

Tiralabran viikkoraportti #1 Kuvan pakkaus

16.5.2013

Mikko Kangasmäki

Toistaiseksi en ole kirjoittanut yhtään riviä valmista koodia, vaan selvitellyt bmp-tiedostojen muotoa ja kuvankäsittelyä javalla.

Aluksi käytin huomattavasti aikaa kuvan lukemiseen bittitasolla ImageInputStream-luokan avulla. Yritin päästä käsiksi pikselidataan, ja kopioida kuvan toiseen tiedostoon sitä kautta. Kun osaisin tehdä sen, voisim käsittellä dataa siinä välissä. Lopulta onnistuin tässä, mutta se oli työlästä sekä minulle, että koneelle, jolta kuvan kopiointi toiseen tiedostoon kesti n. 50 sekuntia.

Vaiva ei kuitenkaan ollut hukkaan heitettyä, sillä sen avulla opin ymmärtämään paremmin bmp-tiedostomuotoa ja vastaan tulleet ilmiöt antoivat ajattelemisen aihetta koko projektista.

Minulla oli, ja on vieläkin, tiettyjä epäilyjksiä aalokkeiden käytön mielekkyydestä, sillä lopputuloksesta saattaa helposti tulla triviaali tai liian vaikea, eli totetutan vain jonkin muun kirjoittaman algoritmin tai yritän itse tehdä jotain, missä en onnistu. Kuvan pikselidataa käsitellessä aloin miettiä joitakin yksinkertaisempia pakkaamismuotoja: esimerkiksi, että ilmoitettaisiin väreille alue, joka väritetään, tai sitten unohdetaan vähän esiintyviä värisävyjä.

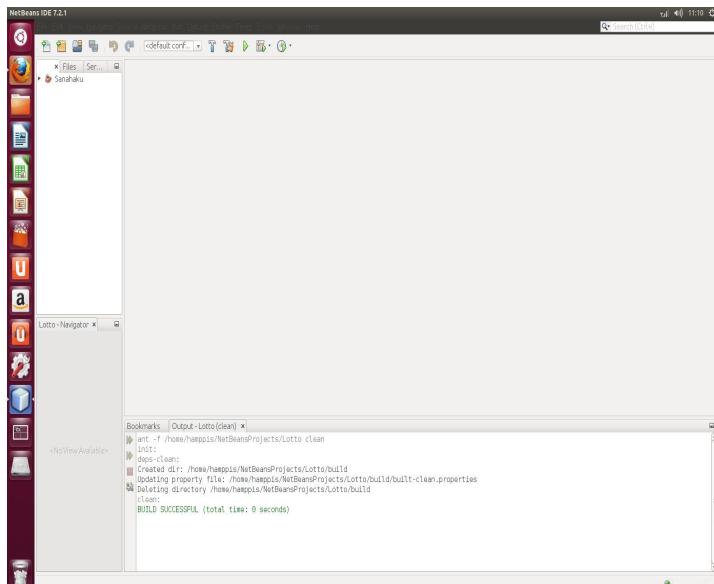
Laskin koneella kultakin kuvan pikseliriviltä siinä esiintyvien punaisen sävyjen määrän. Ensimmäinen testikuva on screenshot netbeansista. Erilaisia punaisen sävyjä esiintyy sen erään satunnaisen rivin 1680 pikselissä tällaisella jakaumalla:

Luvut siis kertovat, että eniten esiintynyt punainen sävyä oli testirivillä 1410 pikselissä. Toiseksi esiintyneintä 205 pikselissä jne. Yhteensä rivillä siis esiintyy vain 48 kaikista 256 punaisen sävystä. Screenshotista tällaista yhdenmukaisuutta voi tietenkin odottaakin, mutta toisessa testikuvassani, joka on talvimaisema, on myös aika vähän eri sävyjä (leveys 3456 pikseliä):

```

215 131 107 102 90 85 85 84 77 65 55 53 30 30 29 28 28 27 25 25 24 24 24 23 23 23 22 22 22 21
21 21 21 20 20 20 19 19 19 19 19 19 17 17 17 17 17 17 17 17 17 17 17 17 17 16 16 16 16 16 16 16
16 16 16 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13 13 13 13
13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 11 11 11 11 11 11 11 11 11 11 11 11 11
11 11 11 11 10 10 10 10 10 10 10 10 10 10 9 9 9 9 9 8 8 8 8 8 8 7 7 7 7 7 7 7 7 7 7 7 6 6
6 6 6 6 5 5 5 5 5 5 4 4 4 3 3 2 2 2 2 2 1 1 1 1 1 1 1 1

```



Näyttäisi järkevältä tiivistää tiedosta unohtamalla värisävyt, joita esiintyy vain pienessä määrässä pikseleitä. Esimerkiksi jättämällä pois kaikki sävyt, joita on alle kymmenessä pikselissä, saadaan sävyjen määrä pienentämään 140:een. Vielä parempi olisi, jos sävyn poisjättäminen voitaisiin ratkaista lähekkäin olevien pikselien määrällä mieluummin kuin niiden kokonaismäärällä.

Tässä vaiheessa kuulin BufferedImage-luokasta, joka olikin huomattavasti nopeampi ja helppokäyttöisempi kuin aiemmat työkaluni.

Aloin kokeilumielessä toteuttaa ohjelmaa, joka lukee riviltä värisävyn (vaikkapa punainen) taulukkoon `int[] värit = new int[256]`. Taulukon arvo `värit[n]` kertoo, kuinka monessa pikselissä rivillä esiintyy punaisen sävyä `n`.

Seuraavaksi ohjelma kirjoittaa kohdekuvan pikseleihin värit esiintymismäärien mukaisessa järjestyksessä suurimmasta pienimpään. Voidaan asettaa jokin minimimäärä, jota vähemmän esiintyneitä sävyjä ei kirjoiteta ollenkaan, ja näin säästetään tilaa, ainakin periaatteessa.

Toistaiseksi ohjelma ei ole vielä toiminut aivan toivotulla tavalla:



Seuraavaksi ajattelin:

1. palauttaa mieleeni aalokkeet
2. selvittää, miten niitä voisi käyttää oikean maailman sovelluksissa
3. selvittää tarkkaan, miten BufferedImagen värit on koodattu (tästä minulla on hajua)
4. kirjoittaa lopullisesta ohjelmasta osat, joissa luetaan kuva levyltä ja kirjoitetaan se levylle.

Tällä viikolla jatkoin kokeilujani kuvankäsittelyssä. Löysin bugin, jonka ansiosta enintään yhdessä pikselissä esiintyvän värisäbyn poistaminen kuvasta oli toiminut katastrofaalisen huonosti. Korjattuani sen vertailin silmämääräisesti erilaisia tapoja vähentää esiintyvien värisävyjen määriä. Kaikki kokeilemani tavat olivat melko yksinkertaisia.

Laskeskelin myös testikuvassa esiintyvien saman säbyn jonojen määriä, ja miten pikseleiden uudelleensäyttäminen vaikuttaa siihen.

M.H:n kehotuksesta lataan kokeilutiedostotkin GitHubiin, vaikka ne ovat jokseenkin lukukelvottomia.

Kuvankäsittelyn lisäksi aloitin kokeilut aallokkeiden kanssa. Tosin, siinä toistaiseksi olen laskeskeillut vain eri välien summia. Ajattelin, että en välitä vielä aallokkeiden laskemisesta, vaan enemmänkin niiden tallennusmuodosta. Ne voi tallentaa kaksiulotteiseen ja selkeään taulukkoon, joka vie paljon tilaa, tai yksiulotteiseen pieneen taulukkoon, jonka kanssa joutuu pelleilemään indekseillä. Tilan säästössä jälkimmäinen on tietenkin toivottavampi.

Sitten aloin kirjoittaa lopullista koodia: Kirjoitin olion, joka lukee kuvan byte-taulukoksi, ja joka pystyy myös tallentamaan sen. Tässä tuli yllättäviä ongelmia, sillä jostain käsittämättömästä syystä se, mikä toimi kokeilukoodissani ei enää toiminut tässä. Periaattessa ongelma oli siinä, että int-arvo 200 esimerkiksi muuttuu bytessä – 56,ksi, ja kun se muutetaan takaisin intiksi, arvo pysyy -56:na.

Ongelman ratkaisuhan ei ole erityisen vaikea, mutta olisin mieluummin tehnyt sen räpellyksissäni toimineella mentelmällä kuin if-lauseella. Tämä laittoi minut myös harkitsemaan shorttien käyttöä bytejen asemesta, koska tämä byten ominaisuus on omiaan aiheuttamaan myöhemmin vaikeita ongelmia. Toisaalta, shorttien käyttö veisi ~3500x2000 pikselin esimerkkikuvassanakin seitsemän megaa enemmän muistia.

Valmista koodia ei tullut toisiaankaan paljoa, ja vielä pahemaksi tilanne muuttui, kun aloin kirjoittaa testejä. Huomasin, että suunnitelmani ohjelman rakenteelle oli kehno. Ne metodit, joita olisi ollut eniten syytä testata olivat privaatteja. Kun yritin selvittää, kuinka sellaisia testataan JUnitilla, törmäsin ajatukseen, että tämä on “code smell”. Ja toisaankin, kaikki ne metodit olivat sellaisia, jotka voisivat olla omassa luokassaan, koska niille todennäköisesti on käytööä muuallakin.

Aloin siis uudelleenjärjestellä ohjelmaa, mikä puuha on vielä kesken, enkä osaa vielä tarkkaan sanoa, miltä tänään jättämäni palautus tulee näyttämään.

Ajattelin, että minulla olisi jokaiselle tiivistämismuodolle, oma luokkansa, joka huolehtii datan tiivistämisestä, tallentamisesta ja aukaisemisesta. Vastaavasti bmp:lle pitää olla oma. Jokaisen luokan käytössä olisi apuvälineitä sisältävä luokka.

Kirjoitan koodin ja javadocin englanniksi silläkin uhalla, että se heikentää laatua, koska englannin käyttö on kuitenkin alalla standardi, ja siihen on varmaankin hyvä totutella ajoissa.

Viime viikolla tajusin testejä kirjoittaessani, millainen sekasotku ohjelmani valmiskin osa oli. Aloin kokonaan alusta, mutta deadlineen ehättääkseni kirjoitin ohjelmasta vieläkin pahemman sekasotkun. Nyt aloitin maanantaina vielä uudelleen, ja tällä kertaa tuhlasin vähän aikaa ajatteluunkin.

Valmiin ohjelman tulisi nyt noudattaa kaavaa, jossa jokaiselle eri tiedon tallennusmuodolle on oma luokkansa, joka huolehtii näistä velvollisuusista:

- kuvan tallentaminen tässä muodossa
- kuvan lukeminen tästä muodosta
- perusdataan muuttaminen tähän muotoon
- perusdataaksi muuttaminen tästä muodosta.

Perusdatalla tarkoitetaan tässä kolmiulotteista byte-taulukkoa $\text{data}[c][x][y]$, joka kuvailee kunkin pikselin perusvärien sävyn s.e. (x,y) on pikselin koordinaatit ja c ilmaisee, minkä perusväriin säväystä on kyse (sininen = 0, vihreä = 1, punainen = 2).

Havaitsin tässä esityksessä yhden ongelman: kun värit esitetään integer-muodossa (0,r,g,b), jokaista sävyä vastaa kokonaisluku 0-255. Kun ne muutetaan byteksi, esimerkiksi laskutoimituksella

green = (byte) ($\text{rgbInteger} >> 8$),

tietoa ei mene hukkaan, mutta se vääristy y: Numero 0,...,255 kuvaautuvat numeroiksi 0,...,126 ,127, -128, -127,..., -2, -1. Näin vierekkäiset värisävyt jhoutuvat etäälle toisistaan, joten tiivistäminen ei tältä osin ota huomioon kuvan luonnollisia ominaisuuksia.

Tämän ongelman kiertämiseksi valitsen ensin värisävyä kuvaavan luvun 0,...,255, vähennän siitä 128, ja sitten vasta muutan byteksi. Tämä tekee koodista yleisesti ehkä vaikeampiselkoista, mutta pyrin kertomaan tämän kaikkialla javadocissa, missä oleellista.

Olen kokeillut jonkin verran itse waveletmuunnostakin. Koska dataa on suuria määriä (koekuvassani kuutisen miljoona pikseliä, eli n. 18 miljoonaa byteä), pyrin käyttämään tilaa säästeliästi, vaikka se tekee koodista epäselvempää. Muunnos olisi helpompaa tallentaa kaksiulotteiseen taulukkoon, mutta siihen tulisi paljon hukkaita, joten olen käyttänyt yksiulotteista taulukkoa, mikä edellyttää vaikeampiselkoista indeksien käsittelyä.

Itse muunnos ei vielä toimi. Tai se toimii, mutta väärin. Sen sijaan tekemäni summataulukko, jonka indeksien pitäisi toimia samoin kuin muunnostaulukossa toimii. Tämä ei ole niinkään osaratkaisu muunnosongelmaan kuin siihen, että kuvan rivit eivät ole välittämättä kakkosen potenssin mittaisia.

Muunnoksessa ongelmaksi tuli, etten ole oikeasti harrastanut sellaisia oikeilla numeroilla tai diskreetissä maailmassa. Matematiikassa niiden käyttö on petollisen helppoa, koska vakioiden numeroarvoihin ei tarvinnut kiinnittää erityistä huomiota. Nyt ne ovat tärkein asia.

Koska muunnoksen kirjoittaminen suoraan datariville ei toiminut, aion perääntyä ja alkaa suosiolla kirjoittaa niiitä erikoistapaukseen, jossa rivin pituus on kakkosen potenssi.

Joitakin ongelmia voi tulla siitä, että muunnoksessa saatavat kertoimet eivät välittämättä ole kokonaislukuja. Pitäisi keksiä keino, jolla taataan, että muunnos pienentää datan määriä.

Viime viikolla torstain palautuksen jälkeen sain aalokemuunnoksen yllättäen toimimaan. Tein sen ensin erikoistapaukselle, jossa rivin pituus on 2^n ja sitten yleiselle tapaukselle palauttamalla siihen. Tein muunnoksen myös kuvalle ja sitten häviöllisen muunnoksen. Sitten tein muunnoksen tallentamista varten oman tiedostamuodon, wtf-tiedoston. Lopuksi tein graafisen käyttöliittymän kuvan muuntamiselle, mikä ei varmaankaan olisi ollut ykkösrioriteetti, mutta se nyt vain sattui huvittamaan.

GitHubissa on nyt ensimmäinen toimiva versio muunnoksesta jar-tiedostona. Graafinen käyttöliittymä avautuu tuplaklikkaamalla, ja komentoriviltä sitä voi käyttää tekstiparametreilla näin:

```
java -jar compression.jar 3 kuva.bmp kohde.wtf,
```

joka muuttaa kuvan wtf-tiedostoksi. Luku 3 on level of loss (lol), jonka tulee olla ei-negatiivinen kokonaisluku. Toiseen suuntaan muunnos toimii kahdella paratmetrilla:

```
java -jar compression.jar pakattu.wtf kohde.bmp
```

Ohjelma päättlee suunnan parametrien määrästä ja lol on kirjoitettu tiedostoon.

Pakaus ei ole erityisen tehokas ajassa eikä tilassa mitattuna. Ensiksi on huomattava, että pakaus on suunniteltu raakadataan, eli bitmapin tiivistämiseen. Jo tiiviistettyjä muotoja kuten .png tai .jpg se ei pähitätä.

Pakkauksen pitkä kesto johtuu osittain vaion huonosta koodaamisesta. Halusin tehdä mahdollisimman nopeasti toimivan version, joten häviöllinen pakkaaminen tekee kuvalle täyden muunnoksen ja unohtaa siitä osan. Tätä ei pitäisi olla vaikea optimoida.

Toinen syy pakkaamisen pitkään kestoon on hidas kirjoittaminen levylle. En tiedä, onko käyttämäni DataOutputStream hidastavaa, vai käytänkö sitä väärin. Luultavasti kokonaisten taulukoiden kirjoittaminen kerrallaan nopeuttaa sitä.

Että pakaus vie paljon tilaa on taas osittain systemaattinen ongelma. osittain ohjelmointitekninen. Teen ohjelmassa muunnoksen yksittäisille riveille, eli funktiolle $N \rightarrow N$. Luulen, että muunnos koko kuvalle. $N^2 \rightarrow N$, olisi tehokkaampi. Alkujaan suunnittelin, että se olisi seuraava vaihe, mutta en enää tiedä, ehdinkö tehdä sitä.

Toinen systemaattinen ongelma on ehkä käyttämäni aalloke, Haarin aalloke, joka on yksinkertainen, mutta ei ehkä parhaiten vastaa tehtävän todellisuutta. Aalokemuunnoksessa nimittäin valittu aalloke voi vaikuttaa paljonkin tulokseen. Olin ajatellut tässäkin kokeilla muita lähestymistapoja, mutta en tiedä, ehdinkö enää.

Ohjelmointiteknisiä ongelmia on tiedon tallentaminen: Muunnoksessa kuvaan esittävien vakioiden määrä periaatteessa puolittuu aina, kun lol kasvaa. Kuvan rivin punaista väriä saattaa esimerkiksi esittää 1024 byteä, ja kun lol = 1, saadaan määrä pienemään 512:aan. Jotkin luvut eivät kuitenkaan enää mahdu byteen, yksittäinen muunnoksesta saatu kerroin voi olla välillä $-128 * 1024, \dots, 127 * 1024$, johon tarvittaisiin int. Jos kertoimet kirjoitettaisiin int:teinä, muunnoksen koon suhde alkuperäiseen olisi suunnilleen

$$\text{sizeOf(int)} * \text{alkupKoko} / 2^{\text{lol}}$$

Onneksi kertoimet eivät kuitenkaan yleensä ole satunnaisesti suuria tai pieniä, vaan alkavat pieninä ja kasvavat loppua kohti (suurempia allokkeita kohti). Tehtyäni ensiksi integer-tallennuksen toimivuutta testatakseni, tein toisen, jossa jokaisen muunnosrivin alussa on kaksi lukua: shortOffset

ja intOffset, jotka kertovat, monesko luku on rivin ensimmäinen short, ja monesko ensimmäinen int. Tällä tavalla pakkaus pienentää tiedoston kokoa jopa silloin, kun lol = 1. (Siis bitmappiin verrattuna).

Olen miettinyt erilaisia keinoja parantaa tästä lähestymistä:

1. Käytän omia lukuja, edellisten lisäksi halfbytejä ja kolmen tavun integereitä.
2. Kerron jokaista lukua ennen, mikä sen tyyppi on, esimerkiksi 00 = halfbyte, 01 = byte, 10 = short, 11 = int
3. Kerron bitillä, onko luvun tyyppi sama kuin edellisen, ja tarvittaessa uuden tyyppin.

On selvää, että mitkään näistä menetelmistä eivät pienennä muunnoksen tekemiseen kuluvalaa aikaa. Niissä on myös se ongelma, että java ei sellaisenaan mahdollista yksittäisten bittien kirjoittamista, jotka tämä edellyttäisi paljon työlästä puuhaa. Tässä kohtaa alkaa näyttää siltä, että C olisi ollut parempi valinta kieleksi. Valitettavasti en vain osaa sitä kovin hyvin.

Seuraavaksi ohjelmassa on:

1. Kirjoittaa wtf-tiedostomuodon määrittely
2. Kirjoittaa uusiksi määrittelydokumentti
3. Kirjoittaa testejä, joita löin laimin tehdäkseni nopeaa toimivan ohjelman.
4. Nopeuttaa ja järkeväyttää muunnoksen tekemistä
5. Tiedoston kirjoittamisen parantaminen.

Jos ohjaajilla on tietoa seuraavista, niin otan ehdotuksia mielelläni vastaan (tulen varmaan ohjausaikaan laitokselle kyselemään neuvooja, kunhan saan vain pyöränkumin paikattua):

1. Mikä olisi paras luokka binäärityestedoston kirjoittamiseen?
2. Onko olemassa jotain systemaattisia tapoja, joilla kirjoittaa dataa tiiviisti, kun yksittäisten lukujen koot vaihtelevat, mutta vaihtelussa on säädöllisyyttä.