6.5.5. Egy egyszerű rendezésen alapuló összekapcsolási algoritmus

A rendezést többféle módon lehet nagy relációk összekapcsolására használni. Mielőtt rátérnénk az összekapcsolási algoritmusok vizsgálatára, hadd jegyezzük meg, hogy összekapcsolások számításakor felmerülhet egy olyan probléma, amely az eddig vizsgált bináris műveletekkel kapcsolatban nem okozott gondot. Egy összekapcsolás alkalmával a két reláció azon sorainak a száma, amelyek az összekapcsolás alapjául szolgáló attribútumokon megegyeznek, és ezért egyidejűleg kell hogy a memóriában legyenek, meghaladhatja a memóriába beférő mennyiséget. A szélsőséges példa talán az lehet, amikor az összekapcsolás alapjául szolgáló attribútum(ok)nak csupán egyetlen értéke van, és így az egyik reláció minden sora összekapcsolódik a másik reláció minden sorával. Ebben az esetben nincs más választásunk, mint hogy vegyük az azonos attribútumértékekkel bíró két sorhalmaznak egy beágyazott cikusú összekapcsolását.

Annak érdekében, hogy ezt az eshetőséget elkerüljük, megpróbálhatjuk az algoritmus más célra felhasznált memóriaigényét csökkenteni, és ezáltal több puffert tudunk elérhetővé tenni az összekapcsolódó sorok befogadására. Ebben a szakaszban azt az algoritmust mutatjuk be, amely a lehető legtöbb puffert teszi elérhetővé a közös értékkel rendelkező sorok tárolására. A 6.5.7. részben megnézünk majd egy másik rendezésen alapuló algoritmust is, amely ugyan kevesebb lemez I/O-műveletet használ, de problémákba ütközhet akkor, ha nagyszámú olyan sor van, amelyek az összekapcsolás attribútumain megegyeznek.

Tegyük fel, hogy az $R(X, Y)$ és az $S(Y, Z)$ relációkat szeretnénk összekapcsolni, és ehhez M darab memóriablokk áll rendelkezésünkre. Ekkor a következőket tesszük:

1. Rendezzük R-et egy kétfázisú többutas összefésüléssel, amelyben Y a rendezési kulcs.
2. S-et hasonló módon rendezzük.
3. Összefésüljük a rendezett R és S relációkat. Ehhez általában csak két puffert használunk, egyet R és egyet S éppen aktuális blokkjára. Az alábbi lépéseket többször megismételjük:

   a) Megkeressük az Y összekapcsolási attribútumoknak azt a legkisebb y értékét, amely éppen az R és S blokkok elején található.
   b) Ha y nem jelenik meg a másik reláció elején, akkor az y rendezési kulcsú sor(oka)t eltávolítjuk.
   c) Egyébként azonosítjuk mindkét relációban az összes y rendezési kulcsú sort. Ha szükséges, addig olvassuk be a rendezett R és S blokkjait, amíg biztosak nem leszünk benne, hogy már egyik relációban sincs y értékű sor. Erre a célra összesen M puffert használhatunk fel.
   d) A kimenetbe írjuk az összes olyan sort, amely R és S közös Y értékkel – jelen esetben éppen y-nal – rendelkező sorainak összekapcsolásával kialakítható.
   e) Ha bármelyik relációban már nincs több megvizsgálatlan sor a memóriában, akkor annak pufferét újra feltöltjük.

6.17. példa: Vegyük ismét a 6.14. példa R és S relációit. Emlékeztetünk rá, hogy a relációk 1000, illetve 500 blokkot foglalnak el, és összesen $M = 101$ memóriapufferünk van. Ha egy relációra kétfázisú többutas összefésüléses rendezést végzünk, akkor minden blokkra négy lemez I/O-műveletet végzünk, mindkét fázisban kettőt-kettőt. R és S rendezéséhez tehát $4\big(B(R) + B(S)\big)$ lemez I/O-művelet szükséges, ami esetünkben éppen 6000.

Amikor az összekapcsolódó sorok megkereséséhez összefésüljük a rendezett R és S relációkat, akkor R és S minden egyes blokkját még egyszer beolvassuk (ez már az ötödik I/O), ami további 1500 lemez I/O-műveletet jelent. Az összefésülés során általában mindössze kettőt használunk a 101 memóriablokkból. Ugyanakkor ha szükség van rá azt is megtehetjük, hogy R és S közös Y értékkel – ez esetben y-nal – rendelkező sorainak befogadására felhasználjuk mind a 101 blokkot. Így elegendő az a feltétel, hogy R és S közös Y értékkel bíró sorai semmilyen y esetén se foglaljanak el összesen 101 blokknál többet.

Vegyük észre, hogy ebben az algoritmusban az összes végrehajtott lemez I/O-műveletek száma 7500, szemben a 6.14. példában szereplő beágyazott ciklusú összekapcsolás 5500 műveletével. Tudjuk azonban, hogy a beágyazott ciklusú algoritmus természeténél fogva négyzetes, így futási ideje $B(R)B(S)$-sel arányos, míg a rendezéses összekapcsolás I/O-művelet költsége lineáris, vagyis a futási ideje $\big(B(R) + B(S)\big)$-sel arányos. Csupán a konstans tényezők értéke és a példa kis mérete (az egyes relációk csak 5, illetve 10-szer nagyobbak annál, mint ami még beférne a memóriapufferekbe) okozza, hogy a beágyazott ciklusú összekapcsolás előnyösebb. Sőt, a 6.5.7. részben látni fogjuk, hogy a rendezéses összekapcsolást általában lehetséges $3\big(B(R) + B(S)\big)$ lemez I/O-művelettel elvégezni, ami esetünkben 4500-at jelentene, ami már alatta van a beágyazott ciklus költségének. □

Ha van olyan y Y érték, amelyre az ezzel az értékkel rendelkező sorok nem férnek be M pufferbe, akkor az

előző algoritmust módosítanunk kell.

1. Ha az egyik reláció, legyen ez mondjuk R, y Y értékkel rendelkező sorai befférnek M–1 darab pufferbe, akkor olvassuk be R ezen blokkjait pufferekbe, majd egyenként olvassuk be S y értékű sorait a fennmaradó pufferbe. Valójában ilyenkor a 6.3.3. rész egymenetes összekapcsolását végezzük el azokra a sorokra, amelyek Y értéke éppen y.
2. Ha mindkét relációnak több y Y értékű sora van annál, hogy azok M–1 pufferbe beférjenek, akkor használjuk fel az M puffert, és hajtsunk végre egy beágyazott ciklusú összekapcsolást a két reláció y Y értékű sorain.

Vegyük észre, hogy mindkét esetben előfordulhat, hogy az egyik reláció sorait beolvassuk, majd figyelmen kívül hagyjuk őket, és így később újra be kell olvasnunk azokat. Például az 1. esetben először lehet, hogy S azon sorainak beolvasásával próbálkozunk, amelyeknek Y értéke y, és azt találjuk, hogy azok nem férnek be M–1 pufferbe. Majd ezután megpróbáljuk R-nek ugyanezen Y értékű sorait beolvasni, és ezek a sorok már befférnek az M–1 pufferbe.

6.5.6. Az egyszerű rendezéses összekapcsolás elemzése

Amint azt a 6.17. példában megfigyeltük, algoritmusunk az argumentum reláció minden blokkjára öt lemez I/O-műveletet végez. Kivételt képezne az az eset, ha olyan sok sor lenne azonos Y értékkel, hogy a szóban forgó sorokat valamilyen speciális módon kellene összekapcsolnunk. Ebben az esetben a további lemez I/O-műveletek száma attól függ, hogy csak az egyik avagy mindkét reláció olyan sok azonos Y értékű sorral rendelkezik-e, hogy azok maguk már M – 1-nél több puffert igényelnek. Az összes esetet itt nem vesszük végig részletesen; a feladatokban megadunk néhány kidolgozásra szánt példát.

Azt is meg kell vizsgálnunk, hogy mekkorának kell M-nek lennie ahhoz, hogy az egyszerű rendezéses összekapcsolás működjön. Az elsődleges korlát az, hogy végre kell tudnunk hajtani R-en és S-en a kétfázisú, többutas összefésüléses rendezéseket. Ahogyan ezt a 2.3.4. részben már megvizsgáltuk, ezeknek a rendezéseknek az elvégzéséhez az szükséges, hogy $B(R) \leq M^2$ és $B(S) \leq M^2$ teljesüljön. Ha ezzel készen vagyunk, akkor már nem fogunk kifogyni a pufferekből, noha – amint már említettük – esetleg el kell majd térnünk az egyszerű összefésüléstől, ha az azonos Y értékkel rendelkező sorok nem férnek be M pufferbe. Összefoglalva, ha ilyen bonyodalmak nem merülnek fel, akkor:

- Az egyszerű rendezéses összekapcsolás $5\big(B(R) + B(S)\big)$ lemez I/O-műveletet használ.
- Működéséhez $B(R) \leq M^2$ és $B(S) \leq M^2$ teljesülése szükséges.

6.5.7. Egy hatékonyabb rendezésen alapuló összekapcsolás

Ha nem kell aggódnunk az összekapcsolási attribútumokon azonos értékkel bíró sorok igen nagy száma miatt, akkor blokkonként 2 lemez I/O-műveletet megtakaríthatunk azáltal, hogy a rendezések második fázisát kombináljuk magával az összekapcsolással. Az ilyen algoritmusokat egyszerűen rendezéses összekapcsolásnak hívjuk. További ismert elnevezések még az „összefésüléses összekapcsolás” és a „rendezéses-összefésüléses összekapcsolás”. Az $R(X, Y) |><| S(Y, Z)$ összekapcsolást M darab memóriablokkot használva a következőképpen számíthatjuk ki:

1. Y-t rendezési kulcsként használva mind R-re, mind S-re M méretű rendezett részlistákat hozunk létre.
2. Az egyes részlisták első blokkjait behozzuk egy-egy pufferbe, ehhez feltesszük, hogy összesen nincs M-nél több részlista.
3. A részlisták soron következő sorai között újra meg újra megkeressük a legkisebb Y értéket, y-t. Mindkét reláció sorai között beazonosítjuk az y értékkel rendelkezőket, ehhez esetleg betesszük azokat az M szabad puffer némelyikébe, feltéve, hogy M-nél kevesebb részlista van. A kimenetbe tesszük az összes olyan R-beli és S-beli sorok összekapcsolását, amelyek az Y attribútum(ok)on y értékkel rendelkeznek. Ha közben bármelyik részlista puffere kiürül, akkor azt lemezről ismét feltöltjük.

6.18. példa: Tekintsük ismét a 6.14. példabeli feladatot: 101 puffer használatával szeretnénk összekapcsolni az egyenként 1000, illetve 500 blokkos R és S relációkat. R-et és S-et felosztjuk 10, illetve 5 darab, egyenként 100

blokk hosszúságú részlistára, majd ezeket rendezzük.[1] Ezután 15 puffert a részlisták aktuális blokkjainak a befogadására használunk. Ha olyan helyzettel kerülünk szembe, ahol sok sornak van azonos Y értéke, ott a fennmaradó 86 puffert használhatjuk ezeknek a soroknak a tárolására. Ha még ennél is több ilyen sor van, akkor valamilyen speciális algoritmust kell használnunk, mint amilyen pl. a 6.5.5. rész végén szerepelt.

Feltéve, hogy az algoritmust nem kell módosítanunk a sok azonos Y értékű sor miatt, adatblokkonként három lemez I/O-műveletet kell végeznünk. Ezek közül kettő a rendezett részlisták létrehozására szolgál. Ezt követően az egyes rendezett részlisták minden egyes blokkját még egyszer beolvassuk a memóriába a többutas összefésülési folyamatban. A lemez I/O-műveletek teljes száma így tehát 4500. □

A fenti rendezéses összekapcsolási algoritmus – amennyiben alkalmazható – hatékonyabb a 6.5.5. rész algoritmusánál. A 6.18. példa kapcsán megjegyeztük, hogy a lemez I/O-műveletek száma $3\big(B(R) + B(S)\big)$. Az algoritmust használhatjuk olyan adatokon, amelyek mérete megközelíti az előző algoritmusét. A rendezett részlisták mérete M blokk, és összesen legfeljebb M darab lehet belőlük. A $B(R) + B(S) \leq M^2$ korlát ennél fogva elégséges.

Elgondolkozhatunk rajta, hogy esetleg elkerülhetők-e mindazok a bonyodalmak, amelyek a nagyszámú azonos Y értékkel rendelkező sor esetén merülnek fel. A következőket érdemes számításba venni:

1. Néha biztosak lehetünk benne, hogy a probléma fel sem merül. Ha például R-nek Y egy kulcsa, akkor egy adott y Y érték R részlistáinak összes blokkjai között csak egyszer fordulhat elő. Amikor éppen y van soron, akkor az R-beli sort a helyén hagyhatjuk, és összekapcsolhatjuk azt S összes megfelelő sorával. Ha a folyamat során S részlistáinak blokkjai kiürülnek, úgy puffereik feltölthetők a következő blokkal. Ekkor egyáltalán nincs szükség pluszhelyre, R és S akárhány sora rendelkezik is az adott y Y értékkel. Ha Y R helyett S-nek kulcsa, akkor a gondolatmenet megismételhető R és S felcserélésével.
2. Ha $B(R) + B(S)$ sokkal kisebb M^2-nél, akkor az azonos Y értékkel rendelkező sorok számára rengeteg kihasználatlan pufferünk lesz, mint azt a 6.18. példa is jelezte.
3. Ha máshogy semmiképpen nem boldogulunk, akkor használhatunk egy beágyazott ciklusú összekapcsolást, leszűkítve az azonos Y értékkel rendelkező sorokra, ami ugyan plusz lemez I/O-műveleteket igényel, de a feladatot kifogástalanul elvégzi. Ezt a lehetőséget a 6.5.5. részben tárgyaltuk.

6.5.8. A rendezésen alapuló algoritmusok összefoglalása

A 6.16. ábra a 6.5. részben ismertetett algoritmusok összefoglaló táblázatát mutatja. A 6.5.5. és a 6.5.7. részekben elmondottak értelmében akkor van szükség az idő- és a memóriakövetelmények módosítására, ha két olyan relációt kapcsolunk össze, amelyek nagyszámú sorára nézve az összekapcsolási attribútumok azonos értéket vesznek fel.

Operátor	*Szükséges* M *kb.*	*Lemez I/O-művelet*	*Rész*
γ, δ	$\sqrt{B}$	$3B$	6.5.1., 6.5.2.
∪, ∩, −	$\sqrt{B\big(R+B(S)\big)}$	$3\big(B(R)+B(S)\big)$	6.5.3., 6.5.4.
|><|	$\sqrt{\max\big(B(R),B(S)\big)}$	$5\big(B(R)+B(S)\big)$	6.5.5.
|><|	$\sqrt{\big(B(R)+B(S)\big)}$	$3\big(B(R)+B(S)\big)$	6.5.7.

6.16. ábra. A rendezés alapú algoritmusok memória- és lemez I/O-művelet követelménye

[1] Technikailag úgy is elrendezhettük volna a részlistákat, hogy mindegyiknek 101 blokk legyen a hossza, továbbá R és S utolsó részlistája 91, illetve 96 blokk hosszú legyen, de a költség ebben az esetben is pontosan ugyanannyi lenne.