Esimerkki 5.2.1

a) $p(a_i) = 1/6$, $i = 1, \ldots 6$. $H(\Sigma) = -\sum(1/6)\log(1/6) = \log 6 = 2.585$ *(bittiä/symboli). 63 symbolin jono tarvitsee keskimäärin vähintään* $63 \log 6 = 162.9$ *(bittiä).*

b) $p(a) = 5/63$, $p(b) = 10/63$, $p(c) = 1/63$, $p(d) = 10/63$, $p(e) = 23/63$, $p(f) = 14/63$. $H(\Sigma) = -(5/63)\log(5/63) - (10/63)\log(10/63) - (1/63)\log(1/63) - (10/63)\log(10/63) - (23/63)\log(23/63) - (14/63)\log(14/63) = 2.241$ *(bittiä/symboli).*

Ehdollinen entropia

- Merkkijonoille voidaan määritellä ehdollinen entropia, jossa merkin s_i todennäköisyys riippuu k:sta edellisestä merkistä. Tämä on *k:nnen asteen entropia H_k*:

$$H_k(\Sigma) = -\sum_{i=1}^{\sigma} \sum_{W \in \Sigma^k} p(Wa_i)\log(p(a_i \mid W)). \tag{5.2}$$

- Jonoa W kutsutaan merkin a_i *k-kontekstiksi*.

- Alkuperäinen entropia $H(\Sigma)$ vastaa nollannen asteen entropiaa $H_0(\Sigma)$.

- Esimerkiksi merkkijonossa $S = XWaYWb$ $p(a \mid W) = p(b \mid W) = 1/2$.

- Yleisemmin määriteltynä H_k on k:nnen asteen Markov-prosessin entropia.

- Huomaa, että merkkijonolle voidaan aina laskea sen *kokeellinen entropia*: Korvataan kukin todennäköisyys $p(a_i \mid W)$ a_i:n suhteellisella osuudella, eli *frekvenssillä*, kontekstissa W ja todennäköisyys $p(Wa_i)$ jonon Wa_i frekvensillä koko jonossa.

Redundanssi

- Keskimääräisen koodinpituuden ja entropian erotusta kutsutaan *redundanssiksi*.
- Esimerkiksi nollannen asteen mallilla

$$R = \sum_{i=1}^{\sigma} p(a_i)k_i - H_0(A), \tag{5.3}$$

missä k_i on a_i:n (keskimääräinen) koodisanan pituus (voi olla tässä reaaliluku; esimerkiksi jatkossa käsiteltävässä aritmeettisessa koodauksessa koodisana ei ole yksikäsitteinen).

- On siis edullista, että symbolin a_i koodisanan pituus k_i on mahdollisimman tarkasti $-\log p(a_i)$.
- Koodi on *universaalinen*, jos koodien keskimääräistä pituutta rajoittaa lauseke $c_1 H + c_2$, missä c_1 ja c_2 ovat vakioita ja H mallin entropia. Jos vielä $c_1 = 1$, on koodi *asymptoottisesti optimaalinen*.

Huomautus 5.2.2 *Voidaan osoittaa, että keskimääräinen koodinpituus on vähintään $H(\Sigma)$. Entropialle H_{-1} tämä seuraa huomiosta: n:n merkin mittaisia merkkijonoja on σ^n kappaletta, joten tarvitaan keskimäärin $\log \sigma^n = n \log \sigma$ bittiä esittämään yhtä jonoa; jos keskimääräinen koodinpituus olisi tätä pienempi, täytyisi kahdella jonolla olla sama koodi. Muille entropioille pätee sama argumentti kun lasketaan keskimääräistä koodinpituutta painotettuna eri jonojen todennäköisyyksillä.*

5.3 Tekstin tiivistys

5.3.1 Koodausmenetelmiä

Olkoon $\Sigma = \{s_1, \ldots, s_\sigma\}$ koodattavien symbolien joukko ja olkoon p_i symbolin s_i esiintymistodennäköisyys.

Yksinkertaisia menetelmiä

- Juoksukoodaus (run-length encoding)
  - Koodataan merkin peräkkäinen toistuminen:
    <merkki><lukumäärä>

  - Soveltuu tilanteisiin, jossa sama symboli esiintyy usein monta kertaa peräkkäin.

- Sanakirjakoodaus
  - Koodataan sanakirjassa (symbolitaulussa) esiintyvä symboli.
  - Mikäli sanakirjassa ei ole riittävästi symboleita (ts. on merkkijonoja, joiden mikään alkuosa ei ole sanakirjassa), on käytettävä koodausta
    <pakomerkki><symboli>
  - Tavallisin valintaheuristiikka: pisin täsmäys.
  - Vakiosanakirjaa käytettäessä voidaan soveltaa optimaalista jäsennystä (eli symbolien valinta). Mukautuvaa sanakirjaa käytettäessä optimaalinen jäsennys on NP-täydellinen ongelma.
  - Jos vakiosanakirjassa jonojen välillä on päällekkäisyyksiä, optimaalinen jäsennys löydetään dynaamisella ohjelmoinnilla (taulukoimalla).
  - Englanninkielistä tekstiä voidaan koodata suhteellisen tehokkaasti vakiosanakirjaa käyttäen.

- Erokoodaus
  - Koodataan lukusarja käyttäen peräkkäisten lukujen erotuksia.

- Aakkostetun luettelon koodaus
  - Koodataan alkion yhteinen alkuosa edeltäjän kanssa alkuosan pituutena.
  - Esim.

```
resume    0resume
retail    2tail
retain    5n
retard    4rd
retire    3ire
```

- Siirrä eteen -koodaus
  - Siirrä eteen -koodaus (MTF, move to front; LRU, least recently used) ylläpidetään jonoa Q, joka sisältää Σ:n symbolit.
  - Symbolin koodi on sen etäisyys jonon alusta.

– Koodauksen jälkeen symboli siirretään jonon alkuun. Siis, jos symboli esiintyy heti uudelleen, sen koodi on 0.

– Esim. merkkijonon `babbaaaaaabbbccccbbccc` siirrä eteen -koodaus on mtf=`1110100000100200010100`.

– Tulokseen sovelletaan tavallisesti edelleen jotain toista koodausta, esim. Huffman-koodausta (ks. alla). Tyypillisesti pienet etäisyydet koodataan lyhyemmin kuin isot etäisyydet.

– Soveltuu tilanteisiin, jossa sama symboli esiintyy usein monta kertaa lyhyellä alueella.

Shannon-Fano-koodaus

- Vaihtuvanmittainen alkuosakoodaus
- Toiminta
  1. Järjestä Σ niin, että $p_i \geq p_{i+1}$ kaikilla $i = 1, \ldots, n-1$.
  2. Aseta jokaisen symbolin s_i koodi ϵ:ksi.
  3. Jaa Σ kahteen osaan $\Sigma_1 = \{s_1, \ldots, s_k\}$ ja $\Sigma_2 = \{s_{k+1}, \ldots, s_n\}$ siten, että $\sum_{i=1}^{k} p_i \approx \sum_{i=k+1}^{n} p_i$.
  4. Lisää joukkoon Σ_1 kuuluvien symbolien koodin perään bitti 0 ja joukkoon Σ_2 kuuluvien symbolien koodin perään bitti 1.
  5. Toista askelia 3-4 rekursiivisesti, kunnes kukin osa sisältää vain yhden symbolin.
- Esim.

s_i	p_i
a	0,2
e	0,3
i	0,1
o	0,2
u	0,1
y	0,1

=>

s_i	ryhmitys				koodi
e	0,3	0,5		0,3	00
a	0,2			0,2	01
o	0,2	0,5	0,3	0,2	100
i	0,1			0,1	101
u	0,1		0,2	0,1	110
y	0,1			0,1	111

- Voidaan osoittaa, että Shannon-Fano-koodauksen tiivistystulos symbolia kohden on keskimäärin pienempi kuin $H+1$, missä H on mallin entropia.

Huffman-koodaus

- Vaihtuvanmittainen alkuosakoodaus
- Toiminta
  1. Aseta jokaisen symbolin s_i koodi ϵ:ksi.
  2. Muodosta joukot $\{s_1\}, \ldots, \{s_n\}$.
  3. Etsi joukot A ja B, joiden todennäköisyyksien summa on pienin.
  4. Lisää joukkoon A kuuluvien symbolien koodin eteen bitti 0 ja joukkoon B kuuluvien symbolien koodin eteen bitti 1.
  5. Yhdistä joukot A ja B.
  6. Toista askelia 3-5, kunnes jäljellä on vain yksi joukko.
- Esim.

yhdistely					s_i	koodi
{e} 0,3	{e} 0,3	{e} 0,3	{a,o} 0,4	•{u,y,i,e} 0,6	a	10
{a} 0,2	{a} 0,2	{u,y,i} 0,3	•{e} 0,3	•{a,o} 0,4	e	00
{o} 0,2	{o} 0,2	•{a} 0,2	•{u,y,i} 0,3		i	011
{i} 0,1	•{u,y} 0,2	•{o} 0,2			o	11
•{u} 0,1	•{i} 0,1				u	0100
•{y} 0,1					y	0101

- Keskimääräinen koodinpituus yo. esimerkissä.
  - Shannon-Fano: 2,5 bit/c
  - Huffman: 2,5 bit/c
  - Kiinteänmittainen koodi: 3 bit/c
  - Teoreettinen (entropia): 2,43 bit/c
- Voidaan osoittaa, että Huffman-koodauksen tiivistystulos symbolia kohden on keskimäärin pienempi kuin $H+p+0,086$, missä H on mallin entropia ja p yleisimmän symbolin todennäköisyys.
- Huffman-koodaus on optimaalinen alkuosakoodaus.
- Huffman-koodauksen virheensieto on suhteellisen hyvä.
  - Tarkastellaan jonoa `011 11 0100 10 00 11 10 = iouaeoa`

- Vaihevirhe
  1. bitti häviää: `11 11 0100 10 00 11 10 = oouaeoa`
  2. bitti häviää: `011 10 10 0100 011 10 = iaauia`
- Amplitudivirhe
  1. bitti kääntyy: `11 11 10 10 0100 011 10 = ooaauia`
  2. bitti kääntyy: `00 11 10 10 0100 011 10 = eoaauia`

- Huffman-koodaus voidaan muodostaa ajassa $\mathcal{O}(\sigma)$, jos symbolit on annettu todennäköisyysjärjestyksessä.
  - Pidetään yllä kahta jonoa nousevassa todennäköisyysjärjestyksessä: 1) alkuperäiset joukot ja 2) yhdistetyt joukot.
  - Pienimmän todennäköisyyden joukot löydetään jonojen alusta.
  - Uusi yhdistetty joukko lisätään toisen jonon loppuun, sillä se on aina siihen asti todennäköisin yhdistetty joukko.

Aritmeettinen koodaus

- Kehittäjät: P. Elias, N. Abramson, J. Rissanen, G. Langdon.
- Menetelmää käytetään useiden tiivistysmallien koodausosana.
- Olkoon $\Sigma = \{s_1, \ldots, s_{n+1}\}$ koodattavien symbolien joukko, missä s_{n+1} on loppusymboli (s_{n+1} aina päättää koodattavan merkkijonon). Olkoon p_i symbolin s_i esiintymistodennäköisyys.
- Sanoma esitetään väliin $[0, 1)$ kuuluvana reaalilukuvälinä $[a, b)$.
  - Jokaisella symbolilla s_i on oma pistevieras välinsä $[a_i, b_i)$ perusvälistä $[0, 1)$ siten, että $p_i = b_i - a_i$. Vaadimme, että $a_1 = 0$, $b_{n+1} = 1$ ja $a_i < b_i = a_{i+1}$, $i = 1, \ldots, n$, ja $a_{i+1} < 1$.
  - Koodaus
    1. Aseta $[a, b) := [0, 1)$.
    2. Suorita askel 3 jokaiselle koodattavan jonon symbolille $u = s_i$ vasemmalta oikealle.
    3. $[a, b) := [a + (b - a) \cdot a_i, a + (b - a) \cdot b_i)$.
  - Purku: koodi $[c, d)$ puretaan seuraavasti:
    1. Aseta $[a, b) := [0, 1)$.

2. Aseta $x := (c-a)/(b-a)$. Etsi i siten, että $x \in [a_i, b_i)$. Tulosta symboli s_i.
3. Päivitä väliä: $[a, b) := [a + (b-a) \cdot a_i, a + (b-a) \cdot b_i)$.
4. Toista askelia 2 ja 3 kunnes $i = n + 1$.

- Esim.

s_i	p_i	väli
`a`	0.2	$[0, 0.2)$
`e`	0.3	$[0.2, 0.5)$
`i`	0.1	$[0.5, 0.6)$
`o`	0.2	$[0.6, 0.8)$
`u`	0.1	$[0.8, 0.9)$
`$`	0.1	$[0.9, 1)$

  – Sanoma `eoe$` koodataan $[0.4082, 0.41)$:

alussa	$[0, 1)$
käsitellään `e`:	$[0.2, 0.5)$
käsitellään `o`:	$[0.38, 0.44)$
käsitellään `e`:	$[0.392, 0.41)$
käsitellään `$`:	$[0.4082, 0.41)$

  – Purussa testimuuttuja x saa arvot 0.4082, 0.694, 0.47 ja 0.9, jotka vastaavat symboleita `e`, `o`, `e` ja `$`.

- Käytännössä koodina käytetään jotain koodausvälillä sijaitsevaa lukua, esim. yllä 0.409. (Mikä tahansa välin luvuista käy.)

  – Purussa käsitellään lukua e välin $[c, d)$ sijasta, $c = e < d$. (Huomaa, ettei purkualgoritmi edes käytä d:tä.)

- Mikäli erillistä loppusymbolia ei olisi, purkaja ei havaitsisi koodin loppua, esim. 0 vastaisi yllä sanomia `a`, `aa`, `aaa`, ...

- *Ongelma*: pitkiä sanomia ei voida käsitellä tehokkaasti rajoittamattomalla tarkkuudella.

- *Ratkaisu*: paloittainen käsittely kokonaislukuaritmetiikassa.

*Aritmeettinen koodaus kokonaislukuaritmetiikassa

- Käytetään kokonaislukuaritmetiikkaa, jolloin reaalilukuväliä $[0, 1)$ vastaava perusväli voi olla esim. $[0, 2^{16})$. Olkoon h perusvälin keskikohta (esim. 2^{15}).
- Koodausväli on $[a, b)$, a ja b kokonaislukuja.
- Idea on seuraava: Lopullinen koodattava väli sisältyy kaikkiin välivaiheiden väleihin. Kunkin merkin kohdalla saadaan selville yhä suurempi osa lopullisen välin bittikoodin alkuosasta. Tulostetaan siis koodin alkuosaa sitä mukaa kun luetaan merkkejä.
- Koodaus ja purku menee suurinpiirtein seuraavasti (sivuutetaan joitain yksityiskohtia).
- Koodaus

```
while b<h or a>=h do
   if b<h then begin
      outputbit(0);
      a:=2·a;
      b:=2·b+1 end
   else begin
      outputbit(1);
      a:=2·(a-h);
      b:=2·(b-h)+1 end
```

  - Koodaus aloitetaan sanoman ensimmäistä symbolia vastaavalla välillä $[a, b)$. Koodauksen jälkeen $[a, b)$:lla on uusi arvo. Olkoon $[a', b')$ seuraavan merkin väli. Päivitetään $[a, b)$ siten, että se vastaa $[a', b')$:n osuutta sen aiemmasta arvosta. Koodataan toinen merkki, jne.
  - *Ongelma:* Jos usean merkin koodauksen aikana pätee $a < h < b$, väliä $[a, b)$ ei päästä kasvattamaan ollenkaan; väli saattaa supistua yhdeksi kokonaisluvuksi.
  - *Ratkaisu:* Kun väli supistuu alueeseen $h/2 \leq a < h \leq b < 3h/2$, skaalataan a:n ja b:n arvoja arvoiksi $2 * (a - h/2)$ ja $2 * (b - h/2) + 1$. Kasvatetaan lisäksi laskuria ℓ, joka kertoo montako kertaa peräkkäin y.o. skaalausta on tehty. Kun myöhemmässä vaiheessa tulostetaan bitti 0 (tai 1), tulostetaan lisäksi 1^ℓ (0^ℓ), ja nollataan laskuri. (Perustelu: ks. [**?**].)

  – Loppumerkin koodauksen jälkeen pätee $a < h \leq b$. Jos $h - a > b-h$ niin tulostetaan vielä 01 muuten 10. Tämä hoitaa samanlaisen tilanteen kuin yllä.

- Purku

```
while b<h or a>=h do
   if b<h then begin
      c := 2·c + inputbit();
      a:=2·a;
      b:=2·b+1 end
   else begin
      c := 2·(c-h) + inputbit();
      a:=2·(a-h);
      b:=2·(b-h)+1 end
```

  – Purku aloitetaan lukemalla c täyteen bittejä. Tämän perusteella voidaan päätellä mikä oli ensimmäinen merkki, ja saadaan rajat a ja b. Tämän jälkeen kutsutaan ylläolevaa koodia, joka täyttää c:n siten, että voidaan jälleen päätellä seuraava merkki, päivittää rajat a ja b kuten koodauksessa, jne. Tapauksen $h/2 \leq a < h \leq b < 3h/2$ huomioonottamiseksi riittää tehdä kuten else-haarassa, mutta vähentää h:n sijasta $h/2$.

- Nyt koodaus/purku toimii oikein ellei yksittäinen p_i ole liian pieni.

Yleistä aritmeettisesta koodauksesta

- Edellä malli oli kiinteä eli symbolien todennäköisyydet eivät muutu. Mukautuvassa mallissa lukuja a_i ja b_i vastaavat kumulatiiviset todennäköisyyssummat $\sum_{j=1}^{i-1} p_j$ ja $\sum_{j=1}^{i} p_j$, joita päivitetään aina, kun malli muuttuu.

- Mukautuvaa mallia voidaan hieman nopeuttaa, kun symbolit järjestetään dynaamisesti siten, että aina pätee $p_1 \leq p_2 \leq \cdots p_n$.

- Tunnetuista koodausmenetelmistä aritmeettinen koodaus antaa parhaan tiivistystuloksen. Käytännössä päästään alle prosentin päähän entropiasta.

  – Huffman-koodauksessa tarvitaan vähintään 1 bitti symbolia kohden — aritmeettisessa koodauksessa voidaan käyttää bitin osatkin.

- Ongelmia
  - Hidas
  - Huono virheensieto: yksikin virhe tekee sanoman lopun palauttamisen mahdottomaksi.
  - (Osa)ratkaisu: syöte pilkotaan blokeiksi.
  - Patentit.

5.4 Ziv-Lempel-menetelmät

- *Idea:* Käytetään sanakirjana aikaisemmin tekstissä esiintyneitä jonoja.
- Menetelmä on käytössä useissa kaupallisissa ohjelmissa — kuten kaikissa mitkä osaavat luoda/lukea .zip ja .gz päätteisiä tiedostoja.

LZ77-perhe

liukuva ikkuna

...cd $\boxed{\text{abcbaabacaadcdbca}}$ $\boxed{\text{bacaadcbac}}$ ab...

käsitelty osa kurkistusosa

- Kurkistusosan pisin alkuosa, joka *alkaa* ikkunan käsitellystä osasta koodataan <positio, pituus>, missä positiot numeroidaan oikealta vasemmalle 0, 1, 2, ...
  - Esim. yllä: <10, 7>
- Sallitaan tapaus, jossa koodattava osajono ja sen aiempi esiintymä voivat *limittyä*, eli olla osittain päällekkäin. (Tämä erikoistapaus ei tuota ongelmia purkuvaiheessa. Miksi?).

Alkuperäinen LZ77

- Koodataan kolmikkoja: <positio, pituus, seuraava merkki>
- Kun kolmikko $< a, b, c >$ on tulostettu, ikkunaa siirretään $b+1$ positiota oikealle.
- Jokaiselle kolmikon osalle on varattu kiinteä määrä bittejä: $\lfloor \log(n - f) \rfloor + \lfloor \log f \rfloor + 8$, missä n on ikkunan koko ja f kurkistusosan koko.
- Esim. `abbbbabbc` koodataan:

$< 0, 0, \texttt{a} >$	ei täsmäystä
$< 0, 0, \texttt{b} >$	ei täsmäystä
$< 0, 3, \texttt{a} >$	kurkistusosan limittäminen sallitaan
$< 2, 2, \texttt{c} >$	

- Kirjallisuudesta löytyy useita hieman tehokkaampia variantteja, kuten LZSS, gzip.

LZ78-perhe

- Sanakirjaan talletetaan merkkijonoja käsitellystä sanoman osasta. Seuraavaksi symboliksi valitaan pisin sanakirjan merkkijono, joka täsmää käsittelemättömään syötteeseen.
- Sanakirja voidaan toteuttaa hajautuksella.
- Toteutuksia: Unixin *Compress* ja *V.42bis-modeemistandardi*.

Alkuperäinen LZ78

- Alussa sanakirjassa on vain tyhjä jono, jonka indeksi 0.
- Koodi on kaksikko $< i, c >$, missä i on sen sanakirjan pisimmän merkkijonon u indeksi, joka täsmää käsittelemättömään syötteeseen ja c on tätä merkkijonoa seuraava merkki.
- Merkkijono u lisättynä merkillä c talletetaan sanakirjaan. Jos sanakirja täyttyy, se tyhjennetään (alkutila palautetaan).
- Esim. `ababcabc` koodataan $< 0, \texttt{a} >< 0, \texttt{b} >< 1, \texttt{b} >< 0, \texttt{c} >< 3, \texttt{c} >$.
- Useita variantteja, kuten LZW, LZJ, LZFG.

5.5 Burrows-Wheeler-menetelmä

- Burrows-Wheeler-muunnos (Luku 4.5.3) muuntaa tekstin helpommin tiivistyvään muotoon.
  - Esim. englanninkielisessä tekstissä `he`-alkuiset rotaatiot tulevat järjestämisen jälkeen peräkkäin, jolloin viimeisessä sarakkeessa on vastaavalla kohdalla paljon kirjaimia `t` ja `T`.
- Tavallisesti muunnettuun tekstiin sovelletaan sirrä eteen -koodausta.
- Siirrä eteen -koodaukseen sovelletaan edelleen esim. Huffman-koodausta tai aritmeettista koodausta.
- Esim. `bzip2`