6.6. Tördelésen alapuló kétmenetes algoritmusok

A tördelésen alapuló algoritmusok családja szintén a 6.5. részben bemutatott problémákat igyekszik megoldani. Az ilyen típusú algoritmusok mögött meghúzódó alapötlet a következő: ha az adatok mennyisége túl nagy ahhoz, hogy azokat az elsődleges memóriapuffereiben tároljuk, akkor végezzünk tördelést az argumentum(ok) összes sorára egy megfelelő tördelőkulcs segítségével. Az összes szokásos műveletre kiválasztható a tördelőkulcs oly módon, hogy a művelet végrehajtásakor együtt tekintendő soroknak ugyanaz legyen a tördelési értéke.

Ezt követően úgy végezzük el a műveletet, hogy egyszerre csak egy kosárral dolgozunk (illetve bináris műveletek esetén az azonos tördelési értékkel rendelkező kosárpárokon dolgozunk). Ezzel elérhetjük azt, hogy az operandus(ok) méretét a kosarak számával arányos mértékben csökkentjük. Ha a rendelkezésre álló pufferek száma M, akkor választhatjuk M-et a kosarak számának, és ezáltal egy M-es szorzóval növelhetjük az általunk kezelhető relációk méretét. Vegyük észre, hogy a 6.5. rész rendezésen alapuló algoritmusai az előzetes feldolgozással szintén egy M-es szorzót nyernek, viszont a rendezéses és a tördeléses megközelítések a hasonló mértékű megtakarítást egészen másféle módon érik el.

6.6.1. Relációk particionálása tördeléssel

Kezdjük a vizsgálódást azzal, hogy áttekintjük, miként particionálnánk az R relációt M–1 darab nagyjából egyforma méretű kosárba, M puffer használatával. Feltesszük, hogy a h tördelőfüggvény argumentumait az R teljes sorai alkotják (azaz R összes attribútuma a tördelőkulcs részét képezi). Minden kosárhoz hozzárendelünk egy puffert. Az utolsó puffer fogadja az R blokkjait, egyszerre csak egyet. A blokk minden egyes t sorát a $h(t)$ kosárba tördeljük, majd bemásoljuk a megfelelő pufferbe. Ha a szóban forgó puffer már tele van, akkor az eredményt kiírjuk lemezre, és ugyanahhoz a kosárhoz egy másik blokkot inicializálunk. Végül minden egyes blokk utolsó kosarát is kiírjuk lemezre, ha az adott kosár nem üres. Az algoritmus további részleteit a 6.17. ábrán láthatjuk. Vegyük észre, hogy noha a sorok változó hosszúságúak lehetnek, az algoritmus azt feltételezi, hogy azok mindig befférnek egy üres pufferbe.

```
inicializáljunk M-1 kosarat M-1 üres puffer felhasználásával;
FOR az R minden egyes b blokkjára DO BEGIN
   olvassuk be a b blokkot az M-edik pufferbe;
   FOR a b blokk minden egyes t sorára DO BEGIN
      IF a h(t) kosárban nincs hely a t számára THEN
         BEGIN
            másoljuk ki a puffert lemezre;
            inicializáljunk egy új üres blokkot a pufferben;
         END;
      másoljuk a t sort h(t) kosárhoz tartozó pufferbe;
   END;
END;
FOR minden egyes kosárra DO
   IF az adott kosárhoz tartozó puffer nem üres THEN
      írjuk ki lemezre az adott puffert;
```

6.17. ábra. *Az R reláció particionálása M–1 kosárba.*

6.6.2. Egy tördelésen alapuló algoritmus az ismétlődések kiküszöbölésére

A továbbiakban a relációs algebra olyan műveleteinek tördelésen alapuló algoritmusait fogjuk megvizsgálni, amelyek kétmenetes algoritmusokat igényelhetnek. Foglalkozzunk először az ismétlődések kiküszöbölésével, vagyis a $\delta(R)$ művelettel. Az R relációt M–1 kosárba tördeljük, amint az a 6.17. ábrán látható. Figyeljük meg, hogy az egyforma t sorok ugyanabba a kosárba kerülnek. A δ művelet így rendelkezik a következő, számunkra lényeges tulajdonsággal: a kosarakat vizsgálhatjuk egyenként, végrehajthatjuk δ-t egyenként az éppen aktuális kosárra, majd válaszként képezhetjük a $\delta(R_i)$-k egyesítését, ahol R_i az R reláció azon része, amelyik tördeléskor az i-edik kosárba kerül. A 6.3.2. rész egymenetes algoritmusa segítségével minden egyes R_i-ből kiküszöbölhetjük az ismétlődéseket, majd a kapott egyedi sorokat kiírjuk a kimenetre.

A módszer mindaddig működik, amíg az R_i-k egyenként elég kicsik ahhoz, hogy beférjenek a memóriába, és így

lehetővé tegyék egymenetes algoritmus használatát. Minthogy azt feltételeztük, hogy a h tördelőfüggvény az R relációt egyforma méretű kosarakba tördeli, valamennyi R_i mérete hozzávetőleg $B(R)/(M–1)$ blokk lesz. Ha ez a szám nem nagyobb M-nél, azaz ha $B(R) \leq M(M–1)$, akkor a kétmenetes, tördelésen alapuló algoritmus működni fog. Amint azt a 6.3.2. részben megmutattuk, valójában csak annyi kell, hogy az egy kosárban található különböző sorok beférjenek M darab pufferbe, viszont abban nem lehetünk biztosak, hogy vannak-e egyáltalán ismétlődések. Így a $B(R) \leq M^2$ becslés, ahol M-et és $M–1$-et egyszerűen azonosnak vettük, megegyezik a δ művelet rendezésen alapuló, kétmenetes algoritmusánál adott becsléssel.

A lemez I/O-műveletek száma ugyancsak hasonló a rendezésen alapuló algoritmuséhoz. Az R minden blokkját egyszer olvassuk be a sorok tördelésénél, és minden kosár valamennyi blokkját kiírjuk lemezre. Ezt követően az egyes kosarak blokkjait ismételten beolvassuk annál az egymenetes algoritmusnál, amely az adott kosarat dolgozza fel. A lemez I/O-műveletek teljes száma tehát $3B(R)$.

6.6.3. Egy tördelésen alapuló algoritmus a csoportosításra és az összesítésre

A $\gamma_L(R)$ művelet végrehajtásához megint csak úgy kezdünk hozzá, hogy R összes sorát $M–1$ kosárba tördeljük. Most azonban ahhoz, hogy ugyanazon csoport összes sora ugyanabba a kosárba kerüljön, olyan tördelőfüggvényt kell választanunk, amely kizárólag az L lista csoportosító attribútumaitól függ.

Ha az R relációt kosarakba particionáltuk, akkor minden egyes kosárra külön-külön használhatjuk a γ művelet 6.3.2. részben megismert egymenetes algoritmusát. A 6.6.2. részben a δ kapcsán megbeszéltük, hogy az egyes kosarakat akkor tudjuk feldolgozni a memóriában, ha $B(R) \leq M^2$.

Ugyanakkor a második menetben az egyes kosarak feldolgozásánál csoportonként csak egy rekordra van szükségünk. Így tehát a kosarat egy menetben tudjuk kezelni még akkor is, ha annak mérete meghaladja M-et, feltéve, hogy azok a rekordok, amik a kosárban lévő csoportoknak felelnek meg, összesen nem igényelnek M-nél több puffert. Egy csoportnak megfelelő rekord rendszerint nem nagyobb R egy soránál. Ha ez így van, akkor $B(R)$-re jobb felső korlátot ad M^2 szorozva a csoportonkénti sorok átlagos számával.

Ennek következményeként, ha kevés csoport van, akkor tulajdonképpen a $B(R) \leq \leq M^2$ szabálynak megfelelő relációknál jóval nagyobb R relációt is képesek vagyunk kezelni. Másrészről viszont, ha M nagyobb a csoportok számánál, akkor nem tudjuk feltölteni az összes kosarat. Az R méretére tehát a tényleges korlátozás M-nek egy bonyolult függvénye; a $B(R) \leq M^2$ csak egy egyszerű becslés. Végül pedig figyeljük meg, hogy γ-ra a lemez I/O-műveletek száma δ-hoz hasonlóan $3B(R)$.

6.6.4. Az egyesítés, a metszet és a különbség tördelésen alapuló algoritmusai

Ha bináris műveletről van szó, akkor ügyelnünk kell arra, hogy mindkét argumentum sorainak tördeléséhez ugyanazt a tördelőfüggvényt használjuk. Az $R \cup_S S$ kiszámításánál például mind az R, mind az S relációt egyenként $M–1$ kosárba tördeljük, jelölje ezeket $R_1, R_2, \ldots, R_{M-1}$, illetve $S_1, S_2, \ldots, S_{M-1}$. Ezek után minden i-re vesszük R_i és S_i halmaz alapú egyesítését, és az eredményt kiírjuk a kimenetre. Vegyük észre, hogy ha egy sor R-ben és S-ben egyaránt előfordul, akkor adott i-re a t sort R_i-ben és S_i-ben is megtaláljuk. Ily módon, amikor ezen két kosár egyesítését vesszük, akkor a t sort csak egyszer írjuk a kimenetbe, és így nincs lehetőség a végeredményben ismétlődések bekerülésére. A $\cup_B$ művelet esetén a 6.3.3. részben bemutatott egyszerű multihalmaz-egyesítési algoritmus a művelet legalkalmasabb megközelítése.

R és S metszetének, illetve különbségének kiszámításakor a $2(M–1)$ darab kosarat pontosan ugyanúgy hozzuk létre, mint a halmaz alapú egyesítés esetében, majd a megfelelő kosárpárokra alkalmazzuk a megfelelő egymenetes algoritmust. Figyeljük meg, hogy itt az összes algoritmus lemez I/O-művelet igénye $B(R) + B(S)$. Ehhez a mennyiséghez hozzá kell még adni azt a blokkonkénti két lemez I/O-műveletet, amely a két reláció sorainak tördeléséhez és a kosarak lemezen való tárolásához szükséges, így a lemez I/O-műveletek száma összesen $3\big(B(R) + B(S)\big)$.

Az algoritmusok működéséhez az szükséges, hogy vehessük R_i és S_i egymenetes egyesítését, metszetét vagy különbségét, amelyeknek mérete körülbelül $B(R)/(M–1)$, illetve $B(S)/(M–1)$. Emlékezzünk rá, hogy ezen műveletek egymenetes algoritmusaihoz az kell, hogy a kisebbik operandus legfeljebb $M–1$ blokkot foglaljon el. Így a tördelésen alapuló kétmenetes algoritmusokhoz jó közelítéssel a $\min\big(B(R),B(S)\big) \leq M^2$ feltételnek kell teljesülnie.

6.6.5. A tördeléses összekapcsolási algoritmus

Ahhoz, hogy $R(X, Y)$ |><| $S(Y, Z)$ összekapcsolást egy tördelésen alapuló kétmenetes algoritmussal számíthassuk ki, majdnem ugyanazt kell tennünk mint a 6.6.4. részben vizsgált többi bináris műveletnél. Az egyetlen különbség az, hogy tördelőkulcsként csak az összekapcsolási attribútumot, Y-t kell használnunk. Ekkor ugyanis biztosak lehetünk benne, hogy ha R és S sorai összekapcsolódnak, akkor adott i-re a megfelelő R_i és S_i kosarakba fognak kerülni. Ezt a *tördeléses összekapcsolás*[1] nevű algoritmust az egymáshoz rendelt kosárpárok egymenetes összekapcsolása teszi teljessé.

6.19. példa: Térjünk vissza a 6.14. példában megismert R és S relációk tárgyalására; ezek mérete egyenként 1000 és 500 blokk, a rendelkezésre álló elsődleges memória-pufferek száma pedig 101. Megtehetjük, hogy mindkét relációt egyenként 100 kosárba tördeljük, azaz R és S esetében az átlagos kosárméret 10, illetve 5 blokk. Mivel a kisebbik szám – ami most 5 – jóval kisebb a rendelkezésre álló pufferek számánál, a kosárpárok egymenetes összekapcsolása várhatóan nem fog akadályba ütközni.

A lemez I/O-műveletek száma az R és S kosarakba történő tördelése közben végzett beolvasáskor 1500; majd újabb 1500 I/O-műveletet kell a kosarak lemezre írásához, végül pedig 1500 I/O-művelet szükséges az egyes kosárpárok memóriába történő beolvasásához, amikor a megfeleltetett kosarak egymenetes összekapcsolását végezzük. A szükséges lemez I/O-műveletek száma tehát 4500, pont úgy mint a 6.5.7. szakasz hatékony rendezéses összekapcsolásánál. □

A 6.19. példát általánosítva kimondhatjuk, hogy:

- A tördeléses összekapcsoláshoz $3\big(B(R) + B(S)\big)$ lemez I/O-művelet kell.
- A kétmenetes tördeléses összekapcsolás addig működik, amíg $\min\big(B(R), B(S)\big) \leq M^2$.

Az utóbbi kijelentés ugyanúgy indokolható, mint a többi bináris műveletnél: a kosárpárok egyik tagjának be kell férnie M–1 darab pufferbe.

[1] Néha a 'tördeléses összekapcsolás' kifejezést a 6.3.3. részben bemutatott egymenetes összekapcsolási algoritmus egy olyan változatának tartják fenn, amelyben a tördelőtáblát elsődleges memóriastruktúraként használják. Ilyenkor az itt bemutatott kétmenetes tördeléses összekapcsolás algoritmusra 'particionált tördeléses összekapcsolás' néven utalnak.