Mika Huttunen

Määrittelydokumentti

Yleistä

Tarkoituksenani on kehittää ohjelma, jolla voi pakata ja purkaa tavallisia tekstitiedostoja ja se tukee myöhemmin ehkä myös muita tiedostotyyppejä.
Ohjelmalla voi pakata tiedostoja ainakin kahta eri pakkausalgoritmia käyttäen; käyttäjä saa valita käyttääkseen joko Huffman -algoritmin tai Lempel-Ziv-Welch -toteutusta.

Valitsin että käytän ym. algoritmeja ongelmanratkaisuun, sillä ne ovat muutenkin yleisesti käytettyjä ja lisäksi minua ohjeistettiin toteuttamaan ohjelmani niitä käyttäen.

Toteutan ohjelman käyttäen tietorakenteina ainakin itsetoteutettua minimikekoa sekä binääripuuta. Lisäksi toteutan hajautustietorakenteen jollain lailla, jota voidaan hyödyntää eri merkkien laskemisessa, kun pakataan tekstitiedostoja Huffman -algoritmia käyttäen.
Muita mahdollisia aputietorakenteita tulee tod. näk. käyttöön myöhemmin, mutta nämä vaikuttavat näin alkuun ohjelman toimimisen kannalta tärkeimmiltä.
Lempel-Ziv-Welcin toteutus toimii hajautustaulua käyttäen ja ei tarvitse muita tietorakenteita.

Syötteenä ohjelma saa tiedoston polun, joka annetaan sille kirjoittamalla tekstikäyttöliittymän kautta. Tiedostot itsessään ovat tavallisia tekstitiedostoja (.txt).

Algoritmien toimintaperiaatteet

Huffman

<u>Pakkaamis- ja purkuvaiheet:</u>
Aikavaativuus: O(n), missä n on syötteenä olevan tekstitiedoston koko
Tilavaativuus: O(m), missä m on siinä olevien erilaisten tavujen lukumäärä

<u>Alustus</u>

Huffmanin algoritmi toimii siten että ennen sitä luetaan pakattava tekstitiedosto ja luoda solmuja jokaista siinä esiintyvää merkkiä varten. Solmuja syntyy siis n kpl, missä n = “merkkien lkm.”. Näille solmuille kirjataan lisäksi tietoa siitä, kuinka monta kertaa ne ovat tiedostoa lukiessa esiintyneet, ja ne asetetaan tämän tiedon nojalla suuruusjärjestykseen minimikekoon.

Huffmanin algoritmi

Huffmanin koodauksessa kaikki solmut poistetaan keosta (ajassa O(1)) luoden samalla myös ylimääräisiä solmuja (n - 1) kpl, joten käsiteltäviä solmuja on siis 2n - 1 = O(n) kpl. Jokainen näistä luoduista solmuista asetetaan myös minimikekoon käsittelyä varten. Tämä lisäysoperaatio vie aikaa O(logn), joten algoritmin aikavaativuus on suuruusluokkaa O(nlogn).

Pakkausvaihe

Kun algoritmi on suoritettu, on saatu aikaiseksi binääripuu, joka kertoo, mitä binääriesitystä kukin tiedoston merkki vastaa. Nyt luodaan uusi tiedosto, kelataan alkuperäinen tiedosto läpi ja kirjoitetaan tähän uuteen tiedostoon alkuperäisen tiedoston data käyttäen siinä olevien merkkien binääriesityksiä, jotka Huffmanin algoritmilla määritettiin. Tähän tiedostoon sisällytetään myös ko. binääripuu, jotta muodostettu pakkaus voidaan myöhemmin purkaa.

Purkuvaihe

Muodostetun pakkauksen purku tapahtuu käytännössä päinvastoin kuin pakkaaminen; tarkastellaan alkuperäisen tiedoston dataa vastaavaa binääriesitystä (pakkauksen datassa) ja käytetään apuna pakkauksessa mukana olevaa Huffman puuta. Näiden avulla löydetään merkit, jotka uuteen muodostettavaan tiedostoon tullaan kirjoittamaan.

Lempel-Ziv-Welch

Pakkausvaihe

Aikavaativuus: O(n), missä n on pakattavan tiedoston koko
Tilavaativuus: O(n * log(m)), missä m on tiedoston lukiessa kirjattavien erilaisten merkkijonojen määrä

Lempel-Ziv-Welchin algoritmissa alkuperäistä tiedostoa kelataan läpi koodaten samanaikaisesti siinä esiintyviä erilaisia merkkijonoja erilaisiksi binääriesityksiksi. Nämä binääriesitykset talletetaan pakkaukseen joka muodostetaan ja näiden binääriesitysten avulla voidaan alkuperäinen tiedosto saada purettua.

Aikavaativuus on O(n), sillä koko teksti on kelattava ainakin kertaalleen läpi. Tilavaativuus on taas O(m), sillä tekstiä kelatessa muodostetaan m kpl erilaisia merkkijonoja, jotka talletetaan muistiin esim. hajautustietorakenteeseen.

Purkuvaihe

Tiedoston purun vaativuusluokat ovat täsmälleen samat kuin pakatessa.
Tiedosto puretaan lukemalla siitä tietynmittaisia bittijonoja ja kääntäen nämä hajautustietorakennetta käyttäen alkuperäisen tekstitiedoston merkeiksi. Purkuvaiheessa muodostetaan samankokoinen hajautustietorakenne kuin lukiessakin (tilav. O(m)) ja tekstin sijasta luetaan bittijonoja, joiden yhteenlaskettu pituus on suurinpiirtein lineaarinen alkuperäisen tekstin pituuteen.

Molempien vaiheiden lopussa kirjoitetaan uuteen muodostettavaan tiedostoon luettu teksti joko bittiesityksenä tai purettuna tekstinä. Tämänkin aikavaativuus O(n) ja tilavaativuus riippuu javan toteutuksesta, todennnäköisesti O(1).

Lähteet

Huffman Coding, Wikipedia, 16.6.2014, http://en.wikipedia.org/wiki/Huffman_coding

Lempel-Ziv-Welch, Wikipedia, 19.6.2014,
http://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93Welch