Tiralabra 2015/periodi 3: Viikkoraportti 3

Kertauksena aimmilta viikoilta: Kakkosviikon palautukseen sain pakkausohjelmani perustoiminnallisuuden kuntoon. Huffman-koodaukseen pohjautuvan tavuittaisen pakkauksen sai ajettua tiedostoille tyypistä riippumatta, tosin pakkaustehokkuus suosi etenkin tekstitiedostoja.

Tiesin tällä viikolla projektiin olevan käytössä aikaa hieman tavallista niukemmin, joten valitsin viikon tavoitteet lähinnä kurssisivulla ilmoitettujen kolmosviikon tavoitteiden mukaan, enkä lähtenyt toteuttamaan varsinaisia uusia ominaisuuksia. Ensimmäisenä ohjelmassa oli Javan valmiiden tietorakenteiden korvaaminen omilla.

Huffman-puuta rakentaessa tarvitsen tietorakenteen, johon lisätään kerran n=256 alkiota, ja josta puunrakennuksen jokaisessa iteraatiossa poistetaan kaksi pienintä, ja lisätään yksi uusi alkio, kunnes listassa on enää yksi alkio (iteraatioiden määräksi tulee siis myös 256). Toteutin tähän tarkoitukseen OrderedLinkedList-listan, joka on aina järjestyksessä. Uuden alkion lisääminen oikeaan paikkaan vaatii O(n), mutta toisaalta pienimmän alkion saa aina ajassa O(1). Aluksi kun tietorakenne pystytetään, siihen lisätään O(n) alkiota, joten kokonaisaikavaativuus on $O(n^2)$. Pystyttämisen jälkeen kussakin O(n) iteraatiossa pienimpien alkioiden haku ja poisto vie 2O(1), mutta aikavaativuutta dominoi taas yhden alkion lisäys O(n). Iteraatioiden aikana listaoperaatioihin kuluu siis kaikkiaan $O(n^2)$.

Itseasiassa aikavaativuutta voisi vielä parantaa vaihtamalla toteutuksen kekoon – molemmat tarvittavat operaatiot (lisäys, pienimmän poisto) saisi toimimaan logaritmisessa ajassa, ja kokonaisaikavaativuudeksi tulisi O(nlogn). Tässä kuitenkin n=256, joten neliöllinenkään aika ei vaikuta koko ohjelman aikavaativuuteen millään tavalla.

Purkuvaiheessa koodisanat luetaan pakatun tiedoston alusta johonkin tietorakenteeseen. Kun itse pakattua dataa luetaan, jokaisen bitin jälkeen tulee tarkistaa vastaako luettu bittijono jonkin tavun tietorakenteeseen luettua koodausta. Hakuja tulee siis O(m) kappaletta, missä m on tiedoston pituus. Jotta tämä tarkastaminen saataisiin toimimaan mahdollisimman nopeasti, toteutin CodewordDictionary-tietorakenteen hajautuksen avulla, jolloin koodisanan tarkistus saadaan miltei vakioaikaiseksi (riittävän suurella hajautustaululla). Hajautusfunktiona käytin suoraan String.hashCodea.

Jälleen viikon viimeiseksi haasteeksi jäi testaamisen hiominen kuntoon... TBC