6.7. Index alapú algoritmusok

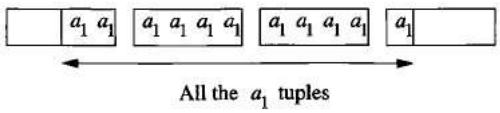
Ha létezik index a reláció egy vagy több attribútumához, akkor elérhetővé válik néhány olyan algoritmus, amely index hiányában nem lenne kivitelezhető. Az index alapú algoritmusok különösen hasznosak a kiválasztás operátorra, de az összekapcsolás és az egyéb bináris műveletek algoritmusai is igen jól kihasználják az indexeket. Folytatjuk az indexszel rendelkező táblák elérésére szolgáló index alapú beolvasás művelet 6.2.1. részben megkezdett tárgyalását is. Ahhoz, hogy ezt a témakört igazán értékelni tudjuk, először teszünk egy kitérőt, és az ún. „nyalábolt" indexekkel foglalkozunk.

6.7.1. Nyalábolt és nem nyalábolt indexek

Emlékezzünk rá a 6.2.3. részből, hogy egy relációt akkor nevezünk nyalábboltnak, ha sorai nagyjából annyi blokkban vannak tárolva, ahányban azok minimálisan elférnének. Az eddig elvégzett összes elemzés azt feltételezte, hogy a relációk nyalábboltak.

Beszélhetünk e mellett még *nyalábolt indexekről* is, amelyek olyan attribútumon vagy attribútumokon értelmezett indexek, amelyeknél a keresési kulcs egy rögzített értékéhez tartozó sorok nagyjából annyi blokkban helyezkednek el, ahány blokkban minimálisan elférnének. Vegyük észre, hogy egy nem nyalábolt relációnak nem lehet nyalábolt indexe[1], viszont egy nyalábolt relációnak lehetnek nem nyalábolt indexei.

6.21. példa: Az a attribútum szerint rendezett $R(a, b)$ reláció, amelyet rendezett sorrendben tárolunk, biztosan nyalábolt. Az a attribútumon értelmezett index is nyalábolt, hiszen az a egy adott a_1 értékére az összes ilyen értékkel rendelkező sor egymás után található. Tehát nyaláboltan jelennek meg a sorok a blokkokban, kivéve esetleg az első és az utolsó a_1 értéket tartalmazó blokkokat, amint azt a 6.19. ábrán is láthatjuk. Egy index a b attribútumon viszont valószínűleg nem nyalábolt, mivel az adott b értékkel rendelkező sorok bárhol elhelyezkedhetnek a fájlban, kivéve, ha az a és b értékei erősen korreláltak. □



6.19. ábra. *Egy nyalábolt indexnek az összes rögzített értékkel rendelkező sora a lehetséges minimális (vagy majdnem minimális) számú blokkban található*

6.7.2. Index alapú kiválasztás

A 6.2.1. részben megmutattuk, hogy miként lehet egy $\sigma_C(R)$ kiválasztást megvalósítani oly módon, hogy beolvassuk az R reláció összes sorát, és a C feltételt kielégítő sorokat kiírjuk a kimenetre. Ha R nem rendelkezik indexszel, akkor ez a legjobb, amit tehetünk. A művelet által felhasznált lemez I/O-műveletek száma $B(R)$, vagy esetleg $T(R)$, az R sorainak száma, ha R nem nyalábolt reláció[2]. Tegyük fel azonban, hogy a C feltétel $a = v$ formájú, ahol az a olyan attribútum, amelyen van indexe, v pedig egy érték. Ekkor rákereshetünk az index v értékére, és megkapjuk azokat a mutatókat, amelyek az R azon soraira mutatnak, ahol az a attribútum értéke éppen v. Ezek a sorok képezik $\sigma_{a=v}(R)$ eredményét, így mindössze annyit kell tennünk, hogy visszanyerjük azokat.

Ha az $R.a$ indexe nyalábolt, akkor a $\sigma_{a=v}(R)$ halmaz visszanyeréséhez szükséges lemez I/O-műveletek száma nagyjából $\boldsymbol{B(R)/V(R, a)}$. A tényleges érték lehet, hogy ennél valamivel nagyobb, mert:

---

[1] Technikai értelemben, ha az indexet a reláció egy kulcsára definiáljuk, azaz az indexkulcs egy adott értékével csak egy sor létezik, akkor az index mindig „nyalábolt", még akkor is, ha a reláció nem nyalábolt. Ugyanakkor, ha index kulcsértékenként csak egy sor van, akkor a tömörségnek nincs semmi haszna, és egy ilyen index teljesítményértékelése ugyanazt adja, mintha nem nyalábolt index lenne.

[2] Idézzük fel a 6.2.3. részben használt jelöléseket: $T(R)$ jelöli az R reláció sorainak számát és $V(R, L)$ a $\pi_L(R)$ művelet egymástól különböző sorainak számát.

1. Gyakran előfordul, hogy az indexet nem tároljuk teljes egészében a memóriában, vagyis kell néhány lemez I/O-művelet az indexben történő kereséshez.
2. Lehet, hogy az összes olyan sor, amelyre $a = v$, belefér b számú blokkba, mégis előfordulhat, hogy $b + 1$ blokkban vannak tárolva, mert nem valamelyik blokk elején kezdődnek.
3. Bár az index nyalábolt, az $a = v$ értékű sorok átnyúlhatnak néhány további blokkba is. Lássunk két okot arra, hogy ez miért fordulhat elő:

   a) R blokkjait esetleg nem a legszorosabban raktuk össze, mert helyet akartunk hagyni R növekedésének a 4.1.6. részben tárgyaltak szerint.
   b) R tárolása elképzelhető olyan sorokkal együtt is, amelyek nem tartoznak R-hez, mondjuk egy nyalábolt fájl elrendezésében.

A fentiek mellett persze kerekítenünk is kell, ha $B(R)/V(R, a)$ érték nem egész szám. Ez leginkább akkor fordul elő, ha a az R kulcsa, ilyenkor $V(R, a) = T(R)$, ami feltehetőleg sokkal nagyobb $B(R)$-nél, nekünk azonban mégis szükségünk van egy lemez I/O-műveletre a v kulcsértékkel rendelkező sor visszanyerésére, és ehhez még hozzájön az index eléréséhez szükséges valahány lemez I/O-művelet is.

Nézzük meg most, hogy mi történik, ha az $R.a$ indexe nem nyalábolt. Első közelítésben elmondható, hogy minden visszanyert sor más blokkban lesz, és $T(R)/V(R, a)$ számú sort kell elérnünk. Ezek szerint éppen $T(R)/V(R, a)$ adja a szükséges lemez I/O-műveletek számának becslését. Maga a tényleges szám ennél nagyobb is lehet, mert előfordulhat, hogy pár indexblokkot lemezről kell beolvasnunk; de lehet a szám ennél kisebb is, ha egyes visszanyert sorok véletlenül ugyanabban a blokkban jelennek meg, és a szóban forgó blokk a memóriapufferben marad.

6.22. példa: Legyen $B(R) = 1000$, és legyen $T(R) = 20000$. Ezek szerint R-nek húszezer sora van, amelyek húszasával találhatók a blokkokban. Legyen a az R egyik attribútuma; tegyük fel, hogy létezik index az a-n, és vizsgáljuk meg a $\sigma_{a=0}(R)$ műveletet. Az alábbiakban felsorolunk néhány lehetőséget, és megadjuk a legrosszabb esetben szükséges lemez I/O-műveletek számát. Az indexblokkok elérésének költségét minden esetben elhanyagoljuk.

1. Ha R nyalábolt, de nem használjuk az indexet, akkor a költség 1000 lemez I/O-művelet. Ez azt jelenti, hogy R minden blokkját vissza kell nyernünk.
2. Ha R nem nyalábolt, és nem használjuk az indexet, akkor a költség 20 000 lemez I/O-művelet.
3. Ha $V(R, a) = 100$ és az index nyalábolt, akkor az index alapú algoritmus 1000/100 = 10 lemez I/O-műveletet használ.
4. Ha $V(R, a) = 10$ és az index nem nyalábolt, akkor az index alapú algoritmus 20 000/10 = 2000 lemez I/O-műveletet használ. Figyeljük meg, hogy **ez a költség magasabb, mint az egész *R* reláció beolvasása** abban az esetben, ha R nyalábolt, de az index nem.
5. Ha $V(R, a) = 20\,000$, azaz a kulcs, akkor az index alapú algoritmus 1 lemez I/O-műveletet igényel, plusz még azt, ami az index eléréséhez szükséges, függetlenül attól, hogy az index tömör vagy sem.
   □

Az index alapú beolvasás mint elérési módszer más típusú kiválasztási művelet során is hasznos lehet.

a) Egy index – pl. egy B-fa – lehetővé teszi az egy adott tartományon belüli keresési kulcs értékek elérését. Ha az R reláció a attribútumára létezik ilyen index, akkor azt használhatjuk arra, hogy a $\sigma_{a\geq 10}(R)$, illetve $\sigma_{a \geq 10 \text{ AND } a\leq 20}(R)$ típusú kiválasztások esetén az R-nek csak a kívánt tartományba eső sorait olvassuk be.
b) Egy komplex C feltételre alapuló kiválasztást bizonyos esetekben megvalósíthatunk úgy, hogy egy index alapú beolvasás után egy másik kiválasztást végzünk a már beolvasott sorokon. Ha pl. a C feltétel $a = v$ AND C' formájú, ahol C' egy tetszőleges feltétel, akkor a kiválasztást két egymás utáni kiválasztásra bonthatjuk fel. Az első kiválasztás csak $a = v$ feltételt ellenőrzi, míg a második a C' feltételt ellenőrzi. Az első kiválasztásnál az index alapú beolvasás operátor használata látszik valószínűnek. A kiválasztás művelet ilyen jellegű kettéosztása csupán egyike annak a sok javításnak, amelyet egy lekérdezés-optimalizáló a logikai lekérdezés-terven eszközölhet, és amellyel a 7.7.1. rész foglalkozik.

6.7.3. Összekapcsolás index segítségével

Az összes eddig vizsgált bináris művelet, továbbá a δ és a γ teljes relációs unáris műveletek is előnyösen

használhatnak bizonyos indexeket. Ezen algoritmusok nagy részét feladatként hagyjuk, és most csak az összekapcsolásokra fogunk összpontosítani. Ezen belül is az $R(X, Y)$ |><| $S(Y, Z)$ természetes összekapcsolást vizsgáljuk meg. Emlékezzünk rá, hogy az X, az Y és a Z attribútumok halmazait jelöli, bár itt gondolhatunk rájuk úgy is, mint konkrét attribútumokra.

Az első index alapú összekapcsolási algoritmushoz tegyük fel, hogy az S relációnak van egy indexe az Y attribútumo(ko)n. Ekkor az összekapcsolás kiszámításának egyik módja az, hogy megvizsgáljuk az R minden egyes blokkját, majd pedig a blokkokon belül minden egyes t sort. Jelöljük az Y attribútum(ok)nak megfelelő t komponenst vagy komponenseket t_Y-nal. Használjuk az indexet arra, hogy megkeressük az S azon sorait, amelyek Y komponense éppen t_Y. Ezek S-nek pontosan azok a sorai, amelyek összekapcsolódnak az R t sorával, ezért minden egyes ilyen sor t-vel való összekapcsolását kiírjuk a kimenetre.

A lemez I/O-műveletek száma több tényezőtől függ. Először is, feltéve, hogy R nyalábolt, $B(R)$ számú blokkot kell beolvasnunk ahhoz, hogy R minden sorát megkapjuk. Ha R nem nyalábolt, akkor akár $T(R)$ számú lemez I/O-műveletre is szükség lehet.

R minden egyes t sorára átlagosan az S reláció $T(S)/V(S, Y)$ számú sorát kell beolvasnunk. Ha S-nek van az Y-ra egy nem nyalábolt indexe, akkor a szükséges lemez I/O-műveletek száma $T(R)T(S)/V(S, Y)$. Ha viszont az index nyalábolt, akkor mindössze $T(R)B(S)/V(S, Y)$ is elégséges[3]. Mindkét esetben előfordulhat, hogy hozzá kell még adnunk Y értékenként néhány lemez I/O-műveletet magának az indexnek a beolvasására.

Függetlenül attól, hogy R nyalábolt vagy sem, mindenképpen az S sorainak elérési költsége a döntő, így ennek az összekapcsolási módszernek a költségét $T(R)T(S)/V(S, Y)$-ként illetve $T(R)\big(\max\big(1,B(S)/V(S, Y)\big)\big)$-ként adhatjuk meg azokra az esetekre, ha az S indexe nem nyalábolt, illetve nyalábolt.

6.23. példa: Térjünk vissza az előző példák adataihoz: az $R(X, Y)$ és az $S(Y, Z)$ relációk egyenként 1000, ill. 500 blokkot foglalnak el. Tegyük fel, hogy mindkét relációból tíz sor fér egy blokkba, azaz $T(R) = 10\,000$ és $T(S) = 5000$. Tegyük fel továbbá, hogy $V(S, Y) = 100$, azaz S sorai között 100 különböző Y érték fordul elő.

Feltételezzük, hogy R nyalábolt, S-nek pedig van egy nyalábolt indexe az Y attribútumo(ko)n. Ekkor – az index elérésére használtakat leszámítva – az R blokkjainak beolvasásához szükséges lemez I/O-műveletek száma körülbelül 1000 (amit a fenti képletekben elhanyagoltunk), ehhez jön még az összekapcsolás költsége, ami $10000 \times 500/100 = 50\,000$. Ez a szám jóval meghaladja az ugyanezekkel az adatokkal végzett, korábban bemutatott módszerek költségét. Ez a költség még tovább nő, ha vagy az R reláció vagy S indexe nem nyalábolt. □

Noha a 6.23. példa alapján azt gondolhatnánk, hogy az index alapú összekapcsolás nem valami jó ötlet, vannak olyan helyzetek amikor az R |><| S összekapcsolást érdemes ilyen módszerrel elvégezni. A leggyakoribb eset az, amikor R sokkal kisebb S-nél, $V(S, Y)$ pedig nagy. A 6.7.5. feladatban utalunk majd egy tipikus lekérdezésre, ahol az összekapcsolást megelőző kiválasztás során R egészen pici lesz. Ebben az esetben S legnagyobb részét az algoritmus egyszer sem vizsgálja meg, hiszen a legtöbb Y érték R-ben meg sem jelenik. A rendezésen, valamint a tördelésen alapuló összekapcsolási módszerek viszont legalább egyszer megvizsgálják S minden egyes sorát.

[3] Ne felejtsük azonban el, hogy $B(S)/V(S, Y)$ helyére 1-et kell írnunk, ha az előbbi hányados 1-nél kisebb lenne; lásd a 6.7.2. részt.