Indeksin luonti ja hävitys

- TKHJ:ssä on yleensä komento create index, jolla taululle voidaan luoda hakemisto
  - Komentoa ei ole standardoitu ja niinpä sen muoto vaihtelee järjestelmäkohtaisesti
- Indeksi voidaan luoda milloin tahansa taulun luonnin jälkeen
  - siis tyhjälle taululle tai
  - olemassa olevalle taululle
    - jos tauluun ladataan suuri määrä dataa voi olla kannattavaa luoda indeksit vasta latauksen jälkeen – saadaan aikaan parempi rakenne eikä jouduta jatkuviin rakenteen tasapainotuksiin
    - Tietueet järjestetään ensin ja lisätään järjestyksessä

1

Indeksin luonti ja hävitys

- Indeksit voidaan myös 'tiputtaa' pois drop index komennolla
  - halutaan uudelleen organisoida indeksi
    - tiputetaan se pois ja luodaan uudelleen
  - suoritetaan iso päivityserä (lisäyksiä/poistoja/indeksointiavaimen muutoksia)
    - on yleensä nopeampaa ja indeksien rakenteen kannalta parempi tiputtaa indeksit ja luoda ne uudelleen erän jälkeen

2

Indeksin luonti ja hävitys

- Pelkästään Create index komennolla voi luoda lähinnä tiheitä oheishakemistoja.
- Harvat hakemistot edellyttävät lisäksi taulumäärittelyn yhteydessä annettuja lisämääreitä. Harvojen hakemistojen tiputtaminen ja uudelleen perustaminen ei ole yhtä yksinkertaista kuin tiheiden hakemistojen käsittely.

3

Dynaamiset hajautusratkaisut

- Aiemmin käsitelty hajautusrakenne perustui siihen, että tietueelle laskettiin soluosoite hajautusavaimen perusteella.
- Hajautusfunktio oli kiinteä ja sen arvoalue (osoiteavaruus) piti kiinnittää funktiota määriteltäessä.
  - Tällöin ei välttämättä ole riittävää tietoa siitä, miten hajautusfunktio jakaa tietueet todellisessa käyttötilanteessa eikä välttämättä edes tietoa siitä, miten nopeasti tietueiden määrä kasvaa. Solua laajentavan ylivuotoketjun pituutta ei voida hallita ja ainakin alussa täytyy varata tilaa paljon yli todellisen tarpeen.

4

Dynaamiset hajautusratkaisut

- B+ -puussa hakupolun pituus pidetään hallinnassa puolittamalla sivu ja pitämällä kummatkin puolikkaat samanmittaisen hakupolun päässä.
- Sivun puolituksen ideaa sovelletaan myös dynaamisissa hajautusrakenteissa. Tunnetuimpia näistä ovat
  - laajeneva hajautus (extendible hashing) ja
  - lineaarinen hajautus (linear hashing)

5

Laajeneva hajautus

- Laajenevassa hajautuksessa on käytössä kiinteä hajautusfunktio h ja sillä kiinteä osoiteavaruus
  - osoiteavaruutta ei kuitenkaan suoraan vastaa mikään tilavaraus, joten se voi olla hyvinkin suuri
  - tilavarauksessa lähdetään liikkeelle pienestä varauksesta ja sitä kasvatetaan tarpeen mukaan
- Rakenteessa käytetään soluhakemistoa, jonka koko on 2^d soluosoitetta. Eksponentti d (≥ 1) kasvaa tiedoston kasvaessa. Sitä kutsutaan rakenteen globaaliksi syvyydeksi (global depth).

6

Laajeneva hajautus

- Soluhakemiston koko on siis 2, 4, 8, …. solua
- Olkoon rakenteen globaali syvyys d. Tällöin avaimen k solu (siis soluhakemiston indeksi) saadaan eristämällä hajautusfunktion antaman osoitteen lopusta d bitin pituinen osa tail_bits(h(k),d).
  - Jos d=1 otetaan viimeinen bitti => {0,1}
  - Jos d=2 otetaan 2 viimeistä bittiä => {00,01,10,11}

7

Laajeneva hajautus

- Lisäysten käsittely
  - Jos soluhakemistosta löytyvässä kotilohkossa on tilaa tietue lisätään sinne kuten aiemmin käsitellyssä hajautuksessa
  - Jos kotilohko (olkoon sen solutunnus binäärisenä {b}) on täynnä, otetaan käyttöön uusi lohko ja jaetaan ylivuotavan kotilohkon tietueet kotilohkon ja uuden lohkon välillä
    - kotilohkolla on paikallinen syvyys (local depth) p (monenko bitin perusteella tietueet on sijoitettu lohkoon)

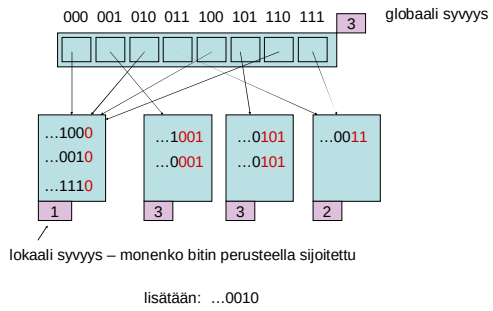
8

Laajeneva hajautus

- Lisäyksen käsittely jatkuu
  - tietueille tehdään uusi sijoittelu ottamalla käyttöön hajauttimen tuottaman osoitteen lopusta päin p+1:s bitti
  - ne tietueet, joilla tämä bitti on 0, jäävät kotilohkoon ja muut siirtyvät (oletetaan, että kaikki eivät menneet samaan lohkoon, näinkin voisi käydä L)
  - kotilohkon ja uuden lohkon paikalliseksi syvyydeksi asetetaan p+1
  - soluhakemiston alkiot, joiden indeksin p+1 viimeistä bittiä ovat 1{b} asetetaan osoittamaan uuteen lohkoon

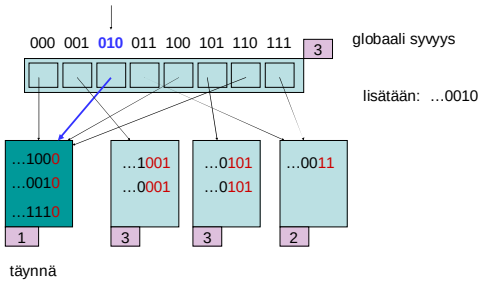
9

Laajeneva hajautus



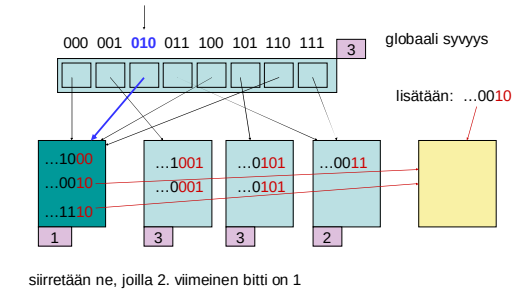
10

Laajeneva hajautus



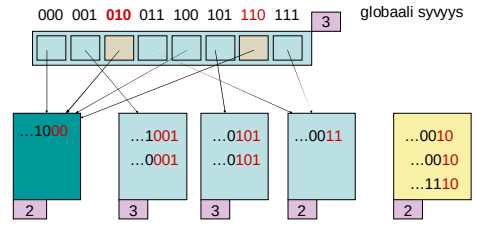
11

Laajeneva hajautus



12

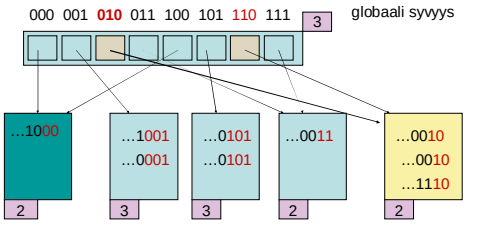
Laajeneva hajautus



siirretään ne, joilla 2. viimeinen bitti on 1
vanhan ja uuden paikallinen syvyys 2
vaihdetaan ne soluosoittimet, joiden indeksi loppuu ..10 osittamaan uuteen

13

Laajeneva hajautus



siirretään ne, joilla 2. viimeinen bitti on 1
vaihdetaan ne soluosoittimet, joiden indeksi loppuu ..10 osittamaan uuteen

14

Laajeneva hajautus

- Jos jaettavan kotilohkon paikallinen syvyys on sama kuin rakenteen globaali syvyys, joudutaan tuplaamaan soluhakemiston koko, (jotta voitaisiin kirjata osoite soluun 1{b})
- Tuplaus tapahtuu helposti kopioimalla nykyinen soluhakemisto
- Tuplaus nostaa rakenteen globaalia syvyyttä yhdellä, kotilohkojen syvyydet säilyvät ennallaan

15

Laajeneva hajautus

- Rakenne laajenee sykäyksittäin
- Jos jakokohtaan osuvissa avaimissa on yhtenäinen bittisekvenssi, voidaan joutua tekemään monta jakoa ennen kuin tietueet saadaan jaettua
- Jos soluhakemisto kasvaa isoksi eikä sitä voida pitää keskusmuistissa voidaan hakuun tarvita 2 levyhakua
- jako ja tuplaus voivat edellyttää useita hakuja

16

Lineaarinen hajautus

- Lineaarisessa hajautuksessa (kotimaista alkuperää – Per Larsson) ei välttämättä tarvita soluhakemistoa
- Hajautusaluetta laajennetaan solu kerrallaan jakamalla jakovuorossa olevan solun sisältö solun itsensä ja uuden solun kesken
- Solut saavat jakovuoronsa järjestyksessä, eikä jaettava solu ole suinkaan välttämättä se jonka kohdalla ylivuoto tapahtuu – soluihin voidaan ylivuotavia tietueita varten liittää ylivuotolista

17

Lineaarinen hajautus

- Solun osoitteen määräämiseksi käytössä on sarja hajauttimia $h_0, h_1, h_2, \ldots$
- Nämä ovat muotoa $\mathbf{h}_i = h(k) \bmod (2^i N)$.
- N voidaan valita kakkosen potenssiksi 2^d, tällöin h_i eristäisi d+i bittiä perushajauttimen h tuottaman arvon lopusta
  - jos d=5, niin h_0 eristää 5 bittiä, h_1 6 bittiä jne
  - tietueen haussa tarvitaan perushajauttimen lisäksi kahta hajautinta h_{taso} ja h_{taso+1}

18

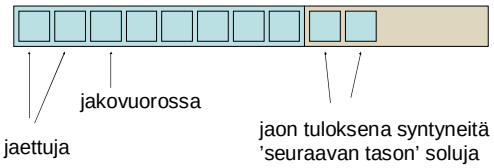
Lineaarinen hajautus

- Haku
  - laske osoite $h_{taso}(h(k))$, eli ota d+taso bittiä lopusta
  - jos kyseessä on jakamaton solu, etsi tietuetta solusta.
  - jos kyseessä on jaettu solu muodosta uusi osoite $h_{taso+1}(h(k))$, eli ota d+1+taso bittiä lopusta
  - etsi tietuetta saadusta solusta
  - Solu on jakamaton, jos sen indeksi on suurempi tai yhtä suuri kuin jakovuorossa olevan solun indeksi

19

Lineaarinen hajautus

Olkoon d=1, taso=2, eli h_2 antaa osoitteet 0..7 ja h_3 osoitteet 0..15



20

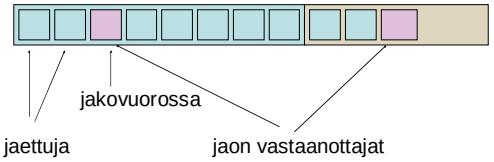
Lineaarinen hajautus

- Lisäyksessä, tietue lisätään haun määräämään soluun
  - jos tietue ei mahdu solun kotilohkoon, se lisätään ylivuotoketjuun.
  - Ylivuoto käynnistää jako-operaation
  - Olkoon jakovuorossa olevan solun taso+d bitistä muodostuva osoite {b}
  - Otetaan käyttöön uusi solu jonka osoite on 1{b}
  - jakovuorossa olevan solun tietueet jaetaan alkuperäisen solun ja uuden solun kesken käyttäen hajautinta h_{taso+1}
  - Jos jakovuorossa oli tason viimeinen solu siirtyy jakovuoro soluun 0 ja tasoa kasvatetaan yhdellä, muuten jakovuoro siirtyy seuraavaan soluun.

21

Lineaarinen hajautus

Olkoon d=1, taso=2, eli h_2 antaa osoitteet 0..7 ja h_3 osoitteet 0..15



22

Lineaarinen hajautus

- Kullakin tasolla jaetaan vuorollaan jokainen tason solu. Kun kaikki on jaettu, on hajautusalue tuplautunut ja siirrytään seuraavalle tasolle.
- Rakenteessa voi olla pitkiäkin ylivuotoketjuja, mutta ne lyhenevät kun taso kasvaa.

23