Tiralabra 2013 periodi III
Testausdokumentti

Mika Viinamäki

21. helmikuuta 2013

1 Testauksen osat

Ohjelmaan liittyvät testit voidaan jakaa kahteen eri osaan:

- Yksikkötestit, jotka testaavat että ohjelma toimii oikein.
- Suorituskykytestit, jotka testaavat ohjelman suorituskykyä erilaisilla syötteillä.

2 Yksikkötestit

Yksikkötestit testaavat algoritmin toimintaa kokonaisuudessaan sekä siihen liittyvien pienempien palasten toimintaa erikseen. Algoritmin osalta testit eivät testaa että LZW-pakkauksen muoto on sellainen niinkuin pitääkin - testeille riittää, että testidata on identtinen pakkaamisen ja purkamisen jälkeen ja että se vie pakattuna tietyn verran vähemmän tilaa kuin alkuperäinen data.

Yksikkötestit voi ajaa vaikkapa komennolla `mvn test` Netbeans-projektin juuressa tai avaamalla projekti Netbeansissa ja ajamalla testit sen kautta.

3 Suorituskykytestit

Suorituskykytestejä on muutamia erilaisia:

- `kauhsa.compression.lzw.benchmarks.PerformanceBenchmark`, joka testaa pakkaus- ja purkunopeutta erilaisilla syötteillä ja asetuksilla
- `kauhsa.compression.lzw.benchmarks.CompressionBenchmark`, joka testaa miten tiiviisti tieto pakkautuu erilaisilla syötteillä ja asetuksilla

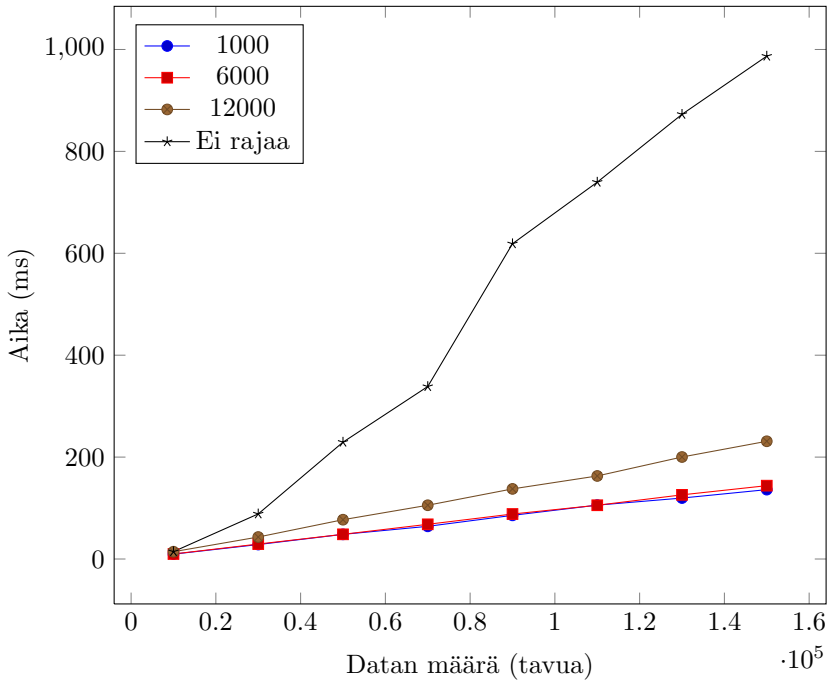
- `kauhsa.utils.hashmap.HashMapBenchmark`, joka testaa hajautustaulun tehokkuutta

Nopeutta testaavat suorituskykytestit käyttävät Caliper[1]-nimistä frameworkia, joka on suunniteltu microbenchmarkien tekemiseen. Nopeustestit tehdään ehkä tarpeettomankin suurella tarkkuudella, ja tästä johtuen niiden ajaminen saattaa kestää todella kauan — jopa puolesta tunnista tuntiin. En kuitenkaan löytänyt frameworkista mitään asetuksia mitkä olisivat ratkaisevasti parantanut testien ajamisnopeutta.

Suorituskykytestit voidaan ajaa vaikkapa ajamalla kyseinen luokka ("Run File") Netbeansin kautta.

3.1 Pakkaus- ja purkunopeus

Pakkausnopeustestien tulokset löytyvät kokonaisuudessaan täältä.



Kuva 1: Pakkausnopeus satunnaisesti generoidulla datalla

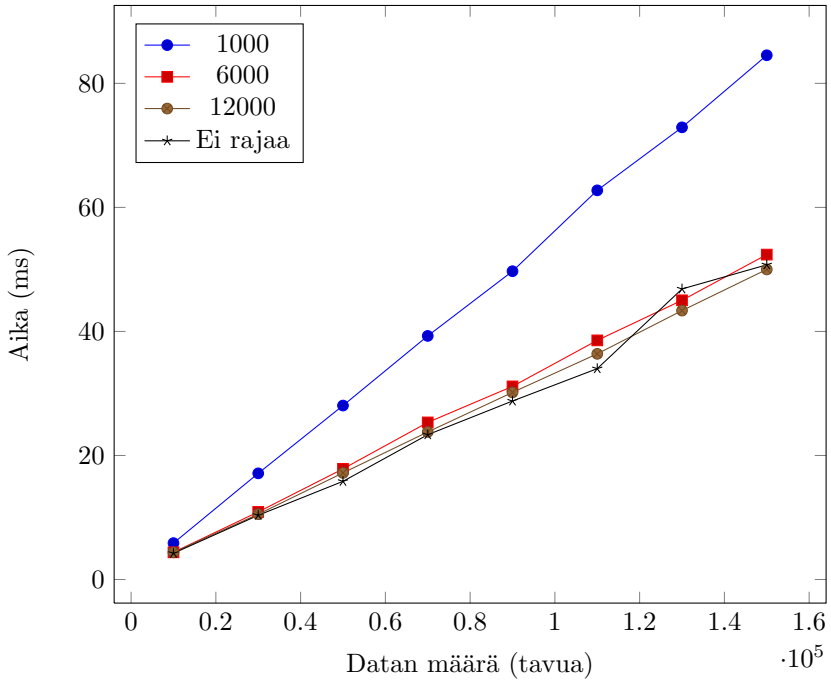
Kuvasta nähdään hyvin, miten sanakirjan rajan poistaminen vaikuttaa pakkausnopeuteen — testin perusteella se pysyy lineaarisena niin kauan kun sanakirjal-

[1] `http://code.google.com/p/caliper/`

la on jokin raja, mutta rajan poistaminen seurauksena pakkausnopeus hidastuu merkittävästi.

Syynä pakkauksen hidastumiseen nostaessa sanakirjan rajaa lienee hajautustaulun operaatioiden hidastuminen, kun siihen lisättyjen alkioiden määrä kasvaa.

Käyttämällä datana suomenkielistä tekstiä — testiaineistona käytettiin Project Gutenbergistä löytyvää *Seitsemän Veljestä*-kirjaa[2] — saadaan kovin erinäköisiä tuloksia:



Kuva 2: Pakkausnopeus suomenkielisellä tekstillä

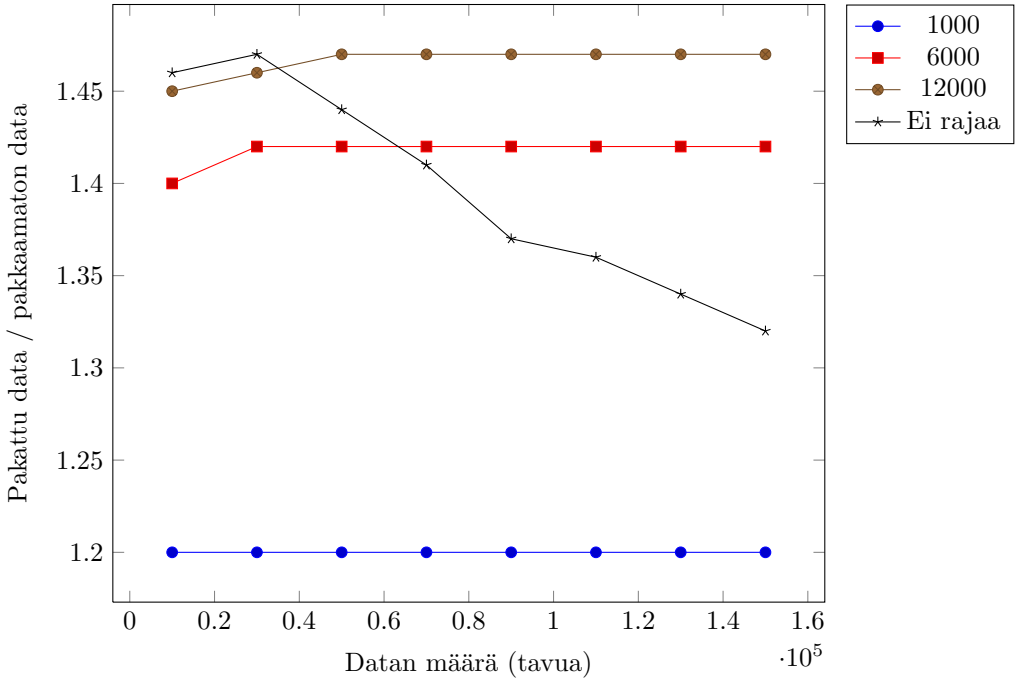
150 000 tavua dataa ei välttämättä ole tarpeeksi että eri sanakirjan sanamäärän rajojen vaikutukset pakkausnopeuteen tulisivat esille, mutta liian pienellä rajalla pakkausnopeus kärsii huomattavasti — satunnaisella datalla näin ei käynyt. Kenties hajautustaulun alkioiden määrä ei ylitä jotain rajaa, joka aiheuttaisi täyttöasteen pienenemisen ja hajautustaulun operaatioista tulisi nopeampia. Tai vaihtoehtoisesti hidasta uudelleenhajautusta joudutaan tekemään 1000 sanan rajalla jatkuvasti.

Purkunopeudessa yhtä merkittäviä eroja ei ole havaittavissa, vaikkakin sanakirjan rajan ja datan sisällön vaikutukset purkunopeuteen ovatkin samansuuntaisia kuin pakkausnopeuteen. Purkaminen on kuitenkin selkeästi pakkaamista

[2] `http://www.gutenberg.org/ebooks/11940`

nopeampaa, etenkin satunnaisella datalla.

3.2 Pakkausteho



Kuva 3: Pakkaussuhde satunnaisella datalla

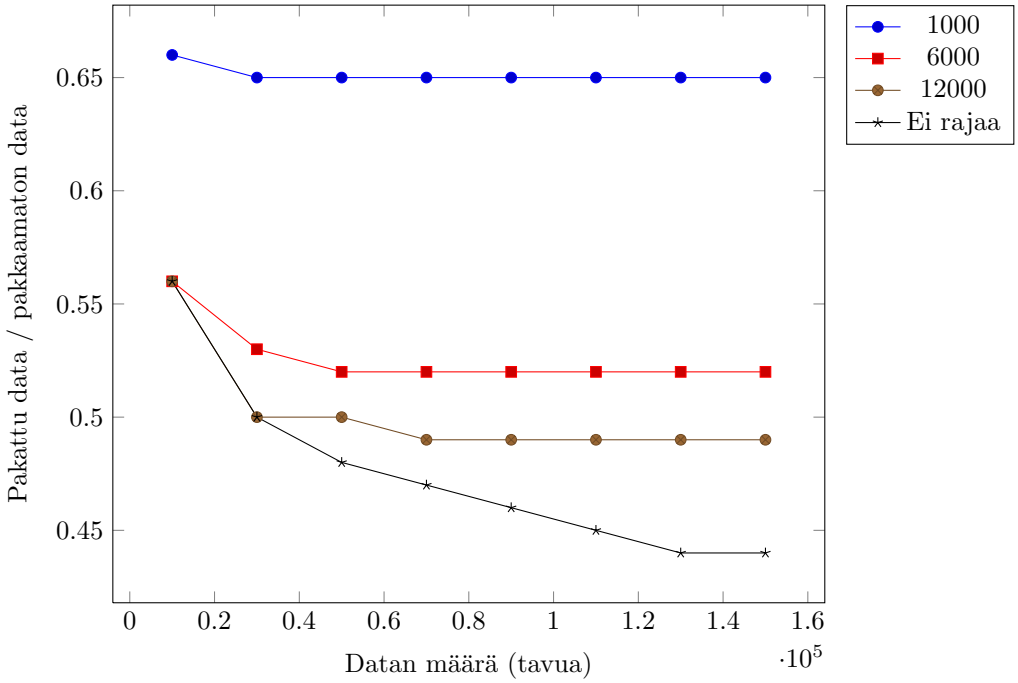
Ensimmäinen huomioitava asia satunnaista dataa pakattaessa on se, että kaikissa testatuissa tapauksissa pakkaussuhde on enemmän kuin 1 — käytännössä tämä tarkoittaa sitä, että pakattu tiedosto on *suurempi* kuin pakkaamaton tiedosto. Implementaatio ei siis sellaisenaan sovellu minkä tahansa datan pakkaamiseen, jos halutaan varmistaa että pakkaus ei kasvata tiedostokokoa.

Sanakirjan rajan muuttamisella on kuitenkin mielenkiintoisia seurauksia pakkaustehoon. Toisin kuin voisi kuvitella, sanakirjan rajan kasvattaminen — joka siis mahdollistaa useampien aiemmin esiintyneiden datanpätkien esittämisen yhdellä koodilla — ei parannakaan pakkaussuhdetta.

Ilmiö on selitettävissä sillä, että satunnaisessa datassa sanakirjasta löytyviä datanpätkiä ei käytännössä pystytä hyödyntämään lainkaan, sillä datassa ei esiinny toistuvia rakenteita. Samalla kuitenkin sanakirjaan luodaan hyvin paljon uusia data-koodi-pareja, joka aiheuttaa sen, että yhden koodin esittämiseen tarvittava bittimäärä kasvaa jatkuvasti. Jos raja on 1000, sanakirjassa ei koskaan ole sitä enempää data-koodi-pareja ja myös koodin esittämiseen tarvittava

bittimäärä pysyy alhaisena. Rajalla 6000 yhden koodin esittämiseen tarvittava bittimäärä kasvaa korkeammaksi, kuitenkaan mitenkään hyötymättä sanakirjan suuremmasta koosta.

Vasta jos raja poistetaan kokonaan sanakirjassa on niin paljon data-koodi-pareja, että niitä voidaan käyttää pakkaamisen apuna. Näin myös pakkaussuhde paranee sitä mukaa mitä enemmän dataa pakataan. Kuten aiemmin todettiin, rajan poistaminen kuitenkin aiheuttaa pakkaamiseen kuluvan ajan räjähdysmäisen kasvun.



Kuva 4: Pakkaussuhde suomenkielisellä tekstillä

Hyvin pakkautuvalla datalla tulokset ovat sellaisia kuin voisi odottaa - sanakirjan rajan kasvattaminen ja lopulta poistaminen parantaa aina pakkaussuhdetta. Kuten satunnaisellakin datalla, käyttäessä sanakirjan rajaa pakkaussuhde ei parane pakattaessa enemmän dataa — tämä johtuu sanakirjan jatkuvasta alustamisesta joka vuorostaan aiheuttaa sen, että aiemmin kerättyjä data-koodi-pareja ei voida hyödyntää sanakirjan resetoinnin jälkeen. Sanakirjan rajan poistaminen poistaa myös tämän ilmiön.

Parhaimmillaan testeissä päästiin pakkaussuhteeseen 0.44, joka siis tarkoittaa, että pakattu tiedosto on 44% alkuperäisen tiedoston koosta.