Testausdokumentti

Huffman-koodauksen ja Lempel-Ziv-koodauksen toteutusta on tarkoitus testata käyttämällä syötteenä erilaisia datatiedostoja, kuten tekstiä, kuvaa, ääntä jne. Keskeisin mielenkiinnon kohde tässä testauksessa on pakkauksen tehokkuus, eli se, kuinka paljon pienempään tilaan alkuperäinen data saadaan tiivistettyä. Tämä riippuu luonnollisesti erittäin paljon käytettävän syötteen laadusta. Testejä on tehty pääasiassa tekstitiedostoilla, ja tehtäväkuvauksessa tavoitteena mainittu 40-60% tiivistys on saavutettu.

Algoritmien aikavaativuuden tutkiminen ei poikkeuksellisesti ole tiivistämisen osalta kovin mielenkiintoinen tutkimuskohde. Kuten jo määrittelydokumentissa on esitetty, on algoritmien aikavaatimus vakiokokoista kooditaulukkoa ja sanakirjaa käytettäessä O(n). Tehokkuuden testaaminen suoritusaikoja mittaamalla on häiriöaltista, koska toteutuksessa käytetään vahvasti levy-I/O:ta, joka vie luonnollisesti paljon aikaa ja on vaikeasti ennustettavissa/toistettavissa esim. välimuistien vaikutuksen takia.

Tekstitiedosto: rautatie.txt

Aineistona on käytetty Gutenberg-projektista ladattua tekstitiedostoa, joka sisältää Juhani Ahon romaanin Rautatie, joka löytyy ohjelman juurihakemistosta nimellä rautatie.txt. Tiedoston koko on 203 075 tavua.

Tulokset – Huffman-koodaus

Pakkaamiseen kulunut aika: 764 ms, 718 ms, 749 ms; keskiarvo 744

Purkamiseen kulunut aika: 1201 ms, 1045 ms, 1029 ms; keskiarvo 1092

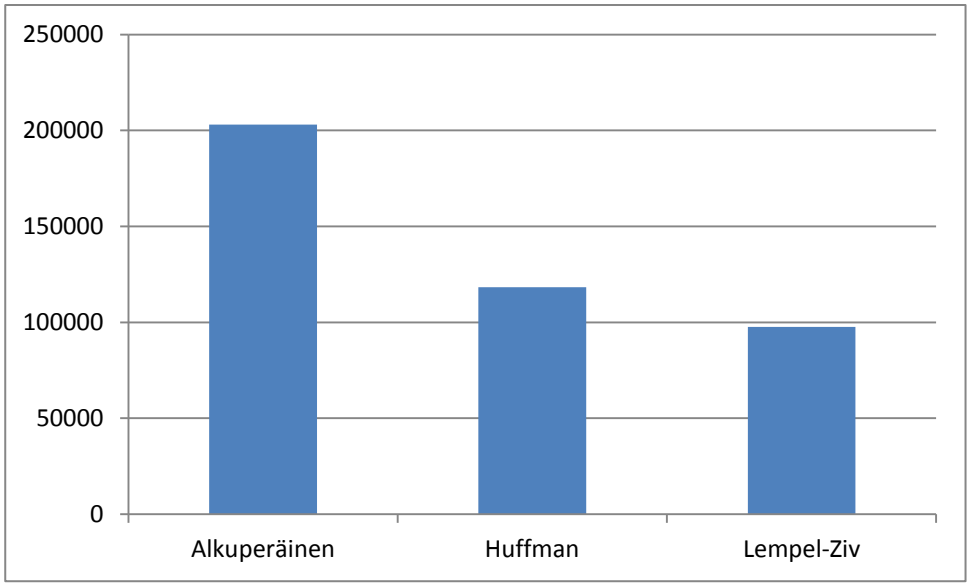
Pakatun tiedoston koko: 118 315 tavua (58 % alkuperäisestä)

Tulokset – Lempel-Ziv-koodaus

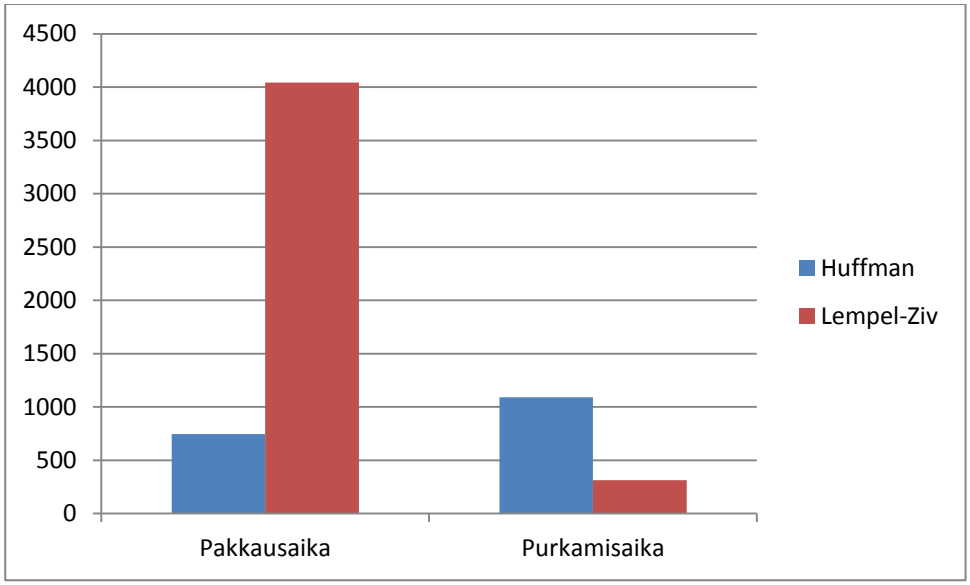
Pakkaamiseen kulunut aika: 3728 ms, 4251 ms, 4150 ms; keskiarvo 4043

Purkamiseen kulunut aika: 484 ms, 250 ms, 202 ms; keskiarvo 312

Pakatun tiedoston koko: 97 623 tavua (48 % alkuperäisestä)



Diagrammi 1: rautatie.txt – alkuperäisen tiedoston ja pakattujen tiedostojen koot tavuina



Diagrammi 2: rautatie.txt – pakkaus- ja purkamisajat Huffman- ja Lempel-Ziv –koodauksella

Tekstitiedosto: testidata.txt

Englanninkielinen The Shakespeare Story-Book Gutenberg-projektista. Tiedoston koko 666 841 tavua.

Tulokset – Huffman-koodaus

Pakkaamiseen kulunut aika: 2200 ms, 2184 ms, 2199 ms; keskiarvo 2194

Purkamiseen kulunut aika: 3120 ms, 3120 ms, 3105 ms; keskiarvo 3115

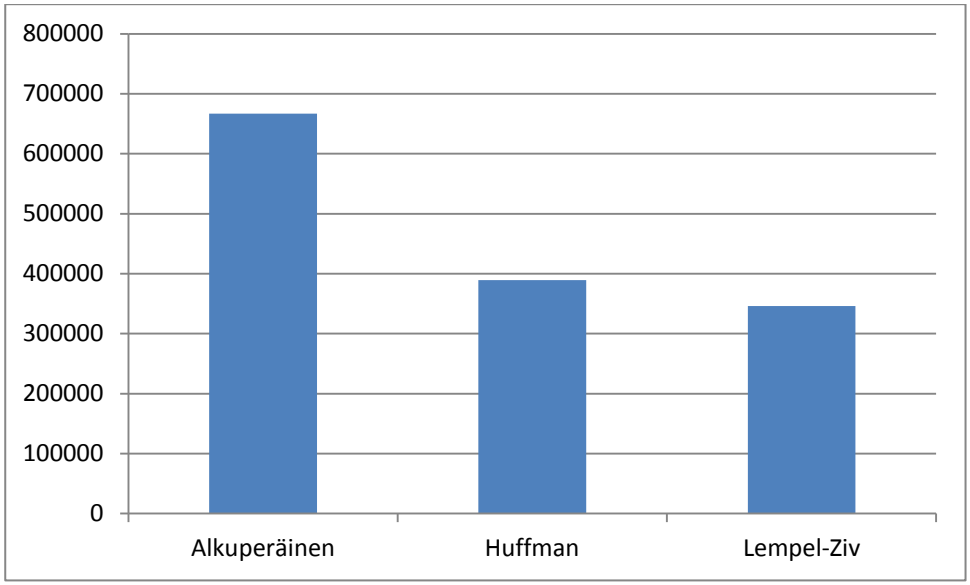
Pakatun tiedoston koko: 389 291 tavua (58 % alkuperäisestä)

Tulokset – Lempel-Ziv-koodaus

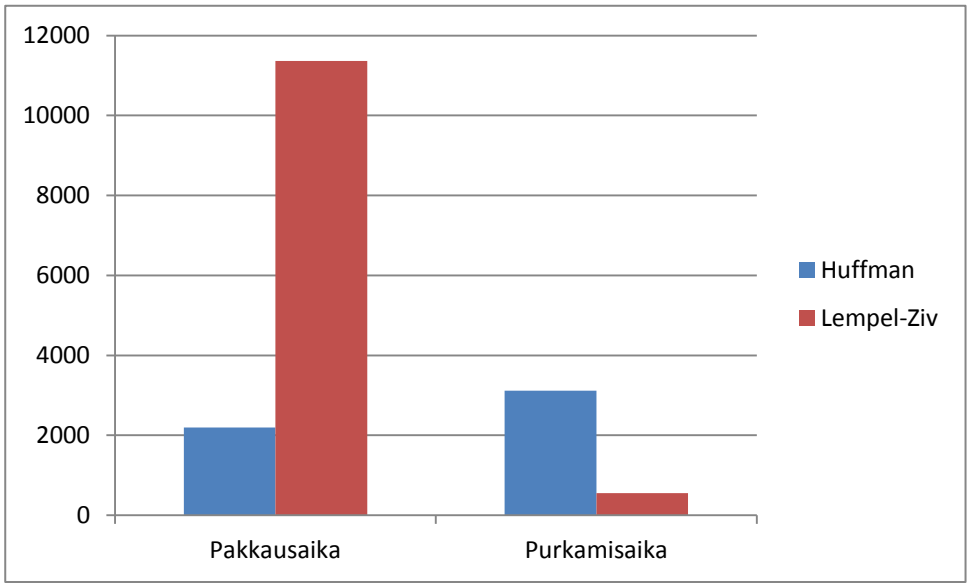
Pakkaamiseen kulunut aika: 11419 ms, 11316 ms, 11357 ms; keskiarvo 11364

Purkamiseen kulunut aika: 562 ms, 499 ms, 608 ms; keskiarvo 556

Pakatun tiedoston koko: 346 146 tavua (52 % alkuperäisestä)



Diagrammi 3: testidata.txt - Alkuperäisen ja pakattujen tiedostojen koko tavuina



Diagrammi 4: testidata.txt – pakkaus- ja purkamisajat Hoffman- ja Lempel-Ziv -koodauksella

Äänitiedosto male.wav

Aineistona on käytetty miesäänen puhetta sisältävää aaltoäänitiedostoa male.wav. Tiedoston koko on 816 496 tavua.

Tulokset - Huffman-koodaus

Pakkaamiseen kulunut aika: 2699 ms, 2745 ms, 2855 ms; keskiarvo 2766

Purkamiseen kulunut aika: 4899 ms, 4914 ms, 5179 ms; keskiarvo 4997

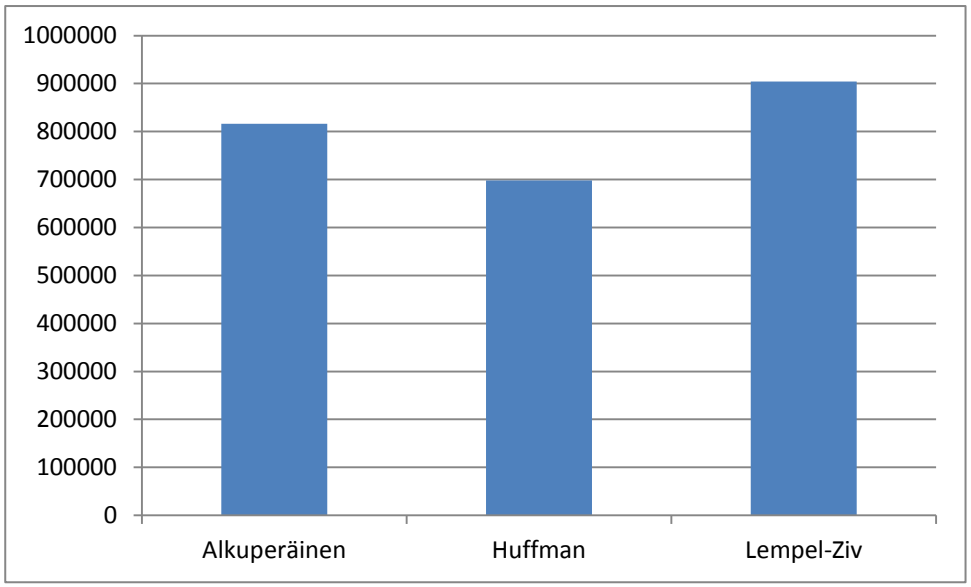
Pakatun tiedoston koko: 698 379 tavua (86 % alkuperäisestä)

Tulokset – Lempel-Ziv-koodaus

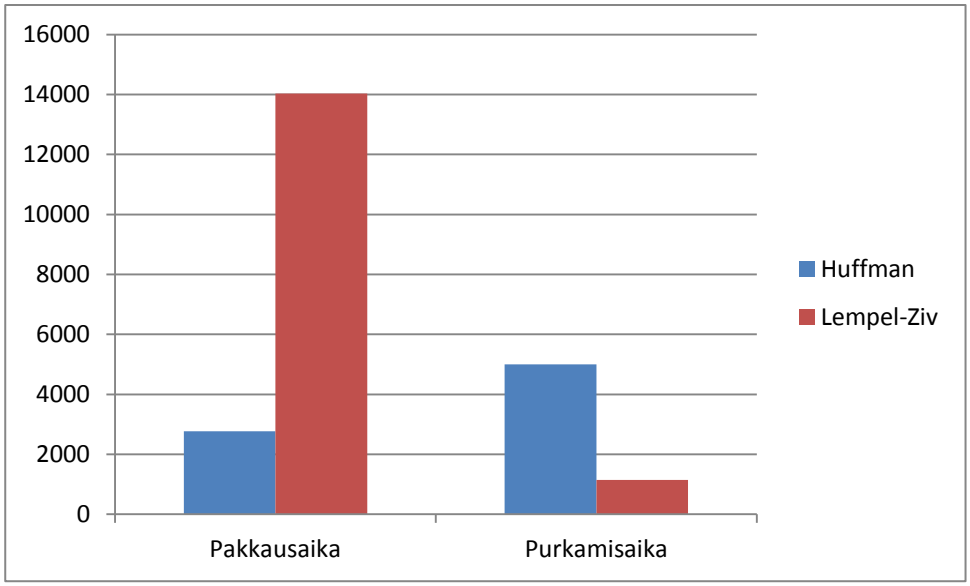
Pakkaamiseen kulunut aika: 15157 ms, 14774 ms, 14849 ms; keskiarvo 14927

Purkamiseen kulunut aika: 1389 ms, 1107 ms, 951 ms; keskiarvo 1149

Pakatun tiedoston koko: 904 047 tavua (111 % alkuperäisestä)



Diagrammi 5: male.wav - Alkuperäisen ja pakattujen tiedostojen koko tavuina



Diagrammi 6: male.wav – pakkaus- ja purkamisajat Hoffman- ja Lempel-Ziv -koodauksella

Bittikarttakuvatiedosto: ruutukaappaus.bmp

Aineistona on käytetty ruutukaappaus.bmp –tiedostoa, jonka koko on 3 098 322 tavua.

Tulokset - Huffman-koodaus

Pakkaamiseen kulunut aika: 10140 ms, 9501 ms, 9625 ms

Purkamiseen kulunut aika: 9922 ms, 8783 ms, 5975 ms

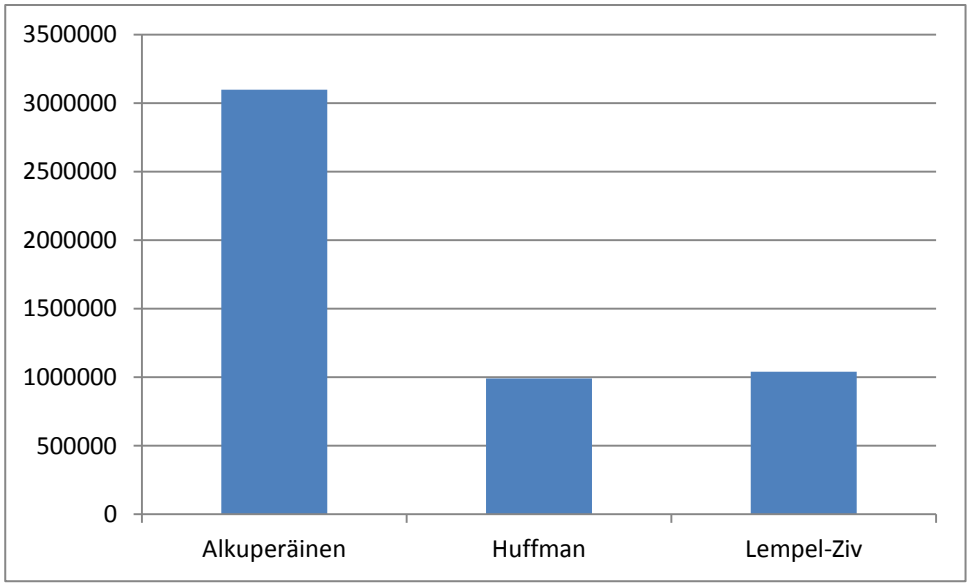
Pakatun tiedoston koko: 990 749 tavua (32 % alkuperäisestä)

Tulokset – Lempel-Ziv-koodaus

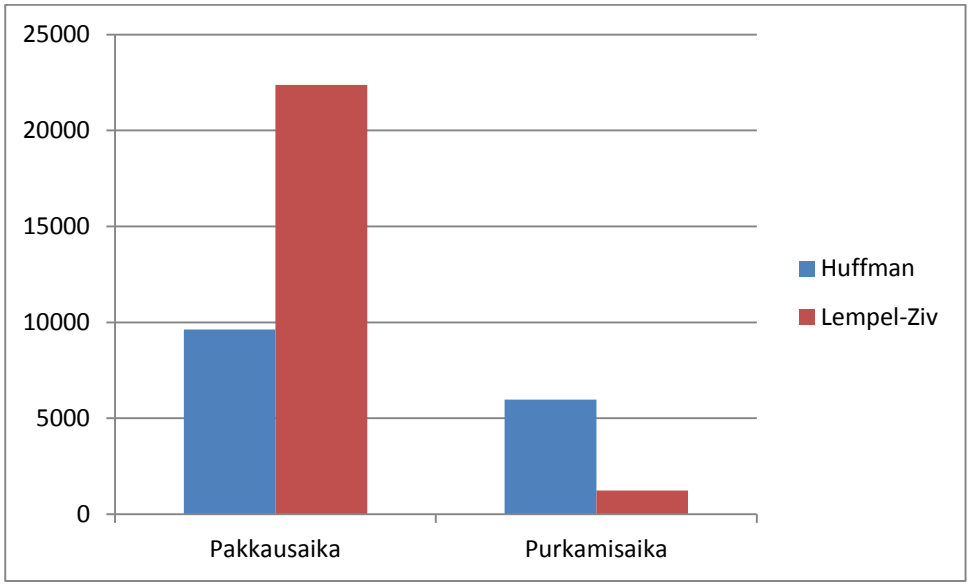
Pakkaamiseen kulunut aika: 21918 ms, 23540 ms, 21669 ms; keskiarvo 22376

Purkamiseen kulunut aika: 1233 ms, 1326 ms, 1138 ms; keskiarvo 1232

Pakatun tiedoston koko: 1 040 379 tavua (34 % alkuperäisestä)



Diagrammi 7: ruutukaappaus.bmp - Alkuperäisen ja pakattujen tiedostojen koko tavuina



Diagrammi 8: ruutukaappaus.bmp – pakkaus- ja purkamisajat Hoffman- ja Lempel-Ziv –koodauksella

Testitulosten analysointia

Tekstitiedostojen pakkaustehokkuusdiagrammeista havaitaan, että molemmat algoritmit ovat varsin tehokkaita tiivistettäessä tekstitiedostoja. Pakkaussuhde on tyypillisesti 50 %:n tienoilla. Lempel-Ziv –koodaus pärjää molemmilla tekstitiedostoilla hieman Huffmania paremmin. Pakkaus- ja purkamisaikoja kuvaavista diagrammeista on helposti nähtävissä, että kaikissa testeissä Huffman-koodauksen tiivistysosuus on ajallisesti huomattavasti nopeampi kuin Lempel-Ziv. Kuitenkin tiivistettyä dataa purettaessa Lempel-Ziv on nopeudessa ylivoimainen. Tämä ero selittyy sillä, että Huffman-algoritmissä tiivistäminen on

yksinkertainen operaatio, jossa vain korvataan jokainen tavu sitä vastaavalla kooditaulusta löytyvällä binääriesityksellä. Lempel-Ziv –koodauksessa puolestaan luodaan tiivistämisen edetessä koko ajan sanakirjaa, lisätään sinne uusia ”sanoja” ja haetaan, löytyykö vastaan tullut merkkijono jo sanakirjassa.

Tiivistetyn datan purkamisessa osat vaihtuvat. Nyt Lempel-Ziv toimii yhtä yksinkertaisesti kuin Huffman pakattaessa. Luetaan vain 12 bitin jaksoja ja korvataan ne sanakirjasta löytyvällä niitä vastaavalla tavujaksolla. Huffmanin koodausta purettaessa taas dataa luetaan biteittäin, ja jokaisen luetun bitin jälkeen on tarkastettava, löytyykö ko. bittijakso kooditaulukosta. Vertailtavaa bittijonoa kasvatetaan bitti kerrallaan aina siihen asti, kunnes vastaava bittijono löytyy kooditaulusta.

Testaamisesta

Jo ohjelman perustoiminnallisuus pitää sisällään testausta. Käyttäjän antama tiedosto pakataan ja puretaan, ja lopuksi suoritetaan testi, joka tutkii, onko purettu tiedosto yhtenevä alkuperäisen käyttäjän antaman tiedoston kanssa. Tämä alkuperäisen datan ja algoritmien käsittelemän datan vertaaminen kertoo, toimiiko algoritmi oikein. Itse ohjelmaan sisältyvän testauksen lisäksi osaa ohjelman komponenteista on testattu luonnollisesti myös JUnit-yksikkötestein. Näin on tehty erityisesti Tarkastaja-luokan osalta, jotta on voitu varmistua siitä, että itse pääohjelman suorittamissa testeissä ei ole virheitä. JUnit-testaus on suoritettu myös itse toteutetulle Prioriteettijono-luokalle, jota tarvitaan Huffman-puun generoimisessa.