R-puut paikkatietojärjestelmien hakemistorakenteina

Jan Lindström

Helsinki 28. huhtikuuta 1997

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos
Paikkatietojärjestelmät seminaari

Sisältö

1 Johdanto

Nopeasti kasvava joukko tietokantasovelluksia tarvitsee mahdollisuuden tallettaa ja hakea suuria määriä useampiulotteisia pisteitä, viivoja ja aluetietoja. Tällaisia tietokantasovelluksia ovat esimerkiksi tietokoneavusteinen suunnittelu (CAD), paikkatietojärjestelmät (geographic information systems, GIS) ja kuvatietokannat.

Yleinen operaatio paikkatiedossa on kaikkien olioiden haku tietyltä alueelta. Esimerkiksi kaikkien maiden haku, joilla on maata 20 kilometrin sisällä tietystä pisteestä laskettuna. Tällainen paikkatietohaku tapahtuu useasti tietokoneavusteisessa suunnittelussa ja paikkatietojärjestelmissä. Siksi on tärkeää pystyä hakemaan olioita tehokkaasti olioiden sijainnin perusteella.

Hakemistorakenne tulisi perustua olioiden sijaintitietoihin. Klassiset yksiulotteiset tietokantojen hakemistorakenteet eivät sovi kovin hyvin useampiulotteisen tiedon hakuun. Avainarvon suoraan yhtäsuuruusvertailuun perustuvat rakenteet kuten hajautus eivät sovi arvovälikyselyihin. Yksiulotteiseen avainarvojen järjestämiseen perustuvat rakenteet kuten B-puut ja ISAM indeksit eivät ole riittäviä, koska hakuavaruus on moniulotteinen.

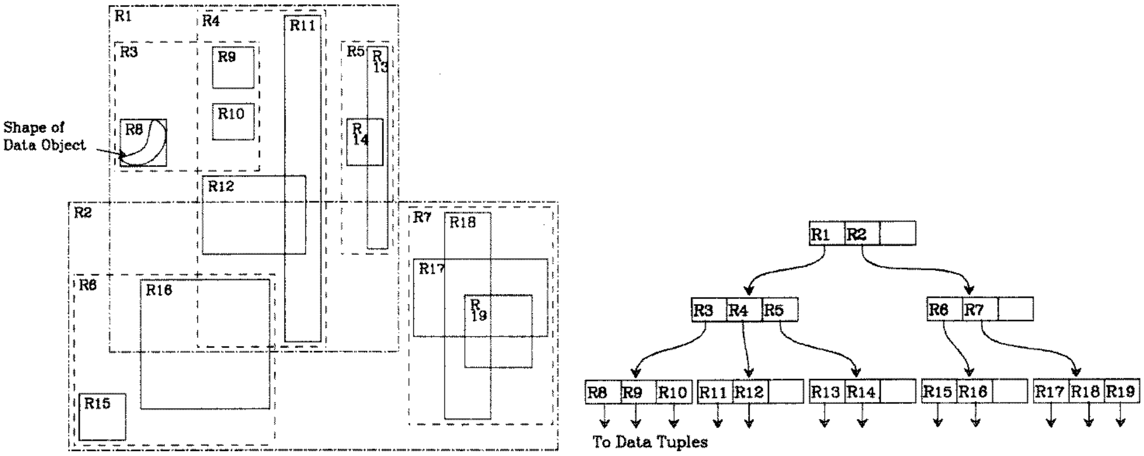
Tässä artikkelissa kuvataan paikkatietojärjestelmissä käytettyjä hakemistorakenteita ja tarkastellaan yhtä hakemistorakennetta tarkemmin esittelemällä tärkeimmät algoritmit. Tämän jälkeen tarkastelua laajennetaan esittelemällä yksi mahdollisuus paikkatietojärjestelmissä käytettävien hakemistorakenteiden samanaikaisen käytön lisäämiseksi.

2 R-puut hakemistorakenteina

Paikkatiedon oliot kuvataan hakemistorakenteissa yleisesti minimisuorakulmioilla (minimum bounding rectangle, MBR). Minimisuorakulmio rajaa varsinaista oliota kaikissa olion ulottuvuuksista siten, että koko olio jää suorakulmion sisäpuolelle.

Lisää minimisuorakulmioiden ominaisuuksista löytyy lähteestä [PaT94].

R-puu [Gut84] on B-puuta muistuttava tasapainotettu puurakenne (kuva 1). R-puun lehtisolmut koostuvat hakemistotietueista, jotka ovat muotoa (I,monikkotunniste). Monikkotunniste viittaa yksikäsitteiseen monikkoon tietokannassa ja I on n-ulotteinen suorakulmio, joka rajaa indeksein 0 – (n-1) indeksoituja moniulotteisia olioita (I=$(i_0, i_1, ..., i_n)$), missä n on ulottuvuuksien lukumäärä ja I suljettu väli [a,b], joka kuvaa olion laajuutta ulottuvuudessa i.



(a) Suorakulmiot (b) R-puu

Kuva 1: Suorakulmioita ja niitä vastaava R-puu [Gut84].

Luonnollisesti rakenteen tulee sallia osittain päällekkäin limittyvät hakemistosuorakulmiot. Siksi ei voida taata, että vain yksi hakupolku riittää olion hakuun puusta. R-puun tason peitolla (coverage) tarkoitetaan kyseisellä tasolla olevien suorakulmioiden yhteenlaskettua alaa. R-puun tason limityksellä taas tarkoitetaan kokonaisalaa, joka sisältyy kahteen tai useampaan solmuun. R-puuta optimoitaessa onkin olennaista minimoida sekä peitto että limitys. Peiton minimointi vähentää tyhjää tilaa suorakulmioissa. Limityksen minimointi on kuitenkin hakujen kannalta vieläkin tärkeämpää, koska haettavaa oliota saatetaan joutua turhaan hakemaan useampaa polkua pitkin [SRF87].

R-puun variaatio R+-puu [SRF87] ideana on sallia suorakulmioiden jakaminen limittymisen estämiseksi. Limittymisen esto saavutetaan rakenteen viemän tilan kustannuksella, josta seuraa puun korkeuden kasvu.

Toinen variaatio R-puista on R*-puu [BKS+90]. R*-puun ideana on optimoida jokaisen hakemistosuorakulmion peittoa, marginaalia ja limitystä. R*-puu onkin eräänlainen tekninen lähestyminen R-puihin. Tarkkaan laadituissa testiolosuhteissa on tutkittu eri ominaisuuksien vaikutusta R*-puun tehokkuuteen ja testien perusteella on valittu parhaimmat toteutusratkaisut. Standardoitu testialusta ja tarkat tehokkuusvertailut osoittivat R*-puun selkeästi tehokkaammaksi kuin muut olemassaolevat R-puun variaatiot [BKS+90].

R*-puusta esitetään jatkossa tarkemmin puun rakenne ja tärkeimmät algoritmit. Kolmas R-puun variaatio on Rlink-puu [KoB95], jossa kaikki solmut jokaisella tasolla linkitetään listaksi vasemmalta oikealle. Rlink-puuta käsitellään tarkemmin kappaleessa 4.

3 R*-puun rakenne ja algoritmit

R*-puu on rakenteeltaan sama kuin R-puu (kuva 1). Hakemistorakenteen sisäsolmut koostuvat tietueista, jotka ovat muotoa (I,lapsiosoitin), missä lapsiosoitin on solmua alemman tason (lapsen) osoite ja I on suorakulmio, joka rajaa kaikki alemman tason solmujen tietueiden suorakulmiot. Olkoon M tietueiden lukumäärän yläraja solmuissa ja m vastaavasti tietueiden lukumäärän alaraja ($2 <= m <= M/2$). R*-puu (R-puu) toteuttaa seuraavat ehdot:

- lehtisolmulla, joka ei ole juuri, on m – M tietuetta,
- jokaisessa lehtisolmun tietueessa I on pienin suorakulmio, joka peittää n-ulotteisen monikkoa vastaavan tieto-olion,
- jokaisella solmulla, joka ei ole lehti eikä juuri, on m – M lasta,
- solmulla, joka ei ole lehti, I on pienin suorakulmio, joka rajaa lapsisolmujen tietueiden suorakulmiot,

- juurella on ainakin 2 lasta, ellei se ole lehti ja
- kaikki lehdet ovat samalla tasolla.

3.1 Tietueen haku puusta

Hakualgoritmi (kuva 2) etenee R-puussa juuresta T lehtiin palauttaen ne tietueet, jotka sisältyvät haettavaan suorakulmioon S. Algoritmi muistuttaa B-puun vastaavaa algoritmia. Hakualgoritmissa hakemistosolmun E suorakulmiota merkitään E.i:llä ja monikkotunnistetta tai lapsiosoitinta E.p:llä.

```
Jos T ei ole juuri [ S1. Etsi alipuut]

    Käy läpi kaikki hakemistotietueet E ja tarkista peittääkö
    niitä vastaava suorakulmio E.i S:n. Kaikille peittäville
    hakemistotietueille käynnistä algoritmi Search parametrillä
    E.p.

Muuten [ S2. Etsi lehti ]

    Käy läpi kaikki E:n tietueet ja tutki peittääkö vastaava
    E.i:n suorakulmio S:n. Mikäli peittää E.p on haettu tietue.
Loppu
```

Kuva 2: Algoritmi Search.

3.2 Lisääminen puuhun

Tietueiden lisäys R-puuhun vastaa B-puiden lisäystä. Uudet hakemistotietueet lisätään lehtiin (kuva 3). Liian täydet solmut jaetaan ja jako etenee puussa ylöspäin. Sekä R-puu että R*-puu ovat epädeterministisiä solmujen valinnassa tietueille eli eri lisäysjärjestys tietueissa tuottaa erilaisen puun. Tästä syystä R*-puu kärsii vanhoista tietueista. Rakenteen rakennuksen alkuvaiheessa lisätyt hakemistosuorakulmiot eivät pysty varmistamaan hyvää hakutehokkuutta tämänhetkisessä tilanteessa. R*-puu pakottaa tietueiden uudelleenlisäyksen lisäysalgoritmin aikana, joka aiheuttaa rakenteen dynaamisen uudelleenorganisoinnin (kuva 4).

```
Algoritmi Insert

I1. Käynnistä algoritmi ChooseSubtree etsimään solmu L, mihin tietue E lisätään.

I2. Jos solmussa L on vähemmän kuin M tietuetta, lisää E solmuun L.
    Muuten
        Käynnistä algoritmi OverflowTreatment parametrillä L.
        Jos solmun jako tapahtui
            Propagoi algoritmi OverflowTreatment puussa ylöspäin
        Jos solmun jako aiheutti juurisolmun jaon
            Luo uusi juurisolmu
    Loppu

I4. Sovita kaikki peittävät suorakulmiot lisäyspolulla siten, että ne
    ovat minimisuorakulmioita lapsisuorakulmioiden suhteen.

Algoritmi ChooseSubtree.

C1. Aseta N:ksi juuri

C2. Jos N on lehti
        palauta N
    Muuten
        Jos N.p osoittaa lehteen [minimoi limityskustannukset]
            Valitse se N:n tietue, jonka suorakulmiota tarvitsee limittää
            vähiten uuden tietosuorakulmion lisäämiseksi. Jos samanarvoisia
            limityksiä on useampia valitse tietue, jonka suorakulmio on
            alaltaan pienin.
        Muuten [minimoi alakustannukset]
            Valitse se N:n tietue, jonka suorakulmio vaatii vähiten
            laajennusta uuden tietosuorakulmion lisäämiseksi. Jos
            samanarvoisia laajennuksia on useampia valitse tietue, jonka
            suorakulmio on alaltaan pienin.
        Loppu
    Loppu
C3. Aseta N:ksi valitun tietueen E.p ja jatka kohdasta C2.

Algoritmi OverflowTreatment.

Jos kyseessä ei ole juurisolmu ja ensimmäinen kutsukerta tälle tasolle
    Käynnistä algoritmi ReInsert
Muuten
    Käynnistä algoritmi Split
Loppu
```

Kuva 3: Algoritmi Insert

Jos uusi tietosuorakulmio lisätään, jokainen ensimmäinen ylivuototilanne aiheuttaa p:n tietueen uudelleenlisäyksen käsittelyssä olevalle tasolle. Tämä saattaa aiheuttaa solmun jaon, joka aiheutti ylivuodon jos kaikki tietoalkiot uudelleenlisätään samaan paikkaan. Beckmannin ja muiden [BKS$^+$90] tutkimus osoitti että p $=$ 30% M:stä

```
Algoritmi ReInsert. Uudelleenlisää p tietuetta R-puuhun.

R1. Kaikkille M+1:lle tietueelle solmussa N laske etäisyys suorakulmioiden
    keskustasta ja solmun N suorakulmion keskustasta.

R2. Lajittele tietueet laskevaan järjestykseen etäisyyksien mukaan

R3. Poista ensimmäiset p tietuetta solmusta N ja sovita N:n suorakulmio.

R4. Alkaen maksimietäisyydestä tai minimietäisyydestä käynnistä
    algoritmi Insert solmujen uudelleenlisäystä varten.
```

Kuva 4: Algoritmi ReInsert

lehtisolmuille ja sisäsolmuille tuottaa parhaimman tehokkuuden.

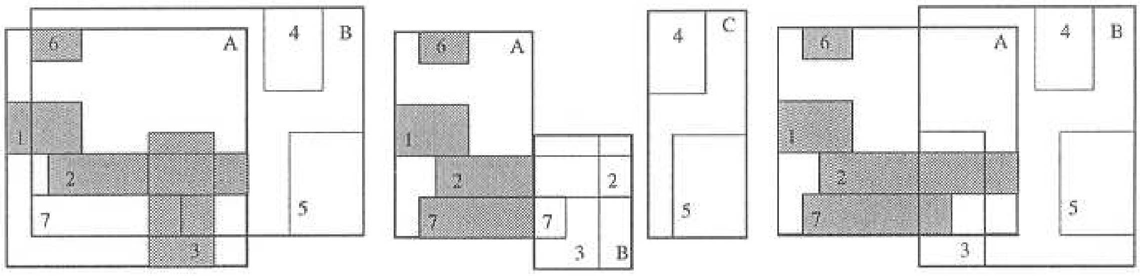
Yhteenvetona voidaan nähdä, että

- Uudelleenlisäys muuttaa tietoalkioiden paikkoja vierekkäisten solmujen välillä vähentäen näin päällekkäisyyttä,
- Sivutuotteena tilankäyttö tehostuu,
- Uudelleenorganisoinnin ansiosta tapahtuu vähemmän solmujen jakoja,
- Koska solmun uloimmat suorakulmiot uudelleenlisätään hakemistosuorakulmion muoto muuttuu enemmän neliömäiseksi.

3.3 Solmujen jakaminen

Koska R-puu sallii solmujen limittymisen, voidaan puussa joutua turhaan tutkimaan useampia hakupolkuja. R+-puu ja R*-puu on kehitetty estämään limittyvien alueiden aiheuttaman tehokkuuden lasku ja ylimääräisen tyhjän tilan poistoon. R+puu sallii solmujen suorakulmioiden jakamisen limittymisen estämiseksi. Toisaalta R*-puu sallii suorakulmioiden limittymisen, mutta yrittää minimoida limittymistä organisoimalla suorakulmiot alkuperäistä R-puuta mutkikkaammilla algoritmeilla.

Kuva (kuva 5) havainnollistaa, kuinka eri algoritmit organisoivat suorakulmiot sisäsolmuissa. Alkuperäinen R-puu muodostaisi hakemistosolmut A ja B. Solmu A sisältää minimisuorakulmiot 1,2,3 ja 6 (harmaat suorakulmiot) ja solmu B sisältää suorakulmiot 7,4 ja 5 (kuva 5a). R+-puu jakaa suorakulmiot siten, että solmut A ja B eivät limity. Tästä aiheutuu duplikaatteja tietueita, joka saattaa kasvattaa hakemistosolmujen määrää (kuva 5b solmu C). R*-puu ryhmittelee solmut eri tavalla saavuttaen pienemmän limittymisen ja peiton sisäsolmuissa (kuva 5c).



(a) (b) (c)

Kuva 5: R-puiden variaatioiden solmujen jaot.

R*-puu käyttää seuraavaa metodia solmujen jakoon [BKS+90]. Tietueet lajitellaan kaikilla akseleilla ensin suorakulmion ala-arvon mukaan ja tämän jälkeen yläarvon mukaan. Jokaisessa lajittelun M-2m+2 jaossa jaetaan M+1 tietuetta kahteen ryhmään, missä k:ssa jaossa (k= 1,...,(M-2m+2)) ensimmäisessä ryhmässä on (m+1)+k tietuetta ja toisessa ryhmässä loput tietueet. Jokaisessa jaossa lasketaan hyvyysarvo. Lopullinen jako valitaan hyvyysarvojen mukaisesti. Kolmea erilaista hyvyysarvoa ja erilaisia hyvyysarvojen kombinaatioita testattiin kokeellisesti, joista valittiin tehokkuudeltaan paras (kuva 6).

3.4 Poistaminen puusta

Tietueiden poistossa alivuotaneiden solmujen käsittely eroaa B-puun poistoalgoritmeista. B-puun tapainen alivuotaneen ja viereisen solmun yhdistäminen on mahdollista R-puissa. Gutmann [Gut84] valitsi kuitenkin solmujen uudelleenlisäyksen. Uudelleenlisäyksellä saavutetaan sama tulos kuin solmujen yhdistämisellä ja uudelleenlisäys on helpompi toteuttaa, koska lisäysalgoritmia voidaan käyttää apuna.

R*-puun poisto-operaatio on sama kuin R-puussa (kuva 7). Jos lehtisolmu L:ssä on

```
Algoritmi Split. Jaa ylivuotanut solmu kahteen solmuun.

S1  Käynnistä algoritmi ChooseSplitAxis määrittämään akseli, jota kohtisuoraan
    jako suoritetaan.

S2  Käynnistä ChooseSplitIndex määrittämään parhaan jaon kahteen luokkaan
    kyseisellä akselilla.

S3  Jaa tietueet kahteen ryhmään.

Algoritmi ChooseSplitAxis. Valitse jakoakseli.

A1  Jokaiselle akselille suorita
        Lajittele tietueet akseleilla ensin suorakulmion ala-arvon mukaan,
        tämän jälkeen yläarvon mukaan ja määritä jaot.
        Laske eri jakojen marginaaliarvojen summa S.
    Loppu

A2  Valitse jakoakseliksi se akseli, jonka S on pienin.

Algoritmi ChooseSplitIndex. Valitse jakoakselin jako.

I1  Valitse jakoakselilla jako, jonka limitysarvo on mahdollisimman pieni.
    Jos useampia samoja limitysarvoja, valitse pienin aluearvon jako.
```

Kuva 6: Algoritmi Split

liian vähän tietueita, poistetaan lehtisolmu ja uudelleensijoitetaan L:n tietueet. Solmujen poistoa propagoidaan ylöspäin tarvittaessa. Poistopolulla olevat suorakulmiot sovitetaan pienemmäksi, jos mahdollista.

4 Samanaikaisuus R-puissa

Sovellusohjelmien kysely- ja lisäystapahtumien yhteydessä täytyy käsiteltävät tietoalkiot lukita sarjallistuvuuden takaamiseksi. Samaan tapaan joudutaan lukitsemaan hakemistorakenne hakemistorakenteen haku-, lisäys- ja poistotapahtumissa. Lisäystapahtuman aiheuttama solmun jakaminen saattaa pahimmassa tilanteessa propagoitua juureen asti, jolloin koko puu on lukittu. Tällöin kaikki muut tapahtumat joutuvat odottamaan lisäystapahtuman päättymistä ja lukkojen vapautumista.

R-puille on toteutettu useita erilaisia lukitusalgoritmeja, jotka ovat lähtöisin B-

```
Algoritmi Delete.

D1  Käynnistä algoritmi FindLeaf, joka etsii E:n sisältävän lehtisolmun L.

D2  Poista tietue E L:stä.

D3  Käynnistä CondenseTree parametrillä L.

D4  Jos juurisolmulla on vain yksi lapsi puun tasapainotuksen jälkeen,
    muodosta lapsesta uusi juurisolmu.

Algoritmi FindLeaf.

L1  Jos T ei ole lehti
        Käsittele jokainen tietue F T:stä ja tarkista limittyykö T.i E.i:n
        kanssa. Jokaiselle limittyvälle tietueelle käynnistä algoritmi
        FindLeaf parametrilla F.p
L2  Muuten
        Käsittele jokainen tietue T:stä ja tarkista vastaako se E:tä.
        Jos E löytyy palauta T.
    Loppu

Algoritmi CondenseTree.

C1  Aseta N = L. Aseta poistettujen solmujen joukko Q tyhjäksi.

C2  Jos N on juurisolmu
        Siirry kohtaan C6
    Muuten
        Aseta P = N:n isäsolmu ja aseta En = n tietue solmussa P.

C3      Jos N:ssä on vähemmän kuin m tietuetta
            poista En tietue P:stä ja lisää N joukkoon Q.
        Loppu

C4      Jos N:ää ei ole poistettu
            Sovita En.i, siten että se sisältää tarkasti kaikki tietueet
            solmusta N.
        Loppu

C5      Aseta N = P ja toista kohdasta C2.
    End

C6  Uudelleenlisää kaikki tietueet joukosta Q. Lehtitason tietueet
    uudelleenlisättään lehtitasolle algoritmilla Insert. Ylemmän
    tason tietueet täytyy asettaa ylemmäksi puussa siten, että
    tietueista riippuvaiset lehtitason alipuut ovat samalla
    tasolla kuin muut lehtisolmut puussa.
```

Kuva 7: Algoritmi Delete

puiden vastaavista algoritmeista [NgK93]. Eräs tällainen B-puun variaatio Blink-puu [LeY81, Sag86] yhdistää solmut jokaisella tasolla oikealle johtavalla linkillä ja kompensoi keskeneräiset solmujen jaot seuraamalla oikealle johtavia linkkejä. Valitettavasti Blink tekniikka olettaa lineaariset avainarvot rakenteelta ja siksi ei sovellu suoraan R-puihin.

Tavoitteena on muodostaa Blink-puuta vastaava rakenne Rlink-puu [KoB95]. R-puu kuitenkin eroaa B-puusta muutamalla tärkeällä tavalla ja siksi Blink-puun strategia on riittämätön. Ongelma on lähtöisin avainarvojen järjestyksen puuttumisesta R-puissa. Rakenteen toteuttamiseksi on pystyttävä vastaamaan kahteen tärkeään kysymykseen: Kuinka havaita onko solmu jaettu ja kuinka rajoittaa oikealle siirtymistä puussa. Blink-puussa vastaus molempiin kysymyksiin piilee lineaarisessa avainarvojen järjestyksessä ja siinä, että solmut tietyllä tasolla ovat järjestetty oikealle johtavan linkin mukaisesti [KoB95].

On mahdotonta soveltaa samaa strategiaa R-puihin. Ensiksikin avainarvot eivät pysty yksiselitteisesti kertomaan onko solmun jako tapahtunut. Avainarvo isäsolmussa saattaa kuulua hakuväliin vaikkakin lapsisolmujen avainarvot eivät kuulu. Vaikka solmujen jako havaittaisiinkin oikealle siirtymisen määrää ei pystytä selvittämään avainarvoja tutkimalla [KoB95].

R-puuhun täytyy lisätä ylimääräistä informaatiota, jonka avulla R-puuta voidaan läpikäydä jatkuvasti muuttuvassa puurakenteessa. Eräs ratkaisu on lisätä puuhun looginen järjestysnumero (LSN, logical sequence number) [KoB95]. Nämä järjestysnumerot vastaavat aikaleimoja siinä, että ne monotonisesti kasvavat aikaa myöten mutta eivät vastaa mitään todellista reaaliaikakelloa. Solmut ja haku- ja lisäysalgoritmit on suunniteltu siten, että niissä otetaan huomioon LSN arvot puussa liikkumiseen.

Rlink-puu eroaa R-puusta kahdella tärkeällä tavalla. Ensinnäkin kaikki solmut samalla tasolla on yhdistetty linkeillä oikealle. On tärkeää huomata, että päinvastoin kuin Blink-puussa yhdistetyt solmut tietyllä tasolla evät esitä avainarvojen järjestystä pienemmästä suurempaan tai yleisemmin eivät esitä järjestystä solmujen isäsolmuihin.

Esimerkiksi (kuva 8) solmu p1 edeltää solmua p2 oikealle johtavissa linkeissä. Kuitenkin p1:n lapsisolmu c4 ei ole ennen lapsisolmua c2. Tällainen tilanne saattaa muodostua, jos solmu p1 jaetaan ja c2:n on siirretty uuteen solmuun p2 [KoB95].

Toinen lisäys rakenteeseen on uniikki LSN jokaiseen solmuun. Jokainen solmu sisältää avainsuorakulmion, linkin lapsisolmuun ja LSN numeron, jonka solmu olettaa lapsisolmussa olevan. Jos solmu jaetaan, uusi oikeanpuoleinen solmu saa uuden LSN numeron. Hakuproseduuri havaitsee solmun jaon vertailemalla isäsolmusta otettua oletettua LSN numeroa lapsisolmussa olevaan todelliseen LSN numeroon. Jos lapsisolmun LSN numero on suurempi, jako on tapahtunut ja jatketaan oikean solmun etsimistä seuraamalla oikealle johtavaa linkkiä.

Rlink-puu voidaan määritellä tasapainotetuksi puuksi, jonka hakemistosolmut sisältävät joukon tietueita ja oikean linkin r. Jokaisella puun tasolla solmujen oikeat linkit muodostavat yksisuuntaisen linkitetyn listan. Tietueet sisäsolmuissa sisältävät avainsuorakulmion k, osoittimen p ja odotetun LSN numeron l siten, että joko

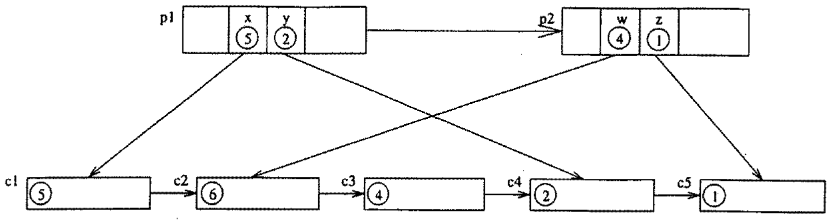
1. Perustapaus, jossa rakenteen lapsitaso vastaa täysin isäsolmua.

   Eli p osoittaa lapsisolmuun N, missä l on N:n LSN. N:n oikea linkki on tyhjä linkki tai jokin solmu R, johon on myös linkki ylemmältä tasolta joltakin tietueelta.

   Esimerkiksi (kuva 8) tietue x osoittaa solmuun c1. Sekä c1:n että x:n LSN ovat samoja ja c1:n oikea linkki osoittaa solmuun c2, johon on linkki tietueesta w solmussa p2.

2. Keskeneräinen jako lapsitasolla, joka on kompensoitu oikealla linkillä. Eli p osoittaa lapsisolmuun N, missä solmun N LSN on suurempi kuin l. On olemassa solmu M, jonka LSN on sama kuin l. Solmu M voidaan löytää seuraamalla oikeita linkkejä solmusta N läpikäyden solmut, joiden LSN on suurempi kuin l. Solmuun M ei ole linkkiä ylemmältä tasolta.

   Esimerkiksi (kuva 8) tietue w solmussa p2. Tietueen w LSN on pienempi kuin solmun c2 ja sama kuin solmun c3. Solmu c3 voidaan tavoittaa seuraamalla oikeaa linkkiä solmusta c2. Solmulla c3 ei ole tietuetta ylemmällä tasolla, mutta sen oikeaan velisolmuun c4 on linkki tietueesta y solmussa p1. Tällainen tilanne aiheutuu, kun solmu c2 jaetaan ja jakoa ei ole propagoitu ylemmälle tasolle vielä.



Kuva 8: Rlink-puun rakenne.

Rlink-puulle voidaan määritellä haku, lisäys ja poistoalgoritmit vastaavasti kuin Blink-puulle [KoB95]. Tarkastelemme tässä artikkelissa tarkemmin vain hakualgoritmia. Lisäys- ja poistoalgoritmit sisältävät niin paljon yksityiskohtia etteivät ne mahdu tämän artikkelin alueeseen.

4.1 Hakualgoritmi

Hakuprosessissa (kuva 9) haetaan kaikki lehtisolmuissa olevat tietueet, jotka ovat täyttävät hakuehdot. Koska avainarvot voivat limittyä, joudutaan yleisessä tapauksessa navigoimaan useita hakupolkuja indeksissä. Kaikkien tarvittavien polkujen läpikäyntiin käytetään pino tietorakennetta. Hakuprosessi alkaa viemällä juuri pinoon. Tutkimaton solmu poistetaan pinosta ja kaikki tietueet, jotka täyttävät hakuehdot viedään pinoon ja koko prosessia jatketaan. Jos lehtisolmu poistetaan pinosta, palautetaan se kyselyn tuloksena. Prosessi päättyy kun pino on tyhjä.

Pinoon työnnetään solmusta sekä osoite että LSN numero. Jos tutkittaessa solmua p löytyy LSN numero, joka on suurempi kuin pinon päällimmäinen LSN numero on solmu jaettu. Solmun jako kompensoidaan tutkimalla kaikki solmut, jotka on jaettu tästä solmusta. Siksi lisäämme p:n oikean linkin solmut ja solmun, jonka LSN on sama kuin odotettu LSN solmulle p.

```
search(Rect r);
    push(stack,[root,root-lsn]);
    return reduceStack(r);

continueSearch(Rect r);
    return reduceStack(r);

reduceStack(Rect r)
    while not empty(stack)
        [p,p-lsn] = pop(stack)

        if (p is pointer to indexed tuple)
            return p
        else
            r-lock(p);
            if p-lsn < LSN(p)
            traverse the rightlink chain starting at rightlink(p)
            to the node with LSN = p-lsn

            for each node n along the rightlink chain:
                r-lock(n)
                push(stack,[n,LSN(n)])
                r-unlock(n)
        endif

        for all entries e of p intersectiong r:
            push(stack,[node-pointer(e),LSN(e)])

        r-unlock(p);
    endwhile
```

Kuva 9: Algoritmi Search

5 Yhteenveto

R-puu on moniulotteisen tai tilassa olevan tiedon tallettamista ja käsittelemistä varten laadittu hakemistorakenne. R-puu muistuttaa rakenteeltaan B-puuta. Huomattavia eroja B-puuhun on kaksi. Ensiksikin avainarvot eivät ole lineaarisesti järjestetty tietyllä tasolla oikealta vasemmalle ja toiseksi avainarvot ovat moniulotteisia

R-puut ja varsinkin R*-puu on tutkimusten perusteella osoittautunut riittävän tehokkaaksi alue- ja pistekyselyihin paikkatiedossa. R*-puu osoittautui testien perusteella tehokkaimmaksi R-puiden variaatioksi.

B-puiden samanaikaisuusalgoritmit on mahdollista toteuttaa myös R-puille ja yksi tällainen algoritmi on Blink-puu, jota vastaava toteutus on Rlink-puu.

Lähteet

[BKS+90] Beckmann, N., Kriegel, H., Schneider, R., Seeger, B., The R*-tree: An efficient and robust access method for points and rectangles. Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, USA, May 23–25, 1990, 322–331.

[Gut84] Guttman, A., R-Trees: A dynamic index structure for spatial searching. Proc. ACM SIGMOD Int. Conf. on Management of Data, Boston, Massachusetts, USA, April 18–21, 1984, 47–57.

[KoB95] Kornacker, M., Banks, D., High-concurrency locking in R-Trees. Proc. of the 21 VLDB Conf, Zurich, Switzerland, Sep 24–28, 1995, 134–145.

[LeY81] Lehman, P., Yao, S., Efficient Locking for Concurrent Operations on B-Trees. ACM Transactions of Database Systems, Volume 6 (4), December 1981, 134–145.

[NgK93] Ng, V., Kameda, T., Concurrent accesses to R-Trees. Advances in Spatial Databases, Singapore, June 23–25, 1993, 142–161.

[PaT94] Papadias, D., Theodoridis, Y., Spatial relations, minimum bounding rectangles and spatial data structures. Technical Report KDBSLAB-TR-94-04.

[Sag86] Sagiv, Y., Concurrent Operations on B*-Trees with Overtaking. Journal of Computer and System Sciences, Vol 33 (2), 1986, 275–296.

[SRF87] Sellis, T., Roussopoulos, N., Faloutsos, C., The R+-tree: A dynamic index for multi-dimensional objects. Proc. of the 13 VLDB Conf, Brighton, England, Sep 1–4, 1987, 507–518.