

# **A bináris keresés elvének alkalmazása algoritmusok hatékonyságának növelése céljából**

**Dr. Ionescu Klára**

**[clara@cs.ubbcluj.ro](mailto:clara@cs.ubbcluj.ro)**

# Bevezetés

Az *Oszd meg és uralkodj* módszer akkor *ajánlott* amikor:

- a feladatot fel lehet bontani *egymástól független részfeladatokra*,
- amelyeket az *eredeti feladathoz hasonlóan oldunk meg*, de kisebb méretű adathalmaz esetében.

# Az algoritmus

1. Az eredeti feladatot *felbontjuk* egymástól *független részfeladatokra*: az eredetihez hasonlóak, de kisebb adathalmazra definiáltak.
2. A részfeladatokkal hasonlóan járunk. A felbontást *akkor állítjuk le*, amikor a feladat megoldása a lehető *legegyszerűbb*.
3. Ezt a legegyszerűbb feladatot *megoldjuk*.
4. A részfeladatok eredményeiből fokozatosan felépítjük mindig a következő méretű feladat eredményeit, ezek *összerakása* által. Az utolsó összerakás az eredeti feladat eredményét adja.

# Megjegyzések

1. A részfeladatok csak méreteikben különböznek az eredeti feladattól  $\Rightarrow$  a *Divide et Impera* módszert **rekurzívan** fejezzük ki.
2. A **felbontás** megtörténik a rekurzióba való **belépéskor**, a **részeredmények összerakása** pedig a **kilépéskor**.

# A módszer általános bemutatása

- A **DivImp(bal, jobb)** algoritmus az  $a_1, a_2, \dots, a_n$  sorozatot dolgozza fel, tehát **DivImp(1, n)** alakban hívjuk meg először.
- Formális paraméterei **bal** és **jobb**, amelyek az aktuális részsorozat bal és jobb indexei.
- Ha nem szeretnénk globális változókkal dolgozni, ehhez hozzáadódnak: **n** és **a**.
- Az *Oszd meg és uralkodj* stratégiát lehet **iteratívan is** implementálni (például, *bináris keresés*).
  - Ezek az algoritmusok mindig gyorsabbak lesznek.

**Algoritmus** DivImp(bal, jobb, eredmény):

**Ha** jobb – bal <  $\varepsilon$  **akkor**

Megold(bal, jobb, eredmény)

**különben**

Feloszt(bal, jobb, közép)

DivImp(bal, közép, eredmény1)

DivImp(közép+1, jobb, eredmény2)

Összerak(eredmény1, eredmény2, eredmény)

**vége(ha)**

**Vége(algoritmus)**

A „leghíresebb” és egyben a leggyakrabban alkalmazott Oszd meg és Uralkodj típusú algoritmus a **„bináris keresés”**.

# Bináris keresés

Adott egy  $n$  egész számból álló, növekvő sorrendbe **rendezett** sorozat:  $x_1 < x_2 < \dots < x_n$ . Állapítsuk meg egy adott szám helyét a sorozatban! Ha az illető szám nem található meg a sorozatban, írjunk ki egy megfelelő üzenetet!

## Elemzés

Az elemet a sorozat közepén fogjuk először keresni.

1.  $\text{keresett} = x_{\text{közép}} \Rightarrow$  **keresett** a sorozatban a **közép** helyen van
2.  $\text{keresett} < x_{\text{közép}} \Rightarrow$  mivel a sorozat rendezett, a keresett számot a sorozat első ( $x_1, \dots, x_{\text{közép}-1}$ ) felében keressük tovább
3.  $\text{keresett} > x_{\text{közép}} \Rightarrow$  a keresett számot a sorozat második ( $x_{\text{közép}+1}, \dots, x_n$ ) felében keressük tovább

A feladat **átalakul ugyan két feladattá**, de **csak az egyiket kell megoldani**.

**Nincs szükség a divide et impera harmadik lépésére (a részeredmények összerakására).**

**Algoritmus** BinKeres(bal, jobb):

**Ha**  $\text{bal} > \text{jobb}$  **akkor**      { közép = a keresett helye, ha megtalálható }  
    **térítsd** -1                              { ha keresett nincs a sorozatban }

**különben**

    közép  $\leftarrow (\text{bal} + \text{jobb}) \text{ div } 2$

**Ha** keresett  $> x_{\text{közép}}$  **akkor**

**térítsd** BinKeres(közép + 1, jobb)

**különben**

**Ha** keresett  $< x_{\text{közép}}$  **akkor**

**térítsd** BinKeres(bal, közép - 1)

**különben**

**térítsd** közép

**vége**(ha)

**vége**(ha)

**vége**(ha)

**Vége(algoritmus)**

**Megjegyzés:** ha a számok nem különbözők, és *keresett* többször is előfordul, az algoritmus a keresett *tetszőleges* előfordulását téríti.



**Algoritmus** IteratívBinKeres( $n, x, keresett$ ):

bal  $\leftarrow 1$

jobb  $\leftarrow n$

**Amíg** bal  $\leq$  jobb **végezd el:**

közép  $\leftarrow (bal + jobb) \text{ div } 2$

**Ha**  $x_{közép} = keresett$  **akkor**

**térítsd** közép

*{ ha megtaláltuk, vége }*

**különben**

**Ha**  $x_{közép} > keresett$  **akkor**

jobb  $\leftarrow közép - 1$

**különben**

bal  $\leftarrow közép + 1$

**vége(ha)**

**vége(ha)**

**vége(amíg)**

**térítsd** -1

*{ csak, ha nem találtuk meg }*

**Vége(algoritmus)**

# A bináris keresés alkalmazásai

- Következik néhány feladat, amelyeknek megoldásaiban felhasználjuk a bináris keresést.
- Ezekben a *keresés* nem jelenik meg, mint explicit részfeladat, hanem az eredmény egy olyan érték, amelynek értéktartományát ismerjük, ugyanakkor lehetséges a „találgatás” a bináris keresés algoritmusának alkalmazásával.
- Tehát: a bináris keresés algoritmusai eredményesen alkalmazható algoritmusok **optimalizálásának** érdekében.
- Bizonyos feladattípusok esetében, a megoldásként javasolt **lineáris** algoritmus **egy logaritmikussal helyettesíthető**, ha felhasználjuk a **bináris keresés elvét**.

# 1. Keresztmetszet (Mat-Info verseny, 2016)

- Legyen két sorozat, amelyeknek elemei különböző természetes számok: az  $a$  sorozat elemeinek száma  $n$  ( $0 < n \leq 10\,000$ ), a  $b$  sorozat elemeinek száma  $m$  ( $0 < m \leq 10\,000$ ) és növekvően rendezett.
- Határozzuk meg azt a  $c$  sorozatot, amelynek  $k$  ( $0 < k \leq 10\,000$ ) eleme lesz, és amely a két sorozat minden közös elemét egyszer tartalmazza.

**Példa:** ha  $n = 4$ ,  $a = (5, -7, -2, 3)$ ,  $m = 5$  és  $b = (-2, 3, 5, 7, 8)$ , a  $c$  sorozatnak  $k = 3$  eleme van:  $c = (5, -2, 3)$ .

# Elemzés

- Ismerjük a klasszikus algoritmust, amely két *nem* rendezett sorozat keresztmetszetét határozza meg (programozási tétel). Ennek bonyolultsága  $O(n*m)$ .
- „Észrevesszük”, hogy a *b* sorozat rendezett. De az említett algoritmus ezt nem „kéri” ...
- De ne maradjon a tulajdonság kihasználatlanul! Így a *b* sorozatban rendre keressük az *a* sorozat elemeit... **bináris kereséssel!**

**Algoritmus** közösElemek( $n, a, m, b, k, c$ ):

$k \leftarrow 0$  { egyelőre a c sorozatnak nincs egyetlen eleme sem }

**Minden**  $i = 1, n$  **végezd el:** { az a elemait keressük rendre b-ben }

megvan  $\leftarrow$  hamis

bal  $\leftarrow 1$ ; jobb  $\leftarrow m$  { a b rendezett, lehetséges a bináris keresés }

**Amíg nem** megvan **és** bal  $\leq$  jobb **végezd el:**

közép  $\leftarrow (bal + jobb) / 2$

**Ha**  $a_i = b_{közép}$  **akkor** { megtaláltuk:  $a_i$  közös }

$k \leftarrow k + 1$

$c_k \leftarrow a_i$  { elhelyezzük a c sorozatban }

megvan  $\leftarrow$  igaz { a keresés leáll }

**különben**

**Ha**  $a_i < b_{közép}$  **akkor**

jobb  $\leftarrow$  közép  $- 1$  { tovább keresünk a b sorozat bal felében }

**különben**

bal  $\leftarrow$  közép  $+ 1$  { tovább keresünk a b sorozat jobb felében }

**vége(ha)**

**vége(ha)**

**vége(amíg)**

**vége(minden)**

**Vége(algoritmus)**

## 2. S összegű elemek kiválasztása

Legyen egy  $n$  elemű ( $3 \leq n \leq 100\,000$ ), különböző természetes számokat tartalmazó sorozat és az  $S$  természetes szám.

Válasszunk ki az adott sorozatból **három elemet, amelyeknek az összege  $S$ !** Adjunk meg minden megoldást!

### Elemzés

- Részletösszegeket kell számítanunk: három elem összegét hasonlítjuk  $S$ -sel.
- A feladat megoldható három egymásba ágyazott **Minden** ciklussal is (az  $n$  értéke miatt, ez a megoldás időigényes; hiába ügyeskednénk **Amíg** ciklusokkal, illetve a lépésszámok csökkentésével, a bonyolultság továbbra is  $O(n^3)$  lenne).

# Megoldás

- Ha a megoldásba beépítjük a bináris keresést, a bonyolultság  $O(n^2 \cdot \log n)$  lesz:
- Ahhoz, hogy alkalmazhassuk, előbb **rendezzük** az adott sorozatot (ennek bonyolultsága pl.  $O(n^2)$ ).
- Két **Amíg** ciklussal kiválasztunk a sorozatból két elemet (legyen ezeknek indexe  $n_1$  és  $n_2$ ). Ennek bonyolultsága  $O(n^2)$ .
- **Megkeressük** az  $S - a_{n_1} - a_{n_2}$  értéket a **bináris keresést alkalmazva**. Ennek bonyolultsága  $O(\log n)$ , de  $n^2$ -szer.
- A megoldást tovább javítjuk (például, ha  $a_{n_1}$  értéke meghaladja  $S$ -t, kilépünk az első **Amíg**-ből stb.).
- Tehát az algoritmus bonyolultsága:

$$O(n^2) + O(n^2 \cdot \log n) = O(n^2 \cdot \log n)$$

**Algoritmus** Generál( $a, n, S$ ):

$i \leftarrow 1$

**Amíg**  $i < n-1$  **és**  $a_i < S$  **végezd el:**

$j \leftarrow i + 1$

**Amíg**  $j < n$  **és**  $a_i + a_j < S$  **végezd el:**

BinKeres( $a, j+1, n, S - a_i - a_j, k$ )

*{ ha  $S - a_i - a_j$  megtalálható a sorozatban, }*

*{ akkor  $k$  értéke egy valid index, különben  $k$  értéke 0 }*

**Ha**  $k \neq 0$  **akkor**

**Ki:**  $i, j, k$

**vége(ha)**

$j \leftarrow j + 1$

**vége(amíg)**

$i \leftarrow i + 1$

**vége(amíg)**

**Vége(algoritmus)**



### 3. Négyzetszámok darabszáma

*Számoljuk meg egy  $n$  ( $1 \leq n \leq 1\,000\,000$ ) elemű sorozat négyzetszámaait!*

A számok nem nagyobbak  $1\,000\,000$ -nál.

#### **Megoldás**

Részfeladat: ellenőriznünk kell, hogy egy szám négyzetszám-e?  
Tudjuk már, hogy a következő algoritmus nem lesz kielégítő hatékonyságú.

**Algoritmus Számol\_1(n, a, p):**

$p \leftarrow 0$

**Minden  $i = 1, n$  végezd el:**

**Ha  $\text{négyzetgyök}(a_i) = \lfloor \text{négyzetgyök}(a_i) \rfloor$  akkor**

$p \leftarrow p + 1$

**vége(ha)**

**vége(minden)**

**Vége(algoritmus)**

1. Tudjuk, hogy a négyzetgyök kiszámítása elkerülendő, mivel időigényesebb, mint gondolnánk. Főleg, ha sokszor kell alkalmaznunk.
2. Először is az **Amíg** feltételéből „kiemeljük” az  $a_i$ -t **szám**-ba, mivel  $a_i$ -t **megkeresni a memóriában**, (az  $i$  index, a tömb első elemének címe és az  $a$  tömb típusának megfelelő elemhossz alapján) több idő, mint egy egyszerű változóban tárolt értéket keresni.
3. Azt hogy egy adott szám négyzetszám-e, hatékonyabban is tudjuk ellenőrizni:

**Algoritmus** Számol\_2( $n, a, p$ ):

$p \leftarrow 0$

*{  $p$  a négyzetszámokat számlálja }*

**Minden**  $i = 1, n$  **végezd el:**

$k \leftarrow 0$

*{  $a$   $k$  négyzetét hasonlítjuk  $a_i$ -vel }*

$\text{szám} \leftarrow a_i$

*{ az  $a_i$ -t „megkeresni” nehezebb }*

**Amíg**  $k * k < \text{szám}$  **végezd el:**

$k \leftarrow k + 1$

**vége(amíg)**

**Ha**  $k * k = \text{szám}$  **akkor**

$p \leftarrow p + 1$

**vége(ha)**

**vége(minden)**

**Vége(algoritmus)**

# Elemzés

- Ha implementáljuk az első, valamint a második algoritmust, és megmérjük a végrehajtási időt, látni fogjuk, hogy **az utóbbi sokkal kevesebb időt igényel, mint az első!**
- De még nem alkalmaztuk a bináris keresés elvét!
- Vegyük észre, hogy **nem érdemes minden  $k$  ( $k * k < \text{szám}$ ) értékre elvégezni a vizsgálatot**, hiszen tulajdonképpen egy **rendezett halmazban keresünk!**
- **$k$**  lehetséges értékei az  **$\{1, 2, \dots, 1000\}$**  halmazhoz tartozhatnak, mivel az ellenőrizendő számok kisebbek vagy egyenlők **1 000 000**-val.

**Algoritmus Számol\_3(n, a, p):**

$p \leftarrow 0$

**Minden**  $i = 1, n$  **végezd el:**

szám  $\leftarrow a_i$

eleje  $\leftarrow 0$

vége  $\leftarrow 1000$  *{ a legnagyobb érték 1 000 000 }*

**Amíg** eleje < vége **végezd el:**

$k \leftarrow (\text{eleje} + \text{vége})/2$

**Ha**  $k*k \geq \text{szám}$  **akkor** vége  $\leftarrow k$

**különben** eleje  $\leftarrow k + 1$

**vége(ha)**

**vége(amíg)**

**Ha** eleje \* eleje = szám **akkor**

$p \leftarrow p + 1$

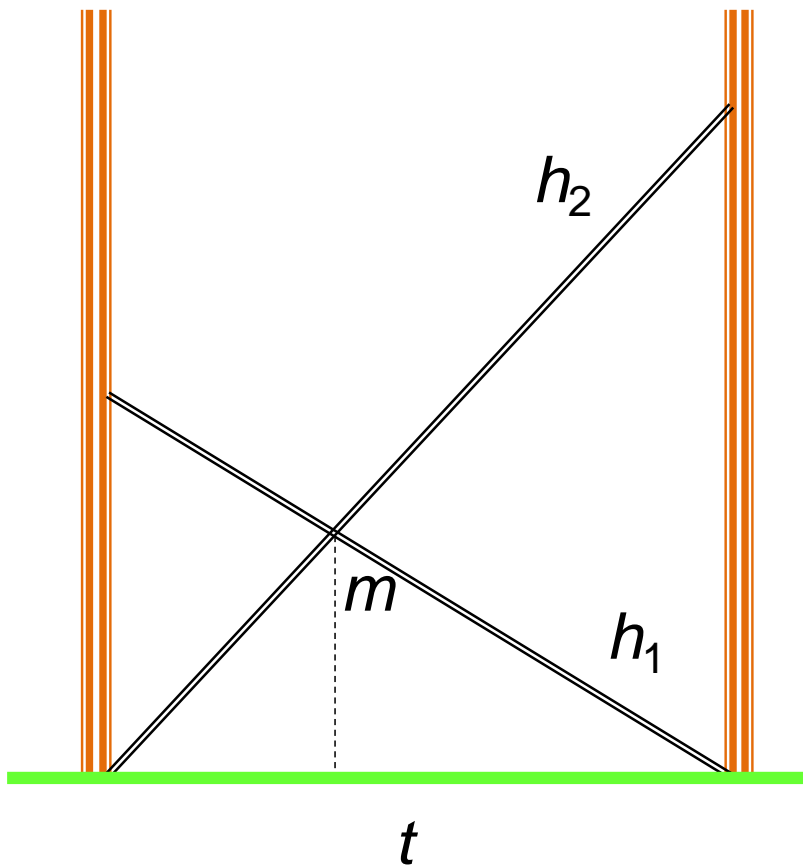
**vége(ha)**

**vége(minden)**

**Vége(algoritmus)**

## 4. Síkmértan

- Két függőleges fal egymástól  $t$  távolságra található.
- Egy  $h_1$  hosszúságú deszkát az egyik fal alapjától a másik falnak támasztunk.
- Egy  $h_2$  hosszúságú deszkát a másik fal alapjától az első falnak támasztunk.
- A két deszka  $m$  magasságban érinti egymást egy pontban, amely valahol a két fal között található.
- Számítsuk ki  $t$ -t  $h_1$ ,  $h_2$  és  $m$  ismeretében (megengedett hibalehetőség  $10^{-5}$ ).



## Észrevétel

*A magasság, ahol a két deszka találkozik nő, ha a keresett **t** távolság csökken és fordítva.*

# Megoldás

- Átfogalmazzuk a követelményt: **keressük meg azt a legnagyobb  $t$  értéket, amelyre a magasság, ahol a két deszka találkozik ne legyen kisebb, mint  $m$ .**
- Felhasználjuk a **bináris keresést** a  $t$  értékének megtalálása érdekében.
- A  $t$  lehetséges legkisebb értéke **0** (a két fal egymás mellett van), legnagyobb értéke pedig a rövidebb deszka hossza.
- Így a  $t$  értékét **0** és  **$\text{Min}(h_1, h_2)$**  között keressük.
- Miután van egy javaslat  $t$ -re,  $h_1$  és  $h_2$  értékeivel kiszámítjuk az érintkezési pont magasságát síkmértani ismeretekkel.
- Ha a kiszámított magasság nagyobb mint az adott  $m$  akkor növeljük  $t$ -t, különben csökkentjük.



**Algoritmus** deszkák( $m, h_1, h_2$ ):

megvan  $\leftarrow$  *hamis*

**Ha**  $h_1 > h_2$  **akkor**  $t \leftarrow h_2$

**különben**  $t \leftarrow h_1$

**vége**(ha)

$\min \leftarrow 0$ ;  $\max \leftarrow t$

**Amíg nem** megvan **végezd el**:

$t \leftarrow (\min + \max) / 2$

számol( $h_1, h_2, t, sz$ )

**Ha**  $\text{abs}(sz - m) \leq 0.00001$  **akkor**

megvan  $\leftarrow$  *igaz*

**különben**

**Ha**  $sz > m$  **akkor**  $\min \leftarrow t$

**különben**  $\max \leftarrow t$

**vége**(ha)

**vége**(ha)

**vége**(amíg)

**térítsd**  $t$

**Vége**(algoritmus)

## 5. Ládák

- *Költözik a múzeum. A tárgyakat kocka alakú, különböző méretű ládába csomagolták. Kicsomagoláskor több személy dolgozik egyidőben, és a rendetlenség elkerülése végett, azokba a helyiségekbe, ahol kicsomagolás folyik, felszereltek egy futószalagot, amelyre az üres ládákat helyezik, a nyitott fedelükkel fölfele.*
- *A futószalag végéhez egy robotot állítottak, amelynek az a feladata, hogy leszedje a ládákat a futószalagról és úgy helyezze egyiket a másikba (ha lehet) hogy végül a ládacsomagok száma a lehető legkisebb legyen. A robotot egy program irányítja úgy, hogy:*

# Ládák

- *A ládákat az érkezésük sorrendjében szedi le a futószalagról.*
- *Az aktuális ládát csak egy nála nagyobb méretű ládába helyezi.*
- *Ha nincs olyan megkezdett csomag, amelybe elhelyezhető az aktuális láda, akkor ez a láda egy új csomag első ládája lesz.*
- *Egy megkezdett csomagba csak egyetlen ládát helyez, vagyis nem helyez két ládát egymás mellé, még akkor sem, ha ez egyébként lehetséges volna.*
- *Egy elhelyezett ládát, többé nem mozgat.*
- *Egy megkezdett csomagot nem helyez egy másik csomagba még akkor sem, ha ez egyébként lehetséges volna.*
- *Egyetlen ládát sem hagy figyelmen kívül.*

# Ládák

- Határozzuk meg a ládák számának ( $0 \leq n \leq 15\,000$ ) és méreteiknek ( $1 \leq \text{ládaMéret} \leq 10\,000$ ) ismeretében a csomagok lehetséges legkisebb számát, valamint, minden csomag esetében az illető csomagban található ládákat.

**Példa:**  $n = 10$ ,  $\text{méretek} = (4, 1, 5, 10, 7, 9, 2, 8, 3, 2)$

**Eredmény:**

**Ládacsomagok száma: 4,**

**Csomagok:**

**1. csomag = (4, 1)**

**2. csomag = (5, 2)**

**3. csomag = (10, 7, 3, 2)**

**4. csomag = (9, 8)**

# Megoldás

- A feladat tulajdonképpen azt kéri, hogy **az adott sorozatot bontsuk fel minimális számú növekvő részsorozatra**.
- A feladat megoldható egy **mohó algoritmussal**, amely mindig a legkisebb olyan ládába csomagol, amelybe lehetséges.
- Észrevétel: a ládacsomagokba utoljára elhelyezett ládák mérete **növekvő sorozatot** alkot, tehát a megfelelő csomag megkeresése lehetséges **bináris kereséssel**.
- Ugyanakkor: nem egy ismert értéket kell megkeresnünk, hanem egy olyat, amely legkisebb az adott számnál nagyobbak között.

# Megoldás

- Gond: az adatok tárolása, hiszen, ha a ládák csökkenő (vagy növekvő) sorrendben érkeznek, a következő két úgynevezett *„legrosszabb esettel”* állunk szemben:
  - Ha a ládák *csökkenő* sorrendben érkeznek, *egyetlen csomag*ba befér minden láda, tehát *egyetlen növekvő részsorozatunk lesz*, aminek a hossza legtovább **15 000**.
  - Ha a ládák *növekvő* sorrendben érkeznek, akkor minden érkező láda új csomagnak felel meg, tehát legtovább **15 000 darab egy elemű részsorozatunk** lesz.

# Megoldás

- A fentieket figyelembe véve egy **15 000 × 15 000** méretű tömböt kellene létrehozzunk, ami (ha lehetséges a választott programozási környezetben) nagyon nagy tárpazarlást jelent, hiszen még tárolnunk kell a csomagok hosszát is egy legtöbb **15000** elemű tömbben.
- A megoldást a **dinamikus tárkezelés** hozza: minden csomag egy verem típusú lista lesz, amelynek a feje az utoljára elhelyezett láda méretét tartalmazza.

A **bináris keresést a veremfejek sorozatán végezzük:**

- ha nem találunk olyan ládát, amelybe az aktuális láda elhelyezhető, akkor új csomagot indítunk,
- különben elhelyezzük az aktuális ládát a megfelelő csomag tetejére.

**Algoritmus** Keres(bal, jobb, új):

**Ha** bal > jobb **akkor** { sikertelen keresés }

jobb  $\leftarrow$  jobb + 1

csomagokszáma  $\leftarrow$  jobb

Helyez(csomagok, csomagokszáma, új)

{ új csomagot kezdünk, a jobb sorszámú után }

**különb**

**Ha** csomagok[bal]^méret > új **akkor** { megvan }

Helyez(csomagok, bal, új)

{ az új méretű ládát a bal sorszámú csomagba tesszük }

**különb**

közép  $\leftarrow$  (bal + jobb)/2 { tovább keresünk }

**Ha** új < csomagok[közép]^méret **akkor**

Keres(bal, közép, új)

**különb**

Keres(közép + 1, jobb, új)

**vége(ha)**

**vége(ha)**

**vége(ha)**

**Vége(algoritmus)**



**Algoritmus** Helyez(csomagok, csomagokszáma, új):

*{ elhelyezzük az új méretű ládát a csomagokszáma  
sorszámú csomagba }*

*helyet kérünk a p mutató által mutatható elem számára*

$p^{\wedge}.méret \leftarrow új$

$p^{\wedge}.köv \leftarrow csomagok[csomagokszáma]$

$csomagok[csomagokszáma] \leftarrow p$

**Vége(algoritmus)**

# Következtetések

- Ha egy maximális értéket kell meghatároznunk, amelynek olyan követelményeknek kell eleget tennie, amelyek ha teljesülnek egy bizonyos értékre, akkor biztosan teljesülnek az ennél kisebbekre, akkor a bináris kereséssel és a feltételek utólagos ellenőrzésével  $\log n$  lépésben eredményt kapunk.
- A szabály alkalmazható minimum esetében is.
- Az elv alkalmazása az algoritmus végrehajtási idejének csökkentését eredményezi, de ehhez előbb be kell látnunk, hogy ez lehetséges, majd meg kell találnunk az alkalmazás módját.