Funktionaalisuus tietorakenteissa

Ville Pirttimäki
vipirtti@jyu.fi

20.5.2008

Tiivistelmä

Tämä raportti tarkastelee funktionaalisten ohjelmointikielien vaikutusta tietorakenteisiin. Lähinnä tarkastellaan persistanssin ja laiskan laskennan vaikutusta tietorakenteen toteutukseen sekä jälkimmäisen ominaisuuden vaikutusta tietorakenteiden operaatioiden vaativuuteen niin tasatun vaativuuden kuin pahimman tapauksen vaativuuden osalta.

1 Pysyvyys

Englanningkieliseltä nimeltä *persistance*. Viittaa tietorakenteiden tapauksessa funktioparadigmassa yleiseen käytäntöön, jossa käsiteltävä data ei tuhoudu mahdollisissa muutoksissa, vaan tietorakenteen vanha versio säilyy myös uuden version luonnin jälkeen. [1, s. 7]

Tämä ominaisuus on oleellinen osa funktio-ohjelmointia ja esiintyy esimerkiksi Haskellissa sisäänrakennettuna ominaisuutena. Muistin käytön kannalta tämä ominaisuus saattaa olla hyvinkin tuhlaava, etenkin huonosti huomioon otettuna.

Otetaan esimerkkinä yhteen suuntaan linkitetty lista. Jotta linkitetyn listan kokoaminen kahdesta osasta, alkioista tai valmiista linkitetyistä listoista, toteuttaisi pysyvyyden, joudutaan listan ensinmäinen osa kopioimaan kokonaisuudessaan. Vain näin kopioidun listan viimeinen alkio pystyy viittaamaan listan uuteen jatkoon. Loppuosaa ei tarvitse kuitenkaan kopioida, vaan alku osa voi viitata suoraan vanhaan kopioon loppuosasta, säästäen näin niin muistia kuin kopiointityötä.

Hyvä esimerkki siitä, miksi pysyvyystä tulisi ymmärtää on listan luominen alkio kerrallaan. Mikäli lista luodaan lisäämällä alkio aina listan loppuun, joudutaan jokainen siihen mennessä lisätty alkio kopiomaan. Lopulta

muistiin on tallennettuna $1 + 2 + 3 + ... + n = n(n + 1)/2$ alkiota. Jos lista luotaisiin esimerkiksi 10 000 alkiosta, tallentuisi muistiin lopuksi noin 50 miljoonaa alkiota. Lisäksi tämä olisi ajallisesti varsin tehotonta, mutta tämä pitää paikkaansa myös epäpysyvillä linkitetyillä listoilla.

Mikäli sama operaatio toteutettaisiin listan alkuun lisäämällä, ei pysyvyys enää aiheuttaisi vastaavaa ongelmaa, vaan jokainen alkio tallentuisi vain kerran. Kahdesti, mikäli lista halutaan kääntää, jolloin lopullinen lista vastaa täysin edellä kuvatun tapauksen tulosta.

Tasapainottamattomassa puussarakenteessa pysyvyys voidaan myös saavuttaa hyvin tehokkaalla tavalla kopioimalla vain lisättyyn tai poistettuun solmuun vievän hakupolun alkiot. Sama ilmiö on todistettavissa myös joissakin tasapainotetuissa puissa, esimerkiksi punamustissa puissa.

2 Laiska laskenta

Toinen varsin yleinen, joskaan ei välttämätön, funktiokielen ominaisuus on laiska laskenta. Kielestä ja sen toteutuksesta riippuu, käytetäänkö laiskaa laskentaa oletuksenomaisesti vaiko vain erikseen käskien. [1, s. 31] Esimerkiksi Haskell-kielen määritelmä ei sisällä laiskaa laskentaa, mutta monet kielet tämän kuitenkin toteuttavat. Toisaalta esimerkiksi Standard ML -kielessä laiska lakenta saadaan aikaan erillisellä notaatiolla.

Käsitteeseen 'laiska laskenta' lasketaan tässä tapauksessa kuuluvaksi kaksi erillistä ominaisuutta. Ensinmäinen näistä on memoisaatio (*memoization*), joka tallettaa funktion tuloksen tietyllä parametrilla tai parametreilla muistiin niin, että jo laskettuja tuloksia ei tarvitse laskea uudestaan, vaan ne voidaan noutaa suoraan muistista. Toisena ominaisuutena on varsinaisen laskutoimituksen lykkäys (*suspension*) siihen pisteeseen asti, kunnes varsinaista tulosta oikeasti käytetään. Tämä mahdollistaa laskutimitusten myöhäistämisen lisäksi niiden täydellisen pois jättämise, mikä suorituksen aikana varsinaista tulosta ei tarvitakkaan. [1, s. 31]

2.1 Tasattu vaativuus

Tasattua vaativuutta voidaan verrata keskimääräiseen vaativuuteen, joskin sillä erotukella että keskimääräinen vaativuus tarkastelee algoritmin yhden operaation keskimääräistä vaativuutta usean mahdollisen syötteen suhteen, kun taas tasattu vaativuus tarkastelee algoritmin keskimääräistä vaativuutta usean operaation sarjassa, jossa syöte voi hyvinkin olla pahin mahdollinen. Yleensä tämä vaatii sen, että operaatio on yleensä nopea toteuttaa, mutta tietyin väliajoin vaaditaan jonkin raskaan operaation, kuten puurakenteen

tasapainotuksen suoritus, jonka ansiosta myöhemmät operaatiot ovat taas kevyempiä. [2, kpl 1.4, 1.5]

Funktionaalisten tietorakenteiden kannalta tasatulta vaativuudeltaan kevyiden tietorakenteiden operaatiot vaativat yleensä destruktiivista tietorakenteen muokkausta ollakseen tarpeeksi tehokkaita. Tämä ei kuitenkaan onnistu pysyvyyttä noudattavissa funktionaalisissa kielissä. Tavalliset keinot näiden tietorakenteiden muokkaamiseksi pysyvyyttä noudattaviksi korottaisivat tasatun vaativuuden pahimman vaativuuden tasolle. [1, kpl 5.6]

2.2 Tasattu vaativuus ja laiska laskenta

Laiska laskenta, etenkin memoisaatio näyttelee merkittävää osaa tasattua aikavaativuutta alennettaessa. Operaatioiden lykkäyksen merkitys on tässä tapauksessa pienempi. Selvä syy tälle on se, että vaikka lykkäys saattaa karsia suoritusajasta joitakin tarpeettomia operaatioita, jokainen tarpeellinen operaatio tullaak kuitenkin suorittamaan jossakin vaiheessa operaatioiden sarjassa. Sen lykkäämin vain vaihtaa tämän suoritusajankohdan paikkaa, joka taas ei vaikuta tasattuun vaativuuteen lainkaan. [1, kpl 6]

Memoisaatio taas vaikuttaa tasattuun vaativuuteen, kun samoja aikaavieviä operaatioita tai samoja lykkäyksiä käyttäviä operaatioita suoritetaan useampia. Vain tällöin memoisoitu data voidaan ottaa käyttöön erikseen laskettavan datan sijaan ja niin operaatioiden suoritusaika ja niiden kautta tasattu vaativuus saadaan laskemaan.

3 Pahimman tapauksen optimointi

Edellisessä kappaleessa keskityttiin tasatun vaativuuden tehokkaaseen toteutukseen funktionaalisilla kielillä. Keskittykäämme nyt vastakkaiseen tapaukseen, pahimman tapauksen vaativuuden optimointiin fuktionaalisilla kielillä. On olemassa tilanteita, joissa on mielekkäämpää keskittyä *kaikkien* operaatioden yhtenäisen aikavaativuuden varmistamiseen keskimääräisen vaativuuden optimoimisen sijaan. Esimerkkejä tällaisista tapauksista ovat esimerkiksi reaaliaka järjestelmät tai interaktiiviset sovellukset, joissa yksittäiset pitkät operaatiot ovat epätoivottuja, vaikka kaikki muut operaatiot olisivatkin jopa vakioaikaisia.

Kuten tasatun vaativuuden kanssa, myös pahimman tapauksen optimoinnissa osoittautuu yksi funktionaalisten kielien yleisistä ominaisuuksista ongelmalliseksi. Tällä kertaa kyseessä on laiskan laskennan laskutoimenpiteitä lykkäävä ominaisuus. Jotta pahimman tapauksen aikavaativuus saataisiin mahdollisimman alas, on muiden operaatioiden suoritettava toimenpiteitä,

jotka alentavat jollain tavalla tulevien operaatioiden aikavaativuutta. Mikäli valittu kieli suorittaa laskutoimitukset oletusarvoisesti laiskasti, eikä asiaa oteta huomioon, voivat nämä operaatiot lykkääntyä, kunnes niiden tulosta tarvitaan. Näin kaikki ennakkoon suoritettujen toimepiteiden varsinainen laskenta kertyy yhden operaation aikaiseksi, mikä on juuri halutun vastainen tilanne.

Toisaalta, mikäli funktionaalinen kieli tai sen käytetty toteutus ei käytä laiskaa laskentaa, tai vaihtoehtoisesti lasikan laskennan käyttöä voidaan hallita, pystytään pahimman tapauksen aikavaativuutta hallitsemaan paremmin.

3.1 Ajoitus

Chris Okasaki toteuttaa kirjassaan [1, kpl 7.1] ja sitä edeltäneissä tutkielmissaan pahimman tapauksen aikavaativuuden optimoinnin käyttämällä tekniikkaa, jota hän kutsuu ajastukseksi (*scheduling*). Tämä tekniikka on suunniteltu tasatulle vaativuudelle optimoitujen tietorakenteiden muuttamiseksi pahimman tapauksen optimoiviksi tietorakenteiksi käyttäen hyväksi laiskan laskennan ominaisuuksia. Lisäksi tekniikan sielittämiseksi on määritelty lykätyn operaation oleellinen hinta (*intrinsic cost*), joka on operaation aikavaativuus, olettaen että kaikki sen sisältämät lykätyt operaatiot on suoritettu jossain vaiheessa etukäteen ja ovat näin memoisoituja ja saatavilla vakioajassa.

Tietorakenteen ajoituksen toteuttamiseksi on ensin muutettava kaikki operaatiot muotoon, jossa niiden oleellinen hinta on samaa tai alempaa vaativuusluokkaa kuin haluttu pahimman tapauksen vaativuus. Seuraava askel on taas varmistaa se, että kun jokin lykätty operaatio viimein lasketaan, on kaikki sen vaatimat lykätyt operaatiot jo laskettu ja memoisoitu. Näin jokainen operaatio on halutun aikavaativuuden mukainen ja tavoite pahimman tapauksen optimoinniksi saavutettu.

Ajoituksen pohjimmainen idea piilee tavassa, jolla jälkimmäinen edellä mainituista tavoitteista saavutetaan. Tietorakenteen lisäksi operaatiolle annetaan aikataulu, joka ainakin periaatteessa osoittaa jokaiseen suorittamattomaan lykkäykseen. Samalla kun operaatio suoritetaan, suoritetaan aikataulusta sopiva, kiinteä määrä lykättyjä operaatioita, jotta näiden tulokset olisivat valmiita kun kyseisiä lykättyjä operaatioita tarvitsevat operaatiot suoritetaan. Luonnollisesti, jotta aikataulun suorituksesta ei muodostuisi uutta pullonkaulaa, on se suunniteltava niin, että operaatioiden suoritus voidaan aloittaa niistä lykkäyksistä, joiden vaatimat lykkäykset on jo suoritettu.

4 Yhteenveto

Tässä raportissa olen pyrkinyt erittelemään niin ongelmia kuin ratkaisuja, joita funktionaalisten ohjelmointikieline pysyvyys- ja laiska laskenta ominaisuudet luovat. Kaikista ongelmista paistaa selvästi läpi tietty tasapaino rakaisun suhteen, jossa jonkin tietyn ominaisuuden luomat ongelmat voidaan kiertää jotakin toista ominaisuutta käyttäen, kuten esimerkiksi pysyvyyden luoma ongelma tasatun vaativuuden suhteen voitiin ratkaista laiskan laskennan memoisaation ansiosta.

Tästä raportis puuttu kuitenkin vielä useita tekniikoita, joita voidaan käyttää funktionaalisten tietorakenteiden muodostukseen, esimerkiksi laiska tietorakenteen uudelleenmuodostus [1, kpl 8], joka käyttää hyväkseen laiskaa laskentaa tietorakenteen tasapainottamisessa. Samoin jäivät puuttumaan useat esimerkkitoteutukset, jotka käyttävät hyäkseen raportissa käsiteltyjä tekniikoita tehokkaan tietorakenteen luomiseen ja ylläpitämiseen.

Funktionaalisia kieliä käyttämällä voidaan rakentaa tietorakenteita hyvin moniin käyttötarkoituksiin, olettaen että tunnetaan niin kielen ominaisuuksien tuomat rajoitukset, kuten persistanssin vaatimukset tietorakenteen muodon suhteen, kuin sen tarjoamat mahdollisuudetkin, kuten memoisaation tuoma teho toistuvaan työhön. Lisäksi yleensä varsin intuitiivisesti abstrahoidut funktionaaliset kielet mahdollistavat monen imperatiivisesti vaikeaselkoisesti toteutettavan tietorakenteen toteuttamisen varsin lyhyesti ja ytimekkästi.

Lähteet

[1] Chris Okasaki, Cambridge University Press, *Purely Functional Data Structures*, 1998

[2] Pekka Orponen & Jarmo Ernvall, Jyväskylän Yliopisto, *Algoritmitekniikka*, 2004