Johdanto

Tietotekniikka on kehittynyt viime vuosikymmenten aikana nopeata vauhtia. Tämä on näkynyt niin tietokoneiden tehoissa kuin myös hinnoissa. Myös tietokoneiden keskusmuistit ovat kasvaneet ja ovat nykyään halvoissakin tietokoneissa on gigatavuja keskusmuistia. Tämän johdosta tietokannat, jotka eivät aikaisemmin mahtuneet kuin kovalevyille, mahtuvat nyt kokonaan tai ainakin suurelta osin keskusmuistiin. Useat yritykset ja tutkimuslaitokset ovatkin alkaneet kehittämään keskusmuistitietokanta-järjestelmiä [HXS07]. Näiden järjestelmien on on tarkoitus tarjota parempaa suoritustehoa kuin perinteiset tietokantajärjestelmät.

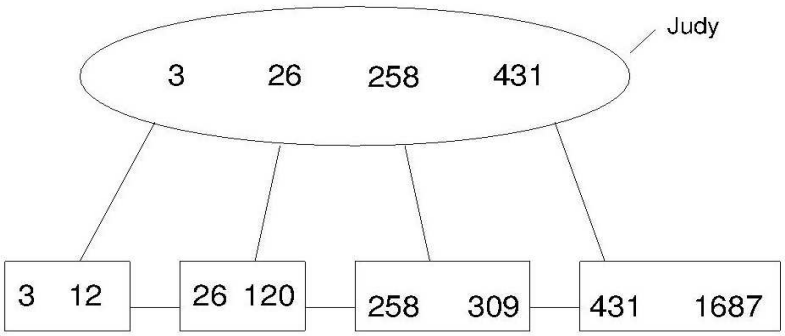
Aivan kuten perinteisissäkin tietokantajärjestelmissä keskusmuisti tietokantajärjes-telmissä tiedolle luotavat hakemistot ovat tärkeitä järjestelmän tehokkuuden kannalta. Muun muassa yhden arvon kyselyt sekä osavälikyselyt hyötyät niistä. Alemmalla tasolla tarkasteltuna nämä operaatiot koostuvat tietyn avaimen hakemisesta tai tietyn hakemiston osan lukemisesta. Molemman operaatioita, jotka hyödyntävät hakemistoja. Hakemistojen luonnissa käytettävät tietorakenteet ovatkin olleet kehityksen kohteena. Rakenteina on käytetty T-puuta sekä B+-puun eri variaatioita. Näistä T-puu on ollut käytössä useissa keskusmuistitietokantajärjestelmissä. On kuitenkin väitetty, ettei T-puu käyttäisi tehokkaasti hyväkseen välimuistia [HXS07].

Tietokoneiden kehittyessä prosessoreiden nopeus on kasvanut nopeammin kuin keskusmuistin nopeus [HXS09] [CGM01]. Prosessoreiden nopeus on kasvanut noin 60% joka vuosi kun keskusmuisti on nopeutunut vain noin 10%. Tämä on ennestään lisännyt välimuistin tehokkaan käyttämisen tärkeyttä. Perinteisissä tietokantajärjestelmissä jopa puolet ajasta voi mennä hukkaan välimuistihutien takia. Keskusmuistitietokantajärjestelmissä tämä aika on vieläkin enemmän.

Vaikka B+-puu käyttävätkin välimuistia tehokkaammin hyväksi kuin T-puut, ne ovat siitä huolimatta varsin huonoja välimuistin käytössä. B+-puusta onkin kehitetty variantteja, jotka ottavat välimuistin paremmin huomioon. Näitä variantteja ovat CSB+puu sekä pB+-puu. Esimerkiksi hakuoperaatiossa välimuistihutien määrä on suhteessa puun korkeuteen, joten CBS+-puussa on pyritty pienentämään puun korkeutta muun muassa lisäämällä solmussa olevien avainten määrää.

Yhdeksi vaihtoehdoksi keskusmuistitietokantajärjestelmien hakemistojen tietorakenteeksi on esitetty J+puuta [HXS07] [HXS09]. Sen on tarkoitus olla tarpeeksi tehokas keskusmuistitietokantajärjestelmien tarkoituksiin ja käyttää välimuistia

tehokkaasti hyväkseen. Puu perustuu Judy-nimiseen digitaaliseen puuhun, joka muodostaa J+-puun yläosan (Kuva 1). Judy-taulukon alla J+-puussa on lehtisolmut, jotka B+-puun tapaan sisältävät puun kaikki tietoalkiot. Judy-taulukkoon tallennetaan vain viitteet lehtisolmuihin. Tämä parantaa J+-puun tehokkuutta osavälikyselyissä verrattuna Judyyn.



Kuva 1: J+-puun rakenne [HXS09].

Judy-taulukko on digitaalinen puu, joka tulkitsee puuhun syötettävät avaimet tavuina, joiden avulla puuta selataan sen eri tasoilla. Käytettäessä 32-tavuisia avaimia puussa on enintään neljä tasoa, 64-tavuisilla avaimilla tasoja on kahdeksan. Judy-puuhun tehtävät hakuoperaatiot ovat tehokkaita. Puun korkeus ei suoraan riipu siihen tallennettavien alkioiden määrästä eikä välimuistihuteja synny paljon. Nämä ominaisuudet siirtyvät myös J+-puuhun.

J+-puusta on myös versio nimeltä pJ+-puu [HXS09]. Tämän version on tarkoitus ennestäänkin parantaa J+-puun välimuistitehokkuutta. pJ+-puu on muuten J+-puun kaltainen, mutta siinä käytetään esihakemista (*prefetch*). Tavallisessa J+-puussa on lehtisolmuissa osoitin vain seuraavaan ja edelliseen solmuun. Esihaetussa puussa lehtisolmusta on kuitenkin osoitin myös johonkin kauempana olevaan solmuun. Tämä parantaa välimuistitehokkuutta piilottamalla välimuistihuteja.

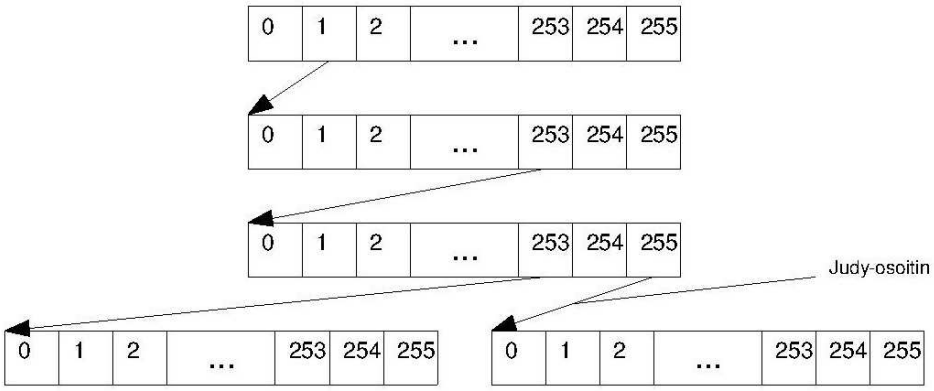
Tässä seminaariraportissa tullaan esittelemään J+-puun rakennetta ja toimintaa. Jotta tämä olisi mahdollista, seuraavassa luvussa luodaan katsaus Judy-puuhun, joka on ratkaiseva osa J+-puuta. J+-puusta esitellään myös esihakemista tukeva versio pJ+-puu. Raportin lopussa luodaan myös lyhyt katsaus testituloksiin, jotka osoittavat J+-puun olevan kilpailevia hakupuita tehokkaampi tietorakenne.

Judy-puu

Judy-puu on ratkaiseva osa J+-puuta. Se toimii puun yläosana ja sen ansiota ovat monet J+-puun hyvistä ominaisuuksista. Tässä luvussa esitellään Judyn keskeisimpiä ominaisuuksia, joiden avulla on parempi ymmärtää J+-puun toimintaa.

Judy-puun ominaisuudet

Judy-puun on kehittänyt Douglas Baskins. Se on 256-haarainen digitaalinen puu (kuva 2). Puun avaimet tulkitaan bitteinä ja jaetaan tavuiksi, joiden avulla puussa liikutaan eteen päin [HXS09]. Tavun perusteella valitaan tasolta oikea haara, joka käytännössä on osoitin seuraavalle tasolle. Avaimen ensimmäisen tavun perusteella valitaan haara puun ensimmäisellä tasolla ja toisen tavun perusteella valitaan haara puun toisella tasolla. Tätä jatketaan kunnes päästään puun loppuun. 32-bittisiä avaimia käytettäessä puussa on enintään neljä tasoa ja 64-bittisillä avaimilla kahdeksan tasoa. Puun toiminta on nopeata, koska jokaisella tasolla tehdään vain kevyitä operaatioita. Puun heikkoutena on kuitenkin, että se vaatii paljon tilaa sillä puussa on osoittimet myös sellaisille avaimille, joita puussa ei ole.



Kuva 2: Digitaalisen puun rakenne [HXS09].

Judy-puun yhteydessä käytetään muutamaa arvoa kuvaamaan puun ominaisuuksia [Bas02]. Laajuus (*expanse*) on mahdollisten avainten joukko. Esimerkiksi kuvan 2 puussa laajuus on juuresta tarkasteltuna 0 - $2^{24}-1$. Alemmilla tasoilla laajuus pieninee ja voi esimerkiksi alimmalla tasolla olla 0 – 255. Alemmilla tasoilla laajuutta kutsutaan myös alilaajuudeksi.

Puun asutus (*population*) taas on puussa olevien avainten määrä. Asutusta voidaan myös tarkastella jossakin alipuussa. Kun tarkastellaan sekä laajuutta että asutusta

jossakin puussa saadaan laskettua myös puun tiheys (*density*), joka on asutus jaettuna laajuudella. Tiheys on siis jokin luku välillä 0 – 1, jolloin 0 on tyhjä puu ja 1 puu, johon on syötetty kaikki mahdolliset avaimet.

Judy-puu on suunniteltu ennen kaikkea tehokkaaksi. Sitä ei ole suunniteltu yksinkertaiseksi, vaan sen rakenne on optimoitu mahdollisimman tehokkaaksi. Tavoitteena on ollut välttää tietojen hakemista välimuistiin. Haku keskusmuistista välimuistiin kestää 50 – 2000 konekäskyn ajan. Jos siis haku pystytään välttämään alle viidelläkymmenellä konekäskyllä, tehdään näin. Myös Judy-puun rakenne tukee välimuistin tehokasta käyttöä. Binääripuussa on paljon enemmän tasoja, minkä takia välimuistiin joudutaan hakemaan enemmän tietoa kuin Judyssa. B-puu taas vaatii käymään läpi kaikki solmut, mikä heikentää sen välimuistitehokkuutta. Judyn tehokkuutta parantaa myös se, ettei sitä tarvitse tasapainottaa puun kasvaessa.

Myös Judy-puun tilantarve on pyritty saamaan mahdollisimman pieneksi. Siinä missä tavallisissa digitaalisissa puissa jokainen solmu on samanlainen, käytetään Judyssa erilaisia solmuja tilanteesta riippuen. Judyn tilantarve onkin saatu hyvin pieneksi ja vain tiivisti täytetty taulukko käyttää tilaa tehokkaammin kuin Judy.

Judyn rakenne

Judy-puita on useita erilaisia. JudyL ja JudySL tyyppisiä käytetään J+-puussa [HXS09]. Tämän lisäksi on olemassa JudyHS ja Judy1. Seuraavaksi esitellään JudyL puun rakennetta tarkemmin. Se tallentaa 32-bittisiä arvoja 32-bittisillä avaimilla ja sitä käytetään JudySL-puun taustalla. JudySL pystyy tallentamaan pidempiä avaimia kuten merkkijonoja. Käytännössä se on taulukko JudyL-puita.

JudyL:ssä käytetään useita eri tietorakenteita, joiden avulla sen tehokkuutta on pyritty optimoimaan. Näistä tärkein on Judy-osoitin, jonka avulla puu haarautuu eri tasojen välillä. Judy-osoitin koostuu kahdeksasta tavusta. Ensimmäiset neljä tavua ovat osoite. Tämän jälkeen seuraavissa kolmessa tavussa tulee tieto tavuista, jotka yhdistävät kaikkia osoittimen alipuussa olevia avaimia, sekä alipuussa olevien avainten määrä. Judy-osoittimen viimeinen tavu kertoo, minkä tyyppiseen solmuun osoitin osoittaa. Mahdollisia tyyppejä ovat tyhjä alijoukko, haara, lehtisolmu tai välitön Judy-osoitin. Näistä haara tarkoittaa käytännössä sisäsolmua.

Judyssä haaroja on useita eri tyyppejä. Lineaariseen haaraan mahtuu korkeintaan seitsemän Judy-osoitinta. Lineaarinen haara on 64 tavun mittainen. Haaraan kuuluu tieto siinä olevien osoittimien määrästä, kuhunkin osoittimeen johtava avaimen tavu,

sekä itse osoittimet.

Bittikarttahaaraan mahtuu lineaarista haaraa enemmän osoittimia. Se jakautuu kahteen osaan. Ensimmäisen osan muodostavat bittikartat, joissa on kerrottu, mihin tavuihin liittyviä osoittimia haarassa on. Toisessa osassa ovat listat haaran osoittimista.

Kolmas mahdollinen haaran tyyppi on tiivistämätön haara, jota käytetään, jos haarassa on paljon avaimia. Se on pelkkä taulukko, jossa on 256 Judy-osoitinta. Osoittimet ovat siis kaikille mahdollisille avaimen arvoille, vaikka avain johtaisi tyhjään alipuuhun. Varsinaisia avaimen tavuja ei ole tallennettu taulukkoon, vaan oikea osoitin lasketaan sen paikasta taulukossa.

Myös lehtisolmuja on Judyssa useita erilaisia. Jos puussa on alle 32 avainta, käytetään juuritason lehtisolmua. Se sijaitsee suoraan juuressa ja siinä on tieto avainten määrästä sekä avaimet ja arvot. Lehtisolmu voi myös olla välitön. Tällöin lehtisolmu on itsessään Judy-osoitin. Arvo tallennetaan joko suoraan osoittimeen tai osoittimessa on linkki kenttään, jossa arvo on.

Lineaarisessa lehtisolmussa taulukko solmuun kuuluvista avaimista sekä lista arvoista. Tällainen lehtisolmu on korkeintaan 256-tavun kokoinen. Neljäs mahdollinen lehtisolmun tyyppi on bittikarttalehtisolmu. Sitä voidaan käyttää vasta puun alimmalla tasolla kun avaimesta on vain tavu jäljellä. Bittikarttalehtisolmu on samanlainen kuin bittikarttahaara. Ainoa ero on, että siihen on tallennettu Judy-osoittimien sijaan puuhun sijoitettuja arvoja.

Judy-puu muodostuu siis yllä esitetyistä osista. Puu valitsee erityyppisistä haaroista ja lehtisolmuista tilanteeseen sopivimman sen mukaan, kuinka monta avainta kyseisessä solmussa on. Jos haarassa on korkeintaan 7 avainta, käytetään lineaarista haaraa. Haaroissa joissa on paljon avaimia käytetään tiivistämätöntä haaraa. Näiden välissä, keskitiheälle haaroille, on tarkoitettu bittikarttahaara.

Lehtisolmuista juuritason solmu on tarkoitettu hyvin pienille, alle 31 avainta tallentaville, puille. Muita lehtisolmun tyyppejä käytetään sen mukaan, kuinka monta avainta on lehtisolmuun tallennettu. Tyypit ovat siis pienimmästä suurimpaan välitön lehtisolmu, lineaarinen lehtisolmu ja bittikarttalehtisolmu.

Judy-puun hyötyjä ovat kevyet hakuoperaatiot sekä pieni muistin kulutus. Nämä ominaisuudet tekevät Judys-puusta houkuttelevan tietorakenteen tietokannan hakemistoksi. Judy-puussa on kuitenkin myös heikkouksia, joiden takia se ei suoraan sovi hakemiston tietorakenteeksi. Osavälikyselyitä ei pysty kunnolla toteuttamaan siinä.

Judy-puuhun ei myöskään voi tallentaa useita saman avaimen omaavia avain-arvo-pareja. Nämä puutteet korjataan J+-puussa ilman, että Judy-puun tehokkuus kärsii.

J+-puu

J+-puu on Judy-puun inspiroima tietorakenne, joka pyrkii ratkaisemaan Judy-puun puutteita. Näin sitä pystytään käyttämään tietokantajärjestelmissä hakemistojen tietorakenteena. J+-puusta on myös kehitetty esihaettu versio pJ+-puu, jonka tarkoitus on ennestäänkin tehostaa J+-puun välimuistin käyttöä [HXS09].

J+-puun rakenne

J+-puu koostuu kahdesta osasta. Puun yläosan muodostaa Judy-puu ja puun alaosan lehtisolmut (kuva 1). Puun yläosan Judy-puuhun ei tallenneta arvoja vaan pelkästään viitearvoista ja osoittimista. Kaikki avaimet tallennetaan lehtisolmuihin nousevassa järjestyksessä. Jokaisen solmun pienin avain tallennetaan Judy-puuhun yhdessä lehtisolmuun osoittavan osoittimen kanssa. Tämä avain toimii nyt Judy-puussa viitearvona, joka osoittaa lehtisolmuun, jonka kaikki avaimet ovat suurempia tai yhtä suuria kuin viitearvo.

Kaikki J+-puun avain-osoitin-parit tallennetaan lehtisolmuihin. Näistä osoitin johtaa tietoalkioon, joka vastaa avainta. Jokaisessa solmussa on tallennettuna enintään m paria ja vähintään m/2 paria. Lehtisolmut ovat linkitetty siten, että jokaisesta solmusta on linkki edelliseen ja seuraavaan solmuun. Tällä tavalla J+-puun lehtisolmut muistuttavat B+-puuta ja tarjoavat hyvän tehokkuuden osavälikyselyissä.

J+-puun operaatiot

Seuraavaksi esitellään, kuinka J+-puun tärkeimmät operaatiot on toteutettu [HXS09]. J+-puun hakualgoritmi alkavat Judy-puun tutkimisella. Ensiksi etsitään haluttua avainta vastaavaa viitearvoa. Jos sitä ei löydy, palautetaan käyttäjälle, ettei haluttua alkiota löytynyt. Käytännössä tämä tapahtuu, kun haluttu avain on pienempi kuin pienin puuhun tallennettu avain. Kun oikea viitearvo löytyy, siirrytään arvoa vastaavaan lehtisolmuun. Tätä lehtisolmua kutsutaan rajaavaksi solmuksi.

Lehtisolmusta haluttua avainta etsitään binäärihaulla. Haun jälkeen käyttäjälle kerrotaan haun lopputuloksesta. Osavälikyselyissä haetaan ensiksi kyselyn alarajaa vastaavan alkio sisältävä lehtisolmu. Tämän jälkeen muut osaväliin kuuluvat alkiot

haetaan selaamalla lehtisolmuja solmujen välisten linkkien avulla.

Lisäysalgoritmi aloitetaan etsimällä lehtisolmu, joka on lisättävän avaimen rajaava solmu. Jos oikeaa viitearvoa ei löydy, lisätään alkio ensimmäiseen lehtisolmuun, koska tällöin lisättävä avain on pienempi kuin yksikään puussa valmiiksi oleva avain. Tällöin myös Judy-puussa olevaa pienintä viitearvoa tulee päivittää.

Kun rajaava solmu löydetään lisätään alkio siihen, jos solmu ei ole täynnä. Solmun ollessa täynnä se joudutaan halkaisemaan. Käytännössä solmun loppupää muodostaa uuden solmun joka lisätään alkuperäisen solmun perään. Uuden solmun viitearvo lisätään myös Judy-puuhun.

Poistoalgoritmi alkaa halutun alkion etsimisellä. Jos alkiota ei löydy, palautetaan tieto käyttäjälle. Jos alkio löytyy ja lehtisolmussa, jossa alkio on, on enemmän kuin m/2 alkiota, poistetaan kyseinen alkio. Vaikka kyseinen alkio olisi solmun pienin, ei Judy-puussa olevaa viitearvoa tarvitse päivittää.

Jos poistettava alkio sijaitsee solmussa, jossa on poiston jälkeen vähemmän kuin m/2 alkiota, tarkastelemme viereisiä solmuja. Jos solmua edeltävässä solmussa on yli m/2 alkiota, siirretään sen suurin arvo vajaaseen solmuun ja päivitetään Judy-puun viitearvoa. Jos tämä ei onnistu tarkastellaan vajaata alkiota seuraavaa solmua. Jos siinä on yli m/2 alkiota, siirretään sen pienin alkio vajaaseen solmuun ja päivitetään Judy-puun viitearvoa. Muussa tapauksessa vajaa solmu yhdistetään joko sitä edeltävään tai sitä seuraavaan solmuun. Tämän jälkeen päivitetään Judy-puun viitearvot vastaamaan uutta järjestystä.

J+-puu toisinnetuilla avaimilla

Judy-puuhun ei voida tallentaa kahta alkiota, joiden avain on sama. J+-puuhun tämän ominaisuuden saa kuitenkin toteutettua. Tämä on tärkeätä, jotta J+-puuta voidaan käyttää tietokantajärjestelmissä hakemistojen toteutukseen. Jotta J+-puuhun voidaan tallentaa toisinnettuja avaimia, täytyy Judy-puusta lehtisolmuihin viittaavien viitearvojen tulkintaa muuttaa. Nyt viitearvo viittaa lehtisolmun pienimpään uuteen arvoon. Tällä tarkoitetaan arvoa, jota ei esiinny edeltävässä solmussa. Esimerkiksi, jos solmussa 1 on arvot 3, 5, 7 ja solmussa 2 arvot 7, 7, 9, on viitearvo ensimmäiseen solmuun 3 ja jälkimmäiseen 9. Jos jossakin solmussa ei ole yhtään uutta arvoa, ei siihen solmuun osoita yhtään viitettä Judy-puusta. Myös uuden arvon lisäämisessä on otettava huomioon toisinnetut avaimet. Jos esimerkiksi edellä esitettyihin solmuihin yritetään lisätä arvo 8, päätyy normaali algoritmi ensimmäiseen solmuun, vaikka arvo tulisi lisätä

jälkimmäiseen solmuun.

pJ+-puu

Kuten jo edellä esiteltyjä tietorakenteita tarkasteltaessa on huomattu, on välimuistin tehokas käyttö tärkeässä osassa koko tietorakenteen tehokkuuden kannalta. Esimerkiksi perinteisen B+-puu suoritusajasta yli puolet menee hukkaan välimuistihutien johdosta. Tämän takia on kehitetty erilaisia hakupuita, jotka pyrkivät optimoimaan välimuistinkäytön [CGM01]. Esimerkiksi CSS-puussa lapsisolmujen osoitteet lasketaan suoraan isäntäsolmujen osoitteista, mikä vähentää tarvetta tuoda tietoa välimuistiin. Näin myös välimuistihutien määrä vähenee. Tämä puu ei kuitenkaan tue päivitysoperaatioita, joten se sopii vain rajattuihin käyttötarkoituksiin.

CSB+-puu taas on B+-puun versio, jossa jokaisessa sisäsolmussa on vain yksi osoitin, joka johtaa lapsisolmujen joukkoon. CSB+-puun hakuoperaatioiden tehokkuus on jonkin verran parempaa kuin tavallisen B+-puun. Se kuitenkin on selkeästi hitaampi päivitysoperaatioissa.

Myös J+-puun välimuistitehokkuutta on pyritty parantamaan uusilla versioilla. Yhden arvon kyselyissä ei ole suuresti parantamisen varaa, koska hakun etenemistä ei voida etukäteen ennustaa. Haku päättyy yhtä todennäköisesti kaikkiin solmuihin. Toiseksi myös J+-puun yläosana toimiva Judy-puu on jo itsessään tehokas käyttämään välimuistia.

Sen sijaan osavälikyselyiden tehokkuutta pystytään parantamaan. Haettuaan osavälin ensimmäisen alkion, lehtisolmuja aletaan selaamaan eteen päin. Tällöin on siis etukäteen tiedossa, mitä solmuja tullaan tarvitsemaan. On siis mahdollista tuoda välimuistiin jo valmiiksi tietoa, jota tullaan vasta tulevaisuudessa tarvitsemaan. Näin pystymme kätkemään osan välimuistihudeista, kun prosessori pystyy jatkamaan operaatioita alkioilla, jotka on jo valmiiksi tuotu välimuistiin.

J+puussa on kuitenkin ongelmana, että lehtisolmuista on lenkit vain edelliseen ja seuraavaan solmuun. Jos ajattelemme tilannetta, jossa olemme lukemassa solmua n ja haluaisimme tuoda solmun n+3 jo ennalta välimuistiin. Jotta solmuun n+3 päästään käsiksi pitää ensiksi lukea solmu n+2. Tämä taas edellyttää solmun n+1 lukemista. Tämä taas sotii koko esihakemisen periaatteita vastaan.

pJ+-puu on J+-puu esihakemista tukeva versio. pJ+-puu on esitelty tutkijoiden Hua, Xiao-Yong ja Shan vuonna 2009 julkaisemassa artikkelissa [HXS09]. pJ+-puu ratkaisee

ongelman lisäämällä jokaiseen lehtisolmuun osoittimen esihakemista varten. Toinen mahdollinen tapa olisi tallentaa osoittimet taulukkoon, kuten pB+-puussa on tehty. Suoraan solmuun tallennettaessa kuitenkin saavutetaan parempi suorituskyky.

J+-puun hakualgoritmit eivät muutu pJ+-puussa suorituksen kuin tehokkuudenkaan suhteen. Lisäys- ja poistoalgoritmien taas tulee muokata myös muiden solmujen esihakuosoittimia. Muokattavien solmujen määrä riippuu siitä, kuinka pitkälle esihakuosoitin osoittaa. Sopivaa esihaun etäisyyttä voidaan kaavoilla, mutta on todettu, että etäisyyden ollessa suhteellisen pieni, tietorakenteet toimivat parhaiten [CGM01].

J+-puun suorituskyky

Tässä luvussa tarkastellaan J+-puun ja pJ+-puun suorituskykyä ja vertaillaan sitä muiden vastaavien tietorakenteiden suorituskykyyn. Suuri ero J+-puulla, pJ+-puulla sekä Judy-puulla muihin vastaaviin hakupuihin, kuten B+-puu, pB+-puu, T-puu sekä CSB+-puu, nähden on, että niiden korkeus on rajoitettu. Judy-puu voi olla enintään neljän tason korkuinen, J+-puu sekä pJ+-puu voivat olla enintään viiden korkuisia. Tällöin myös puiden suorittaman vertailuoperaatioiden määrä hakuoperaatiossa on vakio. Muiden hakupuiden korkeus kasvaa niihin lisättäessä alkioita. Tämä näkyy myös hakualgoritmin suorittamien vertailuiden määrässä. Esimerkiksi, kun n on puussa olevien alkioiden määrä, B+-puu sekä pB+puu suorittavat $O(\log_2^n)$ vertailuoperaatiota hakualgoritmin aikana. Myös hakualgoritmin aikana tapahtuvien välimuistihutien määrä kasvaa näissä puissa puuhun lisättäessä alkioita. Judy-puussa taas huonoimmassa tapauksessa, jossa puun kaikki sisäsolmut ovat bittikartta haaroja sekä lehtisolmu lineaarinen haara, syntyy hakualgoritmin aikana 9 välimuistihutia.

J+-puun suoritustehoa on tutkittu myös kokeellisesti [HXS09]. Puiden suoritustehoa mitattiin suorittamalla puihin hakuoperaatioita. Puiden suorituskykyä testattiin eri avainten määrillä, jotka vaihtelivat 500 000 ja 40 000 000 välillä. Puihin suoritettaessa yksittäisten arvojen hakuja huomattiin, että J+-puu sekä pJ+-puu suoriutuvat selvästi B+-puuta, pB+puuta, CSB+-puuta ja T-puuta nopeammin. Kun J+puu käytti hakuoperaatioihin aikaa 600ms – 900ms, käytti B+-puu variantteineen aikaa 800ms – 1350ms. T-puulla aikaa kului vielä selvästi tätäkin enemmän.

Yksittäisiä avaimia etsittäessä esihakemista hyödyntävä pJ+-puu ei saa mitään etua ja se suoriutuukin hauista yhtä nopeasti kuin tavallinen J+-puu. Osavälikyselyitä tehtäessä esihakemisesta tulee kuitenkin hyötyä. Siinä missä pJ+-puu ja pB+-puu kuluttavat

kyselyyn alle 40ms neljälläkymmenellä miljoonalla avaimella, menee J+-puulla ja B+-puulla aikaa kyselyyn tuplasti kauemmin.

J+puu pärjää myös hyvin vertailtaessa tietorakenteiden tilantarvetta. Sitä testattiin syöttämällä puihin neljäkymmentä miljoonaa. Tällöin J+-puu käytti tilaa 453,6 MB, B+-puu 456,8 MB, pJ+-puu 457,2 MB, pB+-puu 460,4 MB, T-puu 482,4 MB sekä CSB+-puu 645,2 MB.

Yhteenveto

J+-puu on tehokas hakupuu, joka vastaa toimii kilpailijana monille perinteisille hakupuille, kuten B+-puulle. Se käyttää hyväkseen Judy-puuta, joka on tehokkaaksi optimoitu digitaalinen puu. J+-puu lisää Judy-puuhun ominaisuuksia, jotka mahdollistavat sen käytön tietokantajärjestelmissä hakemistojen toteutuksessa. Näitä ominaisuuksia ovat mahdollisuus selata lehtisolmuja solmujen välisten linkkien avulla sekä mahdollisuus käyttää samaa avainta useammalle alkiolle. Sekä J+-puun että Judy-puun suunnittelussa on panostettu välimuistin tehokkaaseen käyttöön. Tätä tehokkuutta pyrkii viemään vielä pidemmälle pJ+-puu, joka tukee esihakemista. Osittain juuri välimuistitehokkuuden ansiosta J+-puu sekä pJ+-puu ovat hyvin tehokkaita tietorakenteita.

Lähteet

Bas02 Baskins D, Judy functions – C libraries for creating and accessing dynamic arrays, http://judy.sourceforge.net

CGM01 Chen S, Gibbons P. B., Mowry T. C., Improving index performance through prefetching. In proc. of the SIGMOD 2001 Conference. 2001, sivut 235-246.

HXS09 Hua Luan, Xiaoyong Du, Shan Wang: Prefetching J -Tree: A Cache-Optimized Main Memory Database Index Structure. J. Comput. Sci. Technol. (JCST) 24(4):687-707 (2009)

HXS07 Hua Luan, Xiaoyong Du, Shan Wang, Yongzhi Ni, Qiming Chen: J+-Tree: A New Index Structure in Main Memory. DASFAA 2007: 386-397