Hur har det gått?

Kodning

Kodning

- En tilldelning av *kodord* för *symboler*
- Formellt en funktion från ett alfabet till ett annat: $C : \Sigma_1 \to \Sigma_2$
- Symboler och kodord kan vara vad som helst, men ett vanligt exempel är att symboler är bokstäver och kodorden är bitsträngar
- Exempel: $\{x \mapsto 01, y \mapsto 101, z \mapsto 0101\}$

Fixed/variable length

- Om varje kodord är lika långt så är kodningen *fixed-length*
  - Exempel: $\{a \mapsto 00, b \mapsto 01, c \mapsto 10\}$
- Annars är det en *VLE, variable-length encoding* (jfr. *VBR, variable bitrate*)
  - Exempel: $\{a \mapsto 0, b \mapsto 10, c \mapsto 1101\}$

Non-singular code

- Om varje kodord är unikt är kodningen *non-singular*
- I det fallet är koden *injektiv*, och kodningen *lossless*
  - Motexempel: $\{a \mapsto 0, b \mapsto 1, c \mapsto 1\}$
  - Strängen aba kodas som 010, men 010 kan avkodas till aba och aca
  - Kodsträngen är kort (bra) men information har förlorats (dåligt)

Uniquely decodable

- Om varje kodsträng kan avkodas unikt är kodningen *uniquely decodable*
- Alltid icke-singulär, men en icke-singulär kod är inte nödvändigtvis unikt avkodbar
- Exempel: $\{a \mapsto 0, b \mapsto 10, c \mapsto 11\}$
- Motexempel: $\{a \mapsto 0, b \mapsto 1, c \mapsto 10\}$
  - 010 kan avkodas till aba eller ac

Prefix/suffix code

(alt. prefix-free/suffix-free)

- I en *prefix* eller *suffix code* är inget kodord ett prefix/suffix till ett annat
- Motexempel: $\{a \mapsto 0, b \mapsto 1, c \mapsto 01\}$
  - Kodordet för a är ett prefix av kodordet för c
  - Kodordet för b är ett suffix av kodordet för c
- *prefix* eller *suffix* är tillräckligt men inte nödvändigt för att vara unikt avkodbar

Shannon entropy

Entropin $H(X)$ för en slumpvariabel X (ett okänt meddelande) över mängden Ω med sannolikhetsfördelning $p(x)$ är

$$H(X) = -\sum_{x_i \in \Omega} p(x_i) \log_2(p(x_i))$$

- Mäter *osäkerheten* av X, eller *mängden information*, i antal bitar
- Kan tolkas som det *genomsnittliga antal bitar* som krävs för att koda en symbol

Shannon's source coding theorem

Ett meddelande med längd N och entropi $H(X)$ kan kodas med $NH(X)$ bitar, men inte mindre (utan att förlora information)

(Shannon 1948)

- Gäller när symboler är *oberoende* och *likafördelde*
- I praktiken är detta är ofta inte fallet, då kan kodning med färre bitar vara möjlig
- Exempelvis text där bokstäver och ord inte följer helt slumpmässigt

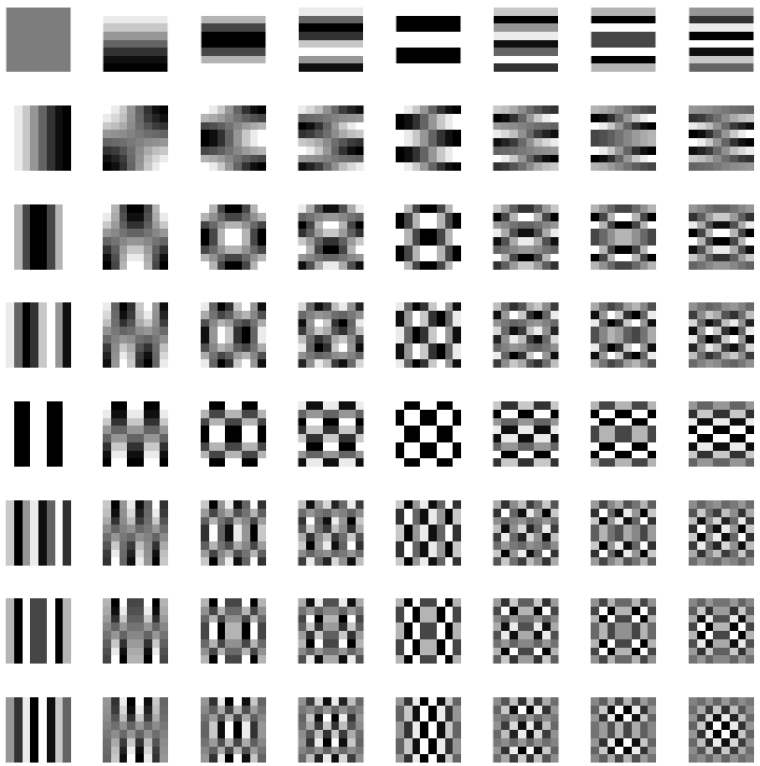
Komprimering

Lossy/destruktiv komprimering

- I destruktiv komprimering går en viss mängd information förlorad
- Funkar för datatyper som bild, video och ljud (lägre kvalitet)
- Funkar inte för binär data eller text (*”H7ej, kag hetr Lussig”*)
- Exempel: JPEG, MPEG4, telefoni

Lossy/destruktiv komprimering: JPEG

- JPEG använder DCT (diskret cosinustransform)
- Delar av bilden beskrivs som summor av cosinusvågor
- Höga frekvenser kan ofta tas bort
- Huffmankodning



Lossless/icke-destruktiv komprimering

- Med icke-destruktiv komprimering förloras ingen data
- Lämpar sig väl för text
- Är ofta mindre effektiv
- Exempel: Huffmankodning, LZ77

Huffmankodning

Huffmankodning

- Bygger på en analys av sannolikheten för vissa tecken/delsträngar
- Frekvent förekommande tecken kodas med få bitar
- Sannolikheten kan bygga på analys av filen (nackdelar?) eller andra antaganden
- Är alltid en prefix-kodning (då också unikt avkodbar)
- Optimal i de fall då Shannon’s source coding theorem gäller (upp till avrundning till helt antal bitar)

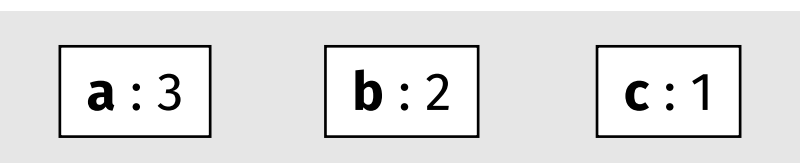
Huffmankodning

- Exempel: Vi ska komprimera strängen **abacab**
- Vi bygger en frekvenstabell:

Tecken	Frekvens
a	3
b	2
c	1

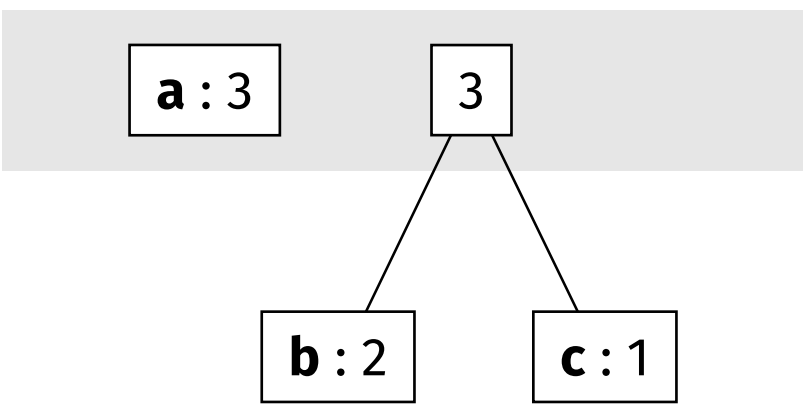
Huffmankodning

- Vi ska nu bygga ett binärt träd med lövnoder som motsvarar tecken
- Vi börjar med att lägga till alla lövnoder (delträd) i en prioritetskö
- Lägst frekvens $\rightarrow$ först i kön



Huffmankodning

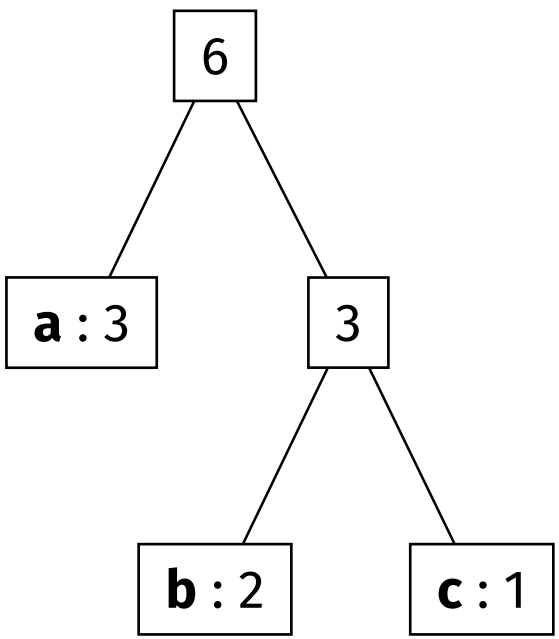
- Vi tar två noder/delträd längst fram i kön, och bygger ett nytt delträd
- Prioriteten hos delträdet är summan av frekvenserna
- Vi lägger till delträdet i kön



Huffmankodning

- Vi fortsätter tills vi bara har ett träd i kön
- Vi kan avläsa kodningen som stigar i trädet
- Vänster $=$ `0`, höger $=$ `1`
- Vi får följande kodningar:

Tecken	Kodning
a	`0`
b	`10`
c	`11`



Huffmankodning

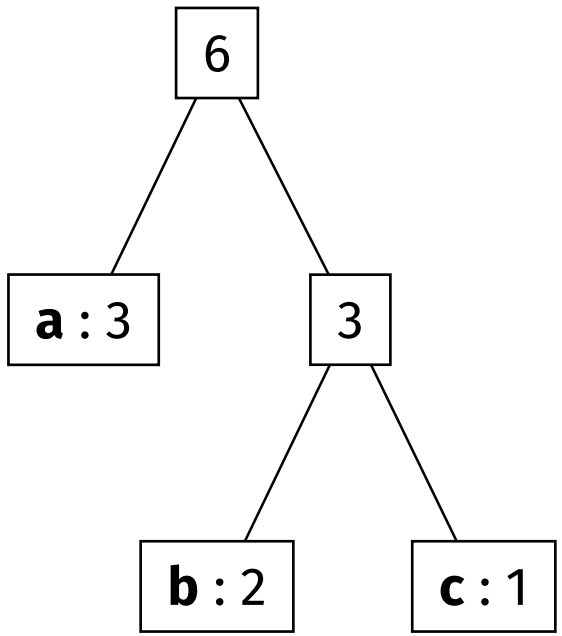
- Vår indata är **abacab**
- Den kodade strängen blir:

0	10	0	11	0	10

Tecken	Kodning
a	`0`
b	`10`
c	`11`

Huffmankodning

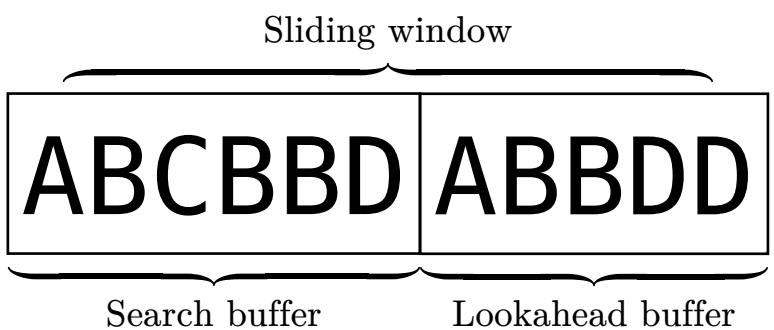
- För att avkoda behöver vi rekonstruera trädet
- Vi läser enskilda bitar och stegar genom trädet
- Trädet lagras vanligtvis i den kodade filen



0	10	0	11	0	10

Andra komprimeringsalgoritmer

- LZ77 använder ett *sliding window*
- Upprepade delsträngar ersätts med referenser
- Referenser består av avstånd, längd och nästa tecken



0,0,A	0,0,B	0,0,C	2,1,B	0,0,D	6,2,B	4,1,D

- LZW, LZSS, LZMA mm. är baserade på LZ77/LZ78
- DEFLATE är en blanding av LZ77 och Huffmankodning
- Används bl.a. i gzip

Läxa

Läxa

- Skriv ett program som kan komprimera filer med Huffmankodning
- Ska både kunna koda och avkoda
- Huffmanträdet ska lagas i filen
- *Hur kan trädet representeras för att lätt kunna serialiseras?*