Ohjelman yleisrakenne

Tällä hetkellä ohjelmassa on kaksi eri tiedostonpakkausalgoritmia, Huffmanin koodaus, LZW ja kesken oleva minimal-viable-product LZ77. Ohjelma ja tämä toteutusdokumentti ovat molemmat tällä hetkellä pahasti kesken.

Huffmanin koodauksen olen toteuttanut käyttäen hyväksi hajautustaulua ja prioriteettijonoa. Hajautustaulun teossa on myöskin käytetty ArrayList tietorakennetta. Nämä kaikki olen itse tehnyt ja vain niin laajana että Huffmanin koodaus toimisi.

Huffmanin koodauksessa pakatessa käytettään pääasiassa luokkaa ”HuffmanCompression”. Ensin käydään teksti läpi ja lisätään kaikkien merkkien esiintyvyydet hajautustauluun. Tästä hajautustaulusta muodostettaan prioriteettijonon avulla Huffmanin puu. Puu käydään läpi ja merkitään jokaista mahdollista merkkiä vastaava Huffmanin koodi hajautustauluun. Tämän jälkeen kirjoitettaan merkkien esiintyvyydet pakatun tiedoston alkuun. Sitten käydään alkuperäinen tiedosto läpi merkki merkiltä ja kirjoitettaan merkkejä vastaavat koodit hajautustaulusta pakattuun tiedostoon. Pakkauksen suoritus on nyt valmis.

Huffmanin koodauksessa purettaessa käytettään pääasiassa luokkaa ”HuffmanDecompression”. Ensin luetaan pakatun tiedoston alkuosasta merkkien esiintyvyydet hajautustauluun. Tästä hajautustaulusta muodostettaan prioriteettijonon avulla Huffmanin puu, samalla tavalla kuten tiedostoa pakatessa. Tämän jälkeen luetaan pakatun tiedoston loppuosa läpi ja puretaan Huffmanin koodi edellä mainitun puun avulla.

LZW algoritmissa olen toteuttanut käyttäen hyväksi jajautustaulua ja ArrayListiä, jotka olen itse toteuttanut niin laajana että LZW toimisi.

LZW algoritmin pakkaus käyttää luokkaa ”LZWCompression”. LZW tiedoston pakkauksessa aluksi alustetaan sanakirja hajautustaulu kaikilla 256 ascii merkillä ja niitä vastaavalla koodilla. Sitten tämän jälkeen käydään tiedosto läpi, jos sanakirja sisältää wc(missä c on luettumerkki) niin w=wc, muuten koodi ArrayListaan lisätään w vastaava koodi sanakirjasta ja sanakirjaan lisätään wc ja uusi w = c. Tämän jälkeen kirjoitettaan koodisto tiedostoon 24 bittiä kerrallaan.

LZW algoritmin purkaus käyttää luokkaa ”LZWDecompression”. Purkaminen sujuu melkein samalla tavalla kuin pakkaaminen. Ensin luetaan tiedostoa 24 bittiä kerallaan ja tästä saadaan tätä vastaava koodi joka lisätään ArrayListaan. Seuraavaksi alustetaan sanakirja kuten pakatessakin. Nyt voidaan koodit purkaa sanakirjan avulla.

LZ77 toiminta perustuu siihen että liikkuvalla ikkunalla etsitään tiedostosta toistoja. Sitten kun semmoinen toisto löytyy niin kirjoitetaan sen paikalle etäisyys ja kuinka pitkä tämä toisto on. Purku tapahtuu sitten niin että kun tullaan kohdalle mihin on kirjoitettu toiston etäisyys ja pituus pari se voidaan purkaa siirtämällä pointteri kohtaan missä alkuperäinen match on ja kirjoitettaan se etäisyyspituus/parin tilalle.

Saavutetut aika- ja tilavaativuudet (m.m. O-analyysi pseudokoodista)

Teen myöhemmin.

Suorituskyky- ja O-analyysivertailu (mikäli työ vertailupainotteinen)

Suoritus kykyyn liittyen suorituskykytestaus.pdf ja suorituskyky viikko5.pdf tiedostossa.

Työn mahdolliset puutteet ja parannusehdotukset

LZW toimii tosi hitaasti isoilla tiedostoilla, eli se pitäisi korjata.

Lähteet

http://en.wikipedia.org/wiki/Huffman_coding

http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch

https://en.wikipedia.org/wiki/LZ77_and_LZ78

Tosi monta eri stackoverflow sivua liittyen aiheseen.