TIETORAKENTEET JA ALGORITMIT -HARJOITUSTYÖ: TIEDON TIIVISTYS MÄÄRITTELYDOKUMENTTI

VILLE TENHUNEN
VILLE.TENHUNEN@GMAIL.COM
013618793
22.12.2013

Tässä harjoitustyössä on tarkoitus toteuttaa kaksi tiedon tiivistys -algoritmia, joilla tekstitiedostoja saadaan pakattua pienempään tilaan. Toteutettavat algoritmit ovat Huffmanin algoritmi ja Lempel-Ziv-Welch:n algoritmi (LZW). Molemmat ovat tietoa hukkaamattomia algoritmeja. Tämän jälkeen on tarkoitus testata näitä algoritmeja erilaisilla tekstitiedostoilla.

Huffmanin algoritmissa merkeille määritetään uudet binäärikoodit sen mukaan kuinka yleinen kyseinen merkki tekstissä on. Yleisemmille merkeille annetaan lyhyemmät binääriyhdisteet kuin harvinaisemmille merkeille ja näin päästään lyhyemmillä binäärijonoilla, koska kaikille merkeille ei ole yhtä pitkiä binääriesityksiä.

Tässä työssä Huffmanin algoritmi toteutetaan niin, että algoritmille annetaan syötteenä teksti sekä kyseiselle tekstille toimiva merkkien esiintymistodennäköisyystaulukko. Tämän lisäksi toteutetaan versio, jossa kyseisiä todennäköisyyksiä ei tiedetä, jolloin merkkien esiintymistiheydet lasketaan ja sen perusteella muodotetaan koodaus. Algoritmia varten toteutetaan minimikeko, jonka painoina on todennäköisyydet/esiintymistiheydet, ja jonka avulla muodostetetaan binääripuu, josta merkkien koodit luetaan. Binääripuun muodostukseen kuluu aikaa O(n log n), jossa n on eri merkkien määrä. Tämä tulee siitä, että jokainen keko operaatio kestää O(log n) verran ja näitä operaatioita tapahtuu lineaarinen määrä.

Puusta koodilla merkin lukeminen tapahtuu logaritmisessa ajassa, jolloin koodia luetaan siten, että juuresta liikkeelle lähtien aina arvolla nolla siirrytään vasempaan lapseen ja vastaavasti ykkösellä oikeaan lapseen. Kun lapsia ei enää ole jäljellä, luetaan kyseisen lehden merkki ja palataan takaisin puun juureen. Näin ollen, mitä lähempänä juurta olevassa lehdessä ollaan, sitä lyhyemmäksi koodi jää. Joten yleisimmät merkit päätyvät näihin lehtiin.

Huffmanin algoritmi voi toimia myös lineaarisessa ajassa, jos eri merkkien todennäköisyydet saadaan algoritmille valmiiksi järjestettynä. Tällöin kahta prioriteettijonoa käyttäen voidaan puu muodostaa lineaarisessa ajassa.Tämän merkitys on kuitenkin vähäisempi, jos järjestäminen pitää kuitenkin toteuttaa ennen algoritmia. Lisäksi tämän nopeuden merkitys ei ole kovin suuri, koska n on kirjainten määrä, joka ei siis kasva kovin suureksi.

Toisena tiivistys algoritmina toteutetaan Lempel-Ziv-Welch:n algoritmi, eli (LZW). Siinä aluksi luodaan kirjasto, johon lisätään kaikki yhden merkin mittaiset merkkijonot. Seuraavaksi pakkausvaiheessa käydään syötettä läpi paloittain niin, että verrataan merkkijono pätkää kirjastoon, ja etsitään sieltä pisin sopiva merkkijono, ja tämän kirjaston alkion indeksikohta kirjastossa laitetaan koodausvirtaan. Tämän jälkeen kirjastoon lisätään uusi alkio, joka on aiempi löydetty merkkijono plus seuraava merkki. Tämän jälkeen syötteen lukua jatketaan kohdasta mihin jäätiin ennen seuraavan merkin lisäystä. Näin kirjasto laajenee samalla kun pakkauskoodia muodostetaan. Pakkauskoodiksi muodostuneet kirjaston indeksit muutetaan vielä vaihtuva mittaisiksi bittijonoiksi, jolloin koodista saadaan lyhyt. Algoritmin nerokkuus muodostuu siitä, että purkuvaiheessa vastaava kirjasto voidaan luoda koodia purkaessa, joten kirjaston tietoja ei tarvitse siirtää koodin mukana.