3.3 Dynaamiset hakemistorakenteet

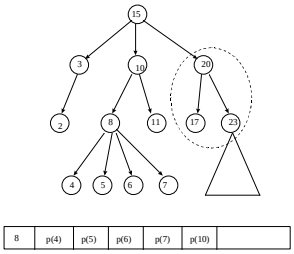
Käsitellyt hakemistot (hajautus, ISAM):
hakemisto-osa on staattinen eli ei muutu muuten kuin uudelleenorganisoinnissa.

Ajan mittaan epätasapainoa:
- hajautuksessa solujen ylivuotoketjut
- ISAM: perusjaksojen ylivuotoketjut

Dynaamiset rakenteet:
- rakennetta ylläpidetään vähittäisin muutoksin jatkuvasti tasapainossa
- lisäys- ja poisto-operaatiot tulevat vain jonkin verran monimutkaisemmiksi, laaja uudelleenorganisointi tarpeettomaksi
- jaksoja ei voida pitää aina täynnä → tilaa kuluu jonkin verran enemmän

B-puu

Johdannoksi puurakenne yleensä:
- juuri, lapset, vanhemmat
- lehtisolmut, sisäsolmut, alipuu
- solmun asteluku: binääripuu, yleinen puu
- solmun taso, puun korkeus



- solmun rakenne:
  - sisältö, esim. hakuavain
  - linkit lapsiin (ehkä myös isään)

Puurakenteen käyttö: esim. hakupuu
- solmut ohjaavat hakua seuraaville tasoille ja edelleen lehtisolmuihin asti
  vrt. ISAM-rakenne: hakemistojakso solmuna

Yleisen hakupuun solmu:
joukko osoittimia ja hakuavaimen arvoja

< P1, K1, P2, ... , K(i-1), Pi, Ki, ... , K(q-1), Pq >

järjestys: K1 < K2 < ... < K(q-1)

Pi osoittaa alipuuhun, jonka solmuille X pätee:
K(i-1) < X < Ki (i > 1, i < q)
X < K1 ; X > K(q-1) (reunimmaiset)

Kun solmu on levyjakso, q voi olla kohtuullisen suuri
=> puun korkeus pysyy "kurissa"
(logaritmisuus, vrt. ISAM-laskelmat)

Solmussa voi olla

- koko tietue (avain + muut kentät), tai
- vain avain K => tarvitaan osoitin tietueeseen (dataan)

Hakupuu ei ole välttämättä tasapainossa; voi olla pieniä alipuita ja pitkiä hakuketjuja
- puun muodostus lisäyksien sarjana?
- luonnin jälkeiset lisäys- ja poisto-operaatiot

Tasapainoinen hakupuu saadaan esim. ehdoilla, jotka määrittelevät B-puun (kertaluokan p B-puu):

1. B-puun sisäsolmu on muotoa

<P1, (K1,D1), P2, ... , (K(i-1),D(i-1)), Pi, (Ki,Di), ... , (K(q-1)D(q-1)), Pq>,

missä Di on osoitin avaimen Ki tietueeseen ja
Pi on osoitin puun seuraavan tason solmuun

Di voi olla jakso- tai tietueosoite.

2. K1 < K2 < ... < K(q-1).

3. Pi:n osoittamassa alipuussa olevat solmut ovat avainarvoltaan K(i-1):n ja Ki:n välissä (vrt. hakupuu).

4. Jokaisessa solmussa on enintään p osoitinta (rakenneosoitinta puuhun).

5. Jokaisessa sisäsolmussa (paitsi juuressa) on vähintään $\lceil (p/2) \rceil$ osoitinta.
Juuressa on vähintään kaksi osoitinta, paitsi jos se on B-puun ainoa solmu.

6. Kohdan 1 mukaisessa solmussa on q puuosoitinta ja q-1 avainta ja dataosoitinta (q <= p).

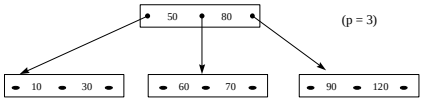
7. Kaikki lehtisolmut ovat samalla tasolla. Lehtisolmun rakenne on muuten sama kuin sisäsolmun, mutta puuosoittimet ovat tyhjiä.

Ominaisuudet 4 ja 5 takaavat B-puun tasapainon. Lehtien samantasoisuus seuraa puun konstruktiosta.

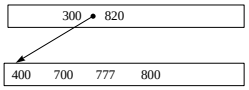
Avainarvon X haku B-puusta:
- haetaan avainta X juuresta
- jos ei löydy, seurataan sitä puuosoitinta, jolle pätee
  K(i-1) < X < Ki
  ja haetaan avainta X osoitetusta alipuusta (solmusta)
  (vrt. ISAM-hakemiston käyttö)

Lisäys ja poisto ovat monimutkaisempia, toteuttavat dynaamisuuden.

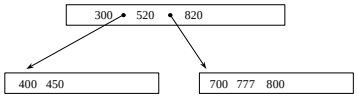
(Toisin kuin ISAM-rakenne, B-puu muodostetaan peräkkäisin lisäyksin; ei erillistä luontivaihetta eikä uudelleenorganisointia.)



Lisäys B-puuhun: (p = 6)



- lisätään 520: mahtuu solmuun
- lisätään 450: ei mahdu



Lisäyksen jälkeen tilaa on monelle uudelle lisäykselle!

Lisäys B-puuhun:
- ensimmäinen lisäys tyhjään juurisolmuun (muita ei ole)
- p-1 lisäyksen jälkeen juureen ei voida enää lisätä
  (ehto 4: enintään p osoitinta => p-1 avainta)
- tällöin tarkastellaan p avaimen joukkoa (juuren solmut +lisättävä):
  - jätetään vain järjestyksessä keskimmäinen avain juureen ja jaetaan muut kahteen osaan: kahdeksi tason 1 solmuksi (uusia solmuja, joille pätee ehto 5)
  - asetetaan osoittimet vastaamaan jakoa

- jaettaessa muu kuin juurisolmu menetellään kuten edellä sijoittamalla kuitenkin puolet solmuista uuteen solmuun samalla tasolla ja viemällä keskimmäinen solmu jaettavan solmun isäsolmuun

- jos isäsolmussa ei ole tilaa, toistetaan menettely tälle jne (periaatteessa juureen asti)

- Jos jaot etenevät juureen asti ja jos juuri on täynnä, juuri jaetaan kahtia (ja perustetaan uusi juuri, joka jää tässä vaiheessa hyvin vajaaksi!). Tässä tilanteessa puun korkeus kasvaa yhdellä.

Poisto B-puusta:

- poisto on yksinkertainen solmunsisäinen operaatio, mikäli poisto ei riko ehtoa 5 (solmu tulisi liian tyhjäksi).

- jos poiston jälkeen solmussa olisi alle p/2 avainta, siihen siirretään vähintään yksi avain viereisestä (veljes)solmusta; jos avaimia on yhteensä liian vähän kahteen solmuun, ne yhdistetään yhdeksi solmuksi

- solmujen yhdistäminen poistaa yhden osoittimen ja avaimen solmujen isäsolmusta; jos siinä olisi nyt liian vähän osoittimia, jatketaan kuten edellä (yhdistäminen voi edetä pahimmillaan juureen asti).

Solmun avainlukumäärän ylä- ja alaraja takaavat sen, että jako ja yhdistäminen eivät toistu usein: yhden jaon tai yhdistämisen jälkeen ei tarvita uutta samaa operaatiota ‘pitkään aikaan’ (samassa kohdassa B-puuta).

Tasapainotilassa (satunnaiset lisäykset ja poistot) solmut ovat keskimäärin 69 %:sti täynnä.

Esimerkki. Olkoon K = 10 tavua, levyjakso 4 KB, puuosoitin (jakso-osoite) 10 B ja dataosoitin (tietueosoite) 15 B.

B-puun solmu (enintään p osoitinta):

10p + (10+15) * (p-1) <= 4096 eli p <= 117:

solmussa on avaimia 58 -116 kpl.

Em. 0.69-tuloksen mukaan osoittimia on noin 80 kpl.

=> avainten määrät eri tasoilla:

juuri	79	(osoittimia 80)
taso 1	6320	6400
taso 2	505600	512000
yhteensä	n. 512000	

Solmun sisällä avaimet ovat järjestyksessä, samoin solmut 'löydetään' järjestykseen perustuen.

Haku solmun sisältä ei vaadi (uusia) levyhakuja.
- erilaisia toteutustapoja (solmurakenteita)
-- millaisia? –

(esim. q = 200-400 aika suuri peräkkäishakuun ...)

Kaikkien avainten sujuva läpikäynti järjestyksessä vaatii useiden jaksojen pitämisen puskurissa (yksi jakso / B-puun taso).

B+ -puu

= B-puun variantti, joka sopii paremmin hakemistoksi:

- kaikki dataosoittimet sijoitetaan B+ -puun lehtitasolle
- sisäsolmuissa säilytetään avainarvoja, jotka ohjaavat hakua (lehtitasolle etenemistä) kuten B-puussa
- lehtisolmut muodostavat tiheän hakemiston perustiedostoon
- lehtisolmut voidaan ketjuttaa => koko tiedoston läpikäynti järjestyksessä on tehokasta

Solmurakenne:

- sisäsolmu:
< P1, K1, P2, ... , K(i-1), Pi, Ki, ... , K(q-1), Pq >

- lehtisolmu:
<K1, D1, K2, D2, ... , K(q-1), D(q-1), P_{next}>

B+ -puun rakenteen ehdot sekä lisäys-ja poisto-operaatiot ovat käytännössä hyvin samanlaiset kuin B-puun (1-7 edellä).

Ehdossa 3 alipuun avaimet voivat olla yhtäsuuria kuin (isä)solmun avainarvo:

K(i-1) < X <= Ki isäsolmussa avainarvo toistuu, "viittana" (= suurin avainarvo)

Sisäsolmujen avainten ei hakuperiaatteen kannalta tarvitse olla lehtisolmuissa esiintyviä ('sopivat' avainarvot ISAM-rakenteen tapaan). Muodostettaessa B+ -puu peräkkäisillä lisäysoperaatioilla kopioidaan tason I solmujen rajakohtia osoittaviksi avaimiksi kuitenkin oikeita avainarvoja, jotka myös säilyvät poistoissa.

Esimerkki solmujen koosta:
Olkoon K 10 tavua, levyjakso 4 KB, puuosoitin (jakso-osoite) 10 B ja dataosoitin (tietueosoite) 15 B (kuten edellisessä esimerkissä).

Sisäsolmu: 10p + 10(p-1) <= 4096 => p <= 205

Lehtisolmu: (15+10)p + 10 <= 4096 => p <= 163

Täyttöasteen 0.69 mukaan sisäsolmussa on keskimäärin 141 ja lehtisolmussa 112 osoitinta.
B+ -puun rakenne muodostuu seuraavasti:

juuri	140	avainta	141	osoitinta
taso 1	19740	"	19881	"
taso 2	2783340	"	2803221	"

lehtitaso 2803221 * 112 = 3.1 * 10^8 tietueosoitinta

(2-tasoisessa hakemistossa 19881 * 112 = 2226672 tietueosoitinta)

Tasojen lukumäärä pysyy siis hyvin pienenä!

B+ -puu muodostaa luontevan tiheän hakemiston perustiedostolle, joka siis voi olla järjestämätön (kasa).

Esim. Oraclen hakemistot:

- perustiedosto on rakenteeltaan kasa:
tietueet voidaan sijoittaa jaksoihin tiiviisti (100 % täyttöaste), niitä ei tarvitse siirtää
(voi olla myös ryväs, jossa tietueita monesta relaatiosta)
- hakemistot B+ -puun muotoisia oheishakemistoja (optimoituja)

- hakemisto luodaan create index -lauseella:

```
create unique index nimihak on employee
(column lname asc, fname asc);
```

- unique valinnainen, asc vastaa oletusarvoa (desc: laskeva järjestys, ei merkitystä):
- unique takaa rivien yksikäsitteisyyden
- tehokkuussyistä perusavaimelle suositellaan eheysrajoitteen (primary key ... tai unique) käyttöä, jolloin järjestelmä tekee hakemiston

- create index -lauseen tarkennuksia (mm.):
muistialue, varausyksiköt
pctfree=30 (jakson varatila),
nosort: perustiedosto järjestyksessä

Harva vai tiheä hakemisto?

- perustiedosto järjestyksessä vai kasa?
- harvassa yksi hakemistotaso vähemmän kuin tiheässä
- tiheän hakemiston tietueet yleensä selvästi perustietueita lyhyempiä
  (=> voi kompensoida korkeuseron)

Oheishakemistojen merkitys levyhakujen kannalta:

Oletetaan, että kasarakenteeseen on luotu 10 ISAM-rakenteista oheishakemistoa, ja jokaisen korkeus on 3.

Operaatio insert into R values (…)
vaatii kaikkiaan
2 + 10 * (2 + 1 + 1) = 42 levyhakua, jos mikään hakemistojaksoista ei ole puskurissa.
Saantiajalla 10 ms aikaa kuluu siis noin 0.4 s.

B+ -puuhakemistoissa ylempien hakemistotasojen päivitys lisää joissakin tapauksissa (harvoin) muutamia levyhakuja. Tasojen määrään vaikuttaa mm. tietuerakenne.

Muita toimintoja:
- muodostetaan lokitietueita elvytystä varten, niiden kirjoitus levylle
- viite-eheyksien tarkistukset (mahdollisesti paljon levyhakuja)

Huom. Hakemistojen sijoittelu levylle on yleisesti tärkeä asia: lähelle perustiedostoa (esim. samalle sylinterille?) vai ei. Esim. 1+1 levyhaun tilanteet (jos hakemisto kaukana, kumpikin levyhaku hidas).

Hakemisto luodaan yleensä vasta tiedoston luonnin (taulun sisällön lataamisen) jälkeen.
- jos luodaan peräkkäisillä lisäyksillä, kaikki lisäykset vaativat hakemiston päivittämisen (ei harva ISAM)
- optimointimahdollisuudet epätasapainon estämiseksi

Taulun koko: aivan pienille tauluille ei kannata tehdä hakemistoa - käytöstä riippumatta.
raja ? (noin 8 jaksoa ? 200-300 tietuetta ?)

Liitosoperaatio tehostuu hakemistoa käytettäessä: tiheä hakemisto (ilman perustietueita) riittää jopa liitosehdon testaamiseen (perustietueita hakematta).

Eri relaatioiden yhdistäminen samaan tiedostoon?

E&N: ‘file of mixed records’

Tarve?
Eri relaatioiden rivien käyttö yhdessä, esimerkiksi osasto ja osaston työntekijät

Tekniikka: fyysinen ryvästäminen (‘klusterointi’):
osaston rivi ja vastaavan osaston tietueet sijoitetaan lähekkäin (samaan jaksoon, ainakin samalle sylinterille)

- monimutkaistaa tiedoston rakennetta: tarvitaan tietuetyyppi; tilan allokointi (eripituisia)

(Ryvästäminen ei ole oikein relaatiomallin henkeen kuuluvaa; paremmin hierarkkiseen tai verkkomalliin sekä oliotietokantoihin.)

Oraclessa:

```
create cluster personnel
      (dept_number number(2))
 create table emp
      ( …
        deptno number(2) not null)
      cluster personnel(deptno)

create table dept
      (deptno number(2),    …)
      cluster personnel(deptno)
```


- ryväs voi olla indeksoitu tai hajautettu; indeksoidussa rypäässä kaikki samalla ryväsavaimella varustetut rivit ovat peräkkäin:

```
(d)  11   Administration …
(e)  12345   Clark     …   11
(e)  23456   Jones     …   11
(e)        . . .           11
(d)  13   Management  …
(e)  88888   Hill      …   13
(e)  99999   Adams     …   13
           . . .
```

Osastonumero voidaan tallentaa vain jaksokohtaisesti (eli jättää pois tietueista) ym. optimointia.

- ryväs on SQL-kyselyjen kannalta ‘läpinäkyvä’, järjestelmä hoitaa sisäisesti.
- säästää tilaa
- hidastaa peräkkäiskäsittelyä (osastot tai työntekijät), lisäyksiä, ryväsavaimen päivityksiä

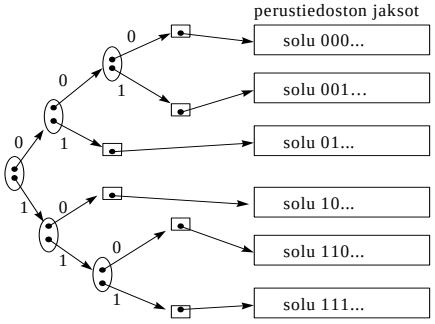
Dynaamiset hajautusmenetelmät

Tavoite: välttää soluun (kotiosoitteeseen) liittyvä tietueiden kasautuminen (ylivuotojaksojen ketjuuntuminen).

Erilaisia menetelmiä: dynaaminen hajautus, laajentuva hajautus, lineaarinen hajautus (ei käsitellä).

Dynaaminen hajautus: binääripuun muotoinen hakemisto, jonka jokaisessa lehtisolmussa on osoitin yhteen soluun.

Hajautusfunktion arvoa h(v) tarkastellaan binääriesityksensä mukaan (bittijonona) ensimmäisestä bitistä alkaen:
0-bitti osoittaa vasempaan alipuuhun, 1-bitti oikeaan



Hakemiston dynaamisuus:

Kun solun jakso tulee täyteen, se jaetaan kahdeksi jaksoksi, jaetaan solun tietueet hajautusarvon seuraavan bitin mukaan kahteen jaksoon ja lisätään hakemistotietue (osoittimet vanhaan ja uuteen jaksoon).

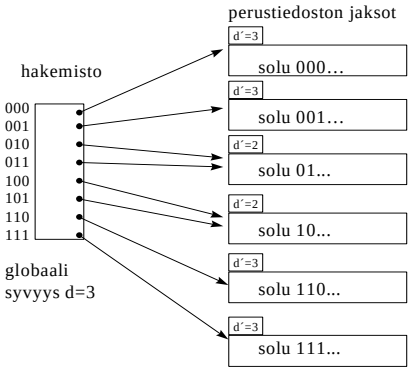
Kun solu (esim. xyz0…) tyhjenee, se poistetaan, sisarussolun (solun xyz1…) hakemistopolulta jätetään yksi hakemistotietue pois ja liitetään sisarussolu edelliseen hakemistotietueeseen (xyz).

Hienojakoisempikin dynaamisuus on mahdollista: puolityhjä solu voidaan yhdistää sisarussoluunsa, jos tila riittää (täyttösuhteelle alaraja) jne.

Hakemistotietueet ovat tässä varsin pieniä, bittitason käsittelyn takia niitä (ja vastaavia tasoja) on tietysti enemmän kuin kokonaisia avainarvoja käsiteltäessä. Hakemisto voidaan kuitenkin usein pitää keskusmuistissa tai vain muutamassa levyjaksossa.

Laajentuva hajautus (extendible hashing)

Ylläpidetään hakemistoa, jossa on 2^d alkiota (hakemistotietuetta). Parametri d on hakemiston globaali syvyys.
Jokaisen solun (perustietueiden jakson) lokaali syvyys d´ kertoo, moneenko hajautusavaimen bittiin solun osoitus perustuu.



Normaalitapauksessa 2 levyhakua: yksi hakemistoon ja yksi soluun (tai hakemisto keskusmuistissa).

Dynaamisuus:

- ylivuoto solussa, jolle d´<d : otetaan käyttöön uusi perustiedoston jakso ja päivitetään uuden jakson osoite hakemistossa (osoitteen paikka on olemassa)
- ylivuoto solussa, jonka d´=d : uuden perusjakson lisäksi kaksinkertaistetaan hakemisto eli asetetaan d ← d+1 (esimerkissä esim. solu 110, hakemisto-osoitteet 1110 ja 1111 käyttöön)

- poisto analogisesti: yhdistetään tyhjät solut ja päivitetään osoitteet; hakemisto voidaan poiston yhteydessä puolittaa, jos kaikkien solujen d´<d (missään ei tarvita hajautusarvon viimeistä bittiä)