15. Külső rendezések

A külső rendezési algoritmusokban alkalmazható műveletek specifikációja.

15.1. File absztrakt adattípus

Típusolt file: File of E

Értékhalmaz: $FileE = \{\langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle : a_i \in E, i = 1,\ldots,n\}$

Műveletek:

$$F : FileE, FN : String, x : E, j : Longint$$

$\{Igaz\}$	$Assign(F,FN)$	$\{F =$ az FN állomány tartalma$\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Reset(F)$	$\{F = \langle\rangle\langle a_0,\ldots,a_i,\ldots,a_n\rangle\}$
$\{F = F\}$	$Rewrite(F)$	$\{F = \langle\rangle\langle\rangle\}$
$\{F = F\}$	$EoF(F)$	$\{EoF = (Pre(F) = \langle a_0,\ldots,a_{n-1}\rangle\langle\rangle)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle \wedge i < n\}$	$Read(F,x)$	$\{x = a_i \wedge F = \langle a_0,\ldots,a_i\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Write(F,x)$	$\{F = \langle a_0,\ldots,a_i,x\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FilePos(F)$	$\{FilePos = i \wedge F = Pre(F)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FileSize(F)$	$\{FileSize = n \wedge F = Pre(F)\}$
$\{F = F \wedge 0 \le j \le FileSize(F)\}$	$Seek(F,j)$	$\{F = \langle a_0,\ldots,a_{j-1}\rangle\langle a_j,\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Truncate(F)$	$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle\rangle\}$

Típustalan (bináris) file: File

Értékhalmaz: $File = \{\langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle : a_i \in Byte, i = 1,\ldots,n\}$

Műveletek:

$$F : File, FN : String, x : E, k, j, r : Longint, V : Array[1..]\,of\,E$$

$\{Igaz\}$	$Assign(F,FN)$	$\{F =$ az FN állomány tartalma$\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Reset(F,Rh)$	$\{F = \langle\rangle\langle a_0,\ldots,a_i,\ldots,a_n\rangle, \lvert a_i\rvert = Rh\}$
$\{F = F\}$	$Rewrite(F,Rh)$	$\{L = \langle\rangle\langle\rangle\}$
$\{F = F\}$	$EoF(F)$	$\{EoF = (Pre(F) = \langle a_0,\ldots,a_{n-1}\rangle\langle\rangle)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle \wedge i < n\}$	$Read(F,x)$	$\{x = a_i \wedge F = \langle a_0,\ldots,a_i\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Write(F,x)$	$\{F = \langle a_0,\ldots,a_i,x\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FilePos(F)$	$\{FilePos = i \wedge F = Pre(F)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FileSize(F)$	$\{FileSize = n \wedge F = Pre(F)\}$
$\{F = F \wedge 0 \le j \le FileSize(F)\}$	$Seek(F,j)$	$\{F = \langle a_0,\ldots,a_{j-1}\rangle\langle a_j,\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Truncate(F)$	$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$BlockRead(F,V,k,[r])$	$\{V[j] = a_{i+j-1}, j = 1,\ldots,r \wedge$ $F = \langle a_0,\ldots,a_{i+r-1}\rangle\langle a_{i+r},\ldots,a_{n-1}\rangle \wedge$ $r = \min(k, n-i)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$BlockWrite(F,V,k)$	$\{F = \langle a_0,\ldots,a_{i-1},V[1],\ldots V[k]\rangle$ $\langle a_{i+k},\ldots,a_{n-1}\rangle\}$

Minden belső tömbös rendezési algoritmus mechanikusan átírható külső rendezési algoritmussá, mivel a `Seek` művelettel megvalósíthóak a pozíció szerinti kiolvasó és módosító műveletek:

```
X:=T[i] - Seek(F,i); Read(F,X)
T[i]:=X - Seek(F,i); Write(F,X)
```

Azonban így olyan algoritmust kapnánk, amely haszálhatatlanul lassú, mert az adatelemeket egyesével mozgatnánk a lemez és a főtár között, ami minden esetben fizikai mozgást igényelhet.

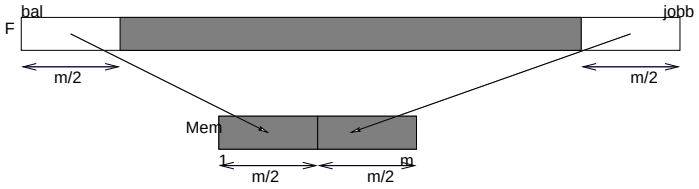
15.2. Gyorsrendezés virtuális memóriával

Ha az operációs rendszer lehetővé teszi virtuális memória használatát, akkor bármely belső rendezés egyszerűen átírható külső rendezéssé. Akkora virtuális memóriát kell foglalni, amekkora a rendezendő file mérete, virtuálisan beolvasni a teljes file-t, belső rendezéssel rendezni, és végül visszaírni a file-ba. Tehát amikor egy elemre (rekordra) hivatkozunk a belső rendezés során, akkor az operációs rendszer gondoskodik róra, hogy bekerüljön a fizikai memóriába.

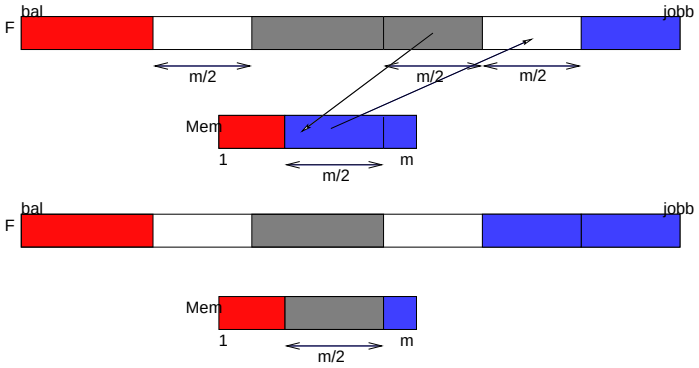
```
 { Külső rendezés virtuális memóriával}
Const MaxN=999999999;
Type
  ElemTip=???;
  MemTip=Array[1..MaxN] Of Elemtip;
Var
  FNev : String;                { a rendezendő file neve}
  N  : Longint;                 { rekordok száma}
  Rh : Longint;                 { rekord méret}
  F:  File;
  Mem:^MemTip;
Begin{külső rendezés virtuális memóriával}
   Rh:=Sizeof(Elemtip);
   Assign(F, FNev); Reset(F, Rh);{ file megnyitás}
   N:=FileSize(F);              { elemszám lekérdezése}
   GetMem(Mem, N*Rh);           { memória foglalás}
   BlockRead(F, Mem^[1], N);    { virtuális beolvasás}
   GyorsRendez(Mem^, N);        { belső rendezés}
   Seek(F,0);
   BlockWrite(F, Mem^[1], N);   { kiíratás (visszaírás)}
   FreeMem(Mem, N*Rh);          { a memória felszabadítása}
   Close(F);
End.
```

Azonban, az operációs rendszer nem tudja, hogy mely elemekre, milyen sorrendben fogunk hivatkozni. Tudjuk, hogy a gyorsrendezés esetén kizárólag a FELOSZT hivatkozik adatelemekre, és tudjuk azt is, hogy adott elem után mely elem következhet. Ezt figyelembe véve lényegesen gyorsíthatjuk az algoritmust. Belátható, hogy ekkor nem a Lomuto, hanem a Hoare-féle felosztást célszerű használni.
Tegyük fel, hogy a rendezésre használható fizikai memória mérete m adatelem, tehát egyszerre ennyi elem lehet a memóriában (feltehetjük, hogy m páros szám). Az F file $bal..jobb$ pozíciói által meghatározott rész adatsor felosztása

úgy kezdődik, hogy a bal pozíciótól beolvasunk $m/2$ elemet, és a jobb végéről, azaz a $jobb-(m/2)+1$ pozíciótól is beolvasunk $m/2$ elemet. Ha $jobb-bal+1 \leq m$, akkor természetesen a teljes részt beolvassuk. Majd választunk egy Fe felosztó elemet, és felosztjuk a memóriában levő részt. A baloldali és jobboldali rész közül az egyik biztosan legalább $m/2$ elemet tartalmaz. Ebből a részből $m/2$ hosszú összefüggő részt ki tudunk írni a file-ba, oda, ahonnan beolvastunk. Az általános helyzetet a 2. ábra mutataja. Tehát a beolvasás/kiírás a BLOCKREAD/BLOCKWRITE



1. ábra. Kezdő lépés; a file-ból bal- és jobb végéről $m/2$ elem beolvasása.



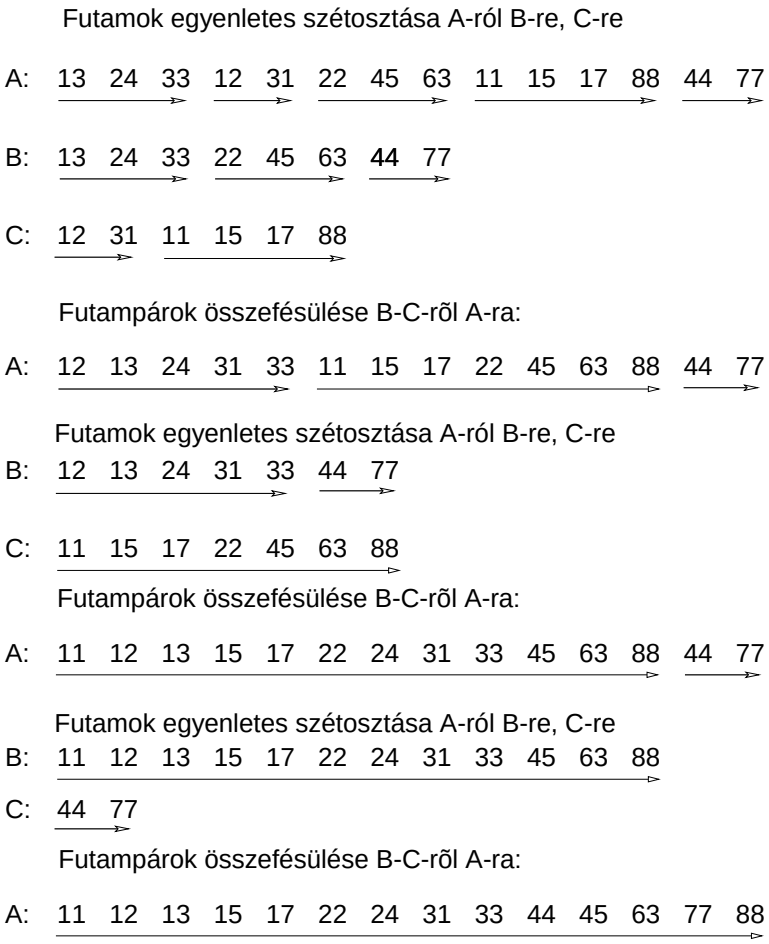
2. ábra. A memóriában felosztott elemekből $m/2$ méretű rész kiírása és újabb $m/2$ méretű szelet beolvasása. A pirossal jelölt részekben az eleme $\leq Fe$, a kékkel jelölt részekben pedig $\geq Fe$. A szürkék a még felosztatlan elemek, a fehér részek jelzik a file-nak azt a részét, ahonnan beolvastunk a memóriába, tehát "üresek".

művelettel mindig $m/2$ méretű szeletekben történik (kivéve, amikor összer a két rész).

15.3. Külső rendezés összefésüléssel

Az $F = \langle a_0, \ldots, a_{n-1} \rangle$ rendezendő adatsorozat rendezett $a_i, \ldots, a_j$ részsorozatát *futamnak* nevezünk. A futam *maximális futam*, ha $a_{i-1} > a_i$ és $a_j > a_{j+1}$.

3 file-os egyenletes összefésülő rendezés.
Tegyük fel, hogy 3 szekvenciális file-t használhatunk a rendezésre: A, B, C, és az A tartalmazza a bemeneti, rendezendő halmazt. Első lépésként osszuk szét a maximális futamokat egyenletesen A-ról B-re és C-re. Az egyenletes szétosztás azt jelenti, hogy a B-re és C-re kerülő futamok száma legfeljebb eggyel térjen el. Ez elérhető úgy, hogy a páratlan sorszámú futamokat B-re, a párosakat C-re másoljuk át. Majd a B-ről és C-ről futampárokat fésüljünk össze A-ra, amely a szétosztás után üresíthető. A szétosztást-összefésülés fázisokat addig ismételjük, amíg egyetlen futam keletkezik.
4 file-os egyenletes összefésülő rendezés.
Látható, hogy ha 4 file-t használhatunk a rendezésre, akkor a futamok szétosztása megtakarítható, kivéve az első, u.n. kezdeti szétosztást.



3. ábra. 3 file-os egyenletes összefésülő rendezés menetei.

Futamok kezdeti szétosztása A-ról C-re és D-re:

A: 13 24 33 12 31 22 45 63 11 15 17 88 44 77

C: 13 24 33 22 45 63 44 77

D: 12 31 11 15 17 88

Futampárok összefésülése C-D-rõl A-B-re:

A: 12 13 24 31 33 44 77

B: 11 15 17 22 45 63 88

Futampárok egyenletes összefésülése A-B-rõl C-D-re:

C: 11 12 13 15 17 22 24 31 33 45 63 88

D: 44 77

Futampárok összefésülése C-D-rõl A-B-re:

A: 11 12 13 15 17 22 24 31 33 44 45 63 77 88

B:

4. ábra. 4 file-os egyenletes összefésülő rendezés menetei.

Többutas egyenletes összefésülő rendezés.
A 4 file-os egyenletes összefésülő rendezés általánosítható tetszőleges p-re, tehát amikor $2p$ darab filet használunk, p input file-ról fésülünk össze p output file-ra.

```
Procedure PUtasEgyenletesFesuloRendezes(Var F);
Var SF:Array[Boolean, 1..P] of Elemtip; {segéd fileok}
  i:Word; be,ki:Boolean;
Begin
  Reset(F); be:=True; ki:=False;
  Futamok kezdeti egyenletes szétosztása F-ről SF[be,1],...,SF[be,p]-re;
  While FutamSzam>1 Do Begin {összefésülő menet}
    For i:=1 To P Do Begin
      Reset(SF[be,i]);
      Rewrite(SF[ki,i]);
    End{For i};
    Futamok egyenletes összefésülése
      SF[be,1..p]-ről SF[ki,1..p]-re
    be:=ki; ki:=Not be;
  End{while};
End;
```

Költségmodell és bonyolultsági mérték.
Olyan költségmodellt és bonyolutsági mértéket keresünk, amelly adeqvát abban az értelemben, hogy hűen adja meg a külső rendezési algoritmusok várható futási idejét. Pontosabban, elég csak az átviteli időt (beolvasás/kiírás) tekinteni, mivel ez nagyságreddel nagyobb, mint a belső müveletek ideje. Azt kell figyelembe venni, hogy a file műveletek végrehajtása fizikai mozgást igényel. Nevezetesen, a lemezmeghajtó író/olvasó fejét pozíciónálni kell, majd egy körülfordulás alatt a kívánt szektor elérhető olvasás/írás művelet céljából. Az író/olvasó fej pozícionálása nagyságreddel lassabb, mint a lemez forgási sebessége. Tekintsük az alábbi három programrészletet. Mindegyik ugyanazt teszi, az F file-ból az i-pozíciótól k adatelemet olvas be.

```
A: For j:= 1 To K Do Begin
     Seek(F,i+j-1); Read(F, V[j])
   End;
B: Seek(F, i);
   For j:= 1 To K Do Read(F, V[j])
C: Sek(F, i);
   BlockRead(F, V[1], k)
```

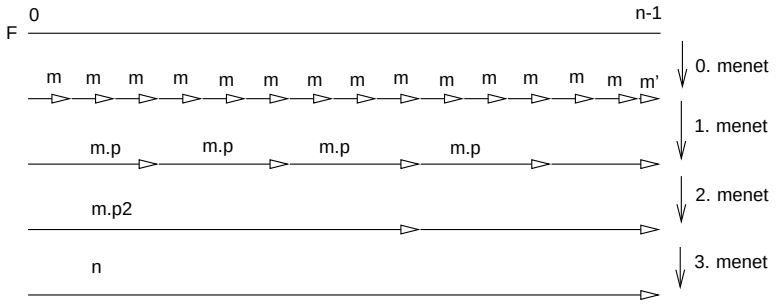
Három bonyolultsági mértékeket vizsgálunk.

1. Minden file művelet költsége 1.
2. A file művelet költsége a művelettel átvitt adatelemek (rekordok) száma.
3. Ha a művelet k db. rekordot visz át, akkor a költsége $\alpha k + \beta$, valamely α és β konstansra.

	1.	2.	3.
A	$2k$	k	$k(\alpha+\beta)$
B	k	k	$k(\alpha+\beta)$
C	2	k	$\alpha k+\beta$

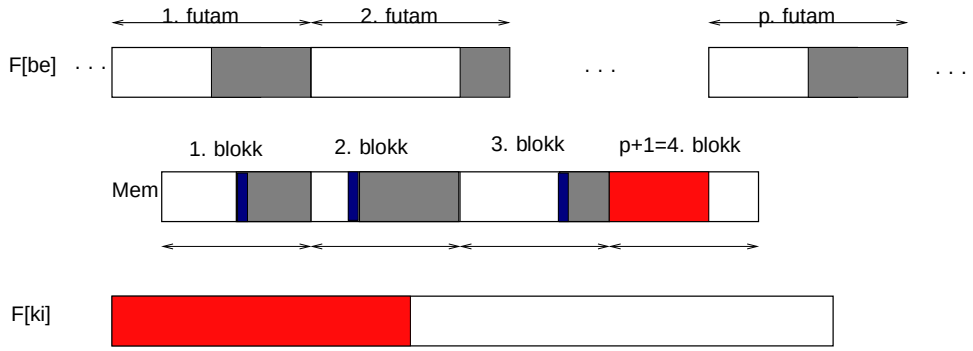
Nyilvánvaló, hogy csak a 3. bonyolultsági mérték elfogadható. Az α és β konstansok értéke függ az operációs rendszertől, pontosabban, az alkalmazott filerendszer megvalósítástól. Azonban majd látni fogjuk, hogy vizsgálatainkban csak a β/α hányados számít, amire könnyű jó becslést adni.

A P-utas egyenletes összefésülő rendezés elemzése.
Jelölje a továbbiakban a rendezendő F file elemszámát n, tehát $FileSize(F) = n$. Feltesszük, hogy rögzített, m adatelemet befogadó memóriat használhatunk az algoritmusban.
A futamok kezdeti szétosztása helyett képezzünk m hosszú futamokat úgy, hogy egy BLOCKREAD művelettel olvas-



5. ábra. A menetek szemléltetése $p = 3$ esetre.

sunk be m adatelemet, rendezzük öket belső gyorsrendezéssel és egy BLOCKWRITE művelettel írjuk vissza a file-ba. Ezt nevezzük 0. menetnek. Ennek az lesz az előnye, hogy a futamok hossza azonos, kivéve az utolsó csonka futamot, amelynek hossza $n \mod m$.
Elegendő lesz egy segéd filet használni, amiben a futamokat tároljuk, hiszen minden futam file-beli pozícióját ki tudjuk számítani a futam sorszámából, és hogy hanyadik összefésülő menetet végezzük. A 0. menet után az i-edik futam kezdőpozíciója $(i-1)m$ és hossza m, kivéve az $\lceil n/m \rceil$-edik utolsót, amelynek hossza $n \mod m$. Ha mindig p futamot fésülünk össze (kivéve az utolsó fésülést, amire esetleg nem marad p futam), az r-edik menet után a futamok hossza $m\,p^r$. A futamok összefésüléséhez mind a p futamból egy, az soron következő adatelemnek a memóriában kell lennie. Ezt úgy biztosítjuk, hogy felosztjuk a memóriát $p+1$ egyenlő részre, az első p blokk lesz az input blokk, ide olvasunk be a futamokból. A $p+1$-edik memória blokk az output blokk, ide visszük át a futamok aktuális elemeinek a legkisebbikét. Ha az output blokk beltelik, kiírjuk az $F[ki]$ output fileba (szekvenciálisan). Ha egy input blokk kifogy, akkor egy blokknyit (illetve csonka futam esetén amennyi maradt a futamból) beolvasunk a megfelelő futamból. A legkisebb elem kiválasztását kupaccal végezzük.
Tegyük fel, hogy k rekord átviteli költsége (egy BlocRead/BlockWrite): $k\alpha+\beta$.
Jelölje $K(n, m, p)$ a rendezési algoritmus átviteli összköltségét, ha a memória mérete m és minden menetben p futamot fésülünk össze.
Milyen p-re lesz $K(n, m, p)$ minimális ?
A kezdeti (m-hosszú) futamok előállításanak
költsége: $K_0(n, m, p)$



6. ábra. Futamok beolvasása és kiírása összefésüléskor.

$$K_0(n,m,p) = 2(n\alpha + \lceil n/m \rceil \beta)$$

A menetek száma, ha minden menetben p futamot fésülünk össze: az a legkisebb r, amelyre $m\,p^r \geq n$.

$$r = \lceil log_p\, n/m \rceil \tag{1}$$

$$p = \left\lceil \sqrt[r]{n/m} \right\rceil \tag{2}$$

p lehetséges értékei: 2 .. $m-1$
r lehetséges értékei: $\lceil log_{m-1}\, n/m \rceil$ .. $\lceil log_2\, n/m \rceil$
Jelölje B a blokkméretet:

$$B = \left\lfloor \frac{m}{(p+1)} \right\rfloor \quad (B(p+1) = m)$$

Egy összefésülő menet költsége $K_1(n,\, m,\, p)$:

$$K_1(n,\, m,\, p) = 2(\alpha n + \lceil n/B \rceil \beta) = 2(\alpha n + \lceil (p+1)n/m \rceil \beta)$$

Tehát a fésülő menetek költsége: $r\,K_1(n,m,p)$.

$$\begin{aligned} K(n,\, m,\, p) &= K_0(n,m,p) + r\,K_1(n,m,p) \\ &= K_0(n,m,p) + \lceil log_p\, n/m \rceil\ (\alpha n + (p+1)\lceil n/m \rceil \beta)) \end{aligned}$$

Mivel $K_0(n,m,p)$ nem függ p-től ezért a $K(n,\, m,\, p)$ kifejezésből hagyjuk el. Továbbá, sok számítást takaríthatunk meg, ha az átviteli költséget nem p, hanem r (a menetek száma) függvényeként fejezzük ki, mert a lehetséges r-ek száma nagyságrenddel kisebb, mint a p-k száma. Ha meghatároztuk, hogy mely r-re lesz minimális az átviteli költség, a (2) képlettel kiszámítjuk a hozzá tartozó p-t.

$$\begin{aligned} K(n,\, m,\, r) &= r\,K_1(n,m,p) \\ &= \alpha n r + r\left(\left\lceil \sqrt[r]{n/m} \right\rceil + 1\right) \lceil n/m \rceil \beta \\ &= r\left(n + \left(\left\lceil \sqrt[r]{n/m} \right\rceil + 1\right) \lceil n/m \rceil \frac{\beta}{\alpha}\right) \end{aligned} \tag{3}$$

Tehát azt kell kiszámítani, hogy milyen *r*-re lesz a (3) kifejezés minimális!
Megvalósítás.

```
 Procedure FutamFesul(FTol :Longint); Var
      S :Array[1..MaxP] Of Longint;    { a prioritási sor (kupac)}
      FVeg,                            { futam vege a fileban }
      FPoz,                            { file rekord sorszám a futamban}
      BKezd,                           { blokk kezdőcíme }
      BVeg: Array[0..MaxP] Of Longint; { blokkvége }
      Sm:Longint;                      { a sor mérete }
      aP:Longint;                      { az összefésülendő futamok tégyl. száma}
      Poz:Longint;                     { file pozíció}
      Ki,                              { az output puffer akt. pozíciója}
      Reksz:Longint;                   { az átvitelben a rekordok száma}

{} Begin {FutamFesul}
    Poz:=(Ftol-1)*FutamHossz;          { az első futam file pozíciója}
    aP:=0;                             { a futamsorszám}
    Repeat                             { minden futamból egy blokk beolvasása}
      BKezd[aP]:=aP*BM;
      FPoz[aP]:=Poz;
      FVeg[aP]:=Poz+FutamHossz-1;
      If FVeg[aP]>=N Then FVeg[aP]:=N-1;
      S[aP+1]:=BKezd[aP];
      Reksz:=FVeg[aP]-Poz+1;
      If Reksz>BM Then Reksz:=BM;
      BVeg[aP]:=BKezd[aP]+Reksz-1;
      Seek(F[be], FPoz[aP]);
      BlockRead(F[be], Mem[BKezd[aP]], Reksz);
      Inc(FPoz[aP], Reksz);
      Inc(Poz, FutamHossz);
      Inc(aP);
    Until (aP=P)Or(Poz>=N);

    BKezd[aP]:=(aP)*BM;       { az output blokk memória-címe}
    BVeg[aP]:=BKezd[aP]+BM-1; { a blokk vége}

    Sm:=aP;
    KupacEpit(Sm);
    Ki:=BKezd[aP];
    While Sm>0 Do Begin
      Sorbol;
      Inc(Ki);
      If Ki>BVeg[aP] Then Begin
        BlockWrite(F[ki], Mem[BKezd[aP]], BVeg[aP]-
```

```
                                        BKezd[aP]+1);
        Ki:=BKezd[aP];
      End;
    End;
    BlockWrite(F[ki], Mem[BKezd[aP]], (Ki-BKezd[aP]));{a maradék kiírása}
  End{FutamFesul};

  Procedure FesuloMenet;
  {Global: FutamSzam, FutamHossz, F}
  Var
    FTol, UFSz:Longint;
  Begin{FesuloMenet}
    be:=ki; ki:=Not ki;             { I/O file váltás }
    Seek(F[ki],0);                  { pozícionálás az output file elejére}
    UFSz:=0;                        { a kelekező új futamok száma}
    FTol:=1;                        { összefésülés az FTol futamtól}
     Repeat
      FutamFesul(FTol);             { futamok összefésülése }
      Inc(UFSz);
      Inc(FTol, P);                 { átlépés a következő P futamra}
    Until FTol>FutamSzam;
    FutamSzam:=UFSz;                { a keletkezett új futamok száma}
    FutamHossz:=P*FutamHossz;       { az új futamok hossza}
  End {FesuloMenet};

Begin{Prog}
  Nyit;                             { megnyitás, optimális P kiszámítása}

  If N<=M Then Begin                { befér a memóriába}
    BlockRead(F[be], Mem[0], N);    { beolvasás}
    BelsoRendez(0, N-1);            { belső rendezés}
    Seek(F[be],0);
    BlockWrite(F[be], Mem[0], N);   { kiíratás (visszaírás)}
    Close(F[be]);
    Exit;
  End Else Begin
    Menet0;                         {kezdeti futamok képzése}
    Repeat
      FesuloMenet;                  {ősszefésülés}
    Until FutamSzam=1;              {amíg egy futamot kapunk}
  End;
  Zar;
End.
```