LZW-pakkaaja

Java 2 Platform Standard Edition 5.0

Teppo Niinimäki

Karstulantie 8 G 115

00550 Helsinki

sähköposti: Teppo.Niinimaki@Helsinki.FI

Helsingin Yliopisto

Tietojenkäsittelytieteen laitos

Tietorakenteiden harjoitustyö

Ohjaaja: Mikko Heimola

Helsingissä 15.10.2006

Sisällysluettelo

Käyttöohje

Yleistä

LZW-pakkaaja on tarkoitettu yksittäisten tiedostojen pakkaamiseen sekä purkamiseen. Ohjelma ei tue monen tiedoston pakkaamista samaan pakettiin. Tämä on kuitenkin kierrettävissä käyttämällä ennen pakkausta jotain tiedostojen yhdistämiseen tarkoitettua sovellusta. Unix maailmassa tällainen on esimerkiksi *tar*. Pakattaessa ja purettaessa alkuperäistä syötetiedostoa ei poisteta automaattisesti. Pakkausalgoritmina LZW-pakkaaja käyttää LZW-algoritmin LZC-nimistä varianttia.

Laitteistovaatimukset

Ohjelman on todettu toimivan ainakin alustalla Java Runtime Environment 5.0. Kääntäminen ja ajaminen saattaa onnistua myös aiemmilla versioilla. Ohjelmaa on testattu vain Linux-ympäristöissä (Ubuntu ja Fedora), mutta todennäiköisesti se toimii myös muillakin alustoilla, joille on saatavilla Java (esim. Windows).

Asentaminen ja kääntäminen

Lähdekoodi koostuu kuudesta java-tiedostosta: Pakkaa.java, PakkausTaulu.java, PakattavaTiedosto.java, Pura.java, PurkuTaulu.java sekä PurettavaTiedosto.java. Kopioi tiedostot haluamaasi hakemistoon ja käännä ohjelma komentamalla kyseisessä hakemistossa:

```
javac Pakkaa.java Pura.java
```

Pakkaaminen

Pakkaamiseen käytetään Pakkaa-luokkaa. Syntaksi on seuraavanlainen:

```
java Pakkaa [valitsimet] pakattava_tiedosto
```

(Huom. Valitsimia voi lisätä myös pakattavan tiedoston nimen jälkeen.)

Seuraavat valitsimet ovat käytettävissä:

Valitsin	**Vaikutus**
-h	Näyttää ohjeen.
-a	Älä käytä kooditaulun alustusta kesken pakkauksen.
-n *pituus*	Määrittää suurimman käytettävän koodipituuden. Pituuden tulee olla väliltä 9-24. Oletuspituus on 19.

Valitsin	Vaikutus
-o *tiedosto*	Määrittää kohdetiedoston nimen. Oletuksena käytettään pakattavan tiedoston nimeä, jonka perään lisätään pääte `.lzc`.
-p	Pakkaustehoa tarkastellaan laskemalla pakkausteho kaiken edellisen kooditaulun *alustuksen* jälkeen pakatun datan sijaan edellisen *tarkastuksen* jälkeen pakatusta datasta.
-r *suhde*	Määrittää suurimman sallitun pakkaussuhteen huonontumisen parhaasta edellisen alustamisen jälkeen saavutetusta arvosta. Kun suhde ylittyy, alustetaan kooditaulu. Annetaan prosentteina väliltä 1-100. Oletus on 5.
-s	Tulostaa tietorivin tilastointia varten. Tämä on testausta varten.
-v	Näyttää tietoa pakkauksen kulusta. Valitsimia voi antaa useita, jolloin näytetään aina enemmän tietoa.

Esimerkkejä pakkaamisesta

Pakataan tiedosto *asiakirja.tx*t tiedostoksi *asiakirja.txt.lzc* oletusasetuksilla:

```
java Pakkaa asiakirja.txt
```

Pakataan tiedosto *arkisto.tar* tiedostoksi *arkisto.tarlz* koodipituudella 21 ilman kooditaulun alustuksia. Lisäksi näytetään jonkin verran tietoa pakkauksen suorittamisesta:

```
java Pakkaa -v -v -a -n 21 arkisto.tar -o arkisto.tarlz
```

Purkaminen

Purkamiseen käytetään Pura-luokkaa. Syntaksi on seuraavanlainen:

```
java Pura [valitsimet] purettava_tiedosto
```

(Huom. Valitsimia voi lisätä myös purettavan tiedoston nimen jälkeen.)

Seuraavat valitsimet ovat käytettävissä:

Valitsin	Vaikutus
-h	Näyttää ohjeen.
-o *tiedosto*	Määrittää kohdetiedoston nimen. Jos purettava tiedosto päättyy päätteeseen `.lzc` käytettään oletuksena nimeä ilman kyseistä päätettä. Muussa tapauksessa täytyy kohdetiedoston nimi antaa manuaalisesti.

Valitsin	Vaikutus
-s	Tulostaa tietorivin tilastointia varten. Tämä on testausta varten.
-v	Näyttää tietoa purkamisen kulusta. Valitsimia voi antaa useita, jolloin näytetään aina enemmän tietoa.

Esimerkkejä purkamisesta

Puretaan tiedosto *asiakirja.txt.lzc* tiedostoksi *asiakirja.txt* oletusasetuksilla:

```
java Pura asiakirja.txt.lzc
```

Puretaan paketti *arkisto.tarlz* tiedostoksi *arkisto.tar*:

```
java Pura arkisto.tarlz -o arkisto.tar
```

Ohjelman toiminta ja rakenne

Luokkarakenne

Pakkaamisessa käytetään luokkia *Pakkaa*, *PakkausTaulu* sekä *PakattavaTiedosto*. Vastaavasti purkamisessa käytetään luokkia *Pura*, *PurkuTaulu* sekä *PurettavaTiedosto*. Näistä luokat *Pakkaa* ja *Pura* ovat ylemmän tason luokkia, jotka prosessoivat ohjelmalle syötetyt parametrit ja suorittavat pakkaamisen / purkamisen käyttäen hyödyksi muita luokkia. *PakkausTaulu* ja *PurkuTaulu* -luokat pitävät kirjaa käytössä olevista koodeista ja niitä vastaavista datapätkistä. Näiden luokkien avulla muunnetaan datapätkät koodeiksi ja toisin päin. *PakattavaTiedosto* ja *PurettavaTiedosto* ovat eräänlaisia puskuriluokkia, joiden avulla lzc-paketteja saa luettua ja kirjoitettua vaihtelevilla koodipituuksilla.

Tarkemmat kuvaukset luokista ja niiden metodeista löytyvät Javadoc-muodossa API-kuvauksina alihakemistosta *LIITE 1 - Javadoc*.

Pakkaaminen

Pakkaamiseen käytetään LZW-algoritmia (tai oikeastaan sen muunnelmaa LZC:tä joka on kuitenkin oleellisilta osin sama). Lyhyesti kuvattuna prosessi etenee seuraavasti: *Pakkaa*-luokan *main*-metodissa käsitellään komentorivillä annetut parametri, aukaistaan tarvittavat tiedostot ja kutsutaan saman luokan *pakkaa*-metodia. *pakkaa*-metodi lukee lähdetiedostoa tavu kerrallaan ja kirjoittaa pakkauksen tuloksena saatavat koodit *PakattavaTiedosto*-luokan välityksellä kohdetiedostoon. Nämä koodit vastaavat aina jotain alkuperäisen datan pätkää.

Pakkaaminen tapahtuu siten, että *pakkaa*-metodi pitää muistissa edellisen kirjoitetun koodin jälkeen lukemaansa datapätkää vastaavaa koodia. Yhden tavun mittaista datapätkää vastaava koodi on sama kuin tavun arvo. Aina uuden tavun luettuaan metodi kysyy luomaltaan PakkasTaulu-oliolta uutta koodia, joka vastaisi muistissaan olevaa (edellistä) koodia vastaavaa datapätkää yhdistettynä uudella tavulla. Jos vastaava koodi löytyy, lukee metodi seuraavan tavun ja jatkaa samalla tavalla. Kun datapätkä on kasvanut niin pitkäksi, että vastaavaa koodia ei löydy kirjoitetaan edellinen palautettu koodi kohdetiedostoon, alustetaan edellinen koodi viimeksi luetuksi tavuksi ja jatketaan prosessia. Näin käydään koko tiedosto läpi.

PakkausTaulu on vakiokokoinen hajautustaulu, joka sisältää periaatteessa kaikki kyseisellä hetkellä käytössä olevat koodit ja niitä vastaavat datapätkät. Käytännössä kuitenkin tallennetaan koodit sekä niitä vastaavat ”edellinen koodi” - ”uusi tavu” -yhdistelmät. Hajautusmetodina on avoin hajautus ja kokeilujono lasketaan kaksoishajautuksella. Taulun koko on noin 1,7 x $2^{\text{maksimikoodipituus}}$, jotta taulu pysyisi tarpeeksi väljänä myös suuremmilla täyttöasteilla ja hakujonot pysyisivät siten lyhyinä. Hajautustaulun avaimena käytetään mainittua koodi-tavu -yhdistelmää, ja vastaavasti alkion arvona on yhdistelmää vastaava koodi. Hajautusfunktion arvo lasketaan kaavalla *edellinen_koodi XOR (uusi_tavu * $2^{\text{maksimikoodipituus} - 8}$)* ja kokeilujonossa käytettävä siirtymä taas kaavalla *hajautusfunktion_arvo / 2 – 1*. Ensimmäinen on sama kuin lähdemateriaalin (http://marknelson.us/attachments/lzw-data-compression/lzw.c) esimerkkikoodissa ja toinen on hieman muunneltu (ja paremman tuntuinen) versio esimerkkikoodin vastaavasta. Kummatkin ovat varsin nopeita laskea ja kokeilujonotkin pysyvät keskimäärin lyhyinä (riippuen datasta yleensä 2-5).

Aluksi vain yhden tavun mittaisilla datapätkillä on vastaava koodi. Aina kun taululta kysytään datapätkää, jolle ei ole vielä määritelty vastaavaa koodia (eli jota vastaavaa koodi-tavu-yhdistelmää ei löydy taulusta), lisätään kyseinen datapätkä (koodi-tavu -yhdistelmä) tauluun ja annetaan sille arvoksi järjestyksessä seuraava vapaa koodi. Tässä tapauksessa palautetaan kuitenkin ilmoitus siitä, että koodia ei löytynyt. Seuraavalla kerralla kyseinen koodi on siis käytettävissä. Näin purettaessa on mahdollista rakentaa vastaava kooditaulu purettavan datan perusteella, eikä koodeja ja niiden merkityksiä tarvitse tallentaa erikseen.

Aikavaativuus

Syöteaineisto käydään kerran läpi ja jokaista syöteaineiston tavua kohti etsitään kooditaulusta vastaava koodi. Koska koodien tallennukseen on käytetty hajautustaulua, on haku käytännössä vakioaikainen (kokemusten perusteella hakuja tulee yleensä keskimäärin väliltä 1,5 - 4, käytännössä aina keskimäärin enintään 6). Osassa tapauksista hajautustauluun myös lisätään uusi alkio, mutta tämäkin operaatio on vakioaikainen. Koska myös lähdetiedoston lukeminen ja koodien

kirjoittaminen syötetiedostoon vaativat aikaa suoraan verrannollisesti datan määrään, on algoritmin aikavaativuus luokkaa *O(n)*, missä *n* on pakattavan datan määrä.

Pienillä tiedostoilla nousee myös kooditaulun alustaminen merkittävään osaan. Koska kooditaulun alustuksessa täytyy käydä koko hajautustaulu läpi kerran on sen aikavaativuus $O(2^k)$, missä *k* on maksimikoodipituus. Näin lopulliseksi aikavaativuudeksi saataisiin $O(n+2^k)$, missä *n* on pakattavan datan määrä ja *k* maksimikoodipituus. Lisäksi jos kooditaulun alustus on käytössä, saatetaan alustus suorittaa pahimmassa tapauksessa tasaisin (pakatun datanmäärän) välein ja alustuksiin kuluisi siis aikaa luokkaa $O(n*2^k)$. Näin ollen pahimmassa tapauksessa aikavaativuus on luokkaa $O(n+n2^k) = O(n2^k)$, missä n on pakattavan datan määrä ja *k* maksimikoodipituus.

Tilavaativuus

Hajautustaulun koko riippuu käytettävästä maksimikoodipituudesta, eikä pakattavan tiedoston koolla ole siihen vaikutusta. Koska tiedostojen lukeminen ja kirjoittaminen tapahtuu tapahtuu pakkauksen kuluessa, ei koko dataa tarvitse säilyttää muistissa. Tilavaativuudeksi saadaan siis *O(1)* pakattavan datan määrän suhteen (ja $O(2^k)$ koodipituuden suhteen).

Purkaminen

Purkaminen on samankaltainen prosessi kuin pakkaaminenkin; *Pura*-luokan *main*-metodi käsittelee komentoriviparametrit, avaa tarvittavat tiedostot ja kutsuu *pura*-metodia. *pura*-metodi lukee syötetiedostoa koodi kerrallaan ja kysyy koodia vastaavaa datapätkää luomaltaan *PurkuTaulu*-oliolta. Saatu purettu data kirjoitetaan kohdetiedostoon.

PurkuTaulu sisältää yksinkertaistettuna taulukon, josta koodia avaimena käyttämällä saadaan vastaava datapätkä. Tässäkin tapauksessa kuitenkin tallennetaan taulukkoon kuitenkin vain koodia vastaava koodi-tavu -yhdistelmä. Kun taululta kysytään koodia vastaavaa dataa, etsitään annettu koodi taulukosta. Koodia vastaavan datan viimeinen tavu otetaan ylös ja haetaan alkuosaa vastaavaa koodi taulukosta. Tätä jatketaan kunnes saavutetaan koodi, joka vastaa yksittäistä tavua (eli sen arvo on pienempi kuin 256). Takaperin luettu data palautetaan kutsujalle. Ennen tätä lisätään kuitenkin edellinen palautettu koodi yhdistettynä tämänkertaisen palautettavan datan ensimmäisellä tavulla taulukkoon. Avaimeksi asetetaan seuraava vapaana oleva koodi. Pakattua dataa purettaessa rakennetaan siis samalla vastaava kooditaulu kuin pakattaessakin.

Aikavaativuus

Jokaista *puretun* tiedoston tavua kohden haetaan taulukosta jokin koodi (ja sitä vastaava koodi-tavu -yhdistelmä). Lisäksi jokaista kysyttyä koodia kohden (vähemmän kuin em. tavuja) lisätään

taulukkoon uusi alkio. Kummatkin operaatiot ovat vakioaikaisia. Koska myös tiedostojen lukeminen ja kirjoittaminen ovat aikavaativuudeltaan suoraan verrannollisia datan määrään, saadaan aikavaativuudeksi *O(n)*, missä *n* on *pakkaamattoman* datan määrä. Huomioitavaa on myös, että kooditaulun alustaminen on purettaessa vakioaikainen operaatio, joten myös pahimman tapauksen aikavaativuus on sama *O(n)*.

Tilavaativuus

Kuten pakkaamisessa hajautustaulun, myös purkamisessa kooditaulukon käyttämä tila on eksponentiaalisesti verrannollinen käytettävään maksimikoodipituuteen, eikä tiedostojen koko vaikuta tähän. Myös puretulle datalle varattava taulukko alustetaan kokoon $2^{\text{koodipituus}}$. Tilavaativuudeksi saadaan siis sama *O(1)* datan määrän suhteen (ja $O(2^k)$ koodipituuden suhteen).

Perusalgoritmista eroavat ominaisuudet

Kuten jo aiemminkin on mainittu, ei käytössä ole puhdas LZW-algoritmi, vaan sen muunnelma: LZC. Tämä tarkoittaa sitä, että koodipituus vaihtelee (normaalisti kasvaa ja kooditaulun alustuksessa laskee) pakkauksen edetessä; Koodit kirjoitetaan käyttäen sellaista koodipituutta, että sillä hetkellä käytössä olevat kaikki koodit olisi mahdollista esittää kyseistä koodipituutta käyttäen. Vaihtelevalla koodipituudella on aina pienentävä (tai neutraali) vaikutus tiedostokokoon. Erityisesti pieniä tiedostoja pakattaessa tästä on hyötyä.

Toinen laajennus alkuperäiseen algoritmiin on mahdollisuus kooditaulun pakkauksenaikaiseen alustukseen. Pakkauksen edetessä algoritmi tarkkailee pakkaustehoa, ja jos sen havaitaan laskevan liikaa, alustetaan kooditaulu ja jatketaan pakkausta pienimmällä koodipituudella. Vastaava alustus täytyy tietenkin suorittaa myös purkamisen yhteydessä, joten suurin mahdollinen (hetkellisellä koodipituudella) käytettävissä oleva koodi on varattu aina alustuksesta ilmoittamiseen. Yleensä pakkauksenaikaiset alustukset joko eivät vaikuta tiedostokokoon (jos niitä ei kuitenkaan tehdä) tai sitten pienentävät sitä. Joissakin tapauksissa vaikutus saattaa kuitenkin olla päinvastainen (katso esimerkkinä videon pakkaus liitteestä LIITE 3). Vaikutus pakkausaikaa riippuu myös pakattavasta tiedostosta: kohtuullinen määrä alustuksia pienentää hajautustaulun keskimääräistä täyttöastetta ja siten vähentää tarvittavien hakujen määrää, mutta toisaalta alustaminen vaatii koko taulukon käymisen läpi kertaalleen, joten jos alustuksia tulee liikaa, kasvaa ajantarve.

Rajoitukset ja parannusehdotukset

Kuten yleensä, muutamia asioita olisi voinut tehdä toisellakin tavalla. Eräs tällainen on pakattaessa käytettävän hajautustaulun koon valitseminen. Tällä hetkellä käytetään vakiopituutta, joka on noin 1,7 kertaa käytettävä maksimikoodipituus. Ehkä parempi ratkaisu olisi ollut tehdä hajautustaulusta

laajentuva ja alustaa se aluksi pieneen kokoon. Näin muistin kulutus olisi pienillä tiedostoilla vähäisempi. Sama asia koskee myös purkamisessa käytettäviä taulukoita: nekin olisi voinut tehdä laajentuviksi samaan tapaan.

Joissakin tilanteissa (esim. todella huonosti pakkaantuvia tiedostoja pakattaessa) saattaa käydä niin, että kooditaulun alustuksia tulee todella usein. Tällöin alustuksessa tehdään paljon turhaa työtä, sillä usein taulun täyttöaste näissä tapauksissa on erittäin pieni. Tarpeeksi pienen täyttöasteen ollessa kyseessä voitaisiinkin alustaa vain jo täytetty osa, koska loput ovat jo valmiiksi alustettuja.

Maksimikoodipituus on keinotekoisesti rajoitettu välille 9-24. Alaraja on toki luonnollinen valinta, sillä pienempien koodipituuksien käyttö ei olisi mahdollista, jos dataa käsitellään tavu (kahdeksan bittiä) kerrallaan. Ylärajan sen sijaan olisi voinut toki asettaa suuremmaksi, laajentuvan hajautustaulun tapauksessa mahdollisesti jopa rajoittamattomaksi. Käytännössä tällä ei ole kuitenkaan merkitystä normaalissa käytössä, sillä suurilla koodipituuksilla muistinkäyttö nousee nopeasti liian suureksi. Esimerkiksi koodipituudella 24 vaatii pakkaaminen muistia yli 350 Mt ja muistin kulutus kaksinkertaistuu aina kun koodipituutta lisätään yhdellä. Lisäksi näin suurista koodipituuksista alkaa olla hyötyä vasta kun pakattavan tiedoston koko nousee tarpeeksi suureksi (satoihin megatavuihin).

Muita vähemmän merkittäviä ja kohtuullisen helposti muutettavissa olevia rajoituksia ovat esimerkiksi kooditaulun alustusrajan rajoittaminen kokonaislukuihin välillä 1-100 sekä rajoittuminen vain yhden tiedoston pakkaamiseen.

Testaus

Pakkauksen ja purkamisen toimivuutta ja oikeellisuutta on testattu eri tyyppisillä ja eri kokoisilla aineistoilla (esimerkiksi tyhjällä tiedostolla, erilaisilla tekstiasiakirjoilla, binääritiedostoilla).Lisäksi on testattu pakkauksen toiminnan kannalta hankalilla aineistoilla (sama datapätkä toistuu monta kertaa peräkkäin). Kaikilla testatuilla aineistoilla pakkaaminen ja purkaminen tuottavat täsmälleen alkuperäisen kanssa samanlaisen tiedoston.

Pakkaustehoa (eli tiedostokoon pienentymistä) sekä pakkaukseen ja purkamiseen kuluvaa aikaa on myös testattu. Pakkausasetusten vaikutusta pakkaustehoon sekä aikaan tutkittiin pakkaamalla Jules Vernen eräästä teoksesta, kahdesta erilaisesta geologisesta mittausdatasta, Helsingin raitiovaunujen reittikartasta sekä auringonpimennystä esittävästä valokuvasta koottua 1,9 megatavun yhdistelmätiedostoa. Paras pakkaussuhde saavutettiin odotetusti suurella koodipituudella ja pienellä kooditaulun uudelleenalustusrajalla. Koska kuitenkin joissakin tapauksissa pieni uudelleenalustusraja heikentää pakkaustehoa on alustusraja asetettu kompromissiksi arvoon 5.

Pakkaustehon tarkkailutavalla ei ollut kovin suurta vaikutusta keskimääräiseen pakkaustehoon. Pätkätarkkailulla (tarkkaillaan vain lyhyttä viimeisen tarkistuksen jälkeen pakattua pätkää) pakkausteho parantui nopeammin. Sen sijaan paras pakkausteho saavutettiin kuitenkin normaalilla tarkkailulla (tarkkaillaan koko edellisen alustuksen jälkeen pakattua dataa). Pakkaus- ja purkuajat olivat pätkäkatkaisulla hieman pienemmät. Testausdata (testien tulokset) löytyy liitteestä *LIITE 2*.

Pakkaustehoa ja -aikoja vertailtiin myös kahden muun yleisen pakkausohjelman, **gzip**:n sekä **bzip2**:n, kanssa. Vertailussa käytettiin eri tyyppisiä ja kokoisia tiedostoja, jotta eroavaisuudet tulisivat selville. Kutakin ohjelmaa käytettiin oletusasetuksilla. Pakatun tiedoston kokoa tarkasteltaessa oli bzip2 odotetusti kärjessä. Lähes aina jäi LZW jälkeen myös gzip:stä. Yllättäen ihmisen genomin pakkaamisessa ylti LZW kuitenkin parhaaseen pakkaussuhteeseen. Pakkaukseen ja purkamiseen käytettyä aikaa mitattaessa oli gzip selvästi ylivoimainen. Sen sijaan LZW:n ja bzip2:n ajantarve oli suunnilleen samaa luokkaa. Pakkausohjelmien vertailun tulokset ovat liitteessä *LIITE 3*.