Haladó informatikai algoritmusok

Ládapakolási feladatok megoldása

Heurisztikus algoritmusok

- A hagyományos algoritmusok előre definiált instrukciók alapján, lépésről lépésre hajtanak végre egy adott feladatot, eredményük pedig egzakt, determinisztikus.
- *Az egyik legegyszerűbb algoritmus például a Pitagorasz-tétel, ami két bemenő paraméter alapján egzakt módon meghatároz egy harmadikat.*
- A heurisztikus algoritmusok próbálgatással, a korábban megszerzett tapasztalatok felhasználásával jutnak eredményre.

(A heurisztika kifejezés a görög heureszisz szóból származik, melynek jelentése rátalálás.)

- *Tekintsük a sakk játékot, elméletileg konstruálható lenne olyan egzakt megoldást biztosító eljárás, ami a sakkbábuk aktuális helyzete alapján, az összes további lehetséges lépés elemzésével kiszámítaná, hogyan nyerhetünk.*
- A problémát az jelenti, hogy adott lépésszám fölött a probléma túl bonyolult lesz.
- Egy heurisztikus algoritmus nem vizsgálja az összes lehetséges lépést, csupán a problématér egy adott részlete alapján, valamilyen logika szerint hozza meg döntését.

Heurisztikus algoritmusok

- A heurisztikus algoritmusok hatalmas előnye, hogy nagy bonyolultságú problémák esetében is képesek viszonylag rövid idő alatt, kevés számítás árán eredményt szolgáltatni.
- Hátrányuk azonban, hogy nem garantálható teljes bizonyossággal az optimális megoldás megtalálása.
- Akkor érdemes használni, ha az adott probléma megoldása hagyományos, egzakt megoldást adó eljárással belátható időn belül nem hajtható végre.
- Nagy problémaméret esetén is képesek optimális, vagy ahhoz közelítő megoldást adni.

Metaheurisztikus algoritmusok

- A metaheurisztikus eljárások olyan univerzális, robosztus heurisztikus algoritmusok, melyek nem csak egy adott típusú feladathoz használhatóak eredményesen.
- Bizonyos korábban szerzett ismereteket eltárolnak, így rendelkeznek probléma specifikus ismeretekkel a megoldás során.
- Ebből következően az esetek nagy részében képesek kikerülni a lokális optimum pontokból.
- Működésüket gyakran sztochasztikus jellemzők is befolyásolják.
- Hatékonyságvizsgálatuk ezért bonyolult feladat, mivel kis túlzással nincs két egyforma futás.
- *Pl. genetikus algoritmus, amely használható a ládapakolási feladat megoldásához is*

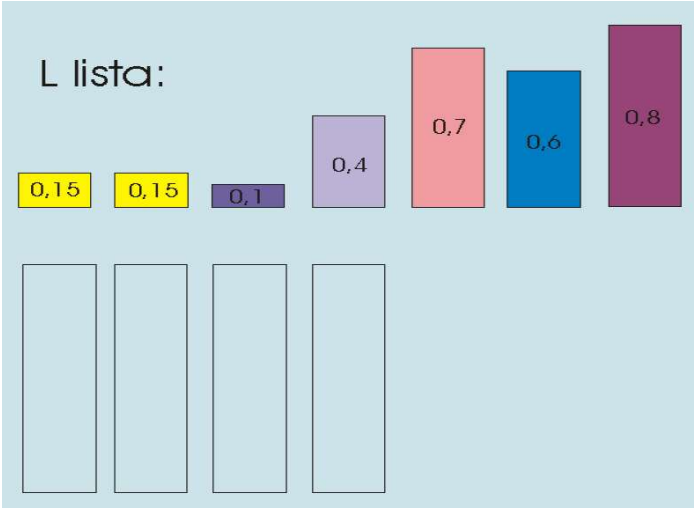
A ládapakolási feladat

A klasszikus egydimenziós ládapakolási feladatban adott tárgyak egy

$L = (a_1, a_2, \ldots, a_n)$ sorozata $(n \geq 1, n \in N)$, ahol

minden tárgy mérete jellemezhető egy (0, 1]-beli valós számmal.

El kell helyeznünk őket minél kevesebb számú egységnyi kapacitású ládába.

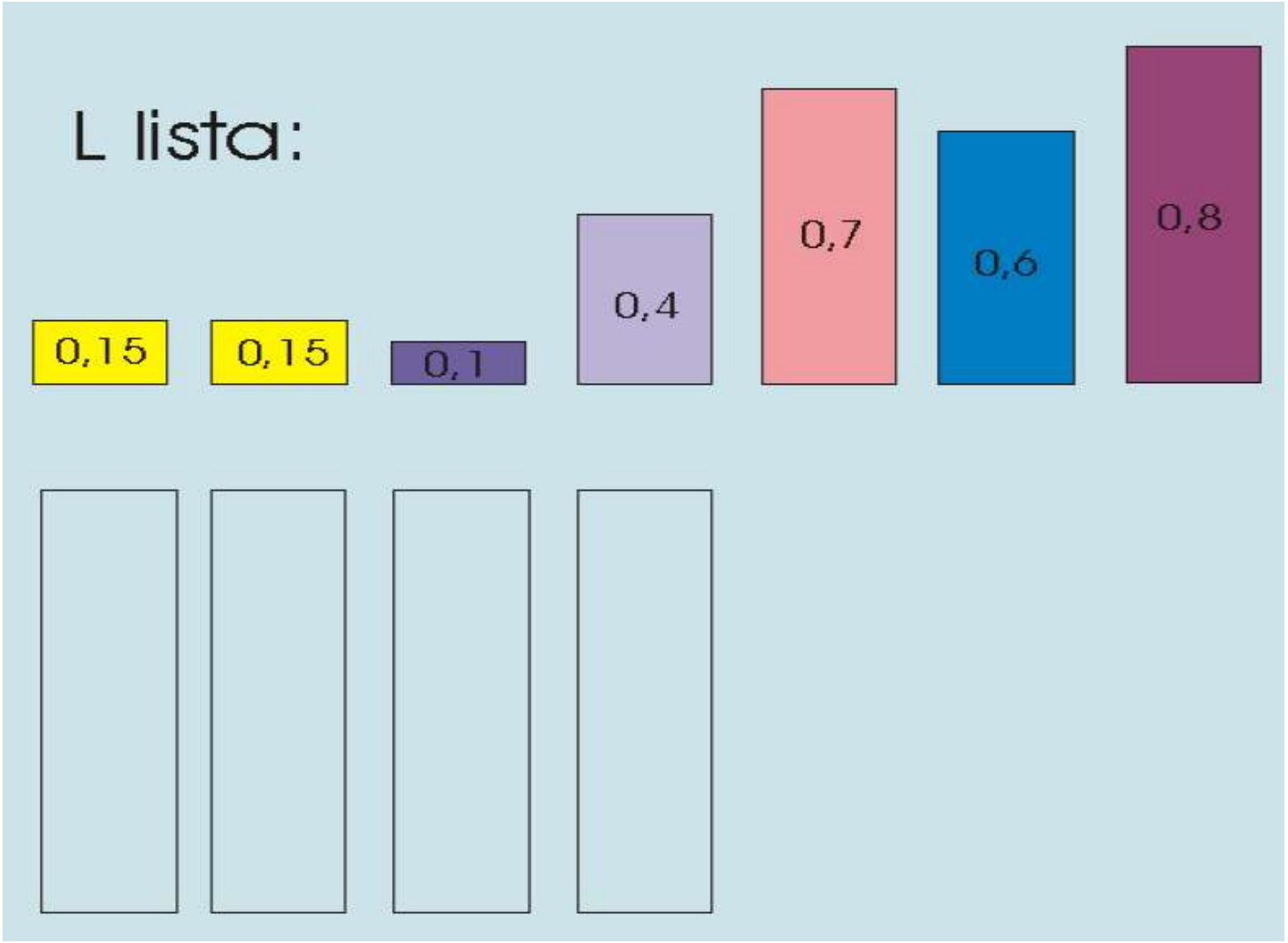


A ládapakolási feladat alkalmazásai

- Konténerek, szállítójárművek megtöltése,
- Memóriaallokáció,
- Üvegipar, pl. üvegrudak vágása,
- Média reklámfelületeinek kiosztása vagy különböző méretű reklámidők kiosztása reklámblokkokba,
- A feladatnak lehet informatikai vonatkozása is:
  - Egymástól független programokat (taszkokat) kell futtatnunk, és mindegyik számítógép (processzor) csak egységnyi ideig áll rendelkezésünkre. Célunk a programok olyan szétosztása a gépek között, hogy minél kevesebb gépet használjunk.
  - Másrészt értelmezhetjük úgy is, hogy egységnyi méretű lemezeken kell adott méretű fájlokat elhelyeznünk, cél a felhasznált lemezszám minimalizálása. Nagymennyiségű adathalmaz, pl. zenefájlok elhelyezése azonos méretű tárolókra (pl. CD-kre).
- A továbbiakban csak az egydimenziós feladattal foglalkozunk, pl. azonos magasságú és szélességű újsághasábokba azonos szélességű, de eltérő magasságú hirdetések elhelyezése.

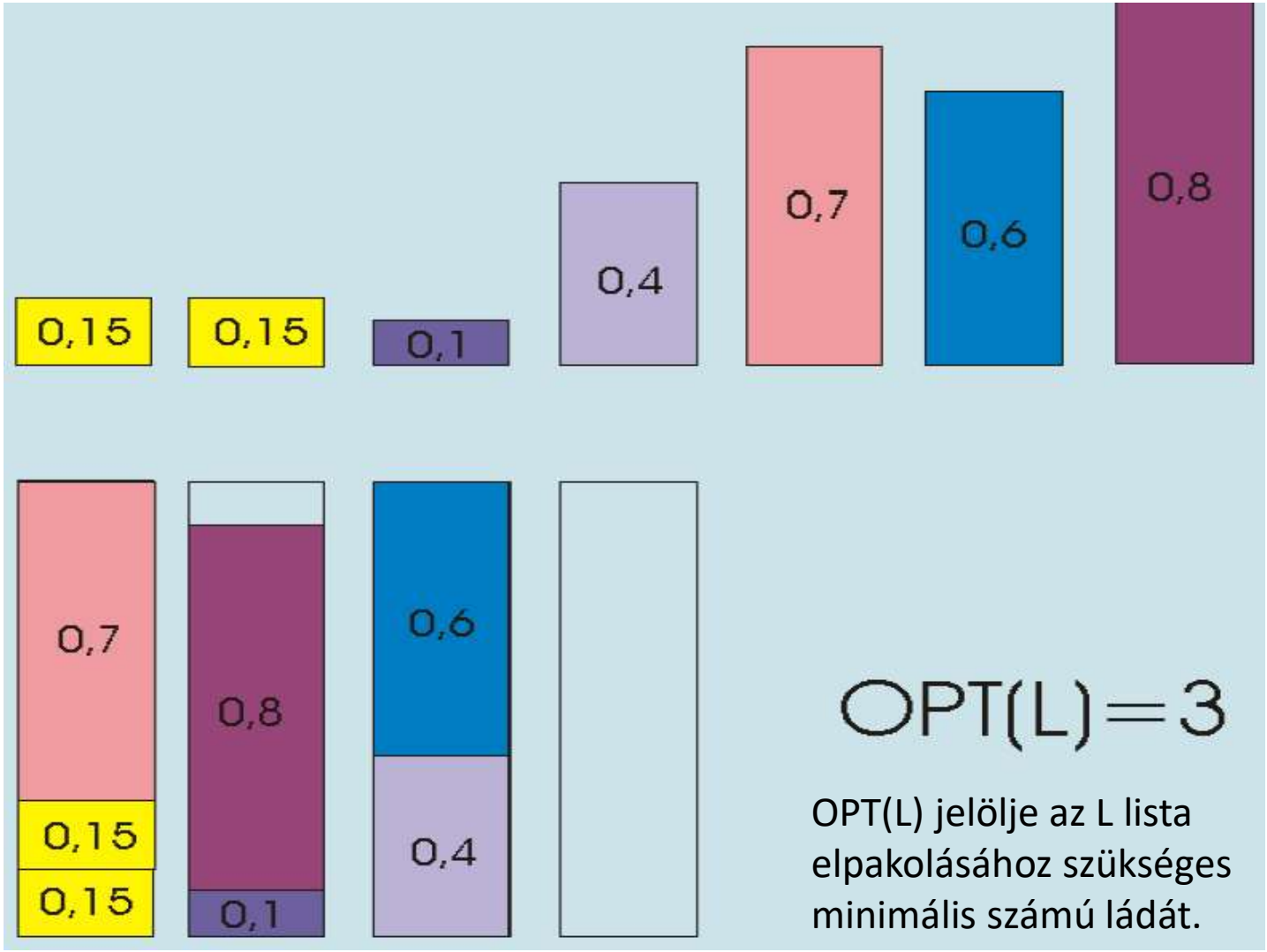
Egy példa

Hány ládában tudjuk az L lista elemeit elhelyezni?



Egy Példa

- Egy lehetséges optimális megoldás



Ez egy NP-nehéz feladat

- Az algoritmusok tervezése és vizsgálata két irányban indult meg:
  - Egzakt algoritmusok keresése - optimális megoldás megtalálása,
  - Közelítő algoritmusok tervezése és vizsgálata –heurisztika, metaheurisztika
- Az ún. approximációs algoritmusokat ezen a területen fejlesztették ki:
  - Olyan algoritmust nevezünk approximációs algoritmusnak, amelytől nem várjuk el, hogy feltétlenül optimális megoldást adjon egy feladatra, de egyrészt gyors (polinomiális idejű), másrészt az általa szolgáltatott megoldás garantáltan "nincs túl messze" az optimum értéktől.

A Next Fit algoritmus

- A Next Fit (NF) algoritmus addig rakja az elemeket a soron következő ládába, amig lehet. Ha egy elem már nem fér az adott ládába, akkor új ládát nyit és abba folytatja az elemek pakolását.
- Az algoritmust „egyszerű” algoritmusnak is nevezik.
- Az algoritmus pszeudo kódja:

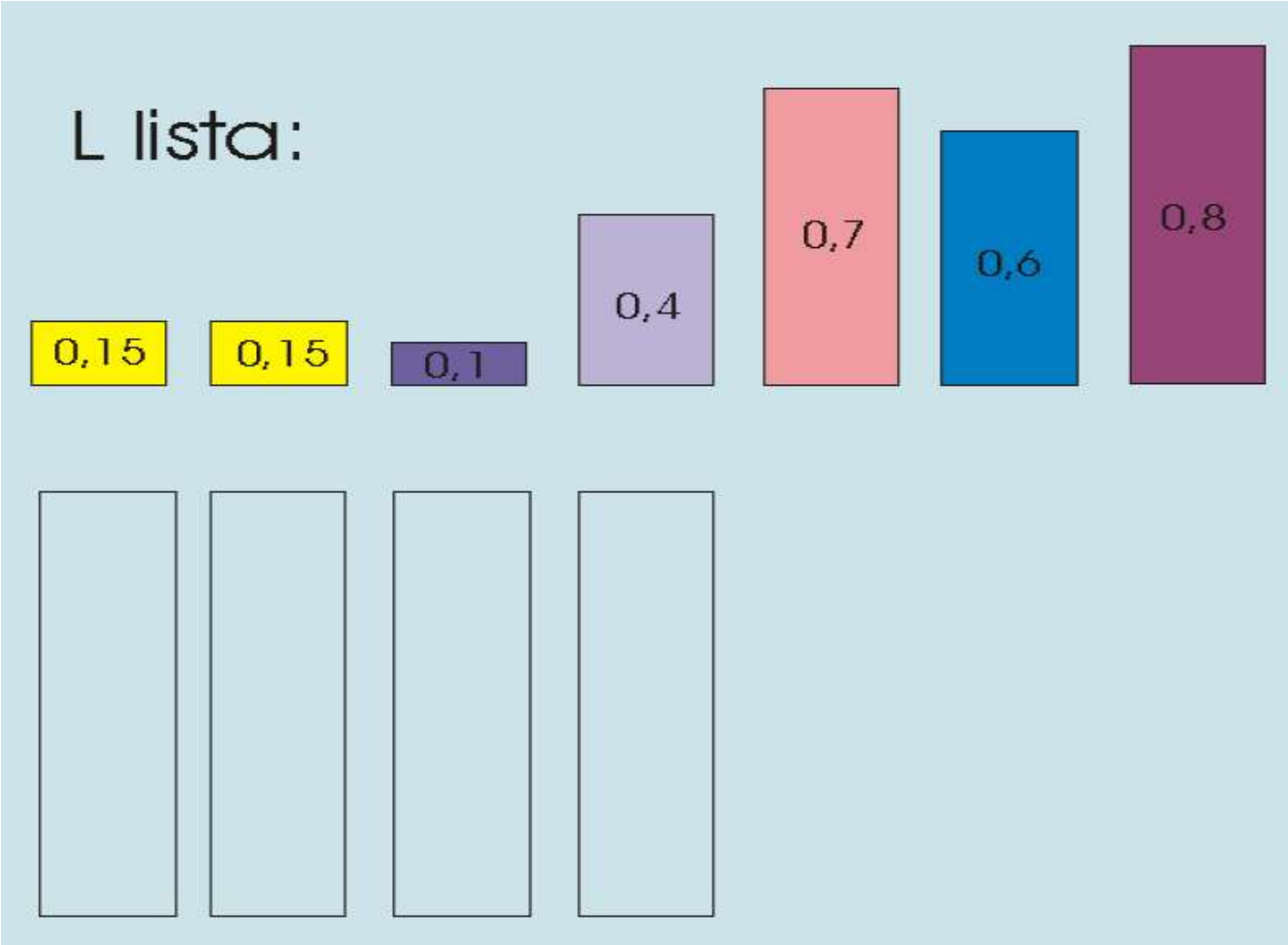
Algoritmus 1.1 $\mathrm{NF}(n, L)$

$b_1 \leftarrow a_1$
$C^{NF} \leftarrow 1$
for $i = 2 \rightarrow n$ **do**
  if $b_{C^{NF}} + a_i \leq 1$ **then**
    $b_{C^{NF}} \leftarrow b_{C^{NF}} + a_i$
  else
    $C^{NF} \leftarrow C^{NF} + 1$
    $b_{C^{NF}} \leftarrow a_i$
  end if
end for

Ennek az algoritmusnak a helyigénye és a futásideje egyaránt ***Θ(n)***.

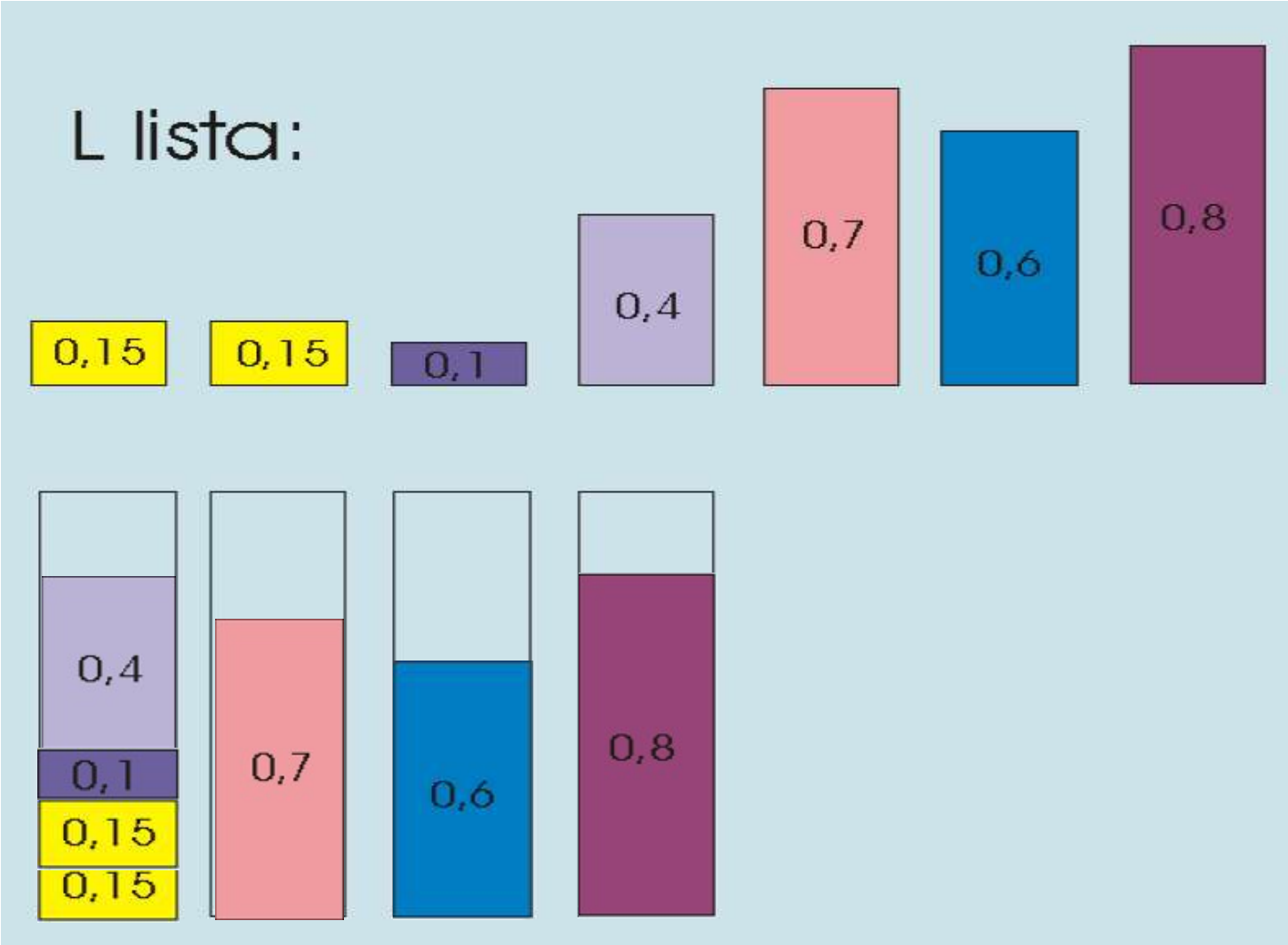
Next Fit (NF)

Hány ládában tudjuk az L lista elemeit elhelyezni?



Next Fit (NF)

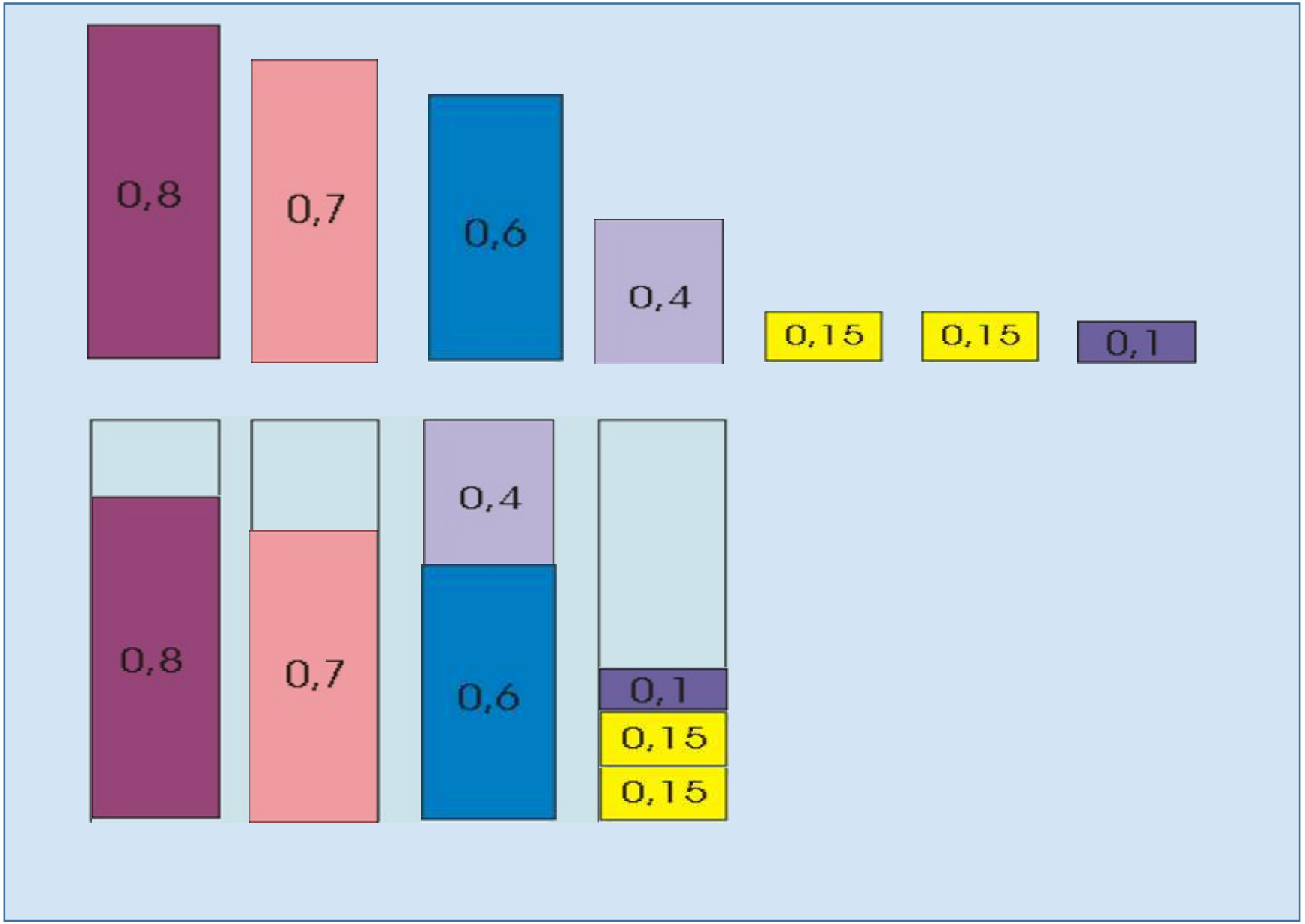
Megoldás: A(L)=4 láda szükséges, nem optimális megoldás



Az NFD algoritmus

- Először a lista elemeit nagyság szerinti csökkenő sorrendbe rendezzük, majd a második részben a rendezett lista elemeire alkalmazza valamelyik korábbi algoritmust. Erre utal az algoritmus nevében szereplő D betű (Decreasing).
- A Next Fit Decreasing (NFD) algoritmus a rendezés után az egyszerű algoritmus (NF) szerint dolgozik.
- Hely- és időigénye az alkalmazott rendező algoritmus és NF megfelelő igényeiből tevődik össze.

Az NFD algoritmus A(L)=4



A Best Fit algoritmus

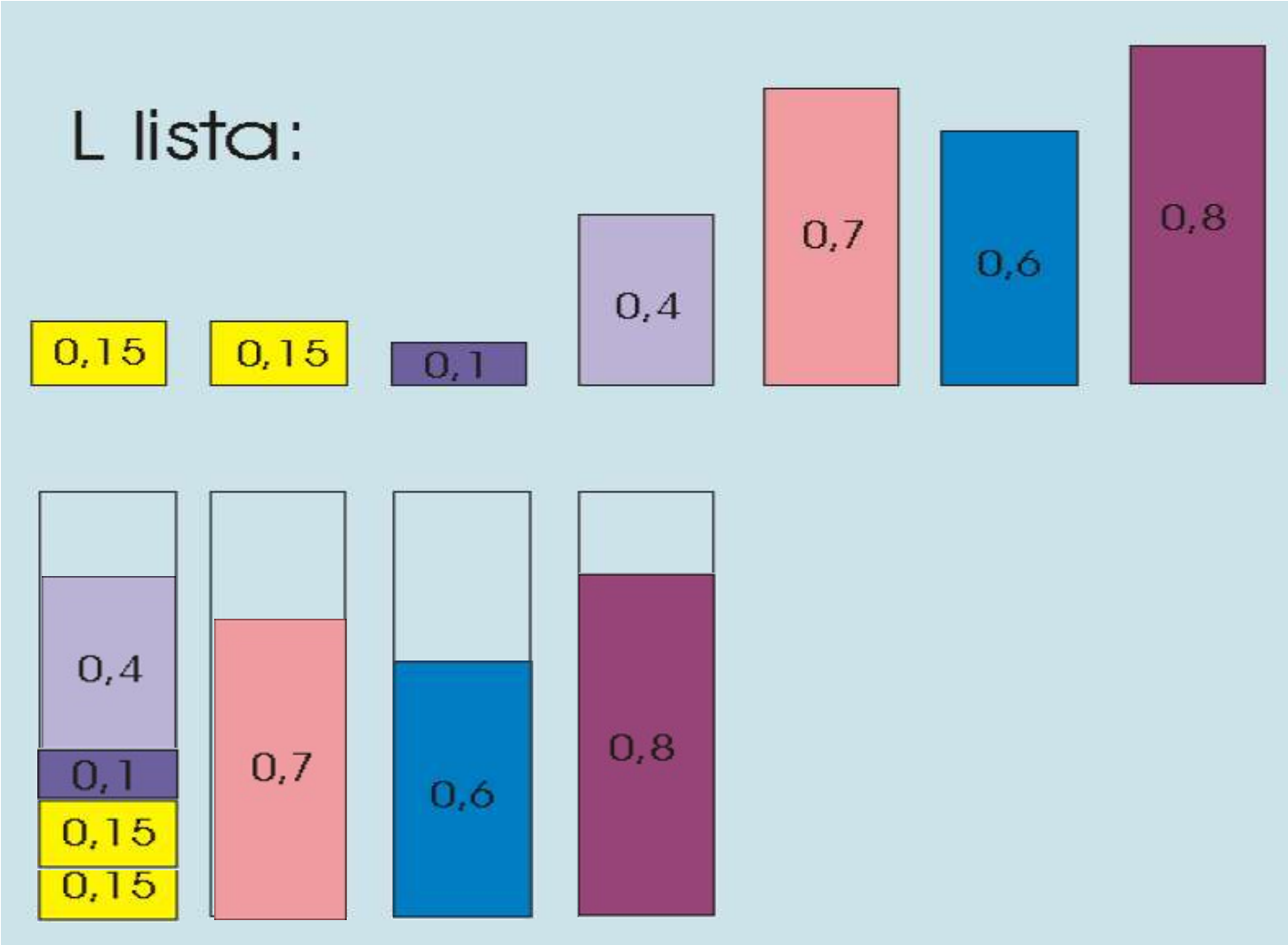
- A Best Fit (BF) algoritmus a soron következő tárgyat mindig abba a ládába helyezi, ahol a lehető legkevesebb üres hely marad.
- Az algoritmust gazdaságos algoritmusnak is hívják.
- Ennek az algoritmusnak a helyigénye ***Θ(n)***, időigenye pedig ***O(n²)***.
- Az algoritmus pszeudo kódja:

Algoritmus 1.2 BF(n, L)

```
C^BF ← 1
for i = 1 → n do
  b_i ← 0
end for
for i = 1 → n do
  szabad ← 1.0
  ind ← 0
  for k = 1 → C^BF do
    if b_k + a_i ≤ 1 and 1 − b_k − a_i < szabad then
      ind ← k
      szabad ← 1 − b_k − a_i
    end if
  end for
  if ind > 0 then
    b_ind ← b_ind + a_i
  else
    C^BF ← C^BF + 1
    b_{C^BF} ← a_i
  end if
end for
```

Best Fit (BF)

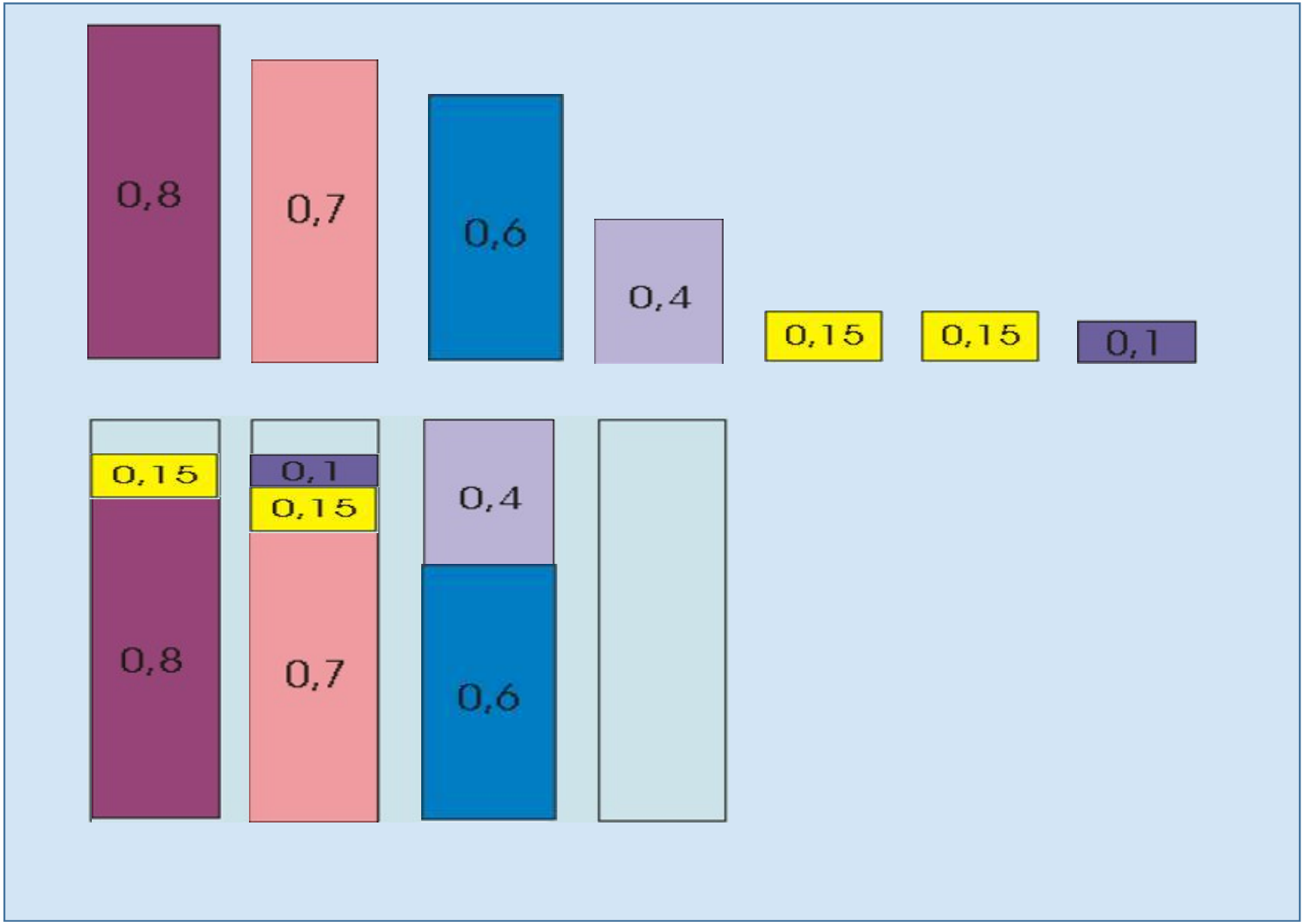
Megoldás: A(L)=4 láda szükséges, nem optimális megoldás



Az BFD algoritmus

- Először a lista elemeit nagyság szerinti csökkenő sorrendbe rendezzük, majd a második részben a rendezett lista elemeire alkalmazza a BF algoritmust.
- A rendező gazdaságos algoritmus (Best Fit Decreasing) a rendezés után a gazdaságos algoritmus (BF) szerint dolgozik, így helyigénye $\boldsymbol{\Theta(n)}$, időigénye pedig $\boldsymbol{O(n^2)}$.

Az BFD algoritmus: A(L)=3, optimális megoldás



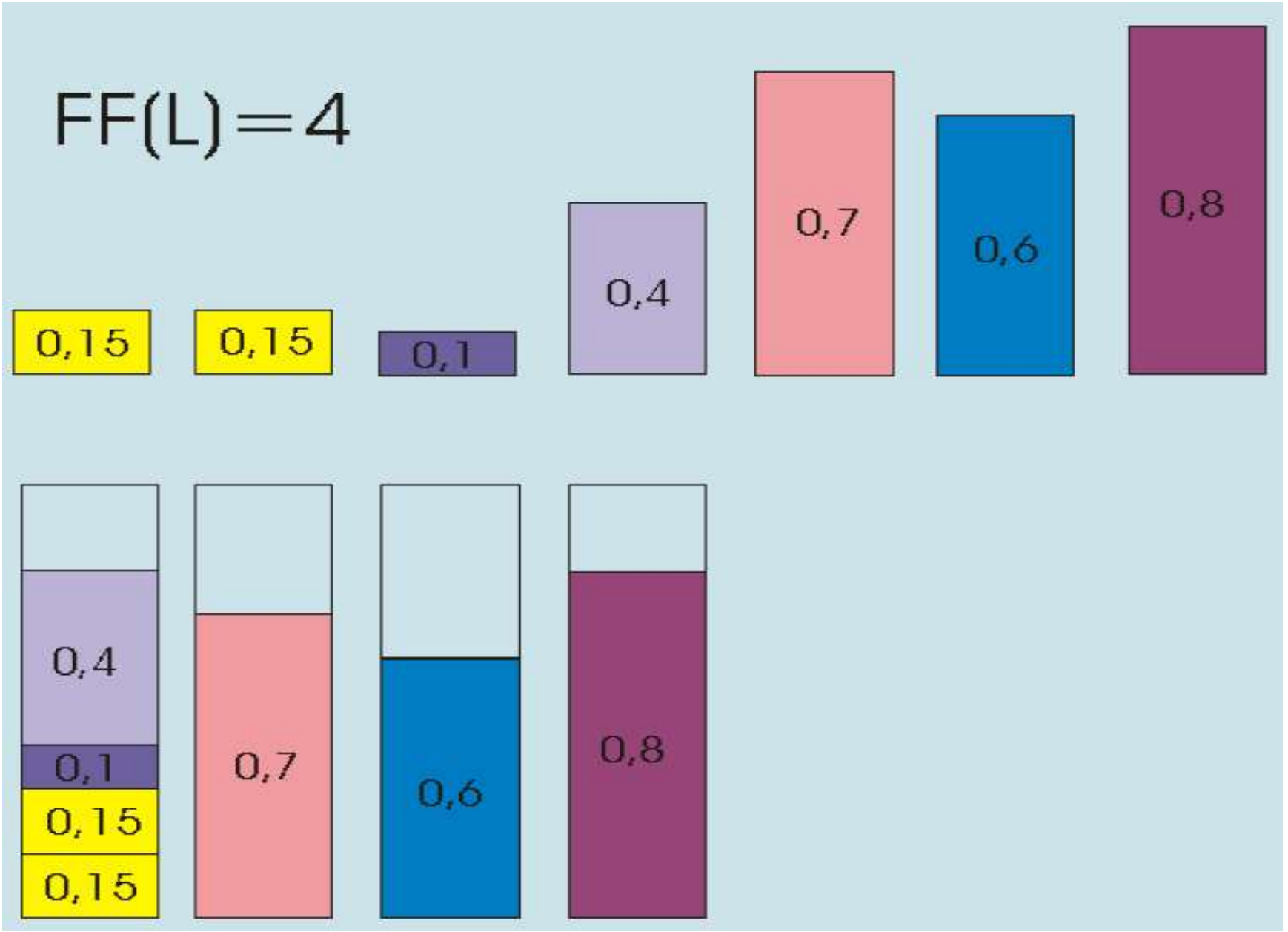
Az FF algoritmus - Mohó algoritmus

- A First Fit alapgondolata, hogy a következő tárgyat mindig az első olyan ládába helyezi, amelybe belefér.
- Azaz, amikor a_i-t kell elhelyeznünk, akkor a legalacsonyabb indexű ládába tesszük azok közül, melyek telitettsége nem nagyobb, mint $1-a_i$.
- Ha nincs ilyen, akkor új ládát nyitunk, melyben a_i lesz az első elem.
- Ennek az algoritmusnak a helyigénye $\Theta(n)$, időigénye $O(n^2)$.

- Az algoritmus pszeudo kódja:

Algoritmus 2.1 $\mathrm{FF}(n, L)$

```
C^FF ← 1
for i = 1 → n do
    b_i ← 0
end for
for i = 1 → n do
    k ← 1
    while b_k + a_i > 1 do
        k ← k + 1
    end while
    b_k ← b_k + a_i
    if k > C^FF then
        C^FF ← C^FF + 1
    end if
end for
```


FF(L)=4
0,15
0,15
0,1
0,4
0,7
0,6
0,8
0,4
0,1
0,15
0,15
0,7
0,6
0,8

A közelítő algoritmusok hatékonyságának mérése – jelölések bevezetése

- Abszolút versenyképességi hányados (**VK**):

  infimum R≥1, úgy, hogy minden inputra:

  $$A(L) \leq R\ OPT(L)$$

- Aszimptotikus versenyképességi hányados (**AVK**):

  infimum R≥1, úgy, hogy minden inputra:

  $$A(L) \leq R\ OPT(L) + c,$$

  ahol ***c*** az input méretétől független konstans.

- ***A(L)*** jelölje az ***L*** lista elpakolásához szükséges ládák számát, ha azt az ***A*** algoritmussal pakoljuk el.

Hatékonyság mérése az FF algoritmus esetében

- Az 40 éve ismert tény, hogy az aszimptotikus versenyképességi hányados

  AVK(FF)=1,7 úgy, hogy minden inputra:

  $$FF(L) \leq 1{,}7\, OPT(L) + 3$$

- Az abszolút versenyképességi hányados, AV(FF) 40 évig nyitott kérdés **VOLT**
- Többen az „additív konstanst” fokozatosan csökkentették. Az abszolút hányadosra azt kapjuk, hogy

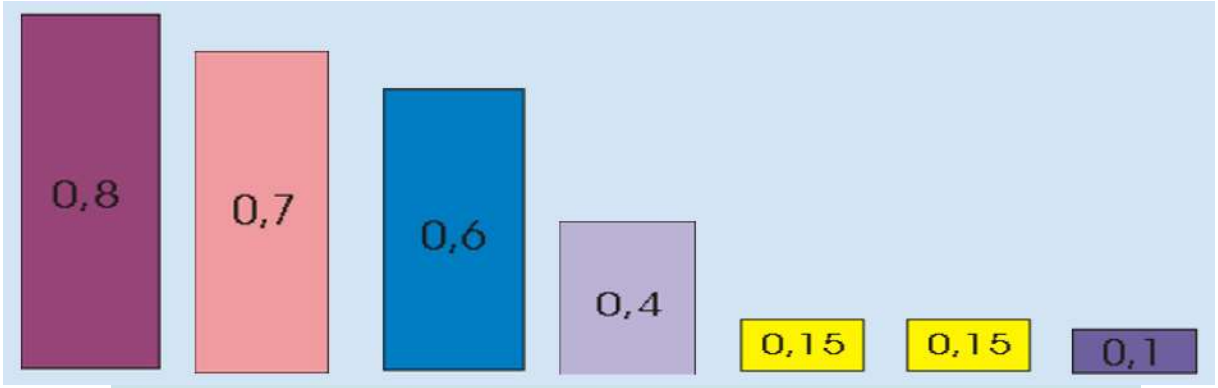
  $$AV(FF) \leq 12/7 = 1{,}7143$$

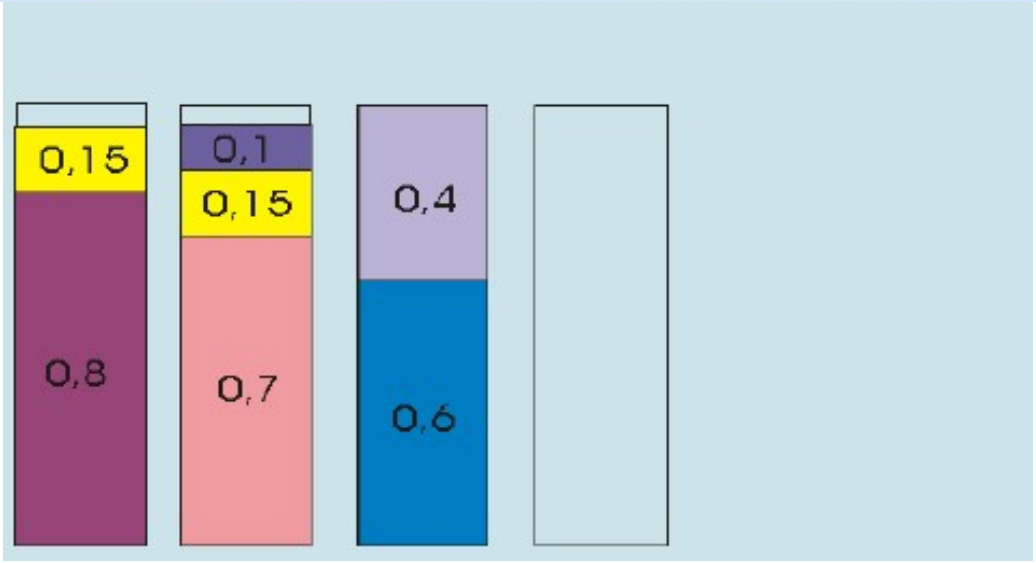
  Ezt többen is bizonyították a közelmúltban, egymástól függetlenül.
- 40 év után bizonyították, hogy

  a First Fit legfeljebb 1,7 OPT(L) alsó egész része számú ládát használ minden L lista esetén.

FFD algoritmus

- Az FF algoritmus eseten is értelmezhető a rendező változat. Az FFD (First Fit Decreasing) algoritmus először csökkenő sorrendbe rendezi a bemenő ***L ∈ D*** lista elemeit, majd FF szerint működik tovább.
- Helyigénye ***Θ(n)***, időigénye pedig ***O(n²)***.





$$AVK_{FFD} = \frac{11}{9} = 1,2222....$$

$$FFD(L) <= \frac{11}{9} OPT(L) + \frac{6}{9}$$

Egy másik jól ismert, már 1,7 alatti (1,69103) AVK-jú algoritmus a Harmonic Fit (HF)

- Az elemeket (és a ládákat is) osztályokba (típusokba) sorolja az alapján, hogy mely ***(1/(i+1), 1/i]*** osztályba esik az adott elem. Minden elem csak saját osztálybeli elemekkel pakolódik NF szabály szerint saját ládaosztályú ládákba ***(i=1,…,k-1)***, az utolsó osztály ***(0,1/k]***.
- Ha a méret ebbe az intervallumba esik:

  I_1= (1/2,1], akkor egy elem van minden ilyen típusú ládában,

  I_2= (1/3,1/2], akkor két elem van minden ilyen típusú ládában,

  …

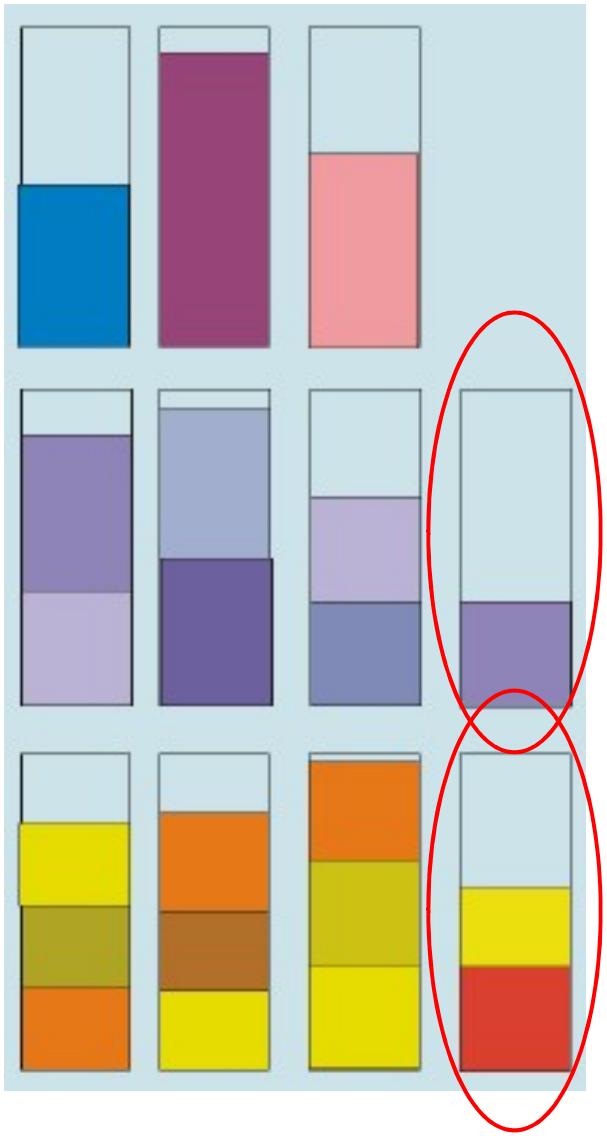
  I_{k-1}=(1/k,1/(k-1)], akkor k-1 darab elem van minden ládában,

  I_k=(0,1/k], akkor legalább k darab elem van (de bármilyen sok lehet)

  osztályonként max. 1 láda kivételével.
- Az utolsó osztálybeli elemeket kivéve, egy elem mérete csak az osztálya szempontjából számít.

Max. egy-egy láda (az utolsó, osztálybeli) kivételével

- i=1, Az 1. osztály minden megnyitott ládája legalább 1/2-ig tele (kivétel nélkül),
- i=2, a második osztályban legalább 2/3-ig tele,
- i=3, a harmadik osztályban legalább 3/4-ig tele,
- i=k-ra legalább 1-1/k-ig (=(k-1)/k) tele. Ha nem így lenne, akkor rá lehetett volna még tenni a következő láda első elemét, amely legfeljebb 1/k méretű.



Online ládapakolási algoritmusok

(„azonnali” vagy „vonali” algoritmusok)

- Abban a sorrendben pakoljuk el az elemeket, ahogy a listába azok „érkeznek”.
- Nem tudunk semmit az aktuális elem után érkező elemek méretéről.
- Nem ismerjük az esetlegesen érkező elemek számát sem.
- „Ami fekszik, nyugszik”. (Az egyszer elpakolt elemek többet nem mozgathatók.)

- A fenti korlátoknak az a következménye, hogy nincs olyan on-line algoritmus, amelynek aszimptotikus vagy abszolút versenyképességi hányadosa 1-hez közeli érték lenne.

Az első, 1,69103 alatti AVK-jú online algoritmus: RFF (Real First Fit)

Definiáljunk négy intervallumot:

- $I_1 = (1/2, 1]$,
- $I_2 = (2/5, 1/2]$,
- $I_3 = (1/3, 2/5]$,
- $I_4 = (0, 1/3]$.

Ezekhez rendeljünk 4 ládaosztályt is, B_1, B_2, B_3, B_4, és pakoljuk I_j elemeit B_j-be (j=1,2,3,4) az FF pakolás szerint, azzal az egy extra szabállyal, hogy I_3 minden 6. elemét pakoljuk mégis a B_1 osztályba, szintén FF szabállyal.

Ezzel a tárkorlátos algoritmusok AVK-jának alsó korlátja is megadható.

Belátható, hogy *RFF(L) ≤5/3 OPT(L)+5* minden L listára, azaz *AVK(RFF)=1,66666*.

Tény: Az 1/3-nál kisebb elemeket FF-fel pakolva 2 láda kivételével legalább 3/4-ig tele lesz mindegyik láda.

Egy példa:

L_1=6n darab 0,15,

L_2=6n darab 0,34,

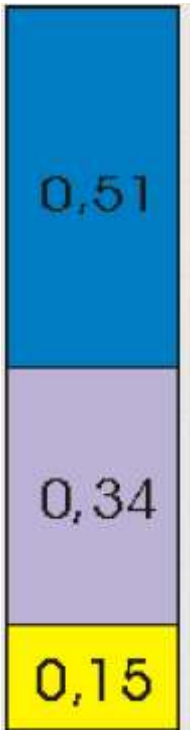
L_3=6n darab 0,51.

A fenti listákra

$OPT(L_1)=n$

$OPT(L_1L_2)=3n$

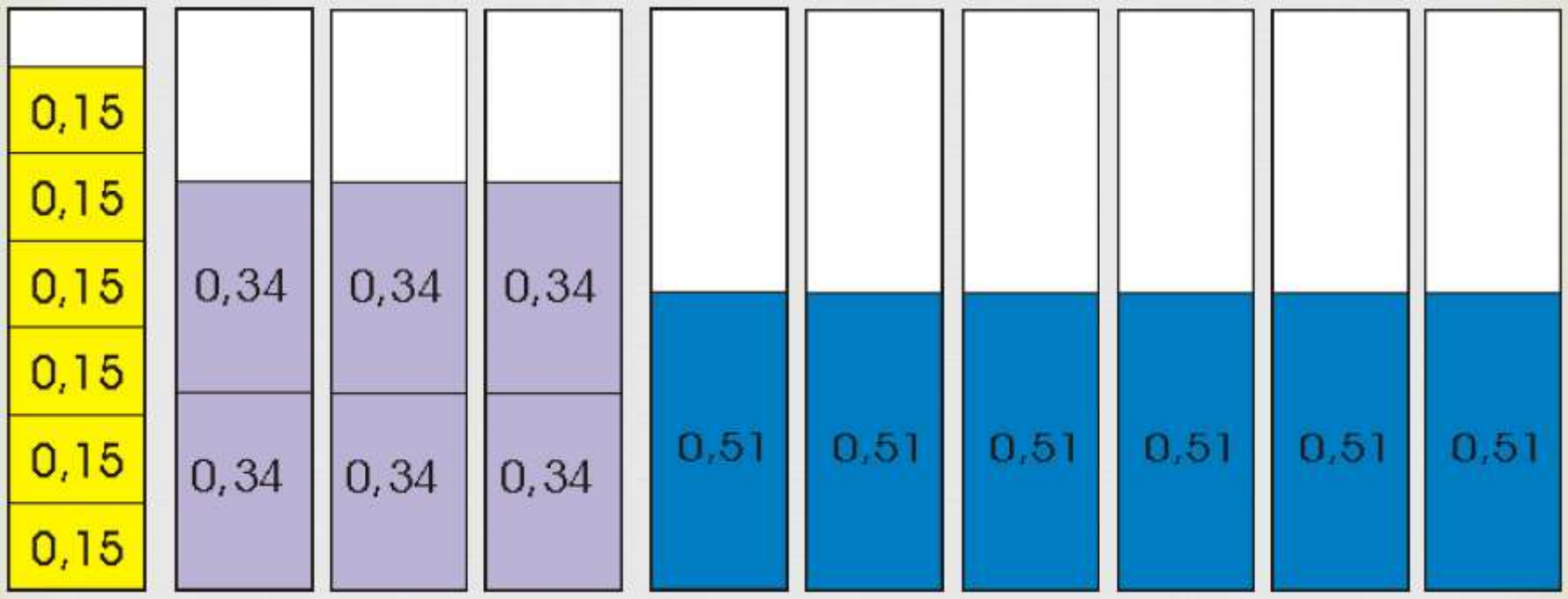
$OPT(L_1L_2L_3)=6n$



Azonban pl. a First Fit (FF) algoritmus ***10n*** ládát hoz létre.

n tetszőlegesen nagy lehet.

AVK(FF) ≥ 10/6 = 1,6666



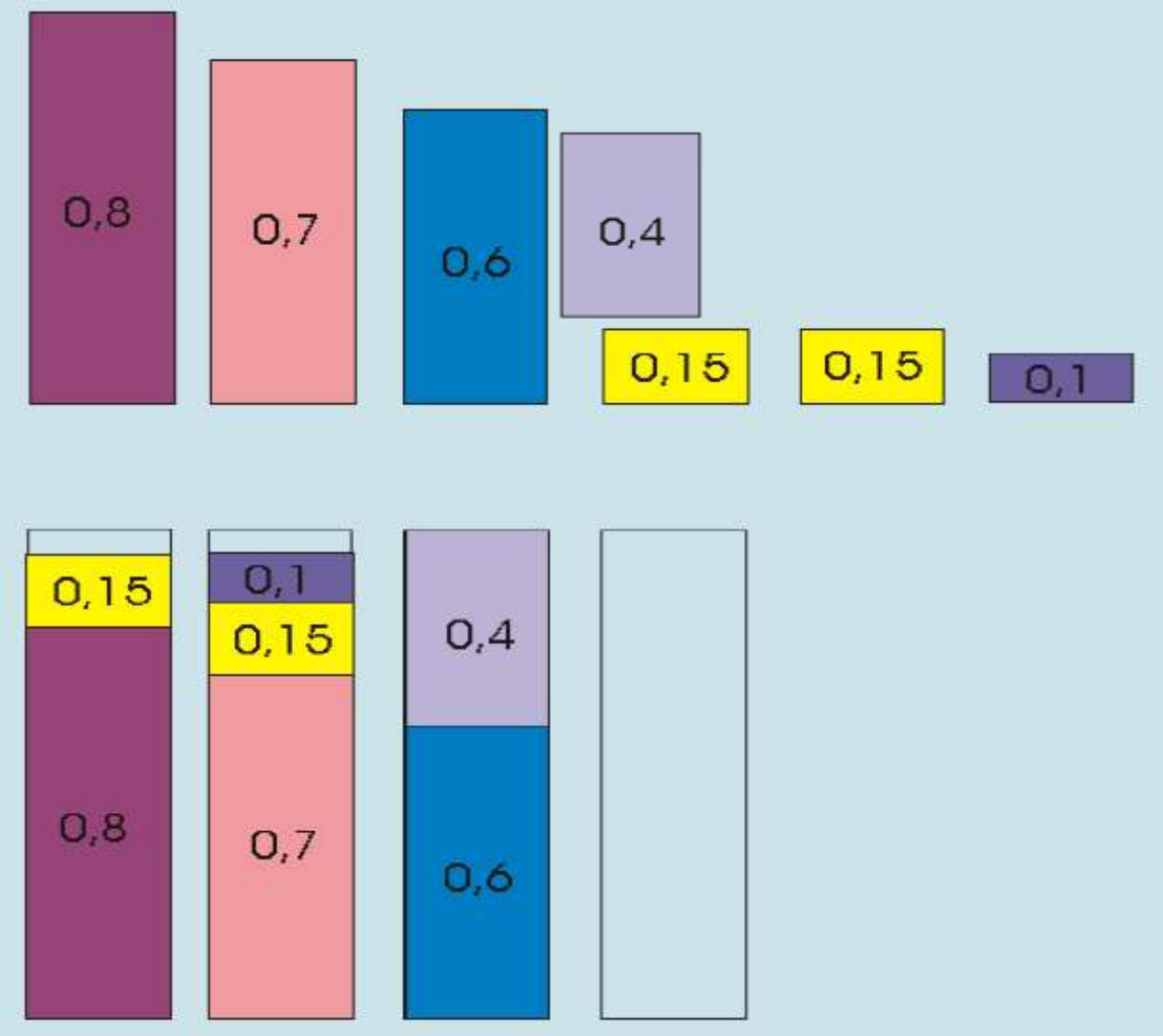
- Az L_1, L_2, L_3 listákkal belátható, hogy egyetlen *A* online algoritmus sem tudja az optimum 1,5-szeresénél jobban elpakolni mindhárom részlistát.
- Azaz nincs 1,5-nél jobb AVK-jú algoritmus.
- 1,5 egy alsó korlát minden online algoritmus AVK-jára.

Online algoritmusok

- Azt már viszonylag korán megfigyelték, hogy abban az esetben, ha az on-line algoritmusoknál feltételezett információnál némileg több információval rendelkezünk, akkor „jobb” algoritmusokat készíthetünk.
- Egy lehetőség: azt feltételezzük (tudjuk), hogy a tárgyak **nagyság szerint rendezett sorrendben** érkeznek, méret szerint csökkenő (nem-növekvő) sorrendben, de a pakoló algoritmusnak továbbra is az online szabályok szerint kell elpakolni az elemeket.

- First Fit Decreasing (FFD): először csökkenő sorrendbe rendezzük az elemeket, majd First Fit szabállyal pakoljuk. Ez, a legjobb ismert rendezett listákat pakoló algoritmus 1979-ből származik (Garey et al.).

1. Feladat megoldása online esetben



- Megmutatható, hogy az FFD AVK-ja 11/9 FFD (Garey et al. 1979):

$$AVK_{FFD} = \frac{11}{9} = 1{,}2222....$$

- Minden L listára (Dósa Gy., 2007): minden L listára

$$FFD(L) <= \frac{11}{9} OPT(L) + \frac{6}{9}$$

- ***Tétel*** *(Békési J., Galambos G., TCS 2012):* Nincs olyan *A* on-line algoritmus, amely az egydimenziós ládapakolási probléma megoldása során csökkenő sorrendbe rendezett listák érkezése esetén $AVK_A < \frac{54}{47} = 1{,}14894$

Félig online típusú ládapakolási algoritmusok

- az online szabályhoz képest megengednek valamiféle könnyebbséget
- Pl. minden lépésben az előző lépés eredményét átrendezhetjük, átpakolhatunk rögzített számú, *k* darab, korábban elpakolt elemet ládák között.
- Pl. az algoritmus előre ismeri, hogy optimálisan hány ládába tudnánk elpakolni az elemeket (megkapja OPT(L) értékét, de ezen kívül az online szabály szerint kell pakolnia).

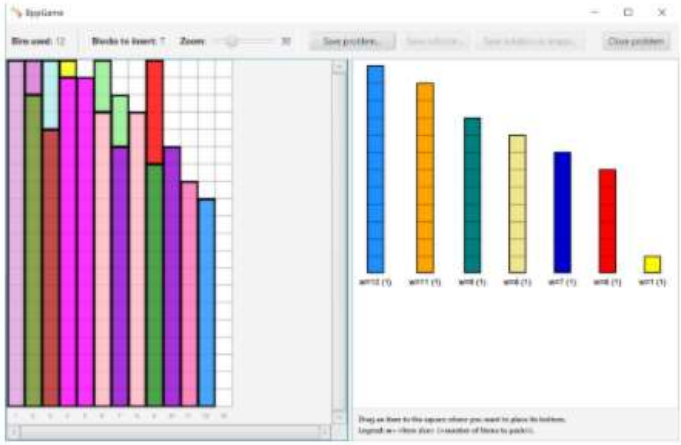
Benchmark-ok előre átgondolt feladatok algoritmusok teszteléséhez

Linkek:

- http://or.dei.unibo.it/library/bpplib
- https://www2.wiwi.uni-jena.de/Entscheidung/binpp/index.htm

BPPLIB - A Bin Packing Problem Library

Page maintained by M. Delorme (m.delorme@tilburguniversity.edu), M. Iori (manuel.iori@unimore.it), and S. Martello (silvano.martello@unibo.it).



The BPPLIB is a collection of codes, benchmarks, and links for the one-dimensional Bin Packing and Cutting Stock problem.

If you need to refer to material taken from this library, please cite M. Delorme, M. Iori, and S. Martello. BPPLIB: A library for bin packing and cutting stock problems. *Optimization Letters*, 12(2):235-250, 2018.

Problem description

The bin packing problem (BPP) can be informally defined in a very simple way. We are given n items, each having an integer weight w_j $(j = 1, \ldots, n)$, and an unlimited number of identical bins of integer capacity c. The objective is to pack all the items into the minimum number of bins so that the total weight packed in any bin does not exceed the capacity.

Similarly, in the cutting stock problem (CSP), we are given m item types, each having an integer weight w_j and an integer demand d_j $(j = 1, \ldots, m)$, and an unlimited number of identical bins (frequently called rolls in the literature) of integer capacity c. The objective is to produce d_j copies of each item type j using the minimum number of bins so that the total weight packed in any bin does not exceed its capacity. A CSP instance can be easily transformed into an equivalent BPP instance and vice-versa.

For a recent **survey** on these problems, see M. Delorme, M. Iori, and S. Martello. Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms. *European Journal of Operational Research*, 255(1):1-20, 2016.

Benchmarks

All instances are given in the two following formats:

- In the BPP format:
  - Number of items (n)
  - Capacity of the bins (c)
  - For each item j ($j = 1,\ldots,n$):
    - Weight (w_j)
- In the CSP format:
  - Number of item types (m)
  - Capacity of the bins (c)
  - For each item type j ($j = 1,\ldots,m$):
    - Weight (w_j) and demand (d_j)

The solution of all instances can be found here. The Excel file also contains the sets of instances that were selected in the experimental evaluation of M. Delorme, M. Iori, and S. Martello. Bin Packing and Cutting Stock Problems: Mathematical Models and Exact Algorithms. *European Journal of Operational Research*, 255(1):1-20, 2016.

Falkenauer (BPP - CSP)

These are the instances used by E. Falkenauer in A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2(1):5-30, 1996. They were downloaded from the OR-library. They are divided into two classes of 80 instances each: the first class has uniformly distributed item sizes ('Falkenauer U') with n between 120 and 1000, and c = 150. The instances of the second class ('Falkenauer T') includes the so-called triplets, i.e., groups of three items (one large, two small) that need to be assigned to the same bin in any optimal packing, with n between 60 and 501, and c = 1000.

Scholl (BPP - CSP)

These are the instances used by A. Scholl, R. Klein, and C. Jürgens in Bison: a fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research*, 24(7):627-645, 1997. They were downloaded from the authors' web page. They are divided into three sets of 720, 480, and 10, respectively, uniformly distributed instances with n between 50 and 500. The capacity c is between 100 and 150 (set 'Scholl 1'), equal to 1000 (set 'Scholl 2'), and equal to 100 000 (set 'Scholl 3'), respectively;

Wäscher (BPP - CSP)

These are some of the instances (those regarded as the most difficult ones) used by G. Wäscher and T. Gau in Heuristics for the integer one-dimensional cutting stock problem: a computational study. *OR Spectrum*, 18(3):131-44, 1996. They were downloaded from the ESICUP website. These 17 hard instances have n between 27 and 239 and c = 10 000.

Schwerin (BPP - CSP)

These are the instances used by P. Schwerin and G. Wäscher in The bin-packing problem: a problem generator and some numerical experiments with FFD packing and MTP. *International Transactions in Operational Research*, 4(5-6):377-389, 1997. They were downloaded from the ESICUP website. They are divided into two sets ('Schwerin 1' and 'Schwerin 2') of 100 instances each with n = 100 and n = 120, respectively, and c = 1000;

Mi volt a probléma?

- A ládapakolási feladatok esetében tárgyakat pakolunk ládákba.
- Minden tárgy rendelkezik egy w_i mérettel, és minden láda egy C kapacitással.
- A pakolást úgy végezzük el, hogy a ládákba pakolt tárgyak összmérete ne lépje túl a láda kapacitását, valamint minimális számú ládát használjunk fel.

Schwerin

- Megpróbáljuk megoldani a feladatokat a legegyszerűbb algoritmussal (pl. FF, BF, NF)
- Ha nem vagyunk elégedettek, akkor jön a következő nehézségi fok (pl. FFD)
- Ha nem vagyunk továbbra sem elégedettek, akkor + heurisztika (pl. FFD+heurisztika)
- További kategóriák - metaheurisztika (pl. hibrid evolúciós algoritmus: Egy egyed egy megoldást ír le a hozzá tartozó ládákkal és tartalmukkal. Az algoritmus rekombináció művelet nélkül dolgozik, két új mutáció műveletet alkalmaz és helyi kereső eljárásokkal javítja a megoldásokat. A mutáció műveletek egy relativ-pár frekvencia mátrix alapján működnek. A frekvencia mátrix minden tárgypár esetén becsült valószínűséget ad arra, hogy egy ládába kerülhetnek.)

Egy sikeres megközelítése a feladat osztálynak

- A Schwerin 1 csoportba tartozó feladatok esetében a láda kapacitása **C = 1000**,
- a tárgyak száma **n = 100**,
- a tárgyak méretei a **[150, 200]** intervallumból kerülnek ki véletlenszerűen, egyenletes eloszlással.
- Az ebbe a csoportba tartozó feladatok mindegyikénél az optimális megoldás **18 darab felhasznált láda**.
- Heurisztika: Ha közelebbről megvizsgáljuk a feladatokat, akkor látható, hogy a 100 tárgyból 40 darabot automatikusan, "gondolkodás nélkül" tudunk pakolni, és csak 60 azon tárgyak száma, ahol már figyelni kell a pakolásokra. Hogyan?

Minta egy feladatra

Ládaméret: 1000
Tárgyak száma: 100
LB: 18

Tárgyak méretei a [150, 200] intervallumból kerülnek ki

Összes tárgy:

200 200 200 199 199 199 199 199 197 197 197 196 196 196 195
195 194 194 194 193 193 192 192 192 189 189 189 189 188 188
187 187 187 186 185 185 184 184 183 182 181 181 180 179 179
179 179 178 178 178 177 177 177 177 175 174 174 174 173 173
173 172 171 171 170 168 168 166 165 164 164 162 161 161 160
160 159 159 159 158 158 157 156 156 155 155 155 155 154 153
153 153 152 152 151 151 151 150 150 150

Két tulajdonságot tudunk észre venni

- Bármelyik hét tárgyat nem tudjuk egy ládába pakolni, ugyanis ha a legkisebb tárgyméretből (150) hetet veszünk, akkor $7 \times 150 = 1050 > 1000$
- Bármely öt tárgyat tudjuk egy ládába pakolni, ugyanis ha a legnagyobb tárgyméretet vesszük (200), akkor $5 \times 200 = 1000$
- Ezért minden láda 5 vagy 6 darab tárgyat fog tartalmazni. Következésképpen 8 láda (a 18-ból) 5 tárgyat tartalmaz.
- Így a 40 legnagyobb tárgy elpakolható 8 ládába úgy, hogy az optimalitás nem sérül.
- Ezután már csak a maradék 60 tárgyat kell elpakolni, itt viszont már jól meg kell gondolni az eljárást!

Feladat: Schwerin1_BPP8_SOLUTION

Ládaméret: 1000
Tárgyak száma: 100
LB: 18

Megoldás lépései:

A(z) 9. láda pakolása

Pakolt tárgyak:

181 181 180 153 153 152

Összes tárgy:

200 200 200 199 199 199 199 199 197 197 197 196 196 196 195
195 194 194 194 193 193 192 192 192 189 189 189 189 188 188
187 187 187 186 185 185 184 184 183 182 181 181 180 179 179
179 179 178 178 178 177 177 177 177 175 174 174 174 173 173
173 172 171 171 170 168 168 166 165 164 164 162 161 161 160
160 159 159 159 158 158 157 156 156 155 155 155 155 154 153
153 153 152 152 151 151 151 150 150 150

A láda töltöttsége: 1000

A(z) 18. láda pakolása

Pakolt tárgyak:

151 151 151 150 150 150

Összes tárgy:

200 200 200 199 199 199 199 199 197 197 197 196 196 196 195
195 194 194 194 193 193 192 192 192 189 189 189 189 188 188
187 187 187 186 185 185 184 184 183 182 181 181 180 179 179
179 179 178 178 178 177 177 177 177 175 174 174 174 173 173
173 172 171 171 170 168 168 166 165 164 164 162 161 161 160
160 159 159 159 158 158 157 156 156 155 155 155 155 154 153
153 153 152 152 151 151 151 150 150 150

A láda töltöttsége: 903

Megoldás: 18 db láda
Összes töltöttség: 17591 egység

Az algoritmus pszeudo kódja

Algorithm 8: REM SW algoritmus

Input: a ládapakolási feladat elemei

Output: az elemek pakolása C méretű ládákba

1 Legyen k a legnagyobb olyan szám, amelyre $0 \leq k \leq 6$, továbbá teljesül az, hogy a k legnagyobb és $6-k$ legkisebb tárgy belefér egy ládába. (Ha nincs ilyen k, az algoritmus megáll.)

2 Pakoljuk a k darab legnagyobb tárgyat egy ládába.

3 Alkalmazzuk az ûtkereső algoritmust a ládában fennmaradó hely lehető legjobb betöltésére.

4 A pakolt tárgyak eltávolítása a rendszerből.

5 Ugorjunk az 1. lépésre, ha van még tárgy, egyébként az algoritmus leáll.

Gy. Ábrahám and Gy. Dósa and T. Dulai and Zs. Tuza and Á. Werner-Stark. Efficient Pre-Solve Algorithms for the Schwerin and Falkenauer_U Bin Packing Benchmark Problems for Getting Optimal Solutions with High Probability. *Mathematics*, 9(13):1540, 2021 (**IF: 2.542**).

Két dimenziós ládapakolási probléma jellemzői

- A klasszikus két dimenziós ládapakolási feladat esetén téglalap alakú tárgyakat kell elhelyezni minél kevesebb 1 egység oldalú ládában.
- A kis téglalapok magassága és szélessége sem nagyobb, mint 1 egység.
- A tárgyakat úgy kell elhelyezni, hogy azok oldalai a láda falaival párhuzamosan helyezkedjenek el, a ládából ne lógjanak ki és egymást ne fedjék.
- Két dimenzióban már felmerül a kérdés, hogy lehessen-e forgatni a téglalapokat, vagy az eredeti helyzetben kelljen-e őket elhelyezni.
- Ha nem lehet őket forgatni, akkor könnyen látható, hogy a téglalap alakú ládába pakolás feladata is modellezhető a négyzetbe pakolással.
- Amennyiben a forgatás megengedett, akkor a téglalap alakú ládába pakolás már nem vezethető vissza olyan egyszerűen a négyzet alakú ládába pakolásra.

Két dimenziós ládapakolási probléma jellemzői

- Fontos szerepe van annak, hogy előzetesen ismerjük-e az összes elpakolandó tárgy méretét, ez alapján megkülönböztethetünk offline és online feladatokat.
- Offline feladat esetén az összes tárgy paramétereit ismerjük, mielőtt elkezdenénk a pakolást.
- Online esetben csak a pakolás során tudjuk meg, hogy milyen tárgyakat kell elpakolnunk; azaz kezdetben egy láda paramétereit ismerjük, és mindig csak akkor ismerünk meg egy új tárgyat, ha az ismertekből egyet elpakolunk.
- Olyan feladat is elképzelhető, hogy egyszerre több, de korlátos sok olyan tárgy lehet, aminek ismerjük a méreteit, de még nem pakoltuk el.
- Online esetben az alapján is lehet osztályozni a feladatokat, hogy mennyi ládát használ egyszerre.

Két dimenziós ládapakolási probléma jellemzői

- Ha nem használunk ilyen megkötést, akkor a feladat megoldására kitalált algoritmus futási ideje nagyon megnőhet. Ugyanis, ahányszor új ládát nyitunk meg, azaz olyanba pakolunk, amibe addig még nem, akkor eggyel növeljük a megnyitott ládák számát.
- Ha az összes nyitott ládát megvizsgálja az algoritmus, mielőtt egy tárgyat elhelyezne, akkor egy-egy tárgy elhelyezése O(n) nagyságrendű időbe is telhet, ahol n az összes elpakolni való tárgy számát jelöli.
- Ezt az időt úgy lehet csökkenteni, hogy maximalizáljuk azoknak a ládáknak a számát, amit az algoritmus megvizsgál az elemek elhelyezésekor. Ehhez definiáljuk a lezárt és aktív ládákat.
- Azokat a ládákat nevezzük aktívnak, amiben már van legalább egy tárgy, és az algoritmus még pakolhat bele tárgyakat.
- Azt mondjuk, hogy egy algoritmus egy aktív ládát lezár, ha utána már soha többé nem nézi meg, hogy rakjon-e bele tárgyat; az ilyen ládát lezártnak nevezzük.
- Ha az aktív ládák maximális számára van megkötés, akkor a feladatot korlátos ládaszámúnak nevezzük.