Adatbázisok elmélete 21. előadás

Csima Judit
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 136/b
csima@cs.bme.hu

2003. Április 23.

A dinamikus hash értékelése

Előnyök:

- egy I/O művelettel minden megvan, ha a vödörkatalógus a belső memóriában van
- a struktúra igazodik az állomány növekedéséhez

Bajok:

- szerencsétlen esetben nagyon nagy vödörkatalógust kell építeni kevés rekord miatt (legrosszabb esetben lényegében a szekvenciális keresést kapjuk vissza)
- a $h(K)$ értéket se lehet a végtelenségig továbbszámolni: ha $h(K)$ már teljes hosszáig kiszámolandó, akkor nem lehet tovább növelni a struktúrát

Indexek

Másik technika adatok jó tárolására. Továbbra is az a cél, hogy a műveletek gyorsan menjenek.

Motiváló ötlet:

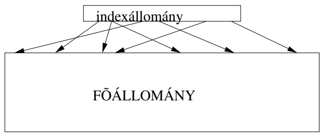
- könyvtárban úgy keresünk, hogy katalóguscetlik között keresünk és az alapján már megvan a könyv (sűrű indexre motiváció)
- ha a polcokon ábécé szerint vannak sorban a könyvek, akkor a polcok elején álló könyvek megvizsgálásával gyorsan eldönthető, hogy melyiken kell keresni (ritka indexre motiváció)

Indexek általános jellemzői:

- rendezettséget figyelembe veszi (ellentétben a hash-el)

- van korlát a legrosszabb esetre is (ellentétben a hash-el, ami viszont átlagosan jobb)
- jól bírja a dinamikus bővülést (ellentétben a hash-el)
- kulcs rendezett halmazból kerül ki (de nem féltétlenül egyedi)
- lehet ugyanarra az állományra több index is készítve

Az indexstruktúra vázlatos felépítése



Az indexállományban vannak a keresést segítő infók, a főállományban vannak a tárolt adatok, a rekordok.

A főállomány lehet esetleg rendezett a keresési kulcs szerint, de nem feltétlenül az.

Az indexállományban indexrekordok vannak, ezek felépítése:

kulcs	mutató

A kulcs valamelyik főállománybeli rekordhoz tartozó kulcsérték, a mutató pedig a főállományba mutat, oda, ahol ez a rekord van.

Aszerint, hogy minden főállománybeli rekordra van rá mutató indexbejegyzés vagy pedig csak néhányra, sűrű illetve ritka indexről beszélhetünk. (Majd látjuk, hogy ez két nagyon más helyzet lesz.)

Ritka index, egyszintű

Feltesszük, hogy a főállomány szabad, azaz elmozgathatók a rekordok szabadon.

Jellemzők:

- motiváló példa: telefonkönyv vagy szótár sarkaiban a bejegyzések, ez alapján keresés
- ritka index, azaz nem minden főállománybeli rekordra van indexbejegyzés, csak blokkonként egyre
- a főállomány vagy rendezett a keresési kulcs szerint vagy lényegében rendezett (blokkokon belül esetleg nem rendezettek a rekordok, de a blokkok egymáshoz képest rendezetten vannak)
- az indexállomány rendezett a keresési kulcs szerint (az indexállomány is háttértéron van)
- az indexbejegyzésben szereplő kulcs a rendezett esetben a blokk legelső kulcsa, a lényegében rendezett esetben a blokk legkisebb kulcsa

- a mutató az indexbejegyzésben arra a blokkra mutat, ahol a bejegyzésben szereplő kulcshoz tartozó rekord van

A fentiek miatt a ritka index kivonata lesz a főállománynak, tartalmazza rendezetten a blokkok legkisebb kulcsait.

Keresés

Adott a keresési kulcs értéke, meg kell találni az ilyen rekordokat.

1. Először az indexállományban megkeressük azt a legnagyobb indexbejegyzést, aminek kulcsa még éppen nem nagyobb, mint az adott keresési érték. Ez átlagosan $\frac{n}{2}$ I/O művelet, ha n blokkból áll az indexállomány.
2. Az előbb megtalált indexbejegyzéshez tartozó blokkban (plusz egy I/O művelet) megkeressük a rekordunkat

Megjegyzések:
1. Ha van valami plusz struktúra, ami segíti a gyors keresést az indexállományban, akkor jobbat is lehet, mint $\frac{n}{2}$.
2. A blokkon belül, a főállományban már bárhogy kereshetünk, de leginkább szekvenciálisan megy

További műveletek

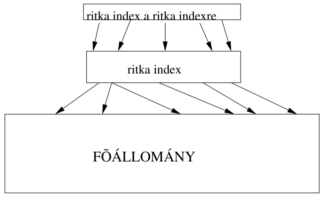
1. **beszúrás**: először egy keresés (hol kellene lennie), aztán blokkon belül a helyére tesszük:
   - *ha rendezett a főállomány*: blokkon belül megkeressük a helyét, beillesztjük; ha nem fér be $\Longrightarrow$ blokkvágás két egyenlő részre, az új blokk minimális (legelső) kulcsára indexbejegyzés beszúrása az indexállományba
   - *ha lényegében rendezett a főállomány*: ha van még hely a blokkban, akkor berakjuk, mindegy, hogy hova; ha nem fér be $\Longrightarrow$ blokkvágás két részre, mindkét részben a minimális elemhez új indexbejegyzés és a régi törlése az indexállományból
2. **törlés**: hasonlóan, mint a beszúrás, először keresés, aztán törlés a blokkból;
   ha épp a minimális rekordot töröltem a blokkból $\Longrightarrow$ indexbejegyzés módosítása
   ha nagyon ritkák a blokkok $\Longrightarrow$ esetleg lapösszevonás (és persze az indexállomány módosítása is ilyenkor), de általában nem érdemes

3. **tól-ig**: valahonnan valameddig terjedő értékű rekordok keresése: a “tól” érték keresése, aztán a főállomány végigolvasása az “ig” értékig (a főállomány folyamatos elérése megoldott)
4. **módosítás**: ha kulcsot nem érint: keresés, átírás; ha kulcsot is érint: törlés, beszúrás

Megjegyzések:
1. Tényleg használtuk, hogy a főállomány szabad (pakolgattuk a rekordokat össze-vissza). Lesz majd technika, amivel elérhető lesz, hogy a főállomány szabadnak látszódjon.
2.Azt is erősen kihasználtuk, hogy (lényegében) rendezett a főállomány.
3. Azért hasznos az indexállomány, mert sokkal kisebb, mint a főállomány, könnyebb benne keresni és a végen csak plusz egy I/O művelet kell a befejezéshez. De ennek ára van: karban kell tartani plusz egy struktúrát.

Többszintes ritka index

Az indexen belül keresni arányos az indexállomány blokkjainak számával. Ez jóval kisebb, mint a főállomány lapszáma, de még mindig nagyon nagy lehet. Ezért: többszintű index, vagyis index az indexre:



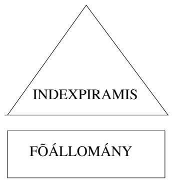
- a felső index még kisebb lesz, könnyebb lesz benne keresni
- a középső szint egyszerre indexe a főállománynak és “főállománya” a felső indexnek

- keresés: a legfelső szinten megkeressük a legnagyobb olyan bejegyzést, ami még kisebb a keresettnél és innen két lap beolvasásával (a megfelelő középső szintű indexlap és aztán a főállomány megfelelő lapja) megvan a keresett rekord
- a többi művelet hasonlóan megy (persze, ha módosul a főállomány, akkor esetleg mindegyik indexállományt is módosítani kell)
- eggyel több szint lesz, azaz eggyel több lapelérés kell a kezdeti keresés után, de a kezdeti keresés lerövidül
- nem csak kétszintű lehet a ritka index, hanem több is $\Longrightarrow$ dinamikusan is változhat $\Longrightarrow$ B-fa

B-fa

A ma ismert egyik legjobb és legelterjedtebb megoldás, m elágazásos B-fa vagy B_m-fa, lényegében ahogy algelből tanultuk:

- a fa levelei: a főállomány blokkjai
- a főállomány (a levelek) rendezett a keresési kulcs szerint
- minden levél ugyanolyan messze van a gyökértől
- a fa belső csúcsai: a különböző szintű indexek lapjai
- egy csúcs gyerekei: az indexlapon levő mutatóknak megfelelő eggyel lejjebb levő indexlapok (illetve alul levelek)
- m: egy lapra m indexrekord fér rá (kicsit más lesz egy belső csúcs szerkezete, mint algelből volt)
- minden lap legalább félig kitöltött, kivéve esetleg a gyökeret (minden csúcsnak legalább $\frac{m}{2}$ gyereke van, kivéve esetleg a gyökeret)



Műveletek

- **keresés**: ahogy algelből volt, a csúcsokban található kulcs-bejegyzések és mutatók mentén; arányos a fa magasságával, ami $O(log_m n)$, ha n blokkja van a főállománynak
- **beszúrás**: ahogy algelből volt, beszúrás után esetleg csúcsvágás(ok), de max. $O(log_m n)$
- **törlés**: ahogy algelből volt, törlés után esetleg csúcsösszevonás(ok), de max. $O(log_m n)$

Megjegyzések:

1. Ha m nagy $\Longrightarrow$ ritkán kell csúcsvágás/csúcsösszevonás.
2. általában m úgy van választva, hogy a fa magassága max. 4 legyen, ha az első lap a belső memóriában van, akkor elég 3 I/O művelet mindenhez

Sűrű index

Eddig feltettük, hogy a főállomány szabad és (lényegében) rendezett a keresési kulcs szerint. Hogyan érjük ezt el? Hogyan lehet több kulcs szerint is keresni?

Erre megoldás a sűrű index:

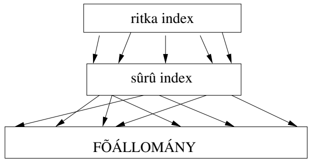
- a főállomány minden rekordjához van egy indexbejegyzés $\Longrightarrow$ ugyanannyi bejegyzés lesz a sűrű indexben, mint ahány rekord van a főállományban, csak persze kisebbek a bejegyzések $\Longrightarrow$ sűrű index = főállomány kicsiben
- ez nem önálló állományszervezési technika (ellentétben a ritka index változataival), hanem csak kiegészítés, ami lehetővé teszi, hogy a főállományt szabadnak és rendezettnek tételezhessük fel

Haszna:

- szabaddá teszi a rekordokat (a főállomány ugyan kötött, de a sűrű index bejegyzései szabadon mozgathatók: építhető rá ritka index)

- rendezettnek mutatja a főállományt: a sűrű indexet úgy rendezzük, ahogy akarjuk
- sokkal kisebb, mint a főállomány, mégis egy az egyben megfelel neki

Tipikus használata:



A sűrű index ráépül a főállományra, erre építjük a valódi állományszervezést. A sűrű index miatt a főállomány szabadnak és rendezettnek tűnik.

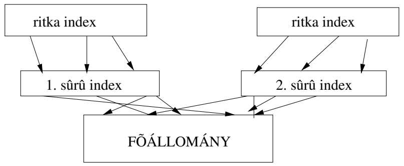
Műveletek ebben a struktúrában

Úgy dolgozunk, mintha a sűrű index lenne a főállomány, innen már csak egy plusz lapelérés a valódi főállomány.

- keresés: keresés sűrűben, onnan egy lapelérés
- beszúrás: beszúrás sűrűbe, aztán valahova berakjuk a főállományba
- törlés: keresés, törlés a főállományból, törlés a sűrűből is

Nagy előnye a sűrű indexnek

Lehetővé teszi, hogy egy főállományra több különböző kulcs szerint is legyen index:



Itt minden sűrű index rendezett a megfelelő kulcs szerint és persze ha változik a főállomány, akkor mindegyik sűrűt is változtatni kell.

Példa:
A Személy(név, telefonszám, személyiszám, ...) sémában a személyiszám az elsődleges kulcs, ezért a rendszer eszerint rendezetten tárolja az adatokat és erre biztosan létre is hoz valami keresési struktúrát. De ha mi szeretnénk a név-re is: kell egy sűrű index: invertált állomány.