Toteutus ja Testaus

Mikko Peltonen 9.12.2012

Helsingin yliopisto, Tietojenkäsittelytieteen laitos

Tietorakenteiden ja algoritmien harjoitustyö, syksy 2012, 2. periodi

ohjaaja Kristiina Paloheimo

Sisältö

1. Johdanto

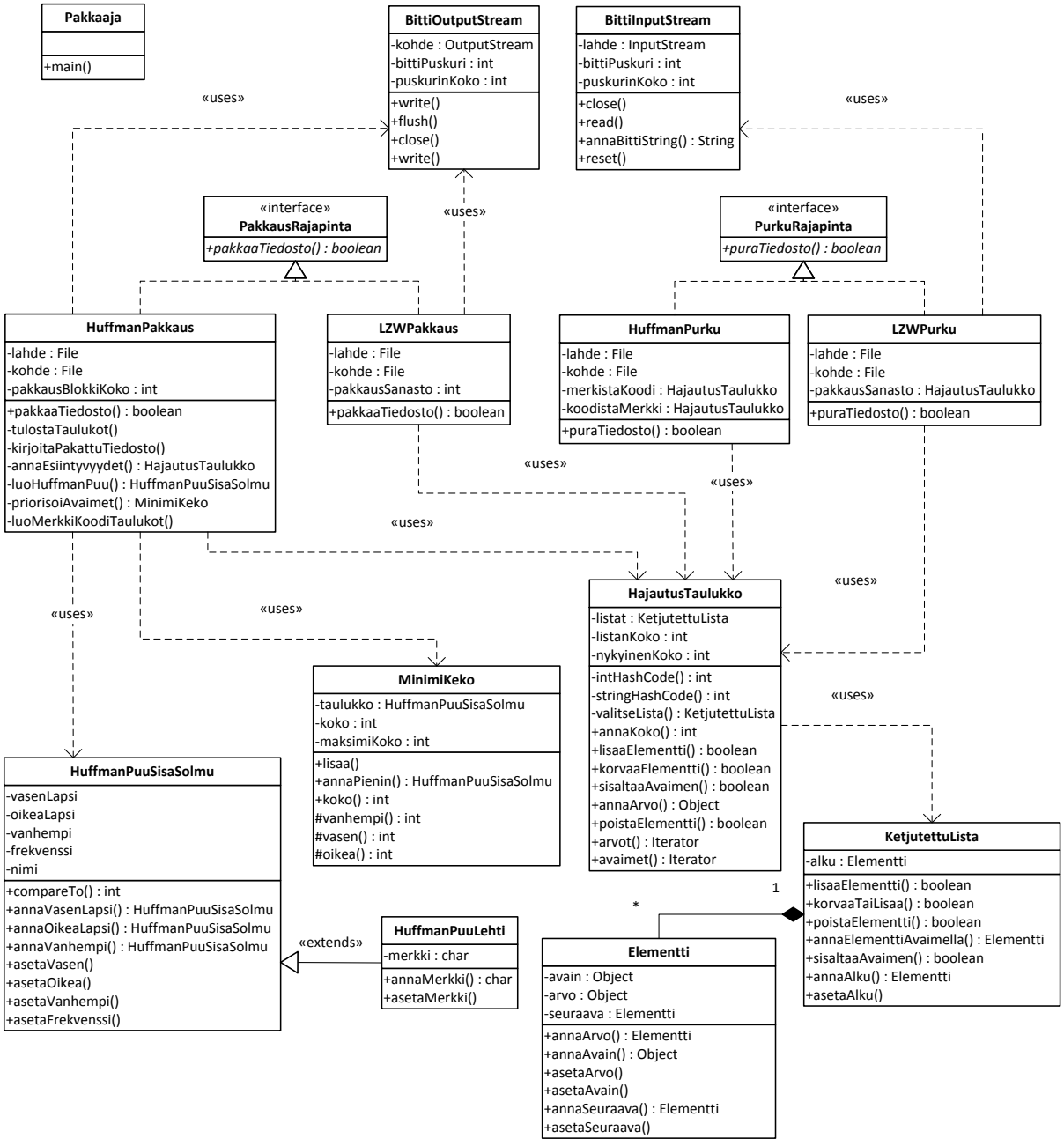
Harjoitustyöni alkuperäinen aihe oli Huffman-koodiin perustuva tiedon pakkaus- ja purkuohjelma. Varsin pian toteutuksen alettua alkoi kuitenkin näyttämään siltä, että aikaa riittäisi myös ennalta sovitun mahdollisen laajennuksen, Lempel-Ziv-pohjaisen algoritmin toteuttamiseen. Lopullinen ohjelma toteuttaa siis molemmat algoritmit. Ohjelma toimii komentoriviltä, ja ottaa argumenteikseen operaation (-pakkaa tai -pura), sekä lähde- että kohdetiedostojen nimet tässä järjestyksessä.

Tämä dokumentti kertoo miten ohjelma on toteutettu ja testattu, ja miten alkuperäiset tavoittelliset aikavaativuudet ovat toteutuneet lopullisessa työssä.

2. Toteutus

2.1. Luokkakaavio

Suoritettavan main()-metodin sisältävä luokka on Pakkaaja. Metodia kutsumalla oikeilla parametreila, tämä main() instantioi Huffman- tai LZW toteutusluokan käytettävän algoritmin ja operaation perusteella. Algoritmin valinta tapahtuu system propertyn ”pakkausalgoritmi” perusteella. Jos tämä system property on asetettu arvoon ”lzw”, käytetään LZW-pakkausta, muutoin Huffman-pakkausta. Alla oleva luokkadiagrammi kuvaa kaikki ohjelman toteutuksessa käyteyt luokat.



Kuva 1: Luokkakaavio.

BittiInputStream ja BittiOutputStream ovat luokkia jotka lukevat ja kirjoittavat yksittäisiä bittejä levyltä. Todellisuudessa levyltä luonnollisesti luetaan / sinne kirjoitetaan tietenkin tavuja, joten nämä luokat käyttävät sisäistä puskuria jossa säilytetään aina yksi ylimääräinen tavu, jossa bittejä käsitellään.

Tietorakenneluokat ovat näkyvissä luokkadiagrammin alaosassa. HuffmanPuuSisaSolmu ja HuffmanPuuLehti toteuttavat Huffman-pakkauksessa tarvittavan Huffman-puun. Minimikekoa käytetään Huffman-pakkauksen tarvitsemana prioriteettijonona. Hajautustaulukkoa käytetään kaikissa pakkaus- ja purkuoperaatioissa sanaston säilyttämiseen. Hajautustaulukko käyttää tietorakenteenaan ketjutettua listaa (KetjutettuLista), joka koostuu Elementeistä. Tietorakenteiden toteutuksesta lisää

2.2. Tiedostojen käsittely

Ohjelma käyttää java.io-paketin tarjoamia BufferedInputStream ja BufferedOutputStream-luokkia kaikkien I/O operaatioiden puskurointiin.

2.2.1. Huffman

Sanasto kirjoitetaan pakattaessa tiedostoon Javan serialisaatiomekanismia käyttäen. Sanastoa käyttämällä pakattu data kirjoitetaan Huffman-koodeina (jotka ovat pituudeltaan 1 bittiä tai pidempiä) tiedostoon suoraan sanaston perään.

2.2.2. LZW

Sanastoa ei Lempel-Ziv algoritmeissa kirjoiteta pakattuun tiedostoon, vaan purkualgoritmi kokoaa sanaston ajonaikaisesti itse. LZW-algoritmin tiedostoon kirjoitetaan ainoastaan pakattua dataa bitteinä.

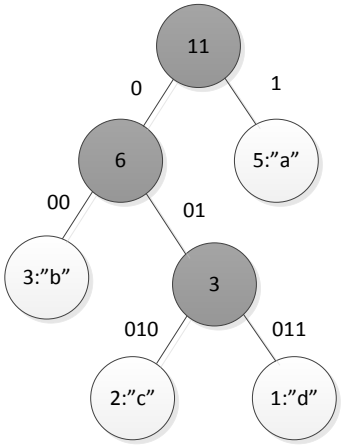
2.3. Käytetyt tietorakenteet ja niiden totetukset

Kaikki tietorakenteet ovat paketissa com.mtspelto.pakkaus.tietorakenteet.

2.3.1. Huffman-puu

Huffman-puuta tarvitaan Huffman-algoritmissa Huffman-koodien luomiseen, kun ensin on selvitetty pakattavan datan sisältämien merkkien esiintyvyydet.

Merkit lisätään Huffman-puuhun esiintyvyyden perusteella siten, että useimmin esiintyvät merkit ovat puussa lähimpänä juurta. Huffman-koodi mille tahansa merkille luodaan hakemalla merkki puusta. Kun puussa siirrytään vasemmalle, lisätään koodiin ”0”. Kun puussa siirrytään oikealle, lisätään koodiin ”1”.



Kuva 2: Huffman-puu merkkijonolle ”aaaaabbbccd”.

Huffman-puun toteutus perustuu kahteen luokkaan, HuffmanPuuSisaSolmu ja sitä laajentava HuffmanPuuLehti. HuffmanPuuSisaSolmu kuvaa yhtä solmua puussa. Koska Huffman-puussa vain lehdissä säilytetään arvoja, lehti toteutuetaan eri luokassa. HuffmanPuuLehti lisää HuffmanPuuSisaSolmuun arvon asettamiseen ja palauttamiseen tarvittavat metodit. Huffman-puun tukemat operaatiot ovat kuvattu taulukossa 1.

Taulukko 1: Huffman-puun tukemat julkiset operaatiot.

Metodi	Toteutusluokka	Kuvaus
asetaVanhempi()	HuffmanPuuSisaSolmu	Asettaa tämän solmun vanhemmaksi parametrina annetun HuffmanPuuSisaSolmun
asetaVasen()	HuffmanPuuSisaSolmu	Asettaa tämän solmun vasemmaksi lapseksi parametrina annetun HuffmanPuuSisaSolmun
asetaOikea ()	HuffmanPuuSisaSolmu	Asettaa tämän solmun oikeaksi lapseksi parametrina annetun HuffmanPuuSisaSolmun
asetaFrekvenssi()	HuffmanPuuSisaSolmu	Asettaa tämän solmun esiintyvyyden
annaFrekvenssi()	HuffmanPuuSisaSolmu	Palauttaa tämän solmun esiintyvyyden
annaVanhempi()	HuffmanPuuSisaSolmu	Palauttaa tämän solmun vanhemman, tai null jos tämä solmu on juuri
annaVasen()	HuffmanPuuSisaSolmu	Palauttaa tämän solmun vasemman lapsen, tai null jos vasenta lasta ei ole
annaOikea()	HuffmanPuuSisaSolmu	Palauttaa tämän solmun oikean lapsen, tai null jos oikeaa lasta ei ole
asetaMerkki()	HuffmanPuuLehti	Asettaa tämän lehden kuvaaman merkin
annaMerkki()	HuffmanPuuLehti	Palauttaa tämän lehden kuvaaman merkin

2.3.2. Hajautustaulukko

Hajautustaulukko käyttää tietorakenteenaan yksinkertaisesti ketjutettuja listoja (ts. Listan jokainen elementti sisältää linkin seuraavaan, mutta ei edelliseen elementtiin). Ketjutettu lista toteutetaan luokassa KetjutettuLista ja sen elementti luokassa Elementti.

HajautusTaulukko hajauttaa lisätyt elementit listoihin hyvin alkeellisen itse tehdyn hajautusalgoritmin avulla. Kokonaisluvut lisätään paikkaan <lisätty arvo> mod <listan koko>. Merkkijono, tai muut objektiarvot lisätään paikkaan <lisätty arvo>.hashcode() mod <listan koko>.

Hajautustaulukko osaa myös palauttaa java.util.Iterator rajapinnan toteuttavat oliot taulukon avaimista sekä arvoista. Iterator-rajapinnan toteutukset ovat tehty Hajautustaulukko-luokan sisällä yksityisinä sisäluokkina (AvaimetIterator ja ArvotIterator).

Hajautustaulukon tukemat operaatiot ovat kuvattu taulukossa 2.

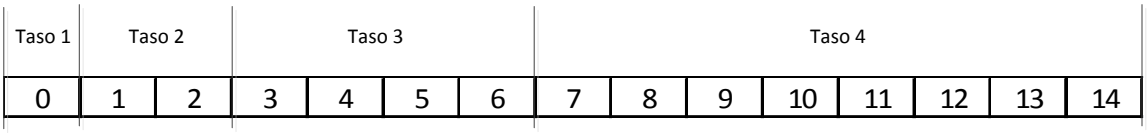
Taulukko 2: Hajautustaulukon tukemat julkiset operaatiot.

Metodi	Palautustyyppi	Kuvaus
annaKoko()	int	Palauttaa tämän hajautustaulukon nykyisten avainten määrän
lisaaElementti()	boolean	Lisää hajautustaulukkoon parametrina annetun objektin. Palauttaa true, mikäli lisäys onnistui, muutoin false
korvaaElementti()	boolean	Lisää hajautustaulukkoon parametrina annetun objektin, tai jos avain on jo oleamassa, korvaa avaimen arvon annetulla arvolla. Palauttaa true, mikäli lisäys tai korvaus onnistui, muutoin false
sisaltaaAvaimen()	boolean	Palauttaa true, mikäli hajautustaulukko sisältää parametrina annetun avaimen, muutoin false
annaArvo()	Object	Palauttaa parametrina annettua avainta vastaavan elementin arvon
poistaElementti()	boolean	Poistaa parametrina annetun elementin
arvot()	Iterator	Palauttaa java.util.Iterator instanssin tämän hajautustaulukon arvoista
avaimet()	Iterator	Palauttaa java.util.Iterator instanssin tämän hajautustaulukon avaimista

2.3.3. Minimikeko

Huffman-algoritmissa merkkien järjestäminen esiintyvyyksien perusteella tehdään prioriteettijonon avulla. Prioriteettijono toteutettiin minimikekona luokassa MinimiKeko. Minimikeko käyttää tiedon säilyttämiseen taulukkorakennetta. Taulukon suuruus kannattaa aina määritella joksikin riittävän suuren kahden potenssin arvoksi (vaikka oikeastaan vain 2^x -2 solua on kulloinkin käytössä).

Elementit järjestetään taulukkoon tasoittain pienimmästä alkaen. Keon pienin elementti on siis aina kohdassa taulukko[0]. Toisen tason elemetit ovat kohdassa taulukko[1] ja taulukko[2] jne. Näin kunkin elementin oikea paikka taulukossa on siis helppo laskea.



Kuva 3: Minimikeon elementtien sijoitus taulukkorakenteeseen.

Keko-ominaisuutta valvotaan lisättäessä ja poistettaessa elementtejä. heapifyAlaspain() ja heapifyYlospain() metodeita kutsutaan kun kekoon on lisätty tai siitä on poistettu elementti. Nämä operaatiot käyvät keon läpi annetusta alkupisteestä alkaen ja palauttavat keko-ominaisuuden.

Taulukko 3: Minimikeon tukemat julkiset operaatiot.

Metodi	**Palautustyyppi**	**Kuvaus**
koko()	int	Palauttaa tämän minimikeon nykyisten elementtien määrän
lisaa()	boolean	Lisää minimikekoon parametrina annetun objektin keko-ominaisuuden säilyttäen. Palauttaa true, mikäli lisäys onnistui, muutoin false.
annaPienin()	HuffmanPuuSisaSolmu	Palauttaa keosta pienimmän esiintyvyyden sisältävän HuffmanPuuSisaSolmun

Operaatiot vasen(), oikea() ja vanhempi() ovat vain MinimiKeon sisäisesti tarvitsemia operaatioita, mutta koska yksikkötestiluokka tarvitsee pääsyä näihin, ovat ne määritelty näkyvyydellä ”protected”.

3. Testaus

Toiminnallinen testaus suoritettiin JUnit-yksikkötesteillä. Yksikkötestit HuffmanPakkaus / HuffmanPurku, sekä LZWPakkaus / LZWPurku-luokille tehtiin siten, että ne pakkasivat ja sitten purkivat 5 erikokoista tiedostoa. Purettua ja alkuperäistä tiedostoa vertailemalla voitiin varmistua kaikkien luokkien toimivuudesta.

Testauksessa mitattiin suoritusaika ohjelman sisäisellä laskurilla, joka ottaa talteen koneen kellonajan ennen suoritusta, ja laskee suorituksen jälkeen kellonaikojen erotuksen ja tulostaa suoritusajan konsolille suorituksen lopuksi. Tällä mittaustavalla eliminoitiin Java-virtuaalikoneen käynnistymiseen ja alustamiseen kuluva aika joka varsinkin pienillä tiedostoilla vääristäisi testituloksia todella paljon.

3.1. Testiaineisto

Testeissä käytettiin seuraavia syötteitä:

1. 180 kB englanninkielinen tekstitiedosto (Charles Dickens: A Christmas Carol)
2. 500 kB XML-tiedosto
3. 13 MB englanninkielinen tekstitiedosto (kokoelma klassikoita)
4. 20 MB XML-tiedosto
5. 115 MB XML-tiedosto

Testiaineiston tekstitiedostot ovat peräisin Project Gutenbergistä (6), XML-tiedostot ovat generoitu Xmark XMLgen-työkalulla (7).

3.2. Testiympäristön tiedot

Kaikki testit ajettiin samalla Dell E6330 kannettavalla tietokoneella. Testiympäristön tärkeimmät tiedot on lueteltu alla.

- Tietokoneen merkki ja malli: Dell Latitude E6330
- Prosessorien lukumäärä, tyyppi ja kellotaajuus: 1*Intel Core i5-3360M @ 2.8 Ghz
- Suoritinytimien lukumäärä: 2 (HyperThreading kytketty päälle)
- Muistin määrä ja tyyppi: 4 GB DDR3 PC3-12800
- Käyttöjärjestelmä: Windows 7 Service Pack 1 (64-bit)
- Java-virtuaalikone: Oracle Java 1.7.0_09-b05, Client HotSpot VM, Mixed Mode 64-bit

3.3. Yleisiä testihavaintoja

Kummatkin algoritmit toimivat mielestäni yllättävän nopeasti. Algoritmit käyttäytyvät kuitenkin hyvin eri tavoin. Huffman-algoritmissa pakkaus on huomattavasti purkamista nopeampaa. Tätä selittänee pakkauksessa käytettävät erittäin tehokkaat tietorakenteet (Huffman-puu ja minimikeko), kun taas purkamisessa erittäin suureen merkitykseen nousee Hajautustaulukon hakuoperaatio, jossa VisualVM:stä keräämieni näytteiden mukaan kulutetaan paljon aikaa purkuoperaation aikana.

LZW-algoritmissa asetelma on toisin päin, eli pakkaus vie huomattavasti enemmän aikaa kuin purkaminen.

3.4. Testitulokset

3.4.1. Huffman

Huffman-pakkauksen ja purkamisen testitulokset on koottu allaoleviin taulukkoihin. Pakkaus / Purku kB / sek on tunnusluku jolla eri pakkaus / purkuoperaatioiden nopeutta voi vertailla.

Taulukko 4: Huffman-pakkauksen testitulokset.

Pakkaamaton koko (tavua)	Pakattu koko (tavua)	Kesto (ms)	Pakkaus kB / sek	Pakkausteho %
182021	108737	170	1045.62	59,73%
581166	353545	190	2987.08	60,83%
12981173	7548537	1991	6367.12	58,14%
23924964	14445670	3021	7733.94	60,37%
118552713	71697108	14452	8010.94	60,47%

Taulukko 5: Huffman-purun testitulokset.

Pakkaamaton koko (tavua)	Kesto (ms)	Purku kB / sek
182021	301	590.55
581166	430	1319.87
12981173	9140	1386.97
23924964	15371	1520.02
118552713	70202	1649.16

3.4.2. LZW

LZW-pakkauksen ja purkamisen testitulokset on koottu allaoleviin taulukkoihin. Pakkaus / Purku kB / sek on tunnusluku jolla eri pakkaus / purkuoperaatioiden nopeutta voi vertailla.

Taulukko 6: LZW-pakkauksen testitulokset.

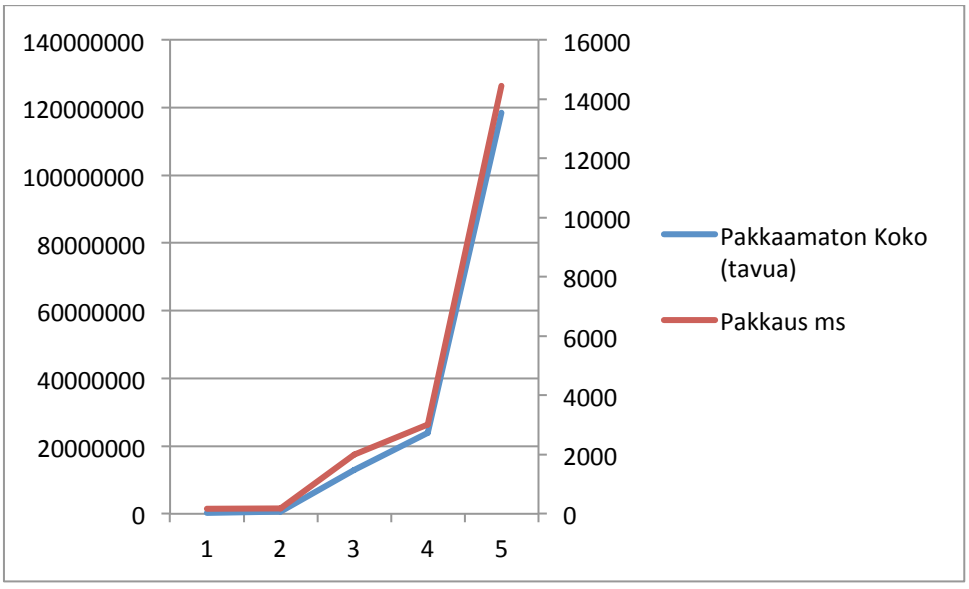
Pakkaamaton koko (tavua)	Pakattu koko (tavua)	Kesto (ms)	Pakkaus kB / sek	Pakkausteho
182021	102104	260	683.67	26,09
581166	314357	470	1207.54	54,09
12981174	7142310	8801	1440.40	55,02
23924964	12998031	15931	1466.59	54,33
118552713	64516701	78632	1472.35	54,42

Taulukko 7: LZW-purun testitulokset.

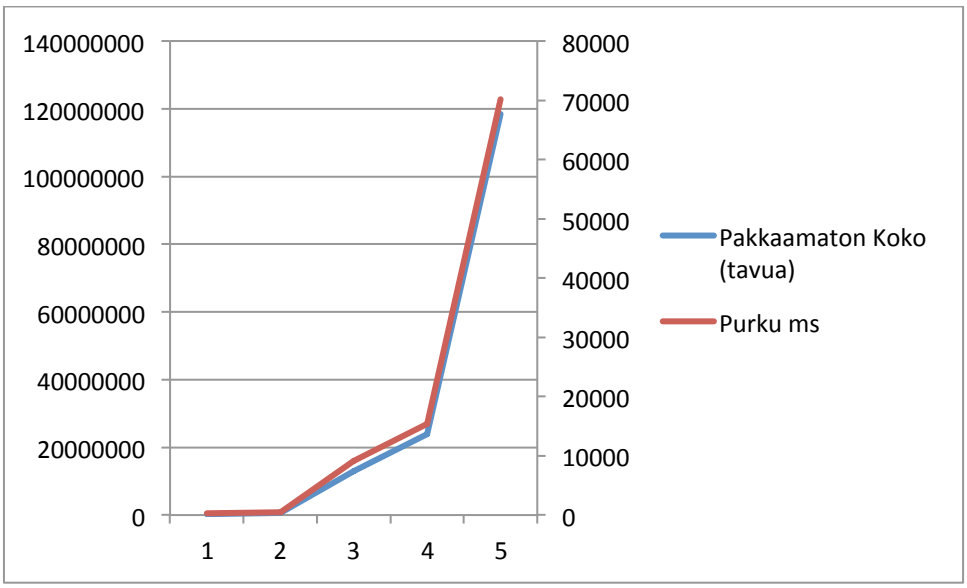
Pakkaamaton koko (tavua)	Kesto (ms)	Purku kB / sek
182021	91	1953.35
581166	40	14188.62
12981174	720	17606.84
23924964	1130	20676.30
118552713	5561	20818.94

3.5. Aikavaativuuksien toteutuminen

Tavoitteellinen aikavaativuus Huffman-algoritmille sekä pakkaus- että purkuoperaatiolle oli O(n).



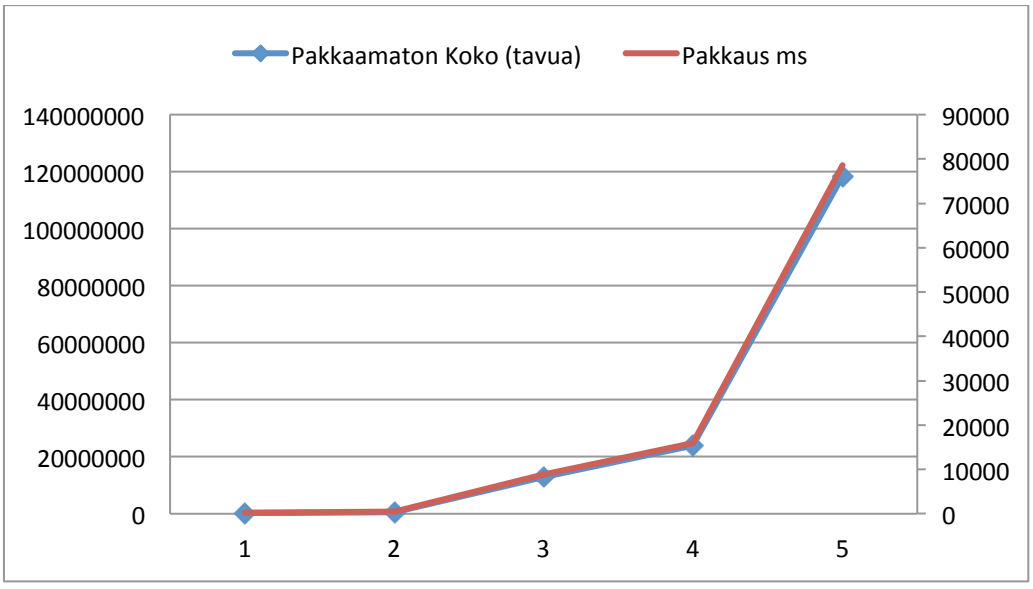
Kuva 4: Huffman-pakkauksen aikavaativuuden toteutuminen.



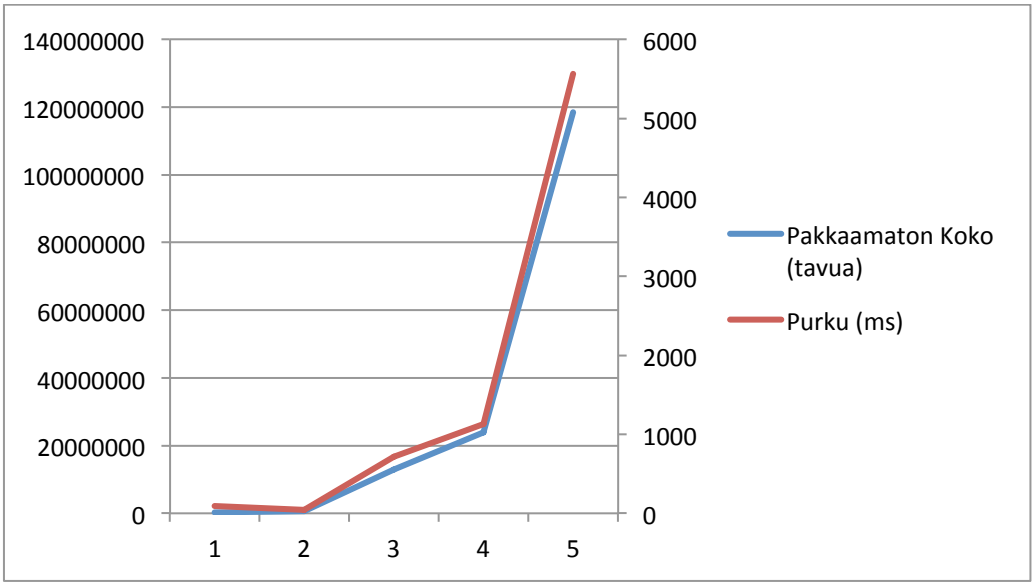
Kuva 5: Huffman-purkamisen aikavaativuuden toteutuminen.

Kuvaajien perusteella aikavaativuustavoite O(n) näyttäisi toteutuvan Huffman-algoritmin kummallekin operaatiolle hyvin.

LZW-algoritmille ei oltu alunperin asetettu aikavaativuustavoitetta, sillä se oli aiheessani valinnainen ominaisuus. LZW-algoritmin molemmat operaatiot toteutuivat kuitenkin varsin odotetusti suuruusluokassa O(n). Allaolevat kuvaajat esittelevät LZW pakkauksen ja purkamisen aikavaativuuksien toteutumisen erikokoisilla syötteillä.



Kuva 6: LZW-pakkauksen aikavaativuus.



Kuva 7: LZW-purkamisen aikavaativuus.

4. Lähteet

1. Tietorakenteet-kurssin (syksy 2005) luentomoniste (Matti Luukkainen & Matti Nykänen)
2. Data structures and algorithms with object-oriented design patterns in Java, Bruno R. Preiss, 2000.
3. http://en.wikipedia.org/wiki/Huffman_coding
4. http://en.wikipedia.org/wiki/LZ77_and_LZ78
5. http://marknelson.us/1989/10/01/lzw-data-compression/
6. Project Gutenberg (http://www.gutenberg.org)
7. XMark XMLGen – An XML Benchmark Project (http://www.xml-benchmark.org)