Pietari Outinen

Häviöttömät ja yleiskäyttöiset tiedonpakkausmenetelmät

Tietotekniikan kandidaatintutkielma

29. huhtikuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Pietari Outinen

Yhteystiedot: pietari.a.outinen@student.jyu.fi

Ohjaaja: Sanna Mönkölä

Työn nimi: Häviöttömät ja yleiskäyttöiset tiedonpakkausmenetelmät

Title in English: Lossless and universal data compression methods

Työ: Kandidaatintutkielma

Sivumäärä: 22+0

Tiivistelmä: Nykyaikaiset verkko- ja mobiilipalvelut eivät olisi mahdollisia ilman tiedon pakkaamista. Yhä useammat laitteet ovat yhteydessä internetiin, sekä eri palvelut siirtävät dataa koko ajan enemmän, joten tiedonpakkaus on tapa hyödyntää rajallinen tiedonsiirtokapasiteetti paremmin. Tässä tutkielmassa selvitetään, miten eri tiedonpakkausmenetelmät toimivat, ja minkälainen on hyvä häviötön ja yleiskäyttöinen pakkausmenetelmä.

Avainsanat: Tiedonpakkaus, Häviötön pakkaus, DEFLATE

Abstract: Modern network and mobile services would not be possible without data compression. More and more devices are on the Internet and online services are constantly transferring more data. So, data compression is a way to make better use of the limited data transfer capacity. This thesis explains how different data compression methods work and what makes a good lossless and universal compression method.

Keywords: Data compression, Lossless compression, DEFLATE

Kuviot

Taulukot

Sisältö

1 Johdanto

Tiedonpakkaus *(eng. Data compression)* tarkoittaa informaatiota esittävän datan muokkaamista vähemmän tilaa vievään muotoon, ja se onkin ollut yksi tärkeimmistä nykyajan mobiili- ja verkkopalveluiden mahdollistavista tekniikoista (Sayood 2006, s. 1; Pu 2006, s.1-2). Vaikka tiedonsiirtotekniikat kehittyvät koko ajan paremmiksi, on tiedonpakkaus edelleen tärkeä osa mahdollistamassa sujuvaa verkkoliikennettä. Verkon yli tarjottavien palveluiden, sekä verkkoon yhdistettävien laitteiden määrä kasvaa jatkuvasti, joten verkkojen käyttöaste tulee vääjäämättä kasvamaan tulevien vuosien aikana.

Tiedonpakkauksella voidaan parantaa tietoverkkojen rajallisia tiedonsiirtonopeuksia, sekä taata tiedonsiirtokapasiteetin riittävyyden kaikille käyttäjille ja laitteille. Erityisesti kehittyvissä maissa ja harvoin asutetulla seudulla, jossa verkot voivat olla vielä melko hitaita, tiedonpakkaus on hyvä keino nopeuttaa tiedonsiirtoa. Tietoverkot eivät ole kuitenkaan ainut sovellus johon tiedonpakkausta voidaan hyödyntää. Esimerkiksi datan tallentamisessa tiedonpakkausta voitaisiin hyödyntää vielä enemmän, sekä sen avulla voitaisiin nopeuttaa suurten datamassojen käsittelyä.

Tässä tutkielmassa ei kuitenkaan tarkastella tiedonpakkauksen soveltuvuutta eri käyttötarkoituksiin, vaan keskitytään tiedonpakkausmenetelmien toteutukseen. Tiedonpakkausta voidaan toteuttaa joko häviöllisesti tai häviöttömästi. Häviötön pakkaus tarkoittaa pakkaustapaa, joka säilyttää täsmälleen alkuperäisen tiedon, ja häviöllisessä pakkauksessa tieto on joiltakin osin puutteellinen. Tässä tutkielmassa keskitytään häviöttömiin menetelmiin, koska ne ovat häviöllisiä menetelmiä yleiskäyttöisempiä, sekä sovellettavissa useampiin käyttötarkoituksiin.

Tutkielman toisessa luvussa käydään läpi tiedonpakkauksen perusteita, jonka jälkeen kolmannessa luvussa tarkastellaan menetelmien kehittämistä, sekä muutamia erilaisia menetelmätyyppejä. Neljännessä luvussa esitellään yleiskäyttöinen DEFLATE-menetelmä, joka on yksi laajimmalle levinneistä yleiskäyttöisistä menetelmistä, sekä vertaillaan sitä muihin nykyaikaisiin menetelmiin.

2 Tiedonpakkaus

Pun (2006) ja Sayoodin (2006) määritelmien mukaan tiedonpakkauksella tarkoitetaan informaatiota esittävän datan muokkaamista eli pakkaamista vähemmän tilaa vievään muotoon. Pakkaaminen toteutetaan erilaisilla operaatioilla, joilla poistetaan informaation säilyttämisen kannalta ylimääräinen data (Pu 2006, s. 1-2; Sayood 2006, s. 1).

Vaikka tiedonpakkauksen voidaan ajatella nimensä perusteella olevan vain datan pakkaamista pienempään muotoon, kuuluu siihen yleisesti myös toinen vaihe, jossa palautetaan eli puretaan pakattu data takaisin alkuperäiseen muotoon (Pu 2006, s. 3-5; Sayood 2006, s. 3). Tässä tutkielmassa pakkaavaa algoritmia kutsutaan pakkaajaksi *(eng. encoder)* ja purkavaa algoritmia purkajaksi *(eng. decoder)*.

Pakkaamisen ja purkamisen jälkeen data voi olla täsmälleen sama kuin alkuperäinen pakkaamaton data, mutta menetelmästä riippuen se voi olla joiltakin osin puutteellinen (Pu 2006, s. 3-5; Sayood 2006, s. 3). Tällöin puhutaan häviöllisestä ja häviöttömästä pakkauksesta, joiden perusteet käydään läpi seuraavaksi.

2.1 Häviötön pakkaus

Sayoodin (2006) mukaan häviöttömällä pakkauksella tarkoitetaan pakkaustapaa, joka säilyttää alkuperäisen tiedon täydellisenä pakkaamisen ja purkamisen välillä. Häviöttömiä menetelmiä käytetään varsinkin silloin, kun pienikin muutos alkuperäisen tiedon ja pakkaus–purkuoperaation jälkeisen tiedon välillä muuttaisi tiedon merkitystä (Sayood 2006, s. 4-5).

Häviöttömät pakkausmenetelmät ovat useimmiten tilastollisia menetelmiä tai hakemistomenetelmiä, joiden operaatioilla vähennetään datan määrää vaikuttamatta tiedon sisältöön. Tilastollisten menetelmien perusteet esitellään luvussa 3.4 ja hakemistomenetelmien luvussa 3.5. Häviöttömät menetelmät soveltuvat hyvin esimerkiksi tekstin pakkaamiseen, jossa jokaisen yksittäisen kirjaimen merkitys on tärkeää informaation pysyvyyden kannalta (Sayood 2006, s. 4). Häviötöntä pakkausta voidaan soveltaa myös muunlaiseen symboliseen dataan kuten kuviin.

Kuvien data koostuu yhdenmuotoisista pikseleistä, jotka sisältävät kullekin pikselille ominaisia lukuarvoja. Pikseleiden rakenne saadaan pakattua hyvin tekstinpakkauksen menetelmillä, sekä pikseleiden lukuarvot käyttäen numeerista menetelmää, kuten differentiaalikoodausta, joka esitellään luvussa 3.6 (Sayood 2006, s. 193). Häviöttömien menetelmien pakkauskyky ei kuitenkaan yleisesti riitä kuvien pakkaamiseen, sillä jo yhden megapikselin kokoinen kuva sisältää tiedon miljoonasta pikselistä. Monissa tapauksissa valitsemalla käyttötarkoitukseen sopiva häviöllinen menetelmä, voidaan saavuttaa parempi pakattavuus ilman merkittävää informaation puuttumista.

2.2 Häviöllinen pakkaus

Sayoodin (2006) mukaan häviöllisellä pakkauksella tarkoitetaan pakkaustapaa, joka ei säilytä aivan kaikkea alkuperäistä tietoa pakkauksen yhteydessä. Tätä tapaa käytetään silloin, kun aivan jokainen tiedon palanen ei ole merkityksellinen tiedon ymmärrettävyyden kannalta. Esimerkiksi äänen ja videon pakkauksessa voidaan hävittää melko paljon tietoa ja silti säilyttää alkuperäisen tiedoston tarkoitus (Sayood 2006, s. 5).

Häviöllisellä menetelmällä pakatusta datasta ei voida palauttaa alkuperäistä dataa, joten sen käyttökohteet ovat rajalliset (Pu 2006, s. 6). Esimerkiksi usean desimaalin reaaliluvut, kuten digitaalisessa muodossa olevat liukuluvut, voidaan pyöristää viemään vähemmän tilaa, mutta samalla menetetään mahdollisuus palauttaa alkuperäinen ja tarkka luku. Häviöttömiä menetelmiä käytetään usein varsinkin multimedian pakkaamiseen, sillä ne kestävät ominaisuuksiensa puolesta hyvin pieniä muutoksia.

Multimedian pakkaamisessa keskitytään osaan, jonka pienet erot tai puute jäävät ihmisen havaintokyvyn ulkopuolelle tai niiden merkitys ymmärrettävyyden kannalta ei ole olennaista (Sayood 2006, s. 200). Vaikka nämä tiedot jäävät ihmisen havaintokyvyn ulkopuolelle, ovat ne mukana alkuperäisessä datassa ja vievät näin ollen turhaa tilaa. Nämä arvot ovat kuitenkin hyvä säilyttää, jos tiedosta halutaan jälkikäteen saada jotakin lisätietoja. Esimerkiksi kuvasta voidaan jälkikäteen tuoda esille asioita, jotka ovat jääneet alkuperäisessä datassa ihmisen havaintokyvyn ulkopuolelle. Toisaalta internetissä esillä olevaa kuvaa ei todennäköisesti tarvitse enää muokata, joten silloin voidaan valita häviöllinen menetelmä.

3 Menetelmien kehittäminen

Tieto voidaan jakaa karkeasti neljään perustyyppiin, jotka ovat teksti, ääni, kuva ja video. Data, jolla tietoa esitetään, sisältää tietotyypistä riippuen erilaisia symboleita eli yksiköitä, joista tieto koostuu (Pu 2006, s. 20). Symboli voi olla esimerkiksi kuvan pikseli, joka sisältää tiedon väriarvoista tietyssä kuvan kohdassa tai kirjain tekstissä, jonka tarkoitus on erottaa se yksilöivällä tavalla muista kirjaimista. Eri symboleilla ovat omat tarkoituksensa ja rakenteensa ja siksi erilaiset tietotyypit tarvitsevat erilaisia pakkausmenetelmiä. Tässä luvussa käydään läpi, miten tiedonpakkausmenetelmiä kehitetään ja arvioidaan, sekä lopuksi esitellään muutamia tekniikoita, joilla dataa saadaan pakattua.

3.1 Mallintaminen ja koodaus

Sayoodin (2006) mukaan tiedonpakkausmenetelmien kehittäminen koostuu yleisesti mallintamisesta ja koodauksesta. Mallintamisvaiheessa pyritään dataa analysoimalla tuottamaan malli, jota voidaan käyttää hyödyksi koodausvaiheessa (Sayood 2006, s.6). Malli voi sisältää esimerkiksi tiedon datan sisältämistä symboleista, erilaisista rakenteista, symbolien tai arvojen suhteesta toisiinsa tai mistä tahansa tiedosta, joka auttaa tiedon pakkaamisessa (Sayood 2006, s. 6 - 11). Mallintaminen on tärkeä osa menetelmäkehitystä, sillä tietylle datalle ja sen piirteille suunnitellut mallit ovat yleensä tehokkaampia (Sayood 2006, s. 23) ja saavuttavat parempia pakkaussuhteita (Sayood 2006, s. 10) kuin yleiskäyttöiset mallit.

Kotimaisten kielten keskus (2018) määrittelee koodauksen esitysmuodon muokkaamisella määriteltyjen sääntöjen mukaisesti. Tiedon pakkauksessa tämä tarkoittaa tietoa esittävän datan pakkaamista pienempää tilaa vievään muotoon, sekä sen palauttamista esitettävään muotoon mallinnusvaiheessa tehdyn mallin pohjalta (Pu 2006, s. 44). Eri mallien pohjalta tehtyä koodaamista on hyvä pystyä vertailemaan ja valitsemaan parhaiten sopiva menetelmä pakattavalle datalle. Seuraavaksi esitellään tapoja, joilla tiedonpakkausmenetelmien toimintaa yleisimmin arvioidaan ja vertaillaan keskenään.

3.2 Menetelmien arviointi

Yksi yleisimmistä tavoista arvioida pakkausmenetelmää on laskea kuinka paljon tiedoston koko pienenee, ja se voidaan ilmaista joko tiedostojen kokojen pakkaussuhteena ($\frac{pakattu}{pakkaamaton}$) tai -kertoimena ($\frac{pakkaamaton}{pakattu}$) (Pu 2006, s. 11). Toinen tapa menetelmien arviointiin on laskea pakkauksen ja purkamisen nopeutta eli käytännössä algoritmin suoritusnopeutta, kompleksisuutta ja muistinkäyttöä (Pu 2006, s. 12). Varsinkin pakkaussuhde, sekä pakkaus- ja purkuoperaatioiden nopeus ovat alalla useimmin käytettyjä arviointitapoja. Häviöllisille menetelmille käytetään lisäksi erilaisia mittareita, joilla verrataan alkuperäistä tietoa pakkauksen läpikäyneeseen tietoon ja arvioidaan kuinka hyvin tieto on pysynyt operaatioiden välissä (Pu 2006, s. 13).

Eri arviointitavat sopivat erilaisiin käyttötarkoituksiin, ja usein on hankala vertailla menetelmien paremmuutta. Menetelmä voi esimerkiksi saavuttaa parempia pakkaussuhteita suoritusnopeuden tai muistinkäytön kustannuksella, joten menetelmän paremmuus riippuu aina käyttötapauksesta. Tietojen pitkäaikaiseen varastointiin sopii hyvän pakkaussuhteen menetelmä, jossa algoritmin nopeus ei ole tärkein kriteeri, kun puolestaan videon suoratoistossa tiedostokoko ja purkunopeus ovat tärkeimmät kriteerit. Reaaliaikainen tiedonsiirto, kuten videopuhelut tai suorat lähetykset, vaatii puolestaan tasapainoilua suoritusnopeuksien, sekä pakkaussuhteen lisäksi verkkojen nopeuksien ja muistinkäytön kanssa. Eri asiat vaikuttavat menetelmän toimivuuteen, joten arviointikriteerit tulee valita huolella, ja paras testi on kuitenkin kokeilla menetelmän toimivuutta käytännössä.

3.3 Jakson pituuden koodaus

Jakson pituuden koodaus on yksi yksinkertaisimmista häviöttömän pakkauksen menetelmistä ja se perustuu peräkkäisten symbolien tai jaksojen koodaukseen. Koodauksessa toistuva peräkkäinen jakso korvataan tiedolla kuinka monta kertaa jakso toistuu ja mitä symboleja jaksoon kuuluu (Pu 2006, s. 49-63). Yksinkertaisimmillaan se voi tarkoittaa kuvion 1 mukaista koodausta.

Kuvio 1. Jakson pituuden koodaus

A BB CCC DDDD EEEEE $\rightarrow$ A 2B 3C 4D 5E

Kuvion 1 mukaisessa koodauksessa on hyvä huomata, että merkkijono ”2B” on koodattuna ”2B” ja purettuna ”BB”, jonka estäminen täytyy huomioida jo kehitysvaiheessa. Jakson pituuden koodaus toimii vain silloin, kun data sisältää paljon peräkkäisiä jaksoja, joten muussa tapauksessa on hyvä valita esimerkiksi tilastollinen menetelmä.

3.4 Tilastolliset menetelmät

Tilastollisissa menetelmissä yhdistyvät vaihtelevan pituuden koodaus ja symboleiden esiintymisen todennäköisyyksien laskeminen, joiden tarkoituksena on minimoida yhden symbolin koodaamiseen tarvittava bittimäärä (Pu 2006, s. 23). Käydään ensin läpi mitä vaihtelevan pituuden koodaus tarkoittaa ja jatketaan siitä tilastollisiin menetelmiin, joilla pakkaussuhdetta saadaan kasvatettua.

Vaihtelevan pituinen koodaus nimensä mukaisesti tarkoittaa symboleiden koodaamista eri mittaisilla koodeilla (Pu 2006, s. 21). Teksti koodataan yleensä digitaaliseen muotoon ASCII-koodauksella tai vastaavalla koodauksella, jossa jokaista merkkiä vastaa vakiomittainen binääriluku. Tilanteessa, jossa pakattava data sisältää vain kolmea eri kirjainta, ei tarvita kaikkia ASCII-koodauksen kahdeksaa bittiä erottamaan kirjaimia toisistaan (Pu 2006, s. 21).

Kuvio 2. Vaihtelevan pituuden koodaus

Symboli	Koodi
A	0
B	10
C	11

$$\text{ABBCCC} \rightarrow 01010111111$$

Kuviossa 2 nähdään, miten merkkijono voidaan koodata vaihtelevan mittaisella koodauksella. Alkuperäinen merkkijono ASCII-koodauksella esitetään 48 bitillä ja kuviossa 2 nähdyssä esimerkissä 11 bitillä. Menetelmää kehittäessä on hyvä kiinnittää huomiota, että mitään koodia tai koodijonoa ei esitetä muiden koodien yhdisteenä, jotta alkuperäinen tieto saadaan purettua. Yksi suosituimmista tavoista tehdä yksikäsitteinen koodaus on käyttää hyödyksi puurakennetta, kuten Huffmanin puuta (kuvio 6).

Kuvion 2 koodaus ei ole kuitenkaan optimaalinen ratkaisu, sillä siinä useimmiten esiintynyt merkki koodattiin pidemmällä koodilla, kuin harvemmin esiintynyt. Koodaus voidaan optimoida tilastollisten menetelmien avulla, joiden tarkoituksena on koodata usein esiintyvät symbolit vähemmän bittejä vievillä binääriluvuilla, jolloin harvemmin esiintyviin symboleihin on varaa käyttää enemmän bittejä (Sayood 2006, s. 41). Yksi tunnetuimmista tilastollisista menetelmistä on Huffmanin koodaus, joka esitellään luvussa 4.1.

3.5 Hakemistomenetelmät

Kun jakson koodauksessa ja tilastollisissa menetelmissä keskityttiin pakkaamaan dataa pääosin symbolien tarkkuudella, hakemistomenetelmät pyrkivät vähentämään toistuvien ja laajempien rakenteiden viemää tilaa. Hakemistomenetelmissä datassa usein toistuvat rakenteet lisätään hakemistoon ja nämä rakenteet korvataan datassa hakemistoon osoittavilla indekseillä (Sayood 2006, s. 117). Kuviossa 3 nähdään yksinkertainen esimerkki hakemistomenetelmien toimintaperiaatteesta.

Kuvio 3. Hakemistomenetelmä

Indeksi	Sana
1	kissa
2	istu

kissa istuu kissanistuimessa $\rightarrow$ 1 2u 1n2imessa

Hakemistot voivat olla joko staattisia tai dynaamisia. Staattisia hakemistoja käytetään, kun tiedetään ennalta datan ominaisuudet eli käytännössä toistuvat rakenteet (Sayood 2006, s. 118-119). Dynaamiset menetelmät puolestaan luovat dataa varten aina omat uniikit hakemistot, jotka perustuvat sillä hetkellä käsiteltävän datan toistuviin rakenteisiin (Sayood 2006, s. 118-121). Hakemistomenetelmissä täytyy pitää huolta, että indeksi vie vähemmän tilaa kuin itse korvattava jakso. Dynaaminen Lempel–Ziv-hakemistomenetelmä esitellään luvussa 4.2.

3.6 Numeeriset menetelmät

Viimeisenä esitellään yksi häviötön, sekä yksi häviöllinen numeerisen datan pakkausmenetelmä. Luvut tallennetaan digitaalisessa muodossa usein vakiomittaisena binäärilukuna, vaikka luvut sisältäisivät eri määrän numeroita. Paljon lukuja sisältävän datan pakkausmenetelmiin on hyvä lisätä jokin tapa esittää muokatut luvut vähemmän bittejä vievällä tavalla.

Ensimmäinen esiteltävä numeerinen menetelmä on differentiaalikoodaus, joka on häviötön menetelmä. Siinä ei tallenneta jokaista arvoa kokonaisena, vaan käytetään hyväksi jotakin lähtöarvoa, ja datan arvot tallennetaan lähtöarvon ja todellisen arvon erona (Sayood 2006, s. 325). Differentiaalikoodaus toimii parhaiten, kun lähtöarvo on todella suuri, ja muiden arvojen vaihtelu suhteessa lähtöarvoon on hyvin pieni.

Kuvio 4. Differentiaalikoodaus

$$\{2000, 1999, 2000, 2001, 2003, 2002\} \rightarrow \{2000, -1, +0, +1, +3, +2\}$$

Toinen esiteltävä numeerinen menetelmä on kvantisointi, joka on häviöllinen menetelmä. Pun (2006) määritelmän mukaan siinä reaaliluvut pyöristetään esimerkiksi kokonaisluvuiksi, joilla luvut saadaan viemään vähemmän tilaa ja mahdollisten arvojen joukko saadaan pienemmäksi. Kvantisointi on skalaarikvantisaatio, jos jokainen näyte kvantisoidaan erikseen ja vektorikvantisaatio, jos muunnettavia näytteitä on vähintään kaksi (Pu 2006, s. 176-178). Kvantisoinnin teho ei tule esiin kovin helpolla, jos desimaaliluvut ja kokonaisluvut tallennetaan samanmittaisina binäärilukuina. Kvantisoinnin jälkeen pyöristetyille luvuille voidaan tehdä jokin vaihtoehtoinen koodaustapa viemään vähemmän tilaa.

Kuvio 5. Kvantisointi

$$\{0.824, 1.128, 2.287, 2.298, 1.937\} \rightarrow \{0.8, 1.1, 2.3, 2.3, 1.9\}$$

4 DEFLATE

DEFLATE on häviötön ja yleiskäyttöinen pakkausmenetelmä tai oikeastaan häviöttömän pakkauksen standardoitu dataformaatti ja sen on suunnitellut Phil Katz (Deutsch 1996). DEFLATE-menetelmä yhdistää hakemistollisen ja tilastollisen menetelmän yhdeksi kokonaisuudeksi. Deutschin (1996) määritelmän mukaan DEFLATE-menetelmän on tarkoitus olla

- alustasta riippumaton, jotta sitä voidaan käyttää eri laitteiden väliseen tiedonsiirtoon
- käytettävissä pitkän datavirran pakkaamiseen pienellä määrällä välimuistia
- verrattavissa parhaisiin yleiskäyttöisiin pakkausmenetelmiin
- vapaasti käytettävissä eli ilman patenttien suojausta
- yhteensopiva yleisesti käytössä olevan gzip-formaatin kanssa.

Deutschin (1996) määritelmän mukaisesti DEFLATE-menetelmän pakkausalgoritmi sisältää Huffmanin koodauksen ja Lempel–Ziv-algoritmin (LZ77). Huffmanin koodaus on yksi yleisimmistä tilastollisista pakkausmenetelmistä, jota käytetään symbolisen datan koodaamiseen vaihtelevan pituisella koodauksella. LZ77 on puolestaan dynaaminen hakemistomenetelmä, jolla pakataan laajempia symbolijonoja pienempään muotoon. Deutschin (1996) julkaisussa ei käydä algoritmien toimintaa yksityiskohtaisesti läpi, vaan sen tarkoitus on kertoa kehittäjille dataformaatin muoto, jotta he osaavat kehittää omat toteutuksensa pakkaus- ja/tai purkualgoritmeista.

Deutschin (1996) määritelmän mukaisessa tiedostomuodossa data on jaoteltu lohkoihin *(eng. block)*, jotka ovat itsenäisesti ja toisista riippumatta pakattu DEFLATE-algoritmilla. Lohko on kaksiosainen, jonka ensimmäisessä osassa on tieto miten lohkon data on pakattu ja toisessa osassa pakkausalgoritmilla pakattu data (Deutsch 1996). Tiedostomuotoa ei kuitenkaan tämän yksityiskohtaisemmin käydä tässä tutkimuksessa, vaan seuraavaksi käymme läpi DEFLATE-algoritmin toimintaa sen sisältämän kahden algoritmin kautta.

4.1 Huffmanin koodaus

David A. Huffman (1952) esitteli tutkielmassaan perusteita, joilla koodaus voidaan optimoida käyttämään keskimääräisesti vähiten bittejä symbolia kohden. Huffmanin esittämät vaatimukset optimaaliselle koodaukselle ovat:

- kaksi eri symbolia tulee koodata uniikilla tavalla
- symbolin koodi on erotettavista toisista koodeista ilman ylimääräisiä indikaattoreita
- useammin esiintyvä merkki koodataan aina lyhyemmällä tai yhtä pitkällä koodilla, kuin harvemmin esiintyvä
- koodauksessa kahden vähiten esiintyvän symbolin koodit ovat yhtä pitkiä ja identtisiä viimeistä merkkiä lukuun ottamatta
- jokainen mahdollinen koodi, jonka pituus on pienempi kuin pisin koodi, täytyy olla käytössä joko symbolin koodina tai pisimmän koodin alkuosana.

Edellä mainittujen periaatteiden pohjalta kehitettiin koodaustapa, jota kutsutaan yleisesti Huffmanin koodaukseksi, ja jota käytetään myös DEFLATE-menetelmässä. Seuraavaksi käydään esimerkin avulla läpi, miten Huffmanin koodaus toteutetaan.

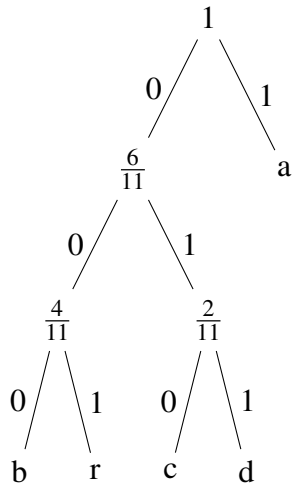
Taulukko 1. Huffmanin koodaus: symbolit, esiintymismäärät ja todennäköisyydet

Symboli	Lukumäärä	Todennäköisyys
”a”	5	0.45
”b”	2	0.18
”r”	2	0.18
”c”	1	0.09
”d”	1	0.09

Tarkastellaan merkkijonoa ”abracadabra”. Lasketaan ensin, kuinka monta kertaa merkit toistuvat ja niiden todennäköisyydet, sekä järjestetään ne todennäköisyyksien mukaan taulukkoon 1. Merkkien ja niiden todennäköisyyksien pohjalta voidaan tehdä Huffmanin puu, jonka tarkoituksena on tehdä jokaiselle merkille uniikki koodaus Huffmanin (1952) esittelemien vaatimusten mukaisesti. Huffmanin puun rakenne tehdään toistamalla seuraavia operaatioita, jotka soveltavat Pun (2006, s. 70-71) ohjeita:

1. Poistetaan joukosta kaksi pienimmällä todennäköisyydellä (p_1, p_2) olevaa alkiota (s_1, s_2). Puu on valmis, kun jäljellä on enää yksi alkio, jonka todennäköisyys on yksi.
2. Yhdistetään s_1 ja s_2 juurisolmuun, asettaen suuremman todennäköisyyden omaava s vasemmaksi lapsisolmuksi ja toinen oikeaksi. Lasketaan puurakenteelle uusi yhteinen todennäköisyys $p_1 + p_2$.
3. Asetetaan saatu puurakenne takaisin joukkoon. Puun paikka määräytyy uuden todennäköisyyden mukaan ja se asetetaan yhtä todennäköisten alkioiden todennäköisimpään päähän.

Kuvio 6. Huffmanin puu



Taulukko 2. Huffmanin koodaus: valmis aakkosto

Symboli	Koodi
”a”	1
”b”	000
”r”	001
”c”	010
”d”	011

Algoritmin tuloksena saadaan koodausta esittävä Huffmanin puu (Kuvio 6), kun kuvataan vasenta siirtymää 0 ja oikeaa siirtymää 1 (Pu 2006, s. 71). Käymällä kaikkien lehtisolmujen reitit läpi saadaan luotua uusi aakkosto eli itse koodaus (Taulukko 2). Uuden aakkoston

mukaisesti merkkijono ”abracadabra” koodautuu muotoon ”100000110101011100000011”. Huffmanin koodauksella merkkijono saatiin koodattua 23 bitillä eli keskimäärin noin 2.1 bitillä merkkiä kohden. Verrattuna ASCII-koodaukseen saatu Huffmanin koodaus vie 3.8 kertaa vähemmän tilaa.

4.2 LZ77

Hakemisto-osa DEFLATE-algoritmia perustuu Abraham Lempelin ja Jacob Zivin (1977) kehittämän dynaamisen Lempel–Ziv-hakemistomenetelmän (LZ77) pohjalle. DEFLATE-algoritmin versio menetelmästä ei kuitenkaan ole ainut LZ77-algoritmin pohjalta syntynyt menetelmä, sillä on olemassa kokonainen Lempel–ziv-sukuisten algoritmien joukko (Pu 2006, s. 129). Monet Lemel–ziv-sukuisista algoritmeista ovat patentoituja, joten Deutsch (1996) kehottaa käyttämään julkaisussaan esitettyä versiota algoritmista.

Algoritmissa Lempelin ja Zivin (1977) määritelmän mukaisesti etsitään toistuvia jaksoja kaksiosaisen puskurin avulla. Ensimmäinen puskuri on muistipuskuri, joka pitää muistissa rajallisen määrän läpikäytyä dataa, ja toinen puskuri määrittää etsittävän jakson, jota kutsutaan tässä tutkielmassa etsintäpuskuriksi. Etsintäpuskurilla on minimi ja maksimikoko. Minimikoko on vähintään yhtä suuri, kuin indeksin tallentamiseen tarvittava määrä dataa, jotta indeksin tallentaminen ei vie alkuperäistä jaksoa enemmän tilaa. Yksinkertaisuudessaan algoritmi etsii etsintäpuskurin määrittämää jaksoa muistipuskurista. Jos jakso löytyy, korvataan etsintäpuskurin jakso indeksillä, joka osoittaa muistipuskurin identtiseen osaan. Deutschin (1996) mukaan DEFLATE-algoritmissa etsintä tapahtuu yksinkertaistettuna seuraavalla tavalla:

1. Asetetaan etsintäpuskurin koko minimiin ja tehdään sen osoittamalle jaksolle tiiviste hajautusfunktiolla, jolla myös muistipuskuri tiivistetään.
2. Etsitään yhtäläistä jaksoa muistipuskurin tiivisteestä. Jos yhtäläistä jaksoa ei löytynyt, edetään datassa eteenpäin.
3. Jos jakso löytyi, kasvatetaan etsintäpuskurin kokoa yhdellä. Tehdään uusi tiiviste ja verrataan sitä muistipuskuriin. Tätä toistetaan, kunnes löytyy ensimmäinen eroava merkki, etsintäpuskuri on maksimikokoinen tai data loppuu.

4. Etsintäpuskurin jakso korvataan muistipuskurin identtiseen jaksoon osoittavalla indeksillä. Sen esitysmuoto on $\langle$o,l$\rangle$, jossa o on siirtymä muistipuskurin jaksoon ja l on jakson pituus.

Toistetaan edellä mainittu algoritmi Huffmanin koodaus kappaleessa (4.1) tuttuun ”abracadabra” esimerkkiin ilman hajautusfunktiota. Asetetaan muistipuskurin maksimikooksi seitsemän merkkiä ja korvattavan jakson minimipituudeksi kolme merkkiä. Edetään merkkijonossa eteenpäin, kunnes ensimmäinen minimimittainen toistuva jakso löytyy.

$$\overbrace{\underline{\text{abr}}\text{acad}}^{\text{Muistipuskuri}} \quad \underbrace{\text{abr}}_{\text{Etsintäpuskuri}} \quad \text{a}$$

Kasvatetaan etsintäpuskurin kokoa, kunnes jokin lopetusehto tulee vastaan. Tässä tapauksessa pakattava data loppuu.

$$\overbrace{\underline{\text{abra}}\text{cad}}\underbrace{\text{abra}}$$

Seuraavaksi korvataan jälkimmäinen jakso indeksillä, joka osoittaa muistipuskurin identtiseen jaksoon ja täten saadaan LZ77-algoritmilla pakattu data.

$$\text{abracad}\langle 7,4\rangle$$

4.3 Arviointi

Yhdistämällä Huffmanin koodauksesta saatu aakkosto (Taulukko2) ja Lempel–Ziv-algoritmi (Kappale 4.2) saadaan DEFLATE-formaattia mukaileva pakkaustulos.

$$\text{abracadabra} \rightarrow 100000110101011\langle 16,8\rangle$$

Kuten edellisessä esimerkissä näkyy, toimii DEFLATE-menetelmä hyvin datalle, joka sisältää vain vähän erilaisia merkkejä, sekä paljon toistuvia jaksoja. Esimerkissä LZ77-algoritmin

hyöty ei tule hyvin esille, mutta laajemmassa aineistossa on se todella tehokas menetelmä. Vertailuja eri menetelmien väliltä on olemassa melko paljon, ja seuraavaksi käydään muutamien testien tuloksia läpi ja arvioidaan menetelmän toimivuutta niiden kautta.

Bansalin, Guptan ja Khandujan (2017) tutkimuksessa vertailtiin eri pakkausalgoritmeja keskenään (DEFLATE, LZMA, BZIP2, PPMd, PPMONSTER) erilaisten tiedostotyyppien pakkaamisessa. Tuloksista näki, että DEFLATE pakkasi huonoimmalla suhteella, mutta oli selkeästi joukon nopein algoritmi sekä pakkauksessa että purkamisessa. Kuitenkin erot pakkaussuhteessa olivat melko pieniä ja tärkeämmäksi perusteeksi osoittautui algoritmin nopeus, jossa DEFLATE oli todella paljon muiden edellä. Muissa menetelmissä tehtiin kompromisseja joko pakkaus- tai purkunopeudessa, joten DEFLATE oli selkeästi testien tasapainoisin menetelmä.

Alakujalan, Kliuchnikovin, Szabadkan ja Vandevennen (2015) tutkimuksessa vertailtiin eri yleiskäyttöisien pakkausalgoritmien toimintaa. *Zlib*-kirjaston DEFLATE-algoritmia testattiin eri laatuasetuksilla, joista toinen pyrkii olemaan mahdollisimman nopea ja toinen pyrkii parempaan pakkaussuhteen. Testeissä olleista menetelmistä vain *Brotli,* tutkimuksen tekijöiden suunnittelema menetelmä, oli verrattavissa DEFLATE-menetelmään. Taulukossa 3 on testattujen dataryhmien keskiarvoiset tulokset DEFLATE-algoritmilta ja brotli-algoritmilta kahdella eri laatuasetuksella.

Taulukko 3. DEFLATE ja brotli suorituskykymittaukset (Alakuijala ym. 2015)

Algoritmi:laatuasetus	Pakkaussuhde	Pakkausnopeus [MB/s]	Purkunopeus [MB/s]
DEFLATE:1	3.3	103.7	323.2
brotli:1	3.8	107,3	357.0
DEFLATE:9	3.9	22.2	349.6
brotli:9	4.5	17.6	380.9

Taulukossa Alakujalan ym. (2015) tutkimuksen kolmen eri mittauksen keskiarvot.

Taulukon 3 tulokset osoittavat brotli-menetelmän tuottavan parempia tuloksia kuin DEFLATE. Tulokset eivät kuitenkaan anna aivan koko kuvaa, sillä Alakujalan ym. (2015) testeissä käytetyt tiedostot olivat pääosin html-dokumentteja, sekä erilaisia tekstejä. Brotli menetelmä on muuten verrannollinen DEFLATE-menetelmään, mutta siihen on lisätty staattinen hake-

misto eri kielten sanoille ja tavuille, sekä html ja css-dokumenteille (Alakujala ym. 2015). Valitut testit suosivat selkeästi brotlia, mutta selkeää eroa menetelmien välille ei kuitenkaan saatu. Voidaan kuitenkin sanoa brotlin sopivan tietyille tiedostoille paremmin, kuin DEFLATE, mutta yleiskäyttöisyyteen ja paremmuuteen ei näiden tulosten pohjalta voida ottaa kantaa.

Googlen kehittämän brotlin lisäksi toinen suuri toimija, Facebook, on kehittänyt oman pakkausmenetelmänsä *Zstandardin*. Colletin ja Turnerin (2016) artikkelissa kerrotaan sen olevan suunniteltu hyödyntämään nykyaikaista laitteistoa paremman ja nopean pakattavuuden saamiseksi. Kun DEFLATE-menetelmä on suunniteltu aiemmalla vuosituhannella ja sen aikaisille laitteistoille, on Zstandard julkaistu noin kaksikymmentä vuotta myöhemmin (Collet ja Turner 2016). DEFLATE-menetelmän ja Zstandardin suorituskykymittausten tulokset Colletin ja Turnerin (2016) tutkimuksesta ovat taulukossa 4.

Taulukko 4. DEFLATE ja Zstandard suorituskykymittaukset (Collet ja Turner 2016)

Algoritmi	Pakkaussuhde	Pakkausnopeus [MB/s]	Purkunopeus [MB/s]
DEFLATE	3.11	23.21	281.52
Zstandard	3.14	136.18	536.36

Tuloksista (Taulukko 4) nähdään, että vastaavalla pakkaussuhteella Zstandard pakkaa moninkertaisella nopeudella ja purkaa lähes kaksinkertaisella nopeudella. Tulosten perusteella voisi kuvitella Zstandardin olevan ylivertainen DEFLATE-menetelmään, mutta on sillä muutamia heikkouksiakin. Zstandardin muistinkäyttöä ei ole rajoitettu kuin laitteiston fyysisillä rajoitteilla, kun puolestaan *Zlib*-kirjaston puskuri on rajoitettu 32 kilotavuun (Collet ja Turner 2016). Tämän takia Zstandard ei sovellu sellaisenaan reaaliaikaiseen tiedonsiirtoon, kun taas DEFLATE-menetelmällä voidaan lähettää aina puskurin ulkopuolinen jo käsitelty data eteenpäin. Muistinkäytön lisäksi *Zstandard* on oma formaattinsa, joka ei ole yhteensopiva minkään aikaisemman formaatin kanssa. Tämä on selkeä hidaste menetelmän leviämiselle, sillä DEFLATE toimii riittävän hyvin ja on lähes aina tuettu.

5 Yhteenveto

Tässä tutkielmassa tutustuttiin tiedonpakkauksen perusteisiin, pakkausmenetelmien kehittämiseen ja arviointiin, sekä muutamiin yleisimpiin menetelmätyyppeihin. Tutkielman lopuksi esiteltiin yleiskäyttöinen ja häviötön pakkausmenetelmä DEFLATE, joka toimii esimerkkinä hyvästä ja tasapainoisesta tiedonpakkauksen toteutuksesta.

Häviötöntä pakkausta käytetään, kun tiedosta ei voida menettää mitään ilman merkityksen muuttumista. Se toteutetaan yleisimmin tilastollisilla ja/tai hakemistomenetelmillä, ja joiden pakkaamismallit toteutetaan analysoimalla pakattavaa dataa. Häviöllisiä menetelmiä käytetään puolestaan silloin, kun tieto sisältää jonkinlaista turhaa dataa, jota voidaan muokata tai poistaa siten, että alkuperäisen tiedon tarkoitus säilyy. Sen toteutustavat riippuvat täysin pakattavan datan ja tiedon ominaisuuksista ja sen pakkaamismallit toteutetaan datan analysoinnin lisäksi myös itse tietoa analysoimalla. Tiedon analysoinnissa pyritään selvittämään osia, jotka voidaan poistaa ilman ymmärryksen menettämistä. Häviöllisiä menetelmiä käytetään useimmiten multimedian pakkaamiseen ja niiden pakkaamiseen on olemassa useimmiten hyviä menetelmiä jo valmiina, kuten kuville PNG- ja JPEG-formaatti.

Useimmiten menetelmiä arvioidaan pakkaussuhteen, algoritmin nopeuden tai muistinkäytön kautta. Menetelmien vertailu ja arviointi voi olla hankalaa, sillä vertailtavia ominaisuuksia voi olla paljon, ja käyttötapauksesta riippuen arvioinnissa painotetaan eri osa-alueita. Tämän takia ei voida yksikäsitteisesti sanoa jonkin menetelmän olevan paras, vaan menetelmiä vertaillaan jonkin ominaisuuden tai käyttötapauksen mukaan. Voidaan sanoa, että paras menetelmä on juuri sillä hetkellä pakattavalle datalle ja käyttötarkoitukselle optimoitu menetelmä.

Yleiskäyttöiset pakkausmenetelmät ovat kehittyneet viimeisen kymmenen vuoden aikana jonkin verran eteenpäin. DEFLATE-menetelmään verrattuna uudet menetelmät saattavat olla parempia jollakin osa-alueella, mutta kokonaisvaltaista paremmuutta ei voida vielä todeta. Zstandard ja brotli ovat tällä hetkellä lupaavimmat menetelmät, joita kehitetään jatkuvasti paremmiksi isojen yritysten tukemina. Marginaalit, joissa erot pysyvät, ovat kuitenkin melko pieniä ja niiden takia DEFLATE-formaattia toteuttavat menetelmät ovat suosion ja tuen takia paremmassa asemassa.

Lähteet

Alakuijala, Jyrki, Evgenii Kliuchnikov, Zoltan Szabadka ja Lode Vandevenne. 2015. “Comparison of brotli, deflate, zopfli, lzma, lzham and bzip2 compression algorithms”. *Google Inc.* https://fossies.org/linux/brotli/docs/brotli-comparison-study-2015-09-22.pdf.

Bansal, Aman, Apoorv Gupta ja Vidhi Khanduja. 2017. “Modern lossless compression techniques: Review, comparison and analysis”: 1–8. doi:10.1109/ICECCT.2017.8117850.

Brotli. https://github.com/google/brotli/releases/tag/v0.2.0. Viitattu: 22.3.2019.

Collet, Yann, ja Chip Turner. 2016. “Smaller and faster data compression with Zstandard”. Viitattu: 8.4.2019, *Core Data.* https://code.fb.com/core-data/smaller-and-faster-data-compression-with-zstandard/.

Deutsch, Peter. 1996. *DEFLATE Compressed Data Format Specification version 1.3.* RFC 1951. Network Working Group, toukokuu.

Huffman, David A. 1952. “A Method for the Construction of Minimum-Redundancy Codes”. *Proceedings of the IRE* 40, numero 9 (syyskuu): 1098–1101. ISSN: 0096-8390. doi:10.1109/JRPROC.1952.273898.

Kotimaisten kielten keskus ja Kielikone Oy. 2018. https://www.kielitoimistonsanakirja.fi/koodata. Viitattu: 6.3.2019.

Lempel, Abraham, ja Jacob Ziv. 1977. “A universal algorithm for sequential data compression”. *IEEE Transactions on Information Theory* 23, numero 3 (toukokuu): 337–343. ISSN: 0018-9448. doi:10.1109/TIT.1977.1055714.

Pu, Ida Mengyi. 2006. *Fundamental Data Compression.* Butterworth-Heinemann. ISBN: 9780750663106.

Sayood, Khalid. 2006. *Introduction to Data Compression.* Nide 3rd ed. Morgan Kaufmann Series in Multimedia Information and Systems. Morgan Kaufmann. ISBN: 9780126208627.

Zlib. `https://www.zlib.net/`. Viitattu: 22.3.2019.

Zstandard. `https://github.com/facebook/zstd`. Viitattu: 8.4.2019.