B^+-puut

Kerttu Pollari-Malmi

Tämä monista on alunperin kirjoitettu syksyn 2005 kurssille osittain Luukkaisen ja Nykäsen vanhojen luentokalvojen pohjalta. Maaliskuussa 2010 pseudokoodiesitys on muutettu vastaamaan Cormenin kolmannen painoksen pseudokoodia.

1 Yleistä B-puusta

Binäärihakupuut toimivat hyvin keskusmuistissa Jos hakemisto on levymuistissa, binäärihakupuu ei kuitenkaan ole hyvä ratkaisu. Tämä johtuu siitä, että tiedon lukeminen levyltä on paljon hitaampaa kuin sen käsitteleminen keskusmuistissa. Lisäksi suuri osa levyhakuun kuluvasta ajasta kuluu oikean levyn kohdan hakemiseen. Kerralla haettava tietomäärä vaikuttaa hakuaikaan vähän, jos haetaan korkeintaan muutamia kilotavuja.

Sopiva hakupuurakenne levymuistiin on *B-puu*. Siinä pyritään puun korkeus ja levyhakujen määrä pitämään pienenä sillä, että jokaisella solmulla voi olla paljon (kymmeniä tai satoja) lapsia. Käytännössä levymuistissa olevan B-puun korkeus on yleensä 2 tai 3.

Tällä kurssilla käsiteltävät B-puun versiot ovat vähän erilaisia kuin Cormenin kirjassa esitetyt, joissa avaimia on tallennettu puun kaikkiin solmuihin. Tällä kurssilla käsitellään B-puun versioita, joissa kaikki rakenteen avaimet (joukon alkiot) ovat lehtisolmuissa. Puun ylemmissä solmuissa (sisäsolmuissa) on *viitta-arvoja*. Ne ovat samantyyppisiä arvoja kuin itse avaimet, mutta eivät itse joukon alkioita. Niitä käytetään vain ohjaamaan avaimen hakua ja muita operaatiota ylempänä puussa.

Tällaisia B-puita kutsutaan B^+puiksi, joissakin lähteissä myös B^*-puiksi. Kurssilla käsitellään niitä Cormenin puiden sijasta, koska

- Käytännössä B^+-puita käytetään paljon useammin kuin “tavallisia” B-puita.
- Operaatiot B^+-puussa ovat vähän yksinkertaisempia kuin “tavallisessa” B-puussa.

2 B^+-puun rakenne

B^+-puu T koostuu solmuista. Yksi solmuista on juurisolmu $T.root$.

Jos solmu x on sisäsolmu, sillä on seuraavat kentät:

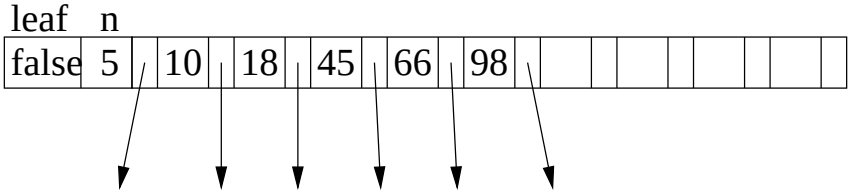
- $x.n$ solmuun tällä hetkellä tallennettujen viitta-arvojen lukumäärä.
- Solmuun tallennetut viitta-arvot järjestyksessä
  $x.router_1 < x.router_2 < \ldots < x.router_{x.n}$
- $x.leaf$, jonka arvo on FALSE kertomassa, että x ei ole lehtisolmu
- $x.n+1$ viitettä $x.c_1, x.c_2, x.c_3, \ldots, x.c_{x.n+1}$ solmun x lapsisolmuihin.

Jos solmu x on lehtisolmu, sillä on seuraavat kentät:

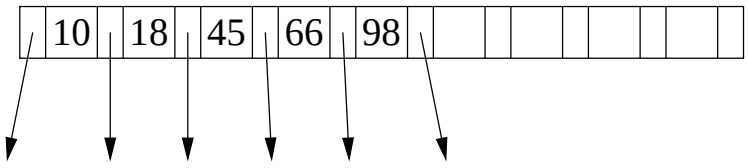
- $x.x$ solmuun tällä hetkellä tallennettujen avainarvojen lukumäärä.
- Solmuun tallennetut avainarvot järjestyksessä
  $x.key_1 < x.key_2 < \ldots < x.key_{x.n}$
- $x.leaf$, jonka arvo on TRUE kertomassa, että x on lehtisolmu

Kun tarkastellaan sisäsolmua x, niin sen viitteille $x.c_1, x.c_2, x.c_3, \ldots, x.c_{x.n+1}$
pätee seuraava ehto:
$k_1 \leq x.router_1 < k_2 \leq x.router_2 < k_3 \ldots < k_{x.n} \leq x.router_{x.n} < k_{x.n+1}$
missä k_i on mikä tahansa viitteen $x.c_i$ päässä olevassa alipuussa oleva avain tai viitta-arvo. Sisäsolmun kaksi peräkkäistä viitta-arvoa antavat siis ala- ja ylärajan sille, kuinka suuria avaimia niiden välissä olevan viitteen päässä olevassa alipuussa voi olla.

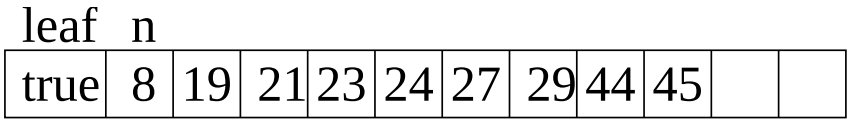
Esimerkki sisäsolmusta, jossa on 5 viitta-arvoa.



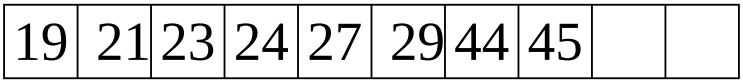
Yleensä solmua x piirrettäessä kenttien $x.n$ ja $x.leaf$ arvoja ei merkitä:



Lehtisolmut sisältävät n ja $leaf$-kenttien lisäksi vain avaimia. (Käytännössä avaimen yhteydessä voi olla alkion muiden kenttien arvoja tai viite paikkaan, johon muiden kenttien arvot on tallennettu. Niitä ei ole mukana kuvissa yksinkertaisuuden vuoksi.)



Yleensä lehtisolmuakaan piirrettäessä kenttien *n* ja *leaf* arvoja ei merkitä:

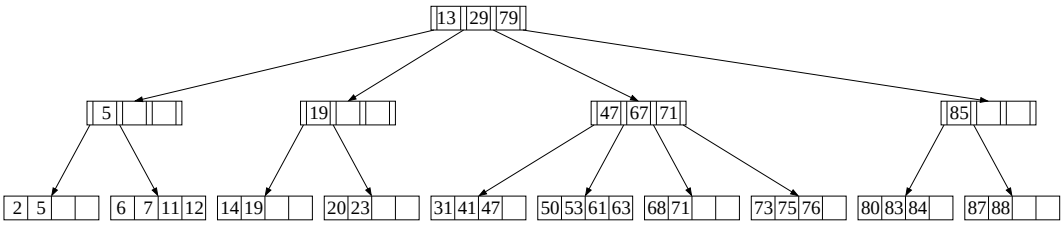


B^+-puun solmun koko valitaan yleensä siten, että solmun koko (tavuina) on käytetyn levymuistin levylohkon koko tai monikerta. Näin kokonainen solmu voidaan hakea levyltä käytännössä lähes samassa ajassa kuin yksi binäärihakupuun solmu (jos se olisi keskusmuistin sijasta levymuistissa).

Tyypillisesti yhdessä solmussa on kymmeniä tai satoja viitta-arvoja tai avaimia. Tällaisen solmun käsittelyyn menee enemmän aikaa kuin yhden pienen solmun käsittelyyn, mutta solmun käsittelyaika on kuitenkin vain pieni osa siitä ajasta, joka kuluu solmun hakemiseen kovalevyltä.

Jos B^+-puussa jokaisella solmulla on 100 lasta ja puun korkeus on 2, mahtuu puuhun miljoona avainta. Mikä tahansa näistä avaimista voidaan saavuttaa sillä, että kovalevyltä haetaan yhteensä 3 solmua.

Esimerkki B^+-puusta (piirustusteknisistä syistä solmun koko on paljon pienempi kuin mitä se on käytännön B^+-puissa):



3 B^+-puun tasapainoehdot

Jos jokaisella B^+-puun solmulla on vain vähän lapsia, B^+-puusta tulee korkea ja kukin operaatio vaatii paljon levyhakuja. Sama ongelma on silloin, jos joku polku B^+-puun juuresta lehteen on paljon pitempi kuin polut juuresta muihin lehtiin.

Käytännössä ei kuitenkaan kannata vaatia sitä, että B^+-puun jokainen solmu olisi aivan täynnä. Tällaisen ehdon ylläpitäminen vaatisi niin paljon ylimääräistä työtä, että operaatioiden tehokkuus kärsisi.

B^+-puun tasapainoehdot ovat seuraavat:

- Jokainen polku juuresta lehteen on yhtä pitkä.
- Jokainen solmu juurisolmua lukuunottamatta on vähintään puolillaan.

Jälkimmäinen tasapainoehto voidaan esittää tarkemmin seuraavasti: Merkitään $s(x)$:llä solmun x lasten lukumäärää, jos x on sisäsolmu, ja solmun x avainten lukumäärää, jos x on lehtisolmu. Olkoon $t \geq 2$ kokonaislukuvakio, joka on määrätty puuta luodessa. Tällöin B^+-puun jokaiselle solmulle pätee

- $1 \leq s(x) \leq 2t$, jos x on puun ainoa solmu.
- $2 \leq s(x) \leq 2t$, jos x on puun juuri ja puussa on muitakin solmuja.
- $t \leq s(x) \leq 2t$, muuten.

Vakio t siis määräytyy sen mukaan, mikä on puun solmun maksimikoko: Solmulla on korkeintaan $2t$ lasta tai avainta ja solmun lasten tai avainten minimimäärä on t, jos solmu ei ole juurisolmu.

Kun B^+-puu toteuttaa nämä tasapainoehdot, voidaan osoittaa, että B^+-puun korkeudelle h pätee

$$h \leq \log_t n$$

kun puussa on n avainta. Tulos saadaan laskemalla solmujen maksimäärä korkeudella 0, mistä saadaan solmujen maksimimäärä korkeudella 1 (koska tiedetään, että jokasella solmulla täytyy olla vähintään t lasta) ja niin edelleen. Puun korkeudelle h pätee, että solmujen maksimimäärä korkeudella h on korkeintaan yksi. Tästä voidaan johtaa ehto puun maksmikorkeudelle.

Siis myös B^+-puu on tasapainoinen. Logaritmin kantalukuna on kuitenkin t eikä 2 kuten binäärihakupuissa.

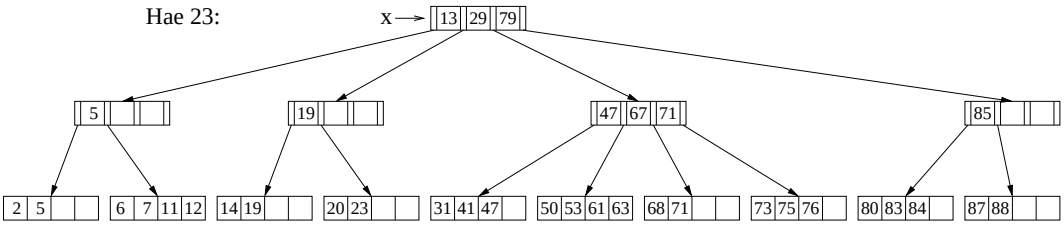
B^+-puuta ei tasapainoteta kierroilla, vaan muuttamalla solmun lasten tai avainten määrää tasapainoehdoissa annetuissa rajoissa. Jos lisäys tai poisto johtaisi tasapainoehdon rikkoutumiseen jossain solmussa, solmu voidaan halkaista tai yhdistää sisarukseen tai solmun ja sen sisaruksen sisältöä voidaan siirtää näiden kahden solmun välillä niin, että tasapainoehdot saadaan voimaan.

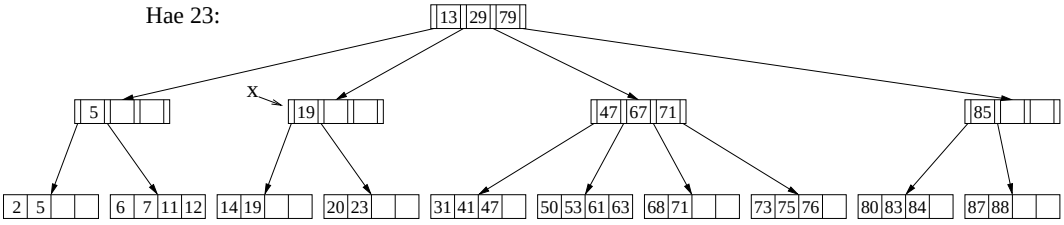
4 Haku B^+-puussa

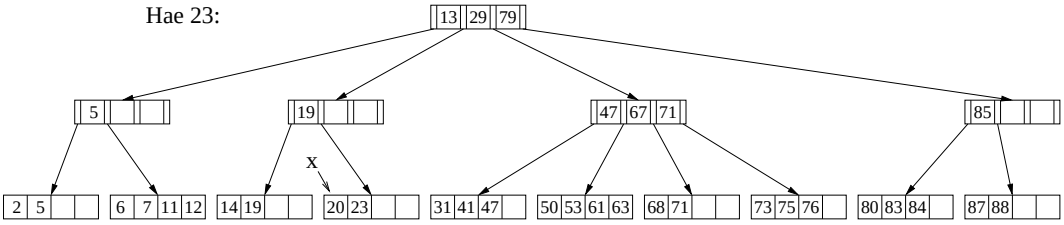
Haku aloittaa juurisolmusta ja etenee jokaisessa solmussa x seuraavasti:

- Jos x ei ole lehtisolmu, etsitään solmun ensimmäinen viitta-arvo $x.router_i$ joka on suurempi tai yhtäsuuri kuin haettava avain k. Tämän jälkeen jatketaan hakua solmusta, johon viite $x.c_i$ viittaa.
- Jos kaikki solmun viitta-arvot ovat pienempiä kuin haettava avain k, jatketaan hakua solmun viimeisen viitteen päässä olevasta solmusta.
- Jos x on lehtisolmu, tutkitaan, onko haettava avain tässä lehdessä.

Esimerkki hakuoperaatiosta (allekkain olevat kuvat kuvaavat tilannetta haun eri vaiheissa)







BTreeSearch(T,k)
1 x = T.root
2 **while not** x.leaf
3 i = 1
4 **while** $i \leq x.n$ **and** $k > x.router_i$
5 i = i+1
6 $x = x.c_i$
7 DiskRead(x)
8 i = 1 // ollaan lehtisolmussa
9 **while** $i \leq x.n$ **and** $k > x.key_i$
10 i = i+1
11 **if** $i \leq x.n$ **and** $k = x.key_i$
12 **return** (x, i)
13 **else return** NIL

B^+-puualgoritmeissa kannattaa merkitä levylohkojen luvut keskusmuistiin ja kirjoitukset levylle, koska ne vievät yleensä paljon enemmän aikaa kuin kaikki algoritmin keskusmuistissa suoritettavat operaatiot. Yllä esitetyssä hakualgoritmissa on oletettu, että B^+-puun juuri on valmiiksi keskusmuistissa (näin on usein käytännössä). Jos juurisolmu ei ole keskusmuistissa, pitää algoritmin alkuun lisätä sen lukeminen levyltä.

4.1 Hakuoperaation aikavaatimus

Hakuoperaation aikana luetaan levyltä keskusmuistiin h solmua, missä h on puun korkeus. Aikaisemmin todettiin, että B^+-puilla $h = O(\log_t n)$, missä n on puuhun tallennettujen avaimien määrä. Jokaisessa solmussa suoritetaan keskusmuistissa peräkkäishaku, joka vaatii ajan $O(t)$. Tällöin hakuoperaation kokonaisaikavaatimus on $O(t \log_t n)$.

Jos solmussa suoritetaan peräkkäishaun sijaan binäärihaku, aikavaatimus saadaan hieman paremmaksi $O(\log_2 t \log_t n)$. Käytännössä kuitenkin levyhakuihin kuluva aika määrää haun suoritusajan, sillä levyhakuun kuluva aika on selvästi suurempi kuin yhdessä solmussa keskusmuistissa suoritettavaan peräkkäishakuun kuluva aika. Usein B^+-puuoperaatioiden tehokkuudesta puhuttaessa lasketaankin vain levyoperaatioiden määrää, eikä puhuta varsinaisesta aikavaatimuksesta.

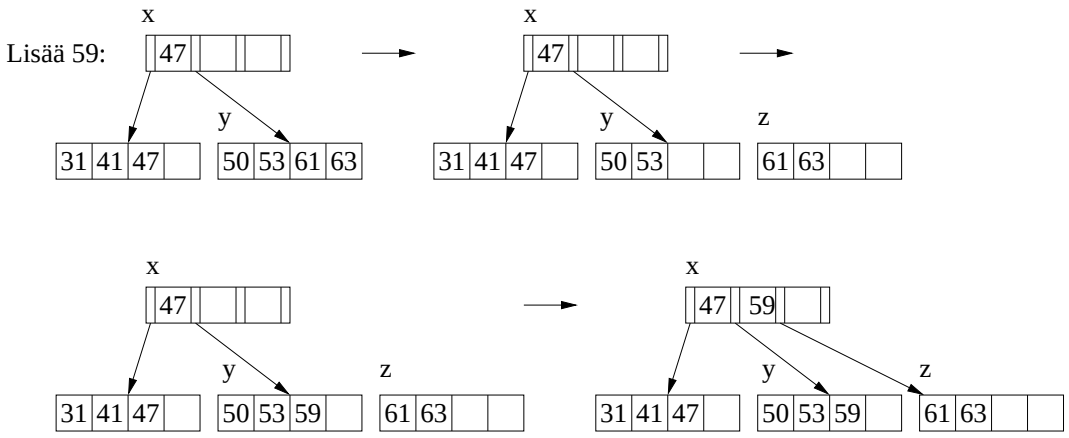
5 Lisäys B^+-puuhun

Avaimen lisäys B^+-puuhun aloitetaan etsimällä se lehtisolmu y, johon lisättävä avain kuuluu. Tämä tehdään kuten hakuoperaatio. Jos löydetyssä lehtisolmussa y on tilaa, avain lisätään solmuun eikä muita toimenpiteitä tarvita.

Jos solmussa y ei ole tilaa lisättävälle avaimelle, se pitää halkaista:

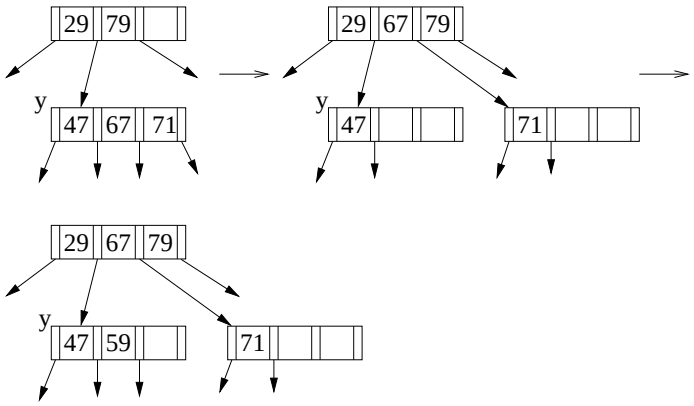
- Varataan tilaa uudelle solmulle z.
- Solmun y ensimmäiset t avainta jätetään solmuun y.
- Solmun y viimeiset t avainta siirretään uuteen solmuun z.
- Lisättävä avain lisätään joko solmuun y tai z. Solmu määräytyy sen mukaan, kumpaan solmuun uusi avain kuuluu suuruusjärjestyksen mukaisesti.
- Jotta myöhemmät operaatiot löytäisivät solmuun z siirretyt avaimet, pitää y:n ja z:n yhteiseen vanhempaan x lisätä viite z:taan ja sopiva viitta-arvo y:n ja z:n viitteiden väliin. Viitta-arvoksi sopii suurin solmun y avainarvoista.
- Jos solmuun x ei mahdu uutta viitettä ja viitta-arvoa, pitää sekin halkaista. Pahimmillaan halkaisut jatkuvat juurisolmuun saakka.

Esimerkki lehtisolmun halkaisusta:



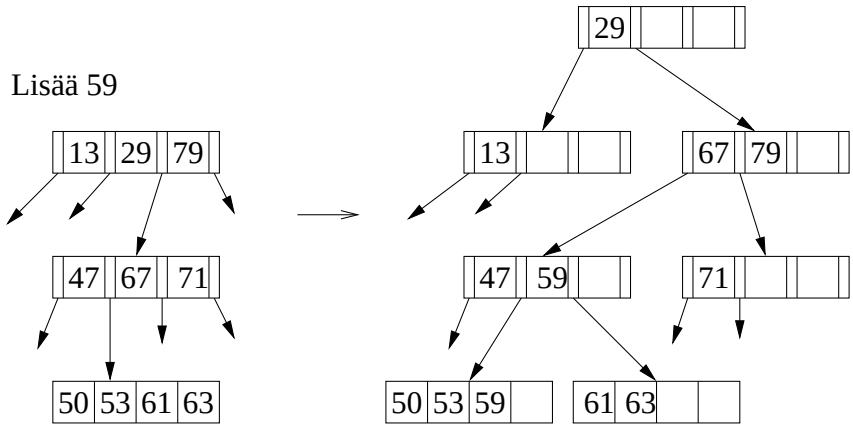
Sisäsolmussa on yksi viitta-arvo vähemmän kuin viitteitä. Sen vuoksi sisäsolmun halkaisussa halkaistavan solmun keskimmäiselle viitta-arvolle ei ole paikkaa kummassakaan solmussa. Se siirtyy halkaistavan sisäsolmun vanhempaan erottamaan halkaisun tuloksena syntyneitä kahta solmua.

Alla esimerkki tilanteesta, jossa solmuun *y* halutaan sijoittaa yksi uusi viite ja viitta-arvo 59. (Viitta-arvo 59 lisätään halkaisun tuloksena syntyneistä solmuista vasempaan, koska 59 on suuruusjärjestyksessä ennen halkaisukohdassa ollutta viitta-arvoa 67. Jos täyteen solmuun olisi lisätty suurempi viitta-arvo kuin 67, olisi se lisätty oikeanpuoleiseen solmuun.)



Jos halkaistavan sisäsolmun vanhempi on täynnä, pitää sekin halkaista, jotta siihen mahtuisi uusi viite ja viitta-arvo. Pahimmillaan halkaisut voi jatkua juureen asti.

Alla esimerkki tilanteesta, jossa arvon 59 lisääminen lehteen saa aikaan kolmen solmun halkaisun. (Kuvassa on vain osa B^+-puusta. Jokaisen viitteen päässä on joku solmu, jolla voi olla myös lapsia niin, että polku juuresta kaikkiin B^+-puun lehtisolmuihin on yhtä pitkä.)



Lisäysalgoritmin ajantarve on $O(t \cdot log_t n)$, missä kerroin t tulee avainten ja viitteiden tutkimisesta siirtelystä keskusmuistissa. Käytännössä suoritusajan määrää kuitenkin levyhakujen määrä, joka on $O(log_t n)$

Esitetyssä algoritmissa tehdään ensin lisäys. Sen jälkeen mennään takaisin ylöspäin ja tehdään tarvittavat solmujen halkaisut. Voitaisiin menetellä myös päinvastoin: jo matkalla juuresta lehteen halkaistaan varmuuden vuoksi kohdatut täydet solmut. Tällöin uusi avain mahtuu aina lehteen, eikä enää tarvitse palata taaksepäin. Näin polku juuresta lehtisolmuun pitää kulkea vain yhteen kertaan, mutta toisaalta solmujen halkaisuja voidaan joutua tekemään turhaan.

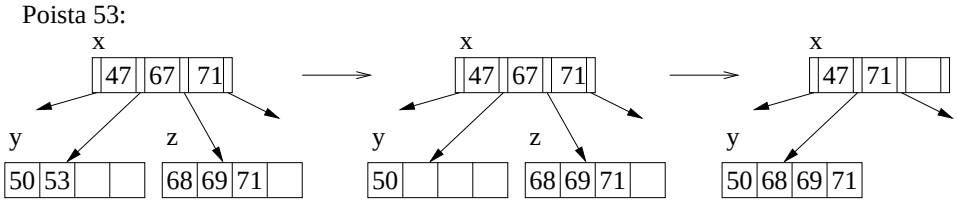
6 Avaimen poisto

Avaimen poisto aloitetaan hakemalla se lehtisolmu, joka sisältää poistettavan avaimen. Vaikka sama arvo olisi myös viitta-arvona ylempänä puussa, riittää lehdessä olevan avaimen poisto. Ylempänä olevaa viitta-arvoa tarvitaan poiston jälkeenkin ohjaamaan poistettavan avaimen kanssa samassa lehtisolmussa tai alipuussa sijaitseviin avaimiin kohdistuvia hakuja ja muita operaatioita.

Koska yksi solmu sisältää useita avaimia, ei itse solmua tarvitse poistaa. Avain vain poistetaan solmusta. Tarvittaessa solmussa olevia muita avaimia siirretään solmun sisällä niin, että ne ovat suuruusjärjestyksen mukaisilla paikoilla poiston jälkeenkin.

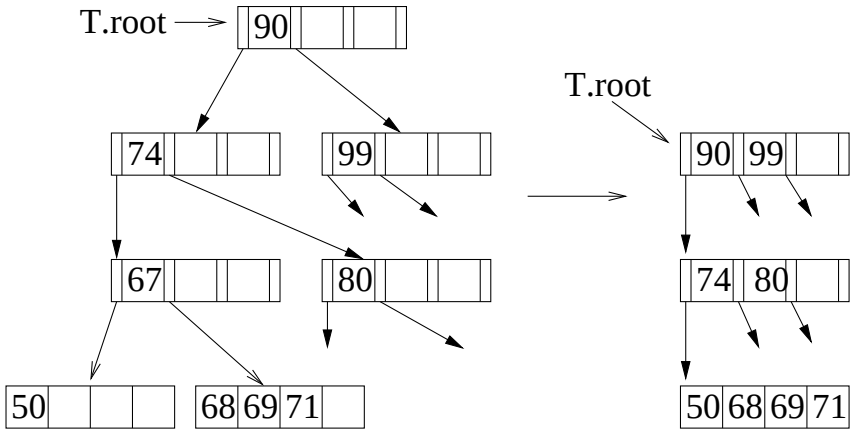
Jos lehtisolmuun jää poiston jälkeen vähintään t avainta, ei muita toimenpiteitä tarvita. Jos avaimien määrä solmussa olisi poiston jälkeen $t-1$, suoritetaan tasapainotus.

Tasapainotukseen otetaan mukaan liian vajaaksi jääneen solmun y sisarus z. Solmussa y on siis $t-1$ avainta. Jos solmussa z on korkeintaan $t+1$ avainta, mahtuvat y:n ja z:n avaimet yhteen solmuun. Tällöin solmun z avaimet siirretään sen sisarukseen y. Samalla poistetaan solmujen yhteisestä vanhemmasta solmuun z viittaava viite sekä y:n ja z:n viitteiden välissä oleva viitta-arvo.

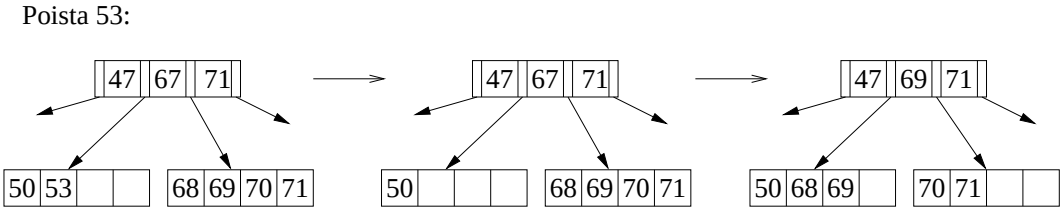


Jos vanhemmalle x jää alle t lasta solmujen y ja z yhdistämisen jälkeen, pitää tasapainotusta jatkaa solmusta x, joka on siis sisäsolmu. *Jos sisäsolmu yhdistetään sisaruksensa kanssa, niin näitä kahta solmua vanhemmassa erottava viitta-arvo "putoaa" yhdistettyyn solmuun.* Pahimmassa tapauksessa yhdistämiset jatkuvat juureen saakka. Jos juurelle jää vain yksi lapsi, niin se poistetaan. Seuraavassa esimerkissä lähdetään suorittamaan tasapainotusta sen jälkeen, kun kuvassa vasemmalla olevasta lehdestä on poistettu yksi avain. Kuvassa on vain osa B$^+$-puusta. Jokaisen viitteen päässä on joku solmu, jolla voi olla myös lapsia niin, että polku juuresta kaikkiin B$^+$-puun lehtisolmuihin on yhtä pitkä.

-

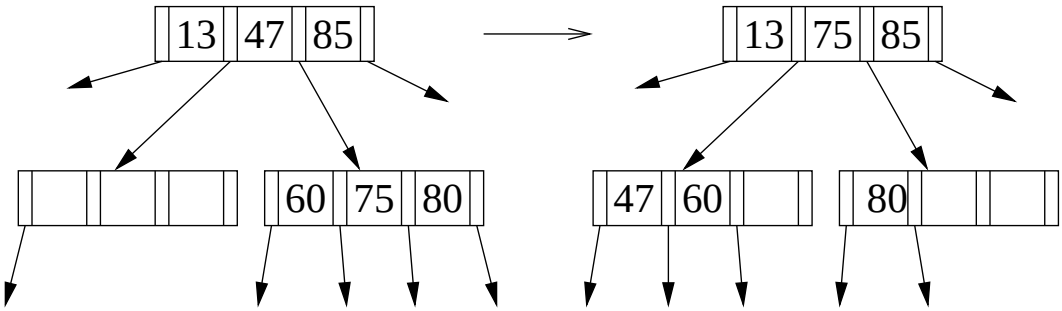


Jos poiston jälkeen liian vajaaksi jääneessä lehtisolmussa y on $t-1$ avainta ja sen sisaruksessa z vähintään $t+2$ avainta, ei solmuja z ja y voi yhdistää, koska niiden avaimet eivät mahdu samaan solmuun. Tällöin solmujen sisältö jaetaan uudelleen näiden kahden solmun kesken niin, että kumpaankin solmuun tulee lähes yhtä monta avainta:



Koska vanhemman lasten määrä ei muuttunut, tasapainotus on valmis eikä sitä tarvitse enää jatkaa ylöspäin. Vanhemmassa pitää kuitenkin päivittää solmujen z ja y viitteiden välissä olevaa viitta-arvoa.

Myös sisäsolmujen sisältöjä voidaan joutua jakamaan solmujen välillä uudelleen, jos solmujen yhdistäminen alempana puussa on saanut aikaan sen, että sisäsolmulla on $t-1$ lasta ja sen sisaruksella on vähintään $t+2$ lasta. Tällöin *solmuja erottava viitta-arvo sisarusten vanhemmassa putoaa toiseen käsiteltävistä solmuista ja uudessa jakokohdassa oleva viitta-arvo nousee vanhempaan*. Seuraavassa esimerkissä vasemmanpuolimmaiselle sisäsolmulle on jäänyt vain yksi lapsi alempana tapahtuneen solmujen yhdistämisen takia:



Poiston aikavaatimus on sama kuin lisäyksellä:

- Kokonaisaikavaatimus on $O(t \cdot \log_t n)$
- Ajantarpeen käytännössä ratkaiseva levyhakujen määrä on $O(\log_t n)$.

Myös poisto-operaatioiden yhteydessä tasapainotus on mahdollista yhdistää hakuvaiheeseen: matkalla juuresta lehteen kohdatut puolillaan olevat solmut tasapainotetaan sisaruksensa kanssa.