4. PROCESSZORSZÁM MINIMALIZÁLÁSA

Ebben a fejezetben egy olyan ütemezési feladatot tárgyalunk, amelyben a taszkok határidőre való végrehajtásához szükséges processzorok számának minimalizálása a cél.

A feladat a következő: $\pi = (\rho, \tau, \mu)$, $\rho = \{\mathcal{P}\}$, $\mathcal{P} = \{P_1,\ P_2, \ldots\}$, $\tau = (\mathcal{T}, \mathbf{t}, \mathbf{d})$, $\mu = N_{proc}$, a processzorok azonosak, a taszkok megszakíthatatlanok, a határidők szigorúak és $\mathbf{d} = (1, 1, \ldots)$, a végrehajtási időkre pedig teljesül $0 < t_i \leq 1$ $(i = 1,\ 2, \ldots, n)$.

Ebben az anyagban technikai okok miatt a feladatot olyan memóriakezelési feladatként tárgyaljuk, amelyben ismert méretű fájlokat kell elhelyezni adott méretű lemezeken és a cél a felhasznált lemezek számának minimalizálása.

Adott a fájlok n száma, valamint az elhelyezendő fájlok méretét tartalmazó $\mathbf{t} = (t_1, t_2, \ldots, t_n)$ vektor, melynek elemeire teljesül $0 < t_i \leq 1$ $(i = 1, 2, \ldots, n)$. A fájlokat úgy kell elhelyezni a lemezeken, hogy nem szabad őket részekre bontani és figyelembe kell venni, hogy a lemezek kapacitása egységnyi.

4.1. Közelítő algoritmusok

Az adott feladat NP-teljes. A gyakorlatban ezért különböző közelítő algoritmusokat használnak.

Ezeknek az algoritmusoknak a bemenő adatai: a fájlok n száma és a fájlméretek $\mathbf{t} = \langle t_1, t_2, \ldots, t_n \rangle$ vektora. A kimenő adatok pedig a szükséges lemezek száma (*lemezszám*) és a lemezek $\mathbf{h} = \langle h_1, h_2, \ldots, h_n \rangle$ szintvektora.

A szereplő pszeudokódokban a kimenő adatokat nem adjuk meg paraméterként. Ezek a kódok nem csak a szükséges lemezek számát, hanem az egyes lemezek telítettségét is megadják.

4.1.1. Lineáris algoritmus (LF)

A lineáris algoritmus (**L**inear **F**it) szerint az F_i fájlt az L_i lemezen helyezzük el. LF pszeudokódja a következő.

LF(n, **t**)

```
1  for i ← 1 to n
2      do h[i] ← t[i]
3  lemezszám ← n
4  return lemezszám
```

Ennek az algoritmusnak mind a helyigénye, mind pedig a futási ideje $\Theta(n)$. Ha azonban a méretek beolvasását és a szintek nyomtatását az 1–2. sorokban lévő ciklusban oldjuk meg, akkor a helyigény $\Theta(1)$-re csökkenthető.

4.1.2. Egyszerű algoritmus (NF)

Az egyszerű algoritmus (**N**ext **F**it) addig rakja a fájlokat a soron következő lemezre, amíg lehet. Pszeudokódja a következő.

NF(n, **t**)

```
1  h[1] ← t[1]
2  lemezszám ← 1
3  for i ← 2 to n
4      do if h[lemezszám] + t[i] ≤ 1
5             then h[lemezszám] ← h[lemezszám] + t[i]
6             else lemezszám ← lemezszám + 1
7                  h[lemezszám] ← t[i]
8  return lemezszám
```

Ennek a algoritmusnak mind a helyfoglalása, mind pedig a futási ideje $\Theta(n)$. Ha a futási idők beolvasását és a szintek kivitelét a 3–7. sorokban lévő ciklusban oldjuk meg, akkor a helyfoglalás $\Theta(1)$-re csökkenthető, a futási idő azonban $\Omega(n)$.

4.1.3. Mohó algoritmus (FF)

A mohó algoritmus (**F**irst **F**it) a fájlokat rendre az első olyan lemezen helyezi el, amelyikre ráférnek.

FF(n, **t**)

```
1  lemezszám ← 1
2  for i ← 1 to n
3      do h[i] ← 0
```

```
4  for i ← 1 to n
5      do k ← 1
6         while t[i] + h[k] > 1
7               do k ← k + 1
8         h[k] ← h[k] + t[i]
9         if k > lemezszám
10          then lemezszám ← lemezszám + 1
11 return lemezszám
```

Ennek az algoritmusnak a helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$. Ha például minden fájlméret 1, akkor az algoritmus futási ideje $\Theta(n^2)$.

4.1.4. Gazdaságos algoritmus (BF)

A gazdaságos algoritmus (**B**est **F**it) a fájlokat rendre a legelső olyan lemezre rakja, ahol a lehető legkisebb szabad kapacitás marad.

BF(n, **t**)

```
 1  lemezszám ← 1
 2  for i ← 1 to n
 3      do h[i] ← 0
 4  for i ← 1 to n
 5      do szabad ← 1.0
 6         ind ← 0
 7         for k ← 1 to lemezszám
 8             do  if h[k] + t[i] ≤ 1 és 1 − h[k] − t[i] < szabad
 9                   then ind ← k
10                        szabad ← 1 − h[k] − t[i]
11                if ind > 0
12                   then h[ind] ← h[ind] + t[i]
13                        elselemezszám ← lemezszám + 1
14                   h[lemezszám] ← t[i]
15 return lemezszám
```

Ennek az algoritmusnak a helyigénye $\Theta(n)$, futási ideje pedig $O(n^2)$.

4.1.5. Párosító algoritmus (PF)

A párosító algoritmus (**P**airwise **F**it) a méretvektor első és utolsó eleméből rendre párt képez, és a két fájlt – a két méret összegének megfelelően – egy, illetve két lemezre rakja.

A pszeudokódban két segédváltozó van: *bind* az aktuális pár első elemének indexe, *eind* pedig az aktuális pár második elemének indexe.

```
PF(n, t)
 1  lemezszám ← 0
 2  bind ← 1
 3  eind ← n
 4  while eind ≥ bind
 5        do if eind − bind ≥ 1
 6              then if t[bind] + t[eind] > 1
 7                        then lemezszám ← lemezszám + 2
 8                             h[lemezszám + 1] ← t[bind]
10                             h[lemezszám] ← t[eind]
11                   else lemezszám ← lemezszám + 1
12                        h[lemezszám] ← t[bind] + t[eind]
13           if eind = bind
14              then lemezszám ← lemezszám + 1
15                   h[lemezszám] ← t[eind]
16                   bind ← bind + 1
17                   eind ← eind − 1
18  return lemezszám
```

Ennek az algoritmusnak a helyigénye $\Theta(n)$, futási ideje $\Theta(n)$. Ha azonban a fájlméretek beolvasását és a lemezszintek kivitelét online módon megoldjuk, a helyigény csak $\Theta(1)$.

4.1.6. Rendező egyszerű algoritmus (NFD)

A következő öt algoritmus két részből áll: először a végrehajtási idők alapján csökkenő sorrendbe rendezik a taszkokat (a rendezés eredménye az **s** vektor), majd a második részben a rendezett taszkokat ütemezik.

A rendező egyszerű algoritmus (**N**ext **F**it **D**ecreasing) a rendezés után az egyszerű algoritmus (NF) szerint dolgozik. Így mind helyigénye, mind pedig időigénye az alkalmazott rendező algoritmus és NF megfelelő igényeiből tevődik össze.

4.1.7. Rendező mohó algoritmus (FFD)

A rendező mohó algoritmus (**F**irst **F**it **D**ecreasing) a rendezés után a mohó algoritmus (FF) szerint dolgozik, így helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$.

4.1.8. Rendező gazdaságos algoritmus (BFD)

A rendező gazdaságos algoritmus (**B**est **F**it **D**ecreasing) a rendezés után a gazdaságos algoritmus (BF) szerint dolgozik, így helyigénye $\Theta(n)$, időigénye pedig $O(n^2)$.

4.1.9. Rendező párosító algoritmus (PFD)

A rendező párosító algoritmus (**P**airwise **F**it **D**ecreasing) a rendezés után rendre párokat képez a legelső és legutolsó taszkokból, majd azokat lehetőleg egy processzorra (ha a végrehajtási idők összege nem nagyobb egynél) ütemezi. Ha ez nem lehetséges, akkor az adott párt két processzorra ütemezi.

4.1.10. Rendező gyors algoritmus (QFD)

A rendező gyors algoritmus (**Q**uick **F**it **D**ecreasing) a rendezés után rendre a legelső fájlt a soron következő üres lemezre teszi, majd ehhez a fájlhoz mindaddig hozzáteszi a lehető legnagyobb fájlt (ezeket rendre a rendezett méretsorozat végétől kezdi keresni), amíg ez lehetséges.

A pszeudokódban használt munkaváltozók: *bind* az első vizsgálandó fájl indexe, *eind* pedig az utolsó vizsgálandó fájl indexe.

QFD(n, **t**)

```
 1  bind ← 1
 2  eind ← n
 3  lemezszám ← 0
 4  while eind ≥ bind
 5        do lemezszám ← lemezszám + 1
 6           h[lemezszám] ← t[bind]
 7           bind ← bind + 1
 8        while eind ≥ bind és h[lemezszám] + s[eind] ≤ 1
 9              do ind ← eind
10              while ind > bind és h[lemezszám] + t[ind − 1] ≤ 1
11        do ind ← ind − 1
12  h[lemezszám] ← h[lemezszám] + t[ind]
13        if eind > ind
14          then for i ← ind to eind − 1
15                     do t[i] ← t[i + 1]
16        eind ← eind − 1
17  return lemezszám
```

Ennek a programnak a helyigénye $\Theta(n)$, futásideje kedvezőtlen esetben $\Theta(n^2)$, a gyakorlatban azonban – egyenletes eloszlású végrehajtási idők esetén – körülbelül $\Theta(n \lg n)$.

4.2. Optimális algoritmusok

4.2.1. Egyszerű hatvány típusú optimális algoritmus (SP)

Ez az algoritmus a fájlokat rendre – egymástól függetlenül – n lemez mindegyikére elhelyezi, így n^n elhelyezést állít elő, majd ezek közül kiválaszt egy optimálisat. Mivel ez az algoritmus minden elhelyezést előállít (feltéve, hogy két elhelyezést azonosnak tekintünk, ha minden lemezhez ugyanazokat a fájlokat rendelik), így biztosan megtalálja az egyik optimális elhelyezést.

4.2.2. Faktoriális típusú optimális algoritmus (FACT)

Ez az algoritmus rendre előállítja a fájlok permutációit (ezek száma $n!$), majd az így kapott listákat helyezi el a NF segítségével.

Az algoritmus optimalitása így látható be. Tekintsünk egy tetszőleges fájlrendszert és

annak egy $S_{\mathrm{OPT}}(\tau)$ optimális elhelyezését. $S_{\mathrm{OPT}}(\tau)$ alapján állítsuk elő a fájlok egy P permutációját úgy, hogy rendre felsoroljuk a $P_1, P_2, \ldots, P_{\mathrm{OPT}}(\tau)$ lemezekre elhelyezett fájlokat. Ha a P permutációt az NF algoritmussal helyezzük el, akkor vagy S_{OPT}-hoz jutunk, vagy egy másik optimális elhelyezéshez (bizonyos taszkok esetleg eggyel kisebb indexű processzorra kerülnek).

4.2.3. Gyors hatványtípusú optimális algoritmus (QP)

Ez az algoritmus (**Q**uick **P**ower) azzal próbálja SP időigényét csökkenteni, hogy a „nagy" fájlokat (melyek mérete nagyobb, mint 0.5) eleve külön lemezre helyezi, és csak a többi fájl (a „kicsi" fájlokat) próbálja mind az n lemezre elhelyezni. Így n^n helyett csak n^K elhelyezést állít elő, ahol K a kicsi fájlok száma.

4.2.4. Gazdaságos hatványtípusú optimális algoritmus (EP)

Ez az algoritmus (**E**conomic **P**ower) azt is figyelembe veszi (amellett, hogy két nagy fájl nem fér el egymás mellett), hogy két kis fájl mindig elfér egy lemezen. Ezért a nagy fájlok számát N-nel, a kicsikét K-val jelölve legfeljebb $N + \lfloor (K+1)/2 \rfloor$ lemezre van szüksége. Így először a nagy lemezeket ütemezzük külön lemezekre, majd a kicsiket a fenti számú lemez mindegyikére. Ha például $N = K = n/2$, akkor ezek szerint csak $(0.75n)^{0.5n}$ elhelyezést kell előállítanunk.

4.3. Listarövidítés (SL)

Bizonyos feltételek mellett igaz, hogy a $\mathbf{t}$ lista felbontható úgy $\mathbf{t}_1$ és $\mathbf{t}_2$ listákra, hogy $\mathrm{OPT}(\mathbf{t}_1) + \mathrm{OPT}(\mathbf{t}_2) \leq \mathrm{OPT}(\mathbf{t})$ (ilyen esetekben szükségképpen az egyenlőség teljesül). Ennek előnye, hogy a rövidebb listákat rendszerint rövidebb összidő alatt tudjuk optimálisan elhelyezni, mint az eredeti listát.

Tegyük fel például, hogy $t_i + t_j = 1$. Ekkor legyen $\mathbf{t}_1 = (t_i, t_j)$ és $\mathbf{t}_2 = \mathbf{t} \setminus \mathbf{t}_1$. Ekkor $\mathrm{OPT}(\mathbf{t}_1) = 1$ és $\mathrm{OPT}(\mathbf{t}_2) = \mathrm{OPT}(\mathbf{t}) - 1$. Tekintsük ugyanis egy optimális elhelyezés esetén azt a két lemezt, amelyekre a $\mathbf{t}_1$ lista elemei kerültek. Mivel mellettük legfeljebb $1-t_1$, illetve $1 - t_2$ összegű fájlok lehetnek, így azok helyfoglalásainak összege legfeljebb $2 - (t_1 + t_2)$, azaz 1. A listát egyszerre mindkét végén vizsgálva $O(n)$ idő alatt kiválaszthatjuk azokat a fájlpárokat, melyekre a végrehajtási idők összege 1. Ezután rendezzük a $\mathbf{t}$ listát. Legyen a rendezett lista $\mathbf{s}$. Ha például $s_1 + s_n > 1$, akkor az első fájl minden elhelyezésben külön lemezre kerül, és így $\mathbf{t}_1 = (t_1)$ és $\mathbf{t}_2 = (t_2, t_3, \ldots, t_n)$ jó választás.

Ha a rendezett listára $s_1 + s_n < 1$, de $s_1 + s_{n-1} + s_n > 1$, akkor legyen s_j a legnagyobb olyan listaelem, amelyet s_1-hez adva nem lépjük túl az egyet.

Ekkor $\mathbf{t}_1 = (t_1, t_j)$ és $\mathbf{t}_2 = \mathbf{t} \setminus \mathbf{t}_1$ választás mellett a $\mathbf{t}_2$ lista két elemmel rövidebb, mint a $\mathbf{t}$ lista volt.

Az utóbbi két művelettel gyakran lényegesen lerövidíthetőek a listák (szerencsés esetben úgy, hogy mindkét listára könnyen megkaphatjuk az optimális processzorszámot).

A rövidítés után megmaradó listát – például a korábbi algoritmusok valamelyikével – természetesen még fel kell dolgozni.

4.4. Becslések (ULE)

A felső és alsó becslésekre (**U**pper and **L**ower **E**stimations) támaszkodó algoritmusok a következőképpen működnek. Valamelyik közelítő algoritmussal előállítják az OPT($\mathbf{t}$) egy A($\mathbf{t}$) felső becslését, majd alulról is megbecsülik OPT($\mathbf{t}$) értékét. Erre alkalmasak például az elhelyezések azon tulajdonságai, hogy két nagy fájl nem helyezhető azonos lemezre, és hogy a méretek összege egyik lemezen sem lehet 1-nél több. Ezért mind a nagy fájlok száma, mind pedig a fájlméretek összege, így ezek MAX($\mathbf{t}$) maximuma is alsó becslést ad. Ha A($\mathbf{t}$) = MAX($\mathbf{t}$), akkor az A algoritmus optimális ütemezést állított elő. Ellenkező esetben például valamelyik időigényes optimumkereső algoritmussal folytathatjuk.

4.5. Algoritmusok páronkénti összehasonlítása

Ha egy ütemezési (vagy más) probléma megoldására több algoritmust ismerünk, akkor az algoritmusok összehasonlításának egyik egyszerű módja annak vizsgálata, hogy megadhatók-e a szereplő paraméterek értékei úgy, hogy a kiválasztott teljesítménymérték értéke az egyik algoritmus esetén kedvezőbb legyen, mint a másik algoritmus esetén.

A most vizsgált elhelyezési probléma esetén a $\mathbf{t}$ méretvektorhoz az A, illetve B algoritmus által rendelt lemezszámot A($\mathbf{t}$)-vel, illetve B($\mathbf{t}$)-vel jelölve azt vizsgáljuk, vannak-e olyan $\mathbf{t}_1$, illetve $\mathbf{t}_2$ vektorok, melyekre A($\mathbf{t}_1$) < B($\mathbf{t}_1$), illetve A($\mathbf{t}_2$) > B($\mathbf{t}_2$). Ezt a kérdést most az előbb definiált tíz közelítő és az optimális algoritmusra vizsgáljuk meg.

Az optimális algoritmusok definíciójából adódik, hogy minden $\mathbf{t}$-re és minden A algoritmusra OPT($\mathbf{t}$) ≤ A($\mathbf{t}$).

A továbbiakban a példavektorok elemei huszadok lesznek.

Tekintsük a következő nyolc listát:

$\mathbf{t_1}$ = (12/20, 6/20, 8/20, 14/20),
$\mathbf{t_2}$ = (8/20, 6/20, 6/20, 8/20, 6/20, 6/20),
$\mathbf{t_3}$ = (15/20, 8/20, 8/20, 3/20, 2/20, 2/20, 2/20),
$\mathbf{t_4}$ = (14/20, 8/20, 7/20, 3/20, 2/20, 2/20, 2/20, 2/20),
$\mathbf{t_5}$ = (10/20, 8/20, 10/20, 6/20, 6/20),
$\mathbf{t_6}$ = (12/20, 12/20, 8/20, 8/20),
$\mathbf{t_7}$ = (8/20, 8/20, 12/20, 12/20).
$\mathbf{t_8}$ = (14, 12, 8, 10).

Ezeknek a listáknak az elhelyezési eredményeit a 4.1. ábrán foglaltuk össze.

A 4.1. ábra mutatja, hogy az első listához LF 4, míg a többi algoritmus ennél kevesebb lemezt igényel. Továbbá azt is mutatja $\mathbf{t}_1$ lista sora, hogy FFD, BFD, PFD, QFD és OPT kevesebb lemezt igényel, mint NF, FF, BF, PF és NFD.

Természetes, hogy nincs olyan lista, amelyre bármelyik algoritmus kevesebb lemezt használna fel, mint OPT.

Az is közvetlenül adódik, hogy nincs olyan lista, melyhez az LF kevesebb lemezt használna fel, mint a többi tíz algoritmus közül bármelyik.

Ezeket a megállapításokat mutatja a 4.2. ábra. Az ábrán a főátlóban lévő X szimbólumok azt jelzik, hogy az egyes algoritmusokat önmagukkal nem hasonlítjuk össze. Az

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
$\mathbf{t}_1$	4	3	3	3	3	3	2	2	2	2	2
$\mathbf{t}_2$	6	2	2	2	3	3	3	3	3	3	2
$\mathbf{t}_3$	7	3	2	3	4	3	2	3	4	2	2
$\mathbf{t}_4$	8	3	3	2	4	3	3	2	4	3	2
$\mathbf{t}_5$	5	3	3	3	3	2	2	2	3	2	2
$\mathbf{t}_6$	4	3	2	2	2	3	2	2	2	2	2
$\mathbf{t}_7$	4	3	3	3	2	3	2	2	2	2	2

4.1. ábra. Lemezszámok összefoglalása.

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
LF	X	1	1	1	1	1	1	1	1	1	1
NF	–	X					1	1	1	1	1
FF	–		X				1	1	1	1	1
BF	–			X			1	1	1	1	1
PF	–				X		1	1	1	1	1
NFD	–					X	1	1	1	1	1
FFD	–						X				
BFD	–							X			
PFD	–								X		
QFD	–									X	
OPT	–	–	–	–	–	–	–	–	–	–	X

4.2. ábra. Algoritmusok páronkénti összehasonlítása.

első oszlopban lévő nagykötőjelek azt jelzik, hogy a sornak megfelelő algoritmushoz nincs olyan lista, melyet az algoritmus több lemez felhasználásával dolgozna fel, mint az oszlopnak megfelelő algoritmus, azaz LF.

Az utolsó sorban lévő nagykötőjelek pedig azt mutatják, hogy nincs olyan lista, melyhez az optimális algoritmus több lemezt igényelne, mint bármelyik másik vizsgált algoritmus.

Végül az egyesek azt jelzik, hogy a $\mathbf{t}_1$ listához az ábra adott mezője sorának megfelelő algoritmus több lemezt használ fel, mint a mező oszlopának megfelelő algoritmus.

Ha tovább folytatjuk a 4.2. ábra lemezszámainak elemzését, a 4.2. ábrát a 4.3. ábrává egészíthetjük ki.

Mivel az első sort és az első oszlopot már kitöltöttük, az LF algoritmussal nem foglalkozunk többet.

A $\mathbf{t}_2$ listához NF, FF, BF és OPT 2 lemezt használ, míg a többi 6 algoritmus hármat. Ezért a „győztesek” oszlopainak és a „vesztesek” sorainak metszéspontjaiba ketteseket írunk (a PF és OPT, valamint az NFD és OPT metszéspontjában azonban a már ott lévő egyest nem írjuk felül, így $4 \times 6 - 2 = 22$ mezőbe kerül kettes). Mivel OPT sorát és oszlopát kitöltöttük, ezért a pont további részében nem foglalkozunk vele.

A harmadik lista hátrányos PF és PFD számára, ezért a soraikban lévő üres mezőkbe hármasokat írunk. Ez a lista arra is példa, hogy NF rosszabb lehet, mint FF, BF rosszabb

lehet FF-nél, BFD az FFD-nél és a QFD-nél.

A negyedik listát csak BF és BFD tudja optimálisan – azaz két lemez felhasználásával – feldolgozni. Ezért a két algoritmus oszlopában a még üres mezőkbe négyest írhatunk.

Az ötödik listához NFD, FFD, BFD és QFD csak két, míg NF, FF, BF, PF és PFD három lemezt használ. Ezért a megfelelő mezőkbe ötös kerülhet.

A $\mathbf{t}_6$ lista „vesztesei" NF és NFD – ezért a soraikban még üresen álló mezőkbe hatost írunk.

A $\mathbf{t}_7$ lista feldolgozását PF jobban végzi, mint FF.

A $\mathbf{t}_8$ lista bizonyítja, hogy a PF lehet jobb, mint a PFD.

További mezők kitöltését segíti a következő

4.1. tétel. *Ha* $\mathbf{t} \in D$, *akkor*

$$\mathrm{FF}(\mathbf{t}) \leq \mathrm{NF}(\mathbf{t}) \ .$$

Bizonyítás. A lista hossza szerinti indukciót alkalmazunk.

Legyen $\mathbf{t} = \langle t_1, t_2, \ldots, t_n \rangle$ és $\mathbf{t_i} = \langle t_1, t_2, \ldots, t_i \rangle$ $(i = 1, 2, \ldots, n)$. Legyen $\mathrm{NF}(\mathbf{t}_i) = \mathrm{N}_i$ és $\mathrm{FF}(\mathbf{t}_i) = \mathrm{F}_i$, továbbá legyen n_i az *utolsó lemez szintje* NF szerint, azaz a legnagyobb indexű, nem üres lemezre tett állományok hosszainak összege akkor, amikor NF éppen feldolgozta $\mathbf{t}_i$-t. Hasonlóképpen legyen f_i az utolsó lemez szintje FF szerint.

A következő invariáns tulajdonság teljesülését bizonyítjuk minden i-re: vagy $F_i < N_i$, vagy $F_i = N_i$ és $f_i \leq n_i$.

Ha $i = 1$, akkor $F_1 = N_1$ és $f_1 = n_1 = t_1$, azaz az invariáns tulajdonság második része teljesül. Tegyük fel, hogy a tulajdonság teljesül az $1 \leq i < n$ értékre. Ha most a t_{i+1} elhelyezése előtt az invariáns tulajdonság első része teljesült, akkor vagy érvényes marad az $F_i < N_i$ egyenlőtlenség, vagy pedig a lemezszámok egyenlők lesznek és $f_i < n_i$ fog fennállni. Ha most a t_{i+1} elhelyezése előtt a lemezszámok egyenlők voltak, akkor az elhelyezés után vagy FF lemezszáma kisebb lesz, vagy pedig egyenlő lemezszám mellett FF utolsó lemezének szintje legfeljebb akkora lesz, mint NF utolsó lemezének szintje. ■

Hasonló állítás bizonyítható az NF–BF, NFD–FFD és NFD–BFD algoritmuspárokra.

Indukcióval belátható, hogy FFD és QFD minden listára ugyanannyi lemezt igényelnek.

Az eddigi megállapításokat a 4.3. ábrán összesítjük.

4.6. Közelítő algoritmusok hibája

Két algoritmus (A és B) relatív hatékonyságát gyakran jellemzik a kiválasztott hatékonysági mérték értékeinek hányadosával, jelen esetben az $\mathrm{A}(\mathbf{t})/\mathrm{B}(\mathbf{t})$ relatív lemezszámmal. Ennek a hányadosnak a felhasználásával különböző jellemzők definiálhatók. Ezeket két csoportba szokás sorolni: egyik csoportba a legrosszabb, míg a másikba az átlagos esetet jellemző mennyiségek kerülnek.

Itt csak a legrosszabb esettel foglalkozunk (az átlagos eset vizsgálata rendszerint lényegesen nehezebb).

Legyen D_n $(n = 1, 2, \ldots)$ azon valós listák halmaza, amelyek n elemet tartalmaznak, és legyen $\mathcal{D}$ az összes valós lista halmaza, azaz

$$D = \cup_{n=1}^{\infty} D_n \ .$$

	LF	NF	FF	BF	PF	NFD	FFD	BFD	PFD	QFD	OPT
LF	X	1	1	1	1	1	1	1	1	1	1
NF	–	X	3	4	7	5	1	1	1	1	1
FF	–	–	X	4	7	5	1	1	1	1	1
BF	–	–	3	X	7	5	1	1	1	1	1
PF	–	2	2	2	X	3	1	1	1	1	1
NFD	–	2	2	2	6	X	1	1	1	1	1
FFD	–	2	2	2		–	X	4		–	2
BFD	–	2	2	2		–	3	X		3	2
PFD	–	2	2	2	8	3	3	3	X	3	2
QFD	–	2	2	2		–	–	4		X	2
OPT	–	–	–	–	–	–	–	–	–	–	X

4.3. ábra. Algoritmusok páronkénti összehasonlításának eredményei.

Legyen $\mathcal{A}_{lsz}$ az ***állományelhelyező,*** (azaz a minden $\mathbf{t} \in \mathcal{D}$ listához egy nemnegatív valós számot hozzárendelő, és így a $\mathcal{D} \to \mathbb{R}_0^+$ leképezést megvalósító) algoritmusok halmaza.

Legyen $\mathcal{A}_{opt}$ a minden listához az optimális lemezszámot rendelő algoritmusok halmaza, és OPT ennek a halmaznak egy eleme (azaz egy olyan algoritmus, amely minden $\mathbf{t} \in D$ listához megadja a listához tartozó fájlok elhelyezéséhez szükséges és elégséges lemezek számát).

Legyen $\mathcal{A}_{köz}$ azon $\mathrm{A} \in \mathcal{A}_{lsz}$ algoritmusok halmaza, amelyekre $\mathrm{A}(\mathbf{t}) \geq \mathrm{OPT}(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $\mathrm{A}(\mathbf{t}) > \mathrm{OPT}(\mathbf{t})$. Legyen A_{becs} azon $\mathrm{E} \in \mathcal{A}_{lsz}$ algoritmusok halmaza, amelyekre $\mathrm{E}(\mathbf{t}) \leq \mathrm{OPT}(\mathbf{t})$ minden $\mathbf{t} \in D$ listára, és van olyan $\mathbf{t} \in D$ lista, amelyre $\mathrm{E}(\mathbf{t}) < \mathrm{OPT}(\mathbf{t})$.

Legyen F_n azon valós listák halmaza, amelyekre $\mathrm{OPT}(\mathbf{t}) = n$, azaz $F_n = \{\mathbf{t} | \mathbf{t} \in D$ és $\mathrm{OPT}(\mathbf{t}) = n\}$ $(n = 1, 2, \ldots)$. A továbbiakban csak $\mathcal{A}_{lsz}$-beli algoritmusokat fogunk vizsgálni. Az A és B algoritmusok $(\mathrm{A}, \mathrm{B} \in \mathcal{A})$ $R_{\mathrm{A,B},n}$ hibafüggvényét, $R_{\mathrm{A,B}}$ hibáját (abszolút hibáját) és $R_{A,\infty}$ aszimptotikus hibáját a következőképpen definiáljuk:

$$R_{A,B,n} = \sup_{t \in F_n} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} ,$$

$$R_{A,B} = \sup_{t \in D} \frac{\mathrm{A}(\mathbf{t})}{\mathrm{B}(\mathbf{t})} ,$$

$$R_{A,B,\infty} = \limsup_{n \to \infty} R_{A,B,n} .$$

Ezek a mennyiségek főleg akkor érdekesek, ha $\mathrm{B} \in \mathcal{A}_{opt}$. Ilyenkor az egyszerűség kedvéért a jelölésekből elhagyjuk a B-t, és az $\mathrm{A} \in \mathcal{A}$, illetve az $\mathrm{E} \in \mathcal{A}$ algoritmusok hibafüggvényéről, hibájáról és aszimptotikus hibájáról beszélünk.

Az NF fájlelhelyező algoritmus jellemző adatai ismertek.

4.2. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$n = \mathrm{OPT}(\mathbf{t}) \leq \mathrm{NF}(\mathbf{t}) \leq 2\mathrm{OPT}(\mathbf{t}) - 1 = 2n - 1 . \tag{4.1}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{NF}(\mathbf{u}_k) \tag{4.2}$$

és

$$k = \mathrm{OPT}(\mathbf{v}_k) \text{ és } \mathrm{NF}(\mathbf{v}_k) = 2k - 1 \ . \tag{4.3}$$

Bizonyítás. a) Az alsó korlát természetes: közelítő algoritmus nem lehet jobb, mint az optimális.

b) A felső korlát belátásához először tegyük fel, hogy NEXT(**t**) páros, azaz $2k$ alakú. A NF által elheyezett fájlok esetén a lemezeken páronként nagyobb a fájlok hosszának összege, mint 1. Ezért a NF $2k$ lemezre mindig úgy helyez fájlokat, hogy hosszaik összege nagyobb, mint k, ezért $\mathrm{NF}(\mathbf{t}) > 2\mathrm{OPT}(\mathbf{t})$ ellentmondásra vezetne

Legyen most $\mathrm{NF}(\mathbf{t}) = 2k+1$. Ekkor az előzőek szerint a hosszak összege nagyobb, mint k, ezért $\mathrm{OPT}(\mathbf{t}) \geq k + 1$, ami elég a kívánt egyenlőtlenséghez.

c) Ha az $\mathbf{u}_k$ lista k darab egyest tartalmaz, akkor ezt a listát mind NF, mind pedig OPT k lemezre helyezi el.

d) A $\mathbf{v}_k$ lista legyen $2k - 1$ olyan lista összeláncoltja, melyekben két elem van: előbb $1/2$, azután pedig $(1/2)/(2k - 1)$. Ezt a listát NF $2k - 1$ lemezre helyezi el, míg OPT csak k lemezt használ fel. ■

Ebből az állításból adódik a NF fájlelhelyező algoritmus hibafüggvénye, abszolút hibája és aszimptotikus hibája.

4.3. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$R_{\mathrm{NF},n} = 2 - \frac{1}{n} \ , \tag{4.4}$$

továbbá

$$R_{\mathrm{NF}} = R_{\mathrm{NF},\infty} = 2 \ . \tag{4.5}$$

Megmutatjuk, hogy a FF fájlelhelyező algoritmus legrosszabb esetben is kevesebb fájlt igényel, mint az optimális érték 1.75-szöröse. A bizonyítás alapja a következő, ú.n. telítettségi lemma.

4.4. lemma. *Legyen k pozitív egész. Ha FF* $k + 1$ *lemez mindegyikén legalább k fájlt helyez el, akkor az ezeken a lemezeken elhelyezett fájlok hosszának összege nagyobb, mint k.*

Ennél valamivel erősebb állítást bizonyítunk be.

4.5. lemma. *Legyenek* k, i_1, i_2, $\ldots$ $i_{k+1} = b$ *pozitív egészek, melyekre* $i_1 < i_2 < \ldots < i_k < b$. *Ha a FF algoritmus az* L_b *lemezen legalább k fájlt helyez el, akkor az* L_{i_1}, L_{i_2}, $\ldots$, L_{i_k}, L_b *lemezeken elhelyezett fájlok méretének összege nagyobb, mint k.*

Érdemes megjegyezni, hogy a telítettségi lemma úgy is megfogalmazható, hogy a lemmában szereplő lemezeken az átlagos telítettség nagyobb, mint $k/(k + 1)$.

Bizonyítás. Legyen a legnagyobb indexű, azaz az L_b lemezen elhelyezett legkisebb fájl mérete a. Ekkor az a méretű fájl nem fért rá a kisebb indexű lemezekre, ezért azok szintje nagyobb, mint $1 - a$. A legnagyobb indexű fájl szintje viszont legalább ka, így a szintek összege valóban nagyobb, mint $k(1 - a) + ka = k$. ■

4.6. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{FF}(\mathbf{t}) < 1.75\mathrm{OPT}(\mathbf{t}) \; . \tag{4.6}$$

Bizonyítás. A bizonyítás alapja, hogy a telítettségi lemma szerint az egy fájlt tartalmazó lemezek átlagos szintje egykettednél, a két fájlt tartalmazó lemezek pedig átlagosan kétharmadnál nagyobb mértékben telítettek. A fájlok hosszának összege legyen H, $\mathrm{OPT}(\mathbf{t}) = p$, $\mathrm{FF}(\mathbf{t}) = f$ és $\mathrm{NF}(\mathbf{t}) = e$.

a) Ha $p \leq 3$, akkor az előző két tétel szerint teljesül az állítás:

$$f \leq e \leq 2p - 1 \; , \tag{4.7}$$

ahonnan már adódik a kívánt egyenlőtlenség.

b) Ha $p > 3$, akkor feltesszük, hogy FF q lemezre egy, a többi $f - q$ lemezre pedig legalább két fájlt tett. A fájlok hosszainak (és egyúttal a lemezek szintjeinek) összege legyen H, ekkor $p \geq H$.

b1) Ha $q = 0$, akkor $p \geq H > 2/3f$, ahonnan $f < 3/2p$.

b2) Ha $q = 1$, akkor $p > 2(f - 1)/3$, ahonnan

$$f < \frac{3}{2}p + 1 = \frac{7}{4}p - \frac{1}{4}p + 1 = \frac{7}{4}p + \frac{4 - p}{4} \; . \tag{4.8}$$

Mivel $p \geq 4$, ebből a kívántnál is erősebb egyenlőtlenség adódik.

b3) Ha $q \geq 2$ és $f - q \leq 2$, akkor $q \leq p$ miatt

$$f \leq q + 2 \leq p + 2 = \frac{7}{4}p - \frac{3}{4}p + \frac{8}{4} = \frac{7}{4}p + \frac{8 - 3p}{4} \; , \tag{4.9}$$

ahonnan a kívántnál kedvezőbb egyenlőtlenség adódik.

b4) Végül ha $q \geq 2$ és $f - q \geq 3$, akkor

$$p > \frac{q}{2} + \frac{2(f - q)}{3} = \frac{2}{3}f - \frac{q}{6} \; , \tag{4.10}$$

ahonnan $p \geq q$ miatt

$$p > \frac{2f}{3} - \frac{p}{6} \; , \tag{4.11}$$

és így

$$f < \frac{7}{6}p\frac{3}{2} = \frac{7}{4}p \; . \tag{4.12}$$

■

Az FF és BF fájlelhelyező algoritmus legrosszabb esetére vonatkozik a következő állítás.

4.7. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{OPT}(\mathbf{t}) \leq \mathrm{FF}(\mathbf{t}), \; \mathrm{BF}(\mathbf{t}) \leq 1.7\mathrm{OPT}(\mathbf{t}) + 2 \; . \tag{4.13}$$

Továbbá, ha $k \in \mathbb{Z}$, *akkor léteznek olyan* $\mathbf{u}_k$ *és* $\mathbf{v}_k$ *listák, melyekre*

$$k = \mathrm{OPT}(\mathbf{u}_k) = \mathrm{FF}(\mathbf{u}_k) = \mathrm{BF}(\mathbf{u}_k) \tag{4.14}$$

valamint

$$k = \mathrm{OPT}(\mathbf{v}_k) \text{ és } \mathrm{FF}(\mathbf{v}_k) = \mathrm{BF}(\mathbf{v}_k) = \lfloor 1.7k \rfloor \ . \tag{4.15}$$

Ebből az állításból adódik FF és BF aszimptotikus hibája, valamint hibafüggvényük jó becslése.

4.8. következmény. *Ha* $n \in \mathbb{Z}$, *akkor*

$$\frac{\lfloor 1.7n \rfloor}{n} \leq R_{\mathrm{FF},n}, R_{\mathrm{BF},n} \leq \frac{\lceil 1.7n + 1 \rceil}{n}, \tag{4.16}$$

továbbá

$$R_{\mathrm{FF},\infty} = R_{\mathrm{BF},\infty} = 1.7 \ . \tag{4.17}$$

Az FF algoritmusra (4.13)-nál kisebb felső korlát is ismert.

4.9. tétel. *Ha* $\mathbf{t} \in F_n$, *akkor*

$$\mathrm{FF} < 1.7\mathrm{OPT}(\mathbf{t}) + 1 \ . \tag{4.18}$$

Ha n osztható tízzel, akkor az (4.15) egyenlőtlenségben FF-re vonatkozó tényleges érték és a (4.18)-ban FF-re vonatkozó felső korlátok megegyeznek, azaz tízzel osztható n esetén az $\mathrm{FF}(\mathbf{t})/(\mathbf{t})$ és $\mathrm{BF}(\mathbf{t})/\mathrm{OPT}(\mathbf{t})$ hányadosok szuprémuma 1,7.

Mit mondhatunk FF abszolut hibájáról?

IRODALOMJEGYZÉK

[1] E. G. Coffman, Jr.: *Computer and Job Shop Scheduling Theory*. John Wiley & Sons, New York, 1972. 1976. MR0629691 (83m:68066a)

[2] A. Iványi: Performance bounds for simple bin packing algorithms. *Annales Universitatis Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica* **5** (1984), 77–82. MR0822601 (87h:68067)

[3] Iványi A. (szerk.): *Informatikai algoritmusok. 1. kötet.* ELTE Eötvös Kiadó, Budapest, 2004. Elektronikusan: http://www.inf.elte.hu/karunkrol/digitkonyv/Jegyzetek2004/InfAlg1.pdf. 11.4. alfejezet, 491–502. oldal. Angolul: *Algorithms of Computer Science. Volume 2.* Mondat Kiadó, Budapest, 2007. 17.4. alfejezet, 836–848. oldal.

Tartalomjegyzék