Toteutusdokumentti

Toteutin harjoitustyökseni suunnitelmien mukaisesti Huffman-koodia käyttävän pakkausohjelman. Lähdin aluksi toteuttamaan aivan tavallista huffman-algoritmia, mutta projektin alkuvaiheissa keksin, että teenkin kieltä sisältäville tekstitiedostoille viritellyn version huffmanista. Versiossani ohjelma luo yhden koodiston sijasta jokaiselle tiedoston merkille oman koodiston. Merkin koodi määräytyy sitten edellisen merkin koodistosta. Eli mitä yleisempi tietty merkki on edellisen merkkinsä seuraajana – sitä lyhyemmän koodin se saa. Tämä parantaa huomattavasti pakkaussuhdetta, kun tiedostot ovat jotain kieltä. Kielissä toistuu aina säännöllisyyksiä, esimerkiksi html:ssä “<” merkkiä seuraa usein “/”.

Ongelmana on toki puiden koon kasvu huomattavan suureksi, mutta vaikka sen ottaa huomioon, on ero melko pienilläkin tiedostoilla huomattavissa.

Ohjelmani on jaettu yksinkertaisiin komponentteihin: pakkaajaan, purkajaan, huffmankoodaajaan ja main-luokkaan, joka huolehtii käyttöliittymästä. Lienee itsestäänselvää, mikä on kunkin osan tehtävä. Tälläinen karkea luokkarakenne oli helppo toteuttaa. Monimutkaisempi luokkarakenne olisi varmasti ollut tyylikkäämpi ja enemmän ohjelmistosuunnittelun periaatteiden mukainen, mutta mielestäni tässä ohjelmassa yksinkertaisempikin versio on siedettävä.

Ohjelman aikavaativuutta on vähän hölmöä analysoida, koska suurin työ on aina tiedoston lukemisessa ja kirjoittamisessa. Se kestää niin pitkään, että muiden osien kuluttaman ajan merkitys jää aika olemattomaksi. Yhden huffman-puun tekeminen tapahtuu kuitenkin ajassa O(n log n). (n = aakkoston merkkien määrä) Logaritmikerroin tulee siitä, että merkit järjestetään käyttäen kekoa, jonka operaatiot toimivat ajassa log(n).

Koska ohjelmani tekee jokaiselle merkille oman huffman-puun, tulee huffman-puiden rakentamisen aikavaativuudeksi siis pahimmassa tapauksessa O(n^2 log n). Mutta jälleen, algoritmin kokonaissuoritusajasta puiden rakentaminen on niin häviävän pieni osa, että tätä lisäystä ei edes huomaa, varsinkin kun merkkien määrä tekstitiedostoissa on melko pieni.

Lisäyksiä ja parannettavia

Projektin alussa minulla oli ideana tehdä montaa ydintä hyödyntävä algoritmi. Aika loppui kuitenkin kesken, joten tämä jää toteutettavaksi tulevaisuudessa. Olen myös vähän epäilevä, pystyykö pakkausohjelma hyödyntämään useaa ydintä tehokkaasti. Tiedoston lukeminen on eniten aikaa vievä osuus, ja en tiedä onko sitä mahdollista rinnakkaistaa.

Huffman-koodini virittelyä voisi myös olla hauska jatkaa. Demotilaisuudessa tuli ehdotus koodin generoimista kahden peräkkäisen merkin perusteella. Tälläinen versio lyhentäisi entisestään usein toistuvien merkkijonojen koodeja, mutta epäilen, että puiden vaatima tila voisi kasvaa liian isoksi. Nykyisessäkin versiossa puut vievät huomattavan osan pakatusta tiedostosta.

Toki puiden koodaamista voisi myös tehostaa. Nyt ne ovat UTF-16 -koodattuja, joka on suoraan sanoen hirvittävää tuhlausta. Tässä versiossa kanonisointi* ei ehkä toimisi, mutta

jonkunlaista muuta virittelyä puiden koodaus kyllä kaipaisi. Ehkä edellisen kappaleen lisäviritysideoiden tapaiset muokkauksetkin voisivat olla järkeviä, jos puille olisi joku kompakti tallennusmuoto.

Lähteinä olen käyttänyt Thomas H. Cormenin kirjaa “Introduction to Algorithms” sekä Wikipedian Huffman-koodista kertovaa sivua (http://en.wikipedia.org/wiki/Huffman_coding)

*Kanonisointi tarkoittaa, että merkkien koodeista voidaan kirjoittaa vain koodien pituudet ylös. Tämä onnistuu, kun aakkosto on ennalta tiedossa. Mielestäni tämä ei toimi omassa versiossani, koska tässä useissa puissa saattaa olla vain muutama merkki. Siten jouduttaisiin tallentamaan paljon turhia koodeja.