Tiralabra 2015/periodi 3: Viikkoraportti 2

Tällä viikolla ensimmäinen tavoitteeni oli poistaa pakkauksesta tiedoston tyyppiä koskevat rajoitukset. Koska viime viikolla olin toteuttanut ohjelmaa niin, että kukin 8-bit-ASCII -merkki sai oman koodisanansa, ei tästä onneksi aiheutunut kohtuuttoman suurta vaivaa. Muutin ohjelman lukemaan tekstin sijasta tiedostoa tavu kerrallaan, ja jokaiselle erilaiselle 256 tavulle luodaan oma Huffman-koodisana. Lopulta muutos viime viikkoiseen olikin varsin helppo tehdä, sillä esim. koodisanoja oli sama määrä. Testailin pakkausta nyt myös pariin jpeg-kuvatiedostoon, ja se näkyi toimivan (joskaan Huffman-koodaus ei tietenkään näissä tapauksissa juurikaan pienennä tiedoston kokoa).

Seuraavana ongelmana oli koodisanojen kirjoittaminen tiedoston alkuun, jotta pakattu tiedosto voidaan myös purkaa. Tässä paloikin hieman turhan paljon aikaa, sillä lähdin toteuttamaan näin jälkikäteen ajateltuna järjetöntä ideaa – lisäsin koodattaviin sanoihin yhden ”erottimen”, jota käytettäisiin alussa erottelemaan koodisanat toisistaan. Tiedosto olisi alkanut siis:

1. erottimen pituus
2. erotin
3. ensimmäinen koodisana
4. erotin
5. toinen koodisana
6. …

Idea oli tietenkin aivan järjetön, sillä Huffman-koodaus on etuliitekoodaus, ts. mikään muu koodisana ei **ala** erottimen koodisanalla, mutta se voi silti esiintyä keskellä jotain muuta koodisanaa. Koodauksen purku tiedoston alusta olikin näin mahdotonta! Onneksi keksin ongelman huomattuani nopeasti toimivamman ratkaisun, nyt koodaus on:

1. ensimmäisen koodisanan pituus
2. ensimmäinen koodisana
3. toisen koodisanan pituus
4. toinen koodisana
5. …

Tämä luonnollisesti toimi paremmin. Koodisanan pituus tallennetaan 8 bittiin, joka sallii pituudet aina 256 bittiin asti. Koodisanoja on 257 (256 eri tavulle + 1 tiedoston lopun merkiksi), ja Huffman-koodauksen ominaisuuksiin kuuluu, että koodisanojen pituus on tällöin 0-256. 8 bittiin mahtuu itseasiassa vain pituudet 0-255, mutta käytännössä maksimipituisia koodisanoja ei muodostu, sillä ne vaatisivat hyvin poikkeuksellisen jakauman eri tavujen esiintymismäärille. (Yleisintä tavua enemmän kuin muita yhteensä, toiseksi yleisintä enemmän kuin loppuja yhteensä, kolmanneksi yleisintä taas enemmän kuin kaikkia loppuja yhteensä jne). Jos ylimääräistä aikaa jää projektin lopulla saatan kyllä kasvattaa tuon 9 bittiin, sillä vaikutus pakatun tiedoston kokoonkin on mitätön.

Viimeisenä toteutuspuolen haasteena tälle viikolle oli tarkastaa, että pakattu data sisältää sen mitä pitääkin, toisin sanoen toteuttaa koodauksen purku. Tähän asti pakatun datan oikeellisuudesta ei ollut oikeastaan mitään varmuutta, joten yllätyin varsin positiivisesti, kun purun ensimmäinen toteutus sai jo datan miltei täydellisesti purettua. Indeksointivirheiden vuoksi ensimmäisissä yrityksissä datan alusta ja lopusta katosi tai niihin ilmestyi yksittäisiä tavuja, mutta nyt nuokin on korjattu, ja ainakin sampleFiles-kansion testitiedostot näkyvät täsmäävän (ajelin sha-tarkistussummia testiksi).

Ohjelman perustoiminnallisuuden toteuttamiseen paloi tällä viikolla reilusti enemmän aikaa kun

olin kuvitellut, joten näiden lisäksi en saanut juuri muuta aikaiseksi kuin testit ja javadocit toteutetuille uusille luokille. Ohjelman vaatimista tietorakenteista itsetoteutettuja on vasta pakkauksen ydin, Huffman-puu. Ensi viikon olennaisimmat tavoitteet liittyvätkin tähän: Huffman-puun rakennuksessa solmujen tallennukseen kannattanee käyttää kekoa (tai muu tietorakenne, johon on nopeaa lisätä oikeaan järjestykseen yksi uusi alkio, ja josta saa nopeasti pienimmän alkion), ja koodisanojen tallennus purussa kannattanee toteuttaa hajautuksen avulla (voidaan nopeasti testata aina luetun bitin jälkeen, vastaako kertynyt bittijono jotain koodisanaa). Kaukaisempina tavoitteina edelleen kokonaan uudet pakkaus- ja muunnostekniikat, kuten jo viime viikolla mainitut run-length encoding ja Burrows-Wheeler -muunos.