# Tiralabra 2013 periodi III
# Toteutusdokumentti

Mika Viinamäki

13. helmikuuta 2013

## 1 Ohjelman yleisrakenne

Ohjelma koostuu muutamasta, jossain määrin itsenäisestä osiosta:

- LZW-pakkaaja ja -purkaja, `kauhsa.compression.lzw`-paketti
- `BitGroup` ja siihen liittyvät apuluokat, jotka mahdollistavat tiedon kirjoittamisen ja lukemisen bitti kerrallaan, `kauhsa.utils.bitgroup`-paketti
- Oma hajautustaulu-implementaatio, `kauhsa.utils.hashmap`-paketti
- Satunnaiset apuluokat, `kauhsa.utils`-paketin juuressa
- Komentoriviltä käytettävä käyttöliittymä, `kauhsa.LZWApp`-luokka

Näiden ohjelman mukana on erinäisiä suorituskykytestejä sekä yksikkötestejä.

## 2 LZW

Algoritmi on toteutettu enimmäkseen Wikipedian LZW-artikkelin[1] perusteella. LZW ei algoritmina ota kantaa siihen, miten sen ytimenä oleva sanakirja on toteutettu — tässä implementaatiossa on käytetty sekä pakkaamisessa että purkamisessa sanakirjana hajautustaulua.

Koska LZW-toteutus pystyy pakkaamaan mitä tahansa binäärimuotoista dataa, sanakirja alustetaan kaikilla mahdollisilla yksittäisillä tavuilla (0-255). Lisäksi sanakirjaan lisätään erityinen EOF-koodi[2], jolla merkitään pakatun datan loppu. Toteutus ei kuljeta näitä tietoja pakatun datan mukana, vaan luottaa siihen että purkaja on yhteensopiva pakatun datan kanssa.

[1] `http://en.wikipedia.org/wiki/LZW`
[2] "End of File"

LZW-implementaatiossa on kaksi eri pakkaustapaa:

- "Tavallinen" pakkaus, joka pakkaa kaiken datan käyttäen samaa sanakirjaa alusta loppuun. Hyvä pakkausteho hyvin pakkautuvalla datalla, mutta saattaa olla todella hidas suurella määrällä huonosti pakkautuvaa dataa.
- Pakkaus lohkoissa, joka aloittaa pakkaamisen alusta aina kun sanakirja kasvaa liian suureksi. Nopeampi kuin tavallinen pakkaus, mutta etenkin hyvin pakkautuvalla datalla pakkausteho on yleensä jonkin verran huonompi. Sanakirjan koon yläraja on säädettävissä.

Tietyllä pakkaustavalla pakattu data on purettava sitä vastaavalla purkutavalla.

# 3 Hajautustaulu

Hajautustaulu on toteutettu Tietorakenteet ja algoritmit -kurssin luentomateriaalissa[3] esitettyä avointa hajautusta sekä kaksoishajautusta käyttäen.

## 3.1 Aikavaativuuden analysointi

Hajautustauluun lisääminen on pseudokoodina[4] esitettynä jokseenkin seuraavanlainen:

```
if current_load_factor() > MAX_LOAD_FACTOR:
        rehash(nextDoubleAsLargePrime())

for i in range(table.length):
    int tableI = hashFunction(key, i, table.length)
    if table[tableI] is None or table[tableI].key == key:
        table[tableI].value = value
        return
```

Koodista nähdään, että aina kun täyttöaste ylittää tietyn rajan, suoritetaan uudelleenhajautus. Uudelleenhajautus on aikavaativuudeltaan $\mathrm{O}(m)$, missä m on nykyisen olioiden tallentamiseen käytetyn taulukon koko. Kaksoishajautus kuitenkin kasvattaa taulun kokoa aina vähintään tuplasti — taulukon kokoina käytetään kaksoishajautuksen takia alkulukuja, joten taulukon uudeksi kooksi tulee taulukon nykyistä kokoa kaksinkertaisena seuraava alkuluku. Tuplasti edellistä isommat alkuluvut on laskettu etukäteen taulukkoon, joten niiden selvittäminen on vakioaikaista. Niinpä uudelleenhajautuksen tekeminen tarpeen vaatiessa on tasoitettuna aikavaativuutena $\mathrm{O}(1)$.

[3] `http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf`, s. 310–325
[4] Ihan oikea Java-koodi löytyy täältä.

Tietorakenteet ja algoritmit -kurssin luentomonisteissa ei erikseen analysoida aikavaativuutta avoimen hajautuksen osalta — hajautustaulun operaatiot ovat kuitenkin tasoitetusti $\mathrm{O}(1)$.