6.9. Több mint kétmenetes algoritmusok

Tudjuk, hogy két menet a műveletek számára elegendő, ha a relációk nem nagyon nagyok. Vegyük azonban észre, hogy a 6.5. és 6.6. részekben tárgyalt technikák olyan algoritmusokká általánosíthatók, amelyek tetszőleges méretű relációkra alkalmazhatók, szükség esetén több menetet használva. Ebben a szakaszban mind a rendezésen alapuló, mind a tördelésen alapuló módszerek általánosítását tárgyalni fogjuk.

6.9.1. Többmenetes, rendezésen alapuló algoritmusok

A 2.3.5. részben utaltunk rá, hogy a kétfázisos, összefésüléses rendezést hogyan lehet kiterjeszteni hárommenetes algoritmussá. Van egy egyszerű rekurzív megközelítése a módszernek, amellyel tetszőlegesen nagy relációt rendezni tudunk. A rendezést úgy is el tudjuk végezni, hogy teljesen rendezzük a relációt, vagy ha arra van szükségünk, akkor n darab rendezett részlistát is létre tudunk hozni vele, tetszőleges n-re.

Tegyük fel, hogy az R reláció rendezéséhez rendelkezésünkre áll M darab puffer. Azt is feltesszük még, hogy a relációt nyaláboltan tároltuk. Ekkor a következőket kell tennünk:

Alap: Ha R belefér az M darab blokkba (vagyis ha $B(R) \leq M$), akkor beolvassuk R-et a memóriába, rendezzük valamilyen rendezési algoritmussal, majd a rendezett relációt kiírjuk a lemezre.

Indukció: Ha R nem fér be a memóriába, akkor osszuk fel R blokkjait M csoportba. Az egyes csoportokat jelöljük R_1, R_2, …, R_M-mel. Rendezzük rekurzívan R_i-t minden i-re ($i = 1, 2, \ldots, M$). Ezután fésüljük össze az M darab rendezett részlistát, ahogyan azt a 2.3.4. részben láttuk.

Ha nem csupán rendeznünk kell R-et, hanem valamilyen egyoperandusú (unáris) műveletet szeretnénk végrehajtani rajta, mint amilyen pl. a γ vagy a δ, akkor módosítsuk a fentieket oly módon, hogy az utolsó összefésüléskor a rendezett részlisták elején levő sorokra elvégezzük a megfelelő műveletet. Vagyis,

- δ esetén minden sornak egy példányát kiírjuk, a többi előfordulását pedig figyelmen kívül hagyjuk.
- γ esetén csak a csoportképző attribútumok szerint rendezünk, és azokat a sorokat, amelyek ezeken az attribútumokon megegyeznek, a szokásos módon összesítjük, ahogyan azt a 6.5.2. részben láttuk.

Ha egy kétoperandusú (bináris) műveletet szeretnénk elvégezni, pl. metszet vagy összekapcsolás, akkor tulajdonképpen ugyanezt az ötletet alkalmazzuk, azzal a különbséggel, hogy először a két relációt összesen M részlistába osztjuk szét. Ezután minden részlistát a fenti rekurzív módszerrel rendezünk. Végül beolvassuk az M darab részlistát a pufferekbe, és elvégezzük a megfelelő műveletet úgy, ahogyan azt a 6.5. rész megfelelő részében bemutattuk.

Az M darab puffert tetszőlegesen oszthatjuk szét az R és S relációk között. A menetek számát azonban úgy minimalizálhatjuk, ha a puffereket a relációk méretével arányosan osztjuk szét. Vagyis R kap $M \times B(R)/\big(B(R) + B(S)\big)$ darab puffert, a többit pedig S-hez rendeljük.

6.9.2. Többmenetes, rendezésen alapuló algoritmusok műveletigénye

Az alábbiakban megvizsgáljuk, hogy milyen összefüggés van a szükséges lemez I/O- műveletek száma, a relációk mérete és a memória mérete között. Jelöljük $s(M, k)$-val annak a legnagyobb relációnak a méretét, amelyet M darab puffer segítségével k menetben rendezni tudunk. Ekkor $s(M, k)$-t a következőképpen számíthatjuk ki:

Alap: Ha $k = 1$, vagyis 1 menet elegendő, akkor szükségképpen $B(R) \leq M$. Másképpen megfogalmazva, $s(M, 1) = M$.

Indukció: Tegyük fel, hogy $k > 1$. Ekkor felosztjuk R-et M részre, ahol szükségképpen minden rész k–1 menet alatt rendezhető. Ha $B(R) = s(M, k)$ akkor $s(M, k)/M$, ami az egyes részek mérete, nem lehet nagyobb mint $s(M, k–1)$. Vagyis $s(M, k) = M\, s(M, k – 1)$.

Ha a fenti rekurziót tovább folytatjuk, a következőt kapjuk:

$s(M, k) = M\, s(M, k–1) = M^2\, s(M, k–2) = \ldots = M^{k–1}\, s(M, 1)$

Mivel $s(M, 1) = M$, ebből azt kapjuk, hogy $s(M, k) = M^k$. Ez azt jelenti, hogy k menet alatt akkor tudunk egy R relációt rendezni, ha $B(R) \le s(M, k)$, ami azt jelenti, hogy $B(R) \le M^k$. Másképpen megfogalmazva, ha egy R relációt k menet alatt akarunk rendezni, akkor ehhez a minimálisan szükséges memóriapufferek száma: $M = B(R)^{1/k}$.

Egy rendezési algoritmus minden menete beolvassa az összes adatot, majd ismét kiírja azokat lemezre. Így egy k menetes rendező algoritmusnak $2k\, B(R)$ lemez I/O-műveletre van szüksége.

Most vizsgáljuk meg egy többmenetes $R(X, Y)$ |><| $S(Y, Z)$ összekapcsolás költségét, ami jó példája a kétoperandusú műveleteknek. Legyen $j(M, k)$ az a maximális blokkszám, amely mellett össze tudunk kapcsolni két relációt k menetben, M darab puffer használatával, ha a relációknak összesen legfeljebb $j(M, k)$ blokkja van. Ez azt jelenti, hogy az összekapcsolás elvégezhető ha $B(R) + B(S) \le j(M, k)$.

Az utolsó menetben összefésüljük a két reláció M darab rendezett részlistáját. A részlisták mindegyikét k–1 menetben rendeztük, így egyikük hossza sem lehet több, mint $s(M, k–1)$, vagyis az összméret maximum $Ms(M, k) = M^k$. Eszerint $B(R) + B(S)$ nem lehet nagyobb, mint M^k, vagyis $j(M, k) = M^k$. A paraméterek szerepét felcserélve azt is kimondhatjuk, hogy a k menetes összekapcsoláshoz legalább $(B(R) + B(S))^{1/k}$ darab pufferre van szükség.

A lemez I/O-műveletek összeszámolásakor ne feledjük, hogy az összekapcsolásnál és más relációs műveleteknél a végeredmény lemezre írásának költségét nem számoljuk, ellentétben a rendezésnél alkalmazott gyakorlattal. Így a részlisták rendezéséhez $2(k–1)(B(R) + B(S))$ lemez I/O-műveletet használunk, míg a végső rendezett részlisták beolvasásához további $B(R) + B(S)$ darabot. A végeredmény összesen $(2k–1)(B(R) + B(S))$ lemez I/O-művelet.

6.9.3. Többmenetes, tördelésen alapuló algoritmusok

Van egy hasonló, rekurzív megközelítése a tördelésen alapuló műveleteknek is, ha azokat nagyon nagy relációkon végezzük. Ha M a rendelkezésre álló memóriapuffek száma, akkor osszuk fel a relációt a tördeléssel M–1 kosárba. Ezután alkalmazzuk a műveletet az egyes kosarakra egyenként, amennyiben a művelet egyoperandusú. Ha a művelet kétoperandusú, mint pl. egy összekapcsolás, akkor a megfelelő kosarakból álló párokra alkalmazhatjuk azt, mintha azok maguk volnának a teljes relációk. Az eddig tárgyalt relációs műveletek esetén – ismétlődések eltüntetése, csoportosítás, egyesítés, metszet, különbség, természetes összekapcsolás, egyenlőséges összekapcsolás – a teljes relációkra alkalmazott művelet végeredménye meg fog egyezni a kosarakra alkalmazott műveletek egyesítésével. Ezt a megközelítést rekurzívan a következőképpen írhatjuk le:

Alap: Egyoperandusú művelet esetén, ha a reláció belefér az M pufferbe, akkor olvassuk be a memóriába, és végezzük el a műveletet. Kétoperandusú művelet esetén, ha bármelyik reláció befér M–1 pufferbe, akkor végezzük el a műveletet úgy, hogy ezt a relációt beolvassuk a memóriába, majd a másik relációt blokkonként beolvassuk az M-edik pufferbe.

Indukció: Ha egyik reláció sem fér be a memóriába, akkor mindkettőt osszuk fel tördeléssel M–1 kosárba úgy, ahogyan azt a 6.6.1. részben láttuk. Végezzük el a műveletet rekurzívan mindegyik kosárra, illetve a megfelelő kosarakból álló párokra, majd gyűjtsük össze a kosarakból, illetve a párokból készülő eredmény kimenetét.

6.9.4. Többmenetes, tördelésen alapuló algoritmusok műveletigénye

A továbbiakban azzal a feltételezéssel fogunk élni, hogy a reláció tördelésekor a sorok a lehető legegyenletesebben oszlanak meg a kosarak között. A gyakorlatban ezt a feltételezést megközelíthetjük, ha tényleg véletlenszerűen tördelőfüggvényt választunk, de valójában mindig lesz bizonyos egyenletlenség a sorok eloszlása tekintetében.

Először nézzük az egyoperandusú műveleteket, mint pl. a γ vagy a δ. A relációt továbbra is R-rel, a memóriapufferek számát pedig M-mel jelöljük. Legyen $u(M, k)$ annak a legnagyobb relációnak a blokkszáma, amelyet egy k menetes tördelő algoritmus kezelni tud. u-t a következőképpen határozhatjuk meg rekurzívan:

Alap: $u(M, 1) = M$, hiszen az R relációnak be kell férnie az M pufferbe, vagyis $B(R) \le M$.

Indukció: Tegyük fel, hogy az első lépés az R relációt M – 1 egyenlő méretű kosárba osztja szét. Ekkor $u(M, k)$-t a következőképpen számolhatjuk ki. A kosaraknak a következő lépés előtt elég kicsinek kell lenniük ahhoz, hogy

őket k–1 lépésben kezelni lehessen, vagyis a kosarak mérete legfeljebb $u(M, k–1)$. Mivel R-et M–1 kosárba osztottuk, így azt kapjuk, hogy $u(M, k) = (M–1)u(M, k–1)$.

Ha tovább folytatjuk a fenti gondolatmenetet, akkor azt kapjuk, hogy $u(M, k) = M(M–1)^{k–1}$, illetve feltételezve, hogy M elegendően nagy, u-ra közelítőleg a következő adódik: $u(M, k) = M^k$. Ez másképpen megfogalmazva azt jelenti, hogy akkor tudjuk az R reláción M puffer segítségével elvégezni az egyoperandusú műveleteket k menetben, ha $M \leq (B(R))^{1/k}$.

Hasonló elemzést végezhetünk a kétoperandusú műveletekre is. Most is az összekapcsolást fogjuk példaképpen venni, mint a 6.9.2. részben. Jelölje $j(M, k)$ a $R(X, Y) |><| S(Y, Z)$ összekapcsolásban résztvevő R és S relációk közül a kisebbiknek a méretére vonatkozó felső korlátot. Most is M jelöli a rendelkezésre álló memóriapufferek számát, k pedig a menetek számát.

Alap: $j(M, 1) = M–1$, vagyis ha az egy menetes algoritmust használjuk az összekapcsolásra, akkor vagy R-nek vagy S-nek be kell férnie $M – 1$ pufferbe. Ezt már láttuk a 6.3.3. részben.

Indukció: $j(M, k) = (M–1)j(M, k–1)$, hiszen az első menetben mindkét relációt M–1 kosárba osztjuk, és elvárásaink szerint az egyes kosarak mérete az eredeti relációnak $1/(M–1)$-ed része lesz, továbbá azt is tudjuk, hogy a megfelelő kosarakból álló párokra a műveletet k–1 menetben el kell tudnunk végezni.

A fenti gondolatmenetet folytatva azt kapjuk, hogy $j(M, k) = (M–1)^k$. Ismét feltételezve, hogy M elegendően nagy, j-re a következő közelítés adódik: $j(M, k) = M^k$. Ez azt jelenti, hogy akkor tudjuk az $R(X, Y) |><| S(Y, Z)$ összekapcsolást elvégezni k mentben M puffer segítségével ha $M^k \geq \min(B(R), B(S))$.