Tiralabra-projekti syksy 2020

Pakkausalgoritmien vertailu –
ongelmakentän ja algoritmien esittely

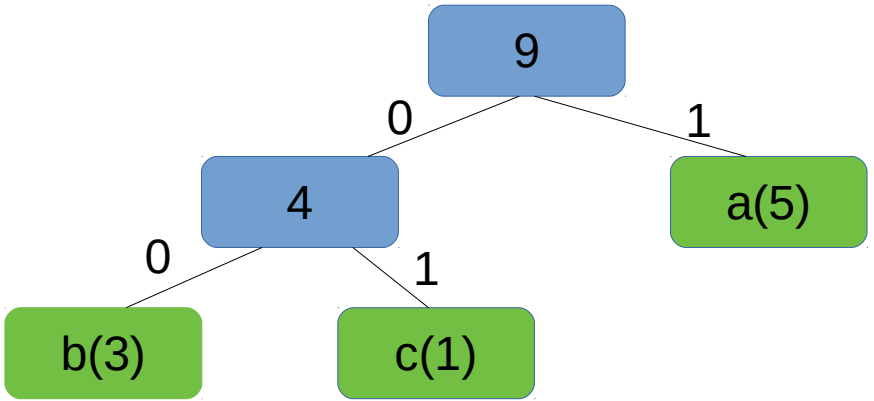
Ongelma

- Data ilmaistaan tietyllä määrällä bittejä
  - Käytännössä data kirjataan tietokoneen muistiin tavuina, eli 8 bitin paloissa.
- Voidaanko sama data ilmaista jossain tiiviimmässä muodossa (ts. vähemmällä määrällä nollia ja ykkösiä)?
  - Tavoitteena talletustilan säästäminen tai esim. verkon yli siirrettävän datan määrän minimoiminen
  - Hintana saa olla se, että purkaminen / pakkaus vie jonkin verran koneaikaa

Huffman koodaus

- Huffman algoritmissa data ilmaistaan vaihtelevan mittaisina bitteinä
- Useimmin käytetty tavu vähäisimmällä määrällä bittejä, harvimmin käytetty useammalla bitillä
  - Esim. tavun kokoiset ascii merkit ’aaaaabbbc’
    - alkuperäinen a = 97 = 01100001, b = 98 = 01100010, c = 99 = 01100011 → 011000010110000101100001011000010110000101100001011000100110001001100010011000100110001001100011
    - pakattu: a = 1, b = 00, c = 01 → 1111100000001
- Koodauksessa käytetään ns. Huffman-puuta
  - Tarvitaan purkuvaiheessa, eli pitää tallentaa tiedostoon mukaan
    - Suurentaa tiedostoa, jolloin todella pienet tiedostot voivat olla pakkauksen jälkeen saman kokokoisia tai jopa suurempia (jos toteutettu huonosti...)!

Huffman puu esimerkki



LZW pakkaus

- Lempel–Ziv–Welch pakkauksessa idea on (hyvin karkeasti yksinkertaistaen) käyttää viittauksia aikaisemmin esiintyneisiin tavuihin / tavujen yhdistelmiin.
- Muodostetaan viittauksista ”sanakirja” sitä mukaa kun dataa luetaan / pakataan
- Sanakirjaa ei tarvitse tallentaa pakatun tiedoston mukaan, vaan se voidaan generoida pakatun datan perusteella

LZW esimerkki 1

- Esim. jos alkuperäinen data 8 bitin ascii merkkejä, a = 01100001 jne.
  - Eli: aaa = 011000010110000101100001 (yht. 8+8+8=24bittiä) jne.
- ”Laajennetaan” merkistöä siten, että käytetäänkin esim. 9 bittistä ”sanakirjaa” ja lisäillään uusia tavuja ja tavuyhdistelmiä ”sanakirjaan”
  - a = 100000000, eli huonompi kuin alkuperäinen!?
  - Mutta tällöin voidaan ilmaista esim. aa = 100000001
    - Eli aaa → 100000001100000000 (yht. 9+9=18bittiä)
  - Mitä pidempiä yhdistelmiä voidaan käyttää, sitä tiiviimpään muotoon data saadaan
  - Purkaessa pitää tietää kuinka monen bitin ”sanakirjaa” on käytetty, eli tieto tästä pitää tallentaa dataan
    - Sovelluksissa voidaan tietty myös käyttää aina jotain tiettyä bittimäärää...

LWZ esimerkki

Data esim. ascii data 'aaaaabbbc'

Koodaus	Tulkinta purkutilanteessa	Sanakirja
000000000 001100001	-a -> a ('a')	256:a
100000000 001100001	256a -> a+a ('aaa')	257:aa
100000001 001100010	257b -> aa+b ('aaaaab')	258:aab
000000000 001100010	-b -> b ('aaaaabb')	259:b
100000011 001100011	259c -> bc ('aaaaabbbc')	260:bc

Tässä '-' tarkoittaa, että merkkiä ei ole ennen kohdattu, ts.
'257b' ja sitä seuraava '-b' tarkoittaa, että on luettu merkkijonoa seuraavasti:
...[a]abb... | a on sanakirjassa, luetaan eteenpäin
...[aa]bb... | aa on sanakirjassa, luetaan eteenpäin
...[aab]b... | aab ei ole sanakirjassa lisätään se sanakirjaan ja dataan
...aab[b]... | b ei ole sanakirjassa lisätään se sanakirjaan ja dataan
jne.

Muun kuin tekstin pakkaaminen?

Tämän pohjimmiltaan yksinkertaisen asian opin itse liian myöhään projektin kannalta ja kantapään kautta :(

- Entäs muu kuin ascii merkeistä koostuva data?
  - Eihän esim. video-tiedosto sisällä ascii merkkejä!? Miten sen muka voi pakata tällä periaatteella!?
  - Eihän kaikkea dataa muutenkaan ilmaista tavun kokoisissa paloissa!?
- Kaiken datan voi tulkita **ikään kuin se koostuisi ascii merkeistä**!
  - Pakkausohjelman ei tarvitse tietää mitä dataa alkuperäinen tiedosto oikeasti sisältää
  - Kaikki data on kirjattu tietokoneen muistiin tiedostoihin tavun paloissa
    - Kaikki tiedostot voidaan siis käsitellä myös tavun paloina, ikään kuin ne koostuisivat 8 bittisistä ascii merkkeistä
  - Tarkemmin sanottuna kaiken datan voi käsitellä tavuina ja pakkauksessa voi käyttää data-yksikkönä tavua (joka on käytännössä sama asia kuin ascii merkki)
  - Käytännössä pakattuihin tiedostoihin tarvitsee vain lisätä tieto siitä, mikä oli alkuperäisen tiedoston pääte
    - Purkamisen jälkeen voidaan vain purkaa data ikäänkuin se koostuisi ascii merkeistä (ts. tavuina) ja lisätä lopuksi puretun tiedoston päätteeksi alkuperäisen tiedoston pääte → käyttöjärjestelmä tietää millä ohjelmalla tiedosto pitää käsitellä