline{1}, .\overline{2})$.
$d \in [.\overline{6}, .8\overline{3}) \Rightarrow I^{(3)} = [.\overline{185}, .2\overline{037})$.
$a \in [.\overline{3}, .\overline{6}) \Rightarrow I^{(4)} \approx [.1\overline{91...}, .\overline{197...})$.
$r \in [0, .\overline{3}) \Rightarrow I^{(5)} \approx [.1\overline{91...}, .\overline{1934...})$.
$\langle eof \rangle \in [.8\overline{3}, 1.0) \Rightarrow I^{(5)} \approx [.1931, .\overline{1934...})$.

Symbol	p_i	P_i
r	$.\overline{3}$	$.\overline{3}$
a	$.\overline{3}$	$.\overline{6}$
d	$.1\overline{6}$	$.8\overline{3}$
<eof>	$.1\overline{6}$	1.0

Ethvert tall i $I^{(5)} = [563/2316, 47/243)$ vil entydig dekode (r, a, d, a, r).

Nytt problem: Velge et desimaltall $q \in I^{(m+1)}$ med færrest mulig biter.

Aritmetisk koding V

Generell fremgangsmåte for å finne $q \in I^{(m+1)}$:

Initialiser $q = L^{(m+1)}$.

- Konverter stegvis q med binærrepresentasjon $c_1 2^{-1} + c_2 2^{-1} + \ldots$.
- Dersom $q \notin I^{(m+1)}$, fjern minst signifikante bit inntil $q \in I^{(m+1)}$.
- Fortsett til det er klart at $q \in I^{(m+1)}$ for alle videre iterasjoner.

Generelle Svakheter med AK:

- Upraktisk med flyttallsrepresentasjon for $L^{(k)}$, $U^{(k)}$.
  - Renormaliser L, U ved å periodisk lagre biter de har til felles.
  - Benytt i stedet triks for heltallsaritmetikk for binærtall.
- Tunge utregninger sammenliknet med Huffman.
- Huffman har ofte bedre kompresjon med flere symboler i alfabetet...
  - ...men aritmetisk koding komprimerer generelt bedre, spesielt for lange sekvenser.

Aasan og Tan

Kompresjon og koding I

Kompendium: 18.1-18.5, 18.7.2, 18.7.4

