Testausdokumentti

Mitä on testattu, miten tämä tehtiin

Olen pyrkinyt JUnit testeillä saavuttamaan 100% testikattavuuden ja koska aiheessani tämä tulee helposti tekemällä testi joka suorittaa tiedoston pakkauksen ja purun ja tarkistaa että tiedostot vastaavat 100% toisiaan olen myös lisäksi pyrkinyt tekemään testejä jokaiselle ohjelman metodille erikseen. Tiedostot jota JUnit testit käyttävät ovat projektissa kansiossa ”testfiles”. ”Käyttöliittymä” luokka on ainut missä ei ole junit testejä. Tähän kuitenkin laitoin koodia jonka pitäisi hylätä suurin osa virhesyötteistä. Tämän lisäksi käyttöliittymän toimivuus tuli testattua ajamalla ohjelmaa satoja kertoja erilaisilla tiedostoilla.

Lisäksi olen käsin tehnyt runsaasti suorituskykyyn liittyvää testausta, tästä tulokset ovat viikon 4 osalta tiedostossa Suorituskykytestaus.pdf ja viikon 5 osalta tiedostoissa suorituskykygraafinenesitysdemo.pdf ja suorituskykytestaus viikko5.pdf. Viikko 6 suorituskyky testaus on mukana tässä tiedostossa. Siinä käytin 6:tta eri tiedostoa ja testasin LZW ja Huffmanin algoritmeja. LZ77 jätin testauksen ulkopuolelle, koska se on pahasti keskeneräinen. Suorituskyky testauksissa käyttämäni tiedostot löytyvät projektista kansiosta ”testfiles/suorituskyky”. Kaikessa testauksessa olen käyttänyt hieman vanhaa ja hidasta kannettavaa tietokonettani. Se on varustettu i3-3110M suorittimella ja siinä on 4GB keskusmuistia. Käyttöjärjestelmänä on henkilökohtainen suosikki pingviini-jakeluversio eli Fedora21. Huomasin demossa, että laitoksen koneella algoritmit toimivat melkein kaksi kertaa nopeammin kuin kannettavallani.

Olen myös testailut viikoilla 5 ja viikoilla 6 luomani tietorakenteita. Viikon 5 tulokset löytyvät tiedostosta. Tietorakenteiden testaus. Viikon 6 tulokset ovat alla.

Huomasin viikko 5 tulosten perusteella että hashmapina ja arraylistin ovat pullonkauloja algoritmin suorituskyvyn kannalta, joten viikolla 6 parantelin niitä niin että ajamalla dataStructuresTests niillä saatiin seuraavanlaiset tulokset:

Oma Arraylist, 60809ms, Javan oma 60236 ms, Oma Hashmap 256 alkiolla 230, Oma hashmap 2^16 alkiolla 212ms, Javan HashMap 221 ms. Käytän LZW algoritmissa 2^16 alkiosta hashmapia ja huffmanin koodauksessa 256 alkioista. Tiedosto jota käytin tietorakenteita testatessa on suorituskyky testauksesta tuttu medium.txt. Eli johtopäätös, arraylistini on nyt melkein yhtänopea kuin javan oma ja LZWn käyttämä hahmappini on nopeampi kuin javan oma. Tietorakenteiden testaus on yksi kahdesta eniten merkitsevästä syystä miksi algoritmieni suorituskyky on nousut viimeisen viikon aikana huimasti.

Toinen syy tähän on, että testasin käsin kuinka paljon eri algoritmi vaiheiden suoritus kestää ja huomasin niistä ison pullonkaulan, tiedoston luvun ja kirjoituksen. Tämän korjasin laittamalla algoritmit käyttämään nopeammin toimivia input- ja outputstreamejä.

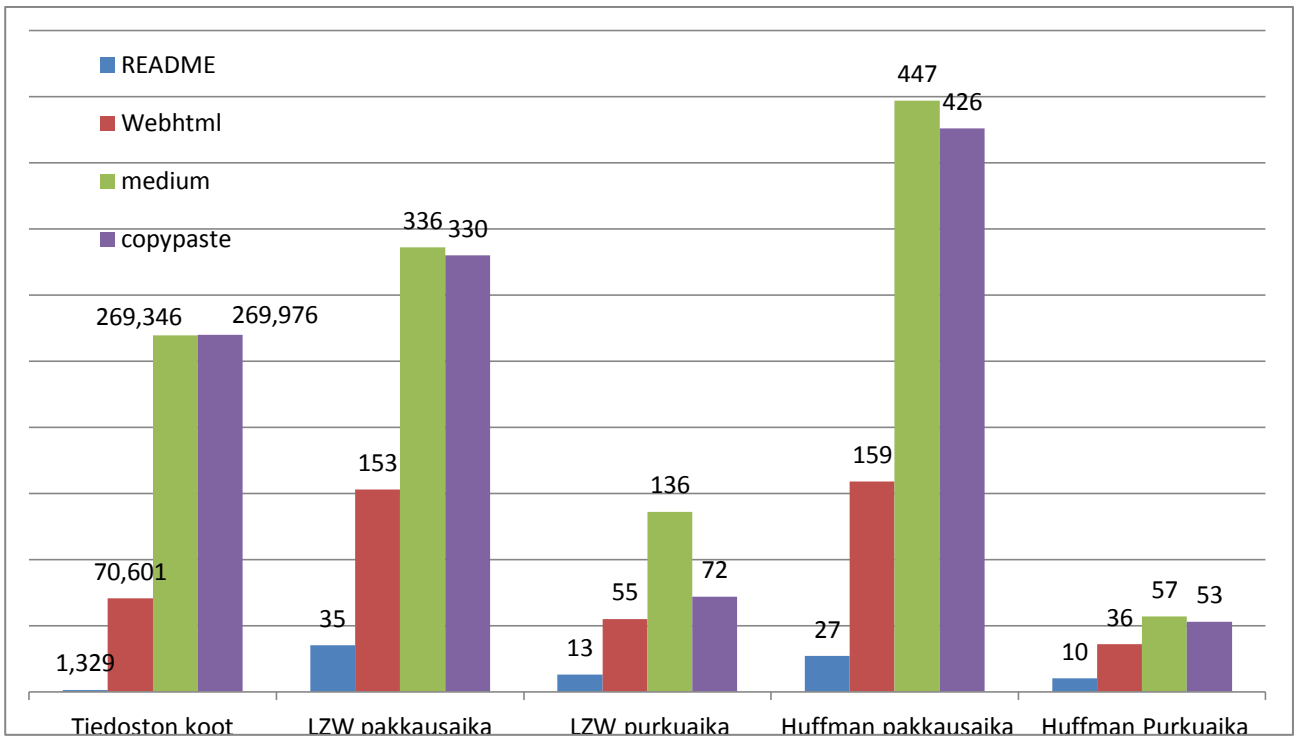
Minkälaisilla syötteillä testaus tehtiin (vertailupainotteisissa töissä tärkeätä)

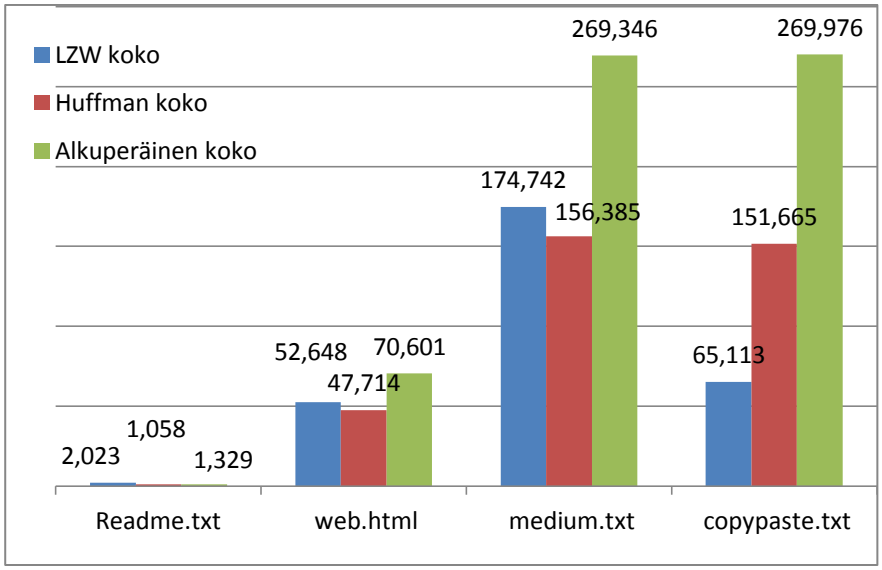
Tiedostot jota käytin suorituskyky testauksessa ovat: pieni englanninkielinen tekstitiedosto(1.3k tavua). html tiedosto(71k tavua) ja kesikokoinen copypaste tekstiä sisältävä tiedosto(269k tavua), keskikokoinen(269k tavua) englanninkielistä tekstiä sisältävä tiedosto, king james bible(5499k tavua) ja valtava liirum-laarumia sisältävä tekstitiedosto(28876k tavua). Tulokset tästä ovat alempana.

Miten testit voidaan toistaa

JUnit testit voidaan ajaa tietty automaattisesti. Lisäksi käyttäjä voi testata manuaalisesti ajamalla erilaisia tiedostoja kansiossa missä .Jar tiedosto sijaitsee, käyttöohjeen mukaan.

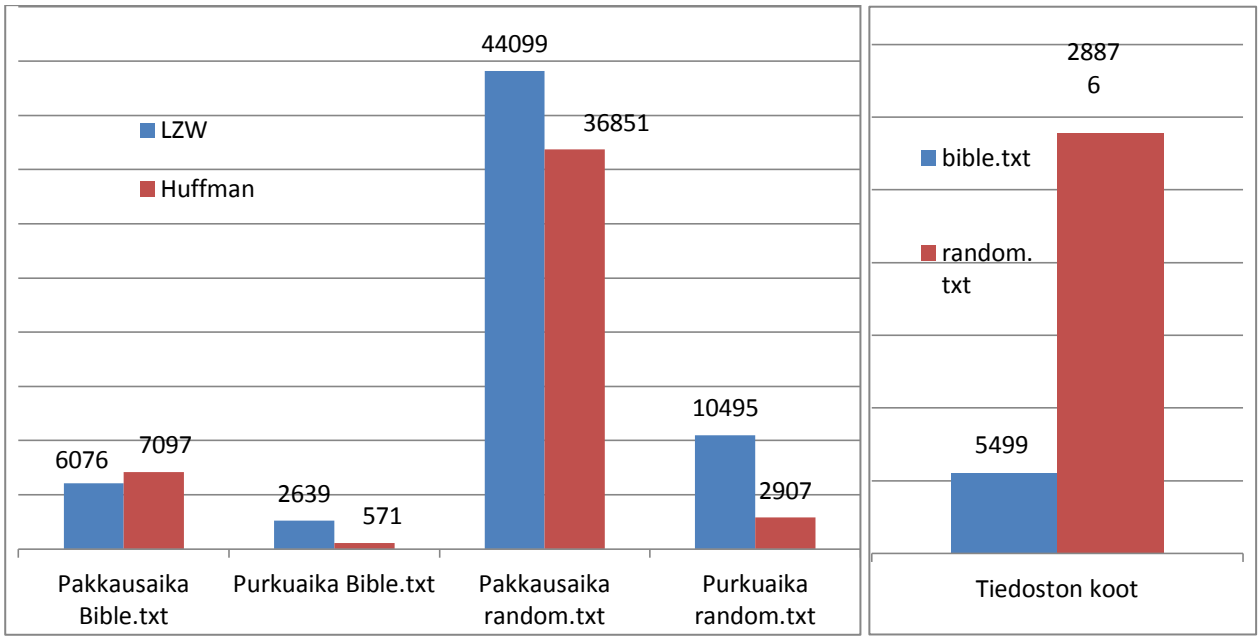
Ohjelman toiminnan empiirisen testauksen tulosten esittäminen graafisessa muodossa.





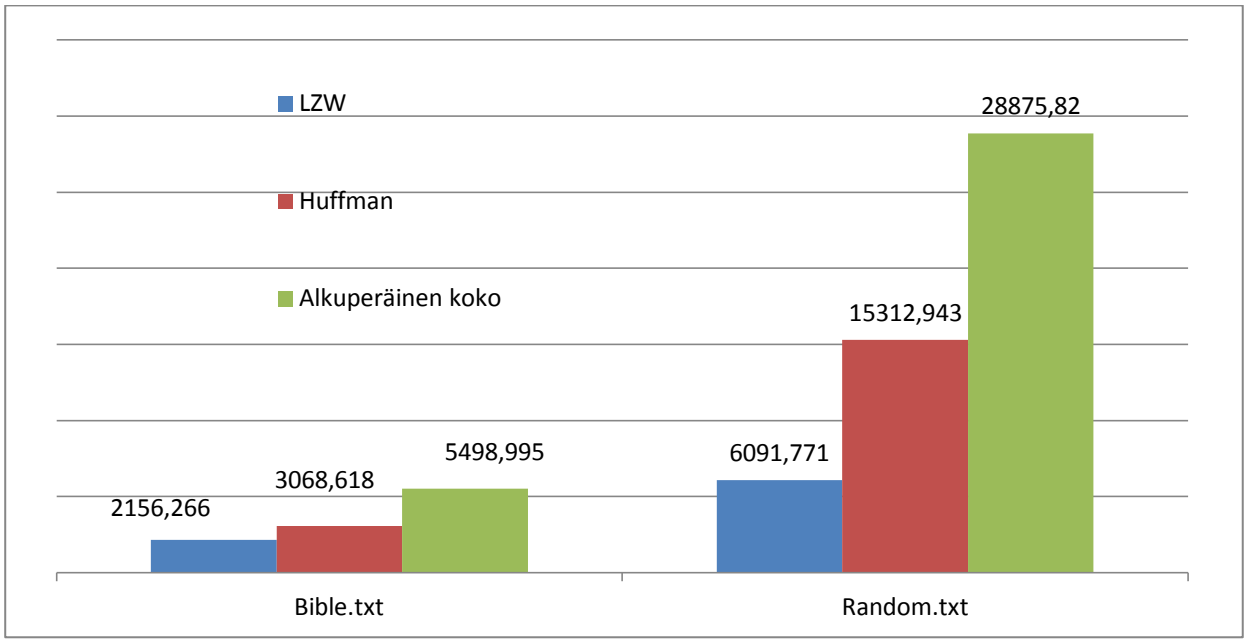
Ensimmäisessä kahdessa kuvassa on LZW ja Huffmanin koodauksella pakattu 4 eri tiedostoa. Ylemmässä kuvassa on tarkoitus havainnollistaa miten Tiedoston koko vaikuttaa pakkaus- ja purkuaikaan LZW ja Huffmaninkoodaus algoritmeilla. Tiedoston koko on kilotavuina ja pakkaus- ja purkuajat millisekunteina. Kuten näkyy tiedoston koko vaikuttaa aika lineaarisesti pakkaus ja purkuaikoihin. LZW:llä pakkaaminen on nopeampaa kuin Huffmanilla, mutta purkamisessa tilanne on toisinpäin.

Toisessa kuvassa näkyy miten tiedoston koko muuttuu pakatessa eri tiedostoja LZW ja Huffmanin algoritmeilla. Tiedoston koot ovat kaikki kilotavuina. Selkeästi voidaan huomata, että LZW algoritmi pärjää erinomaisesti jos tiedostossa on paljon copypastea. Muuten Huffmanin koodaus on hieman parempi.



Yllä olevassa kuvassa näkyy isojen tiedostojen pakkaus ja purkuajat LZW ja Huffmanin algoritmeilla. Ajat on millisekunteina ja tiedostojen koot kilotavuina. Pakatessa algoritmien nopeus

on lähellä toisiaan, mutta purettaessa Huffmanin algoritmi on selkeästi nopeampi kuin LZW algoritmi. Myöskin isoilla tiedostoissa tiedoston koko näyttäisi vaikuttavan lineaarisesti sekä pakkaus- että purkausaikoihin.



Viimeisessä kuvassa näkyy miten tiedoston koko muuttuu pakatessa isoja tiedostoja LZW ja Huffmanin algoritmeilla. Tiedoston koot ovat kaikki kilotavuina. LZW näyttäisi pärjäävän Huffmania paremmin isoissa tiedostoissa ja luultavasti random.txt liirum-laarum generaattori toisti aika paljon samoja sanoja niin tämä myös nostaa LZW:n suorituskykyä.