

# 1. előadás

## Az algoritmusok szerepe a számításokban

Algoritmusok mint technológia

*Adatszerkezetek és algoritmusok előadás*

2018. február 6.

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

### Algoritmusok mint technológia

- Hatókonyiság
- Technológia

### Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonyiság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonyisége
- Oszd meg és uralkodj

# Általános tudnivalók

Ajánlott irodalom:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- Gyakorlati aláírás
  - 2 ZH
- Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj



- Informálisan algoritmusnak nevezünk bármilyen jól definiált számítási eljárást (**elemi lépések sorozatát**), amely
  - bemenetként bizonyos értéket vagy értékeket kap és
  - kimenetként bizonyos értéket vagy értékeket állít elő.
- Eszerint az algoritmus olyan számítási lépések sorozata, amelyek a bemenetet átalakítják kimenetté.
- Az algoritmusokat tekinthetjük olyan eszköznek is, amelynek segítségével pontosan meghatározott számítási feladatokat oldunk meg.
- Ezeknek a feladatoknak a megfogalmazása általában a bemenet és a kimenet közötti kívánt kapcsolat leírása.

## Algoritmusok

Algoritmusok fogalma

Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

Algoritmusok mint  
technológia

Hatékonyság

Technológia

Algoritmusok  
elemzése

Algoritmusok helyessége

Ciklusinváriáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Algoritmusok

## Kicsit általánosabban

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Olyan eljárás (**elemi lépések sorozata**), melynek során a következők teljesülnek:

- jól meghatározott objektumokon jól meghatározott műveleteket végzünk
- minden lépés elvégzése után egyértelműen definiált helyzet áll elő
- **véges sok lépés után véget ér**
- nem csak egy feladatra, hanem egy feladatosztály tagjaira érvényes

### Algoritmusok

#### Algoritmusok fogalma

Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

#### Algoritmusok mint technológia

Hatékonyság

Technológia

#### Algoritmusok elemzése

Algoritmusok helyessége

Ciklusinvariáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Algoritmusok megadásának módjai

Algoritmusokat a következő módon lehet megadni:

- természetes (beszélt) emberi nyelven
- pontokba szedett természetes nyelvi „utasításokkal”
- folyamatábrával
- pszeudonyelvvel (lásd gyakorlaton)
- valamelyen programozási nyelven

Az egyetlen követelmény az, hogy a leírás pontosan, azaz félreérthetetlen módon adja meg a követendő (számítási) eljárást.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma

## Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság

Technológia

## Algoritmusok elemzése

Algoritmusok helyessége

Ciklusinváriáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Algoritmusok megadásának módjai

Algoritmusokat a következő módon lehet megadni:

- természetes (beszélt) emberi nyelven
- pontokba szedett természetes nyelvi „utasításokkal”
- folyamatábrával
- pszeudonyelvvel (lásd gyakorlaton)
- valamelyen programozási nyelven

Az egyetlen követelmény az, hogy a leírás pontosan, azaz félreérthetetlen módon adja meg a követendő (számítási) eljárást.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma

Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság

Technológia

## Algoritmusok elemzése

Algoritmusok helyessége

Ciklusinvariáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Példa elefánttal és zsiráffal

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Hogyan tegyünk be egy elefántot a hűtőszekrénybe?

- 1 Nyissuk ki a hűtőszekrény ajtaját.
- 2 Tegyük be az elefántot a hűtőszekrénybe.
- 3 Zárjuk be a hűtőszekrény ajtaját.

## Hogyan tegyünk be egy zsiráfot a hűtőszekrénybe?

- 1 Nyissuk ki a hűtőszekrény ajtaját.
- 2 Vegyük ki az elefántot a hűtőszekrényből.
- 3 Tegyük be a zsiráfot a hűtőszekrénybe.
- 4 Zárjuk be a hűtőszekrény ajtaját.

[Algoritmusok](#)

[Algoritmusok fogalma](#)

[Algoritmusok megadása](#)

[Algoritmusok építőelemei](#)

[Példa: rendezési feladat](#)

[Algoritmusok mint  
technológia](#)

[Hatékonyság](#)

[Technológia](#)

[Algoritmusok  
elemzése](#)

[Algoritmusok helyessége](#)

[Ciklusinvariáns](#)

[Hatékonyság elemzés](#)

[A RAM modell](#)

[A bemenet mérete](#)

[A futási idő](#)

[A beszúró rendezés  
hatékonysága](#)

[Oszd meg és uralkodj](#)

# Példa elefánttal és zsiráffal

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Hogyan tegyünk be egy elefántot a hűtőszekrénybe?

- ① Nyissuk ki a hűtőszekrény ajtaját.
- ② Tegyük be az elefántot a hűtőszekrénybe.
- ③ Zárjuk be a hűtőszekrény ajtaját.

## Hogyan tegyünk be egy zsiráfot a hűtőszekrénybe?

- ① Nyissuk ki a hűtőszekrény ajtaját.
- ② Vegyük ki az elefántot a hűtőszekrényből.
- ③ Tegyük be a zsiráfot a hűtőszekrénybe.
- ④ Zárjuk be a hűtőszekrény ajtaját.

[Algoritmusok](#)

[Algoritmusok fogalma](#)

[Algoritmusok megadása](#)

[Algoritmusok építőelemei](#)

[Példa: rendezési feladat](#)

[Algoritmusok mint  
technológia](#)

[Hatékonyság](#)

[Technológia](#)

[Algoritmusok  
elemzése](#)

[Algoritmusok helyessége](#)

[Ciklusinváriáns](#)

[Hatékonyság elemzés](#)

[A RAM modell](#)

[A bemenet mérete](#)

[A futási idő](#)

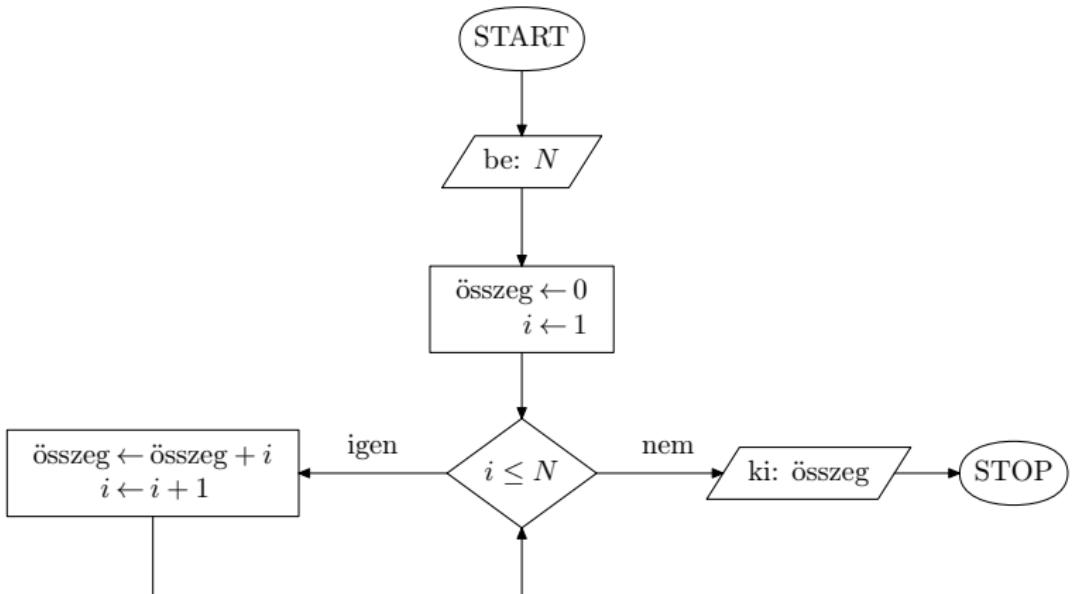
[A beszúró rendezés  
hatékonysága](#)

[Oszd meg és uralkodj](#)

# Példa folyamatábrával

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma

### Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság

Technológia

### Algoritmusok elemzése

Algoritmusok helyessége

Ciklusinvariáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj



## Vezérlési szerkezetek

- 1 szekvencia  
(utasítások végrehajtása a felírás sorrendjében)
- 2 szelekció  
(elágazások)
- 3 iteráció  
(ciklusok)
- 4 alprogramok hívása  
(„kész” algoritmusok bevonása a részfeladatok megoldásába)

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása

### Algoritmusok építőelemei

Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

## Példa: rendezési feladat

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**Bemenet:** n számot tartalmazó  $\langle a_1, a_2, \dots, a_n \rangle$  sorozat.

**Kimenet:** a bemenet sorozat olyan  $\langle a'_1, a'_2, \dots, a'_n \rangle$  permutációja (újrarendezése), hogy  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

- Ha például a bemenet **31, 41, 59, 26, 41, 58**, akkor egy rendező algoritmus
- kimenetként a **26, 31, 41, 41, 58, 59** sorozatot adja meg.
- Egy ilyen bemenet a rendezési feladat **egy esete**.
- Általában egy feladat (probléma) **egy esete** az a bemenet, (a feladat megfogalmazásában szereplő feltételeknek eleget tevő bemenet,) amely a feladat megoldásának kiszámításához szükséges.

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei

#### Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűró rendezés  
hatékonysága  
Oszd meg és uralkodj

## Példa: rendezési feladat

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**Bemenet:** n számot tartalmazó  $\langle a_1, a_2, \dots, a_n \rangle$  sorozat.

**Kimenet:** a bemenet sorozat olyan  $\langle a'_1, a'_2, \dots, a'_n \rangle$  permutációja (újrarendezése), hogy  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

- Ha például a bemenet **31, 41, 59, 26, 41, 58**, akkor egy rendező algoritmus
- kimenetként a **26, 31, 41, 41, 58, 59** sorozatot adja meg.
- Egy ilyen bemenet a rendezési feladat **egy esete**.
- Általában egy feladat (probléma) **egy esete** az a bemenet, (a feladat megfogalmazásában szereplő feltételeknek eleget tevő bemenet,) amely a feladat megoldásának kiszámításához szükséges.

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei

#### Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

## Példa: rendezési feladat

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**Bemenet:** n számot tartalmazó  $\langle a_1, a_2, \dots, a_n \rangle$  sorozat.

**Kimenet:** a bemenet sorozat olyan  $\langle a'_1, a'_2, \dots, a'_n \rangle$  permutációja (újrarendezése), hogy  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

- Ha például a bemenet **31, 41, 59, 26, 41, 58**, akkor egy rendező algoritmus
- kimenetként a **26, 31, 41, 41, 58, 59** sorozatot adja meg.
- Egy ilyen bemenet a rendezési feladat **egy esete**.
- Általában egy feladat (probléma) **egy esete** az a bemenet, (a feladat megfogalmazásában szereplő feltételeknek eleget tevő bemenet,) amely a feladat megoldásának kiszámításához szükséges.

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei

#### Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűrő rendezés  
hatékonysága  
Oszd meg és uralkodj

## Példa: rendezési feladat

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A rendezés az informatikában alapvető művelet  
(számos program alkalmazza közbeeső lépésként),  
ezért nagyszámú jó rendező algoritmust dolgoztak ki.  
Az, hogy adott alkalmazás esetén melyik  
rendező algoritmus a „legjobb” – más tényezők mellett – függ

- a rendezendő elemek számától,
- rendezettségétől,
- a rájuk vonatkozó lehetséges korlátoktól és
- a felhasználandó tároló fajtájától
  - központi memória,
  - lemezek,
  - szalagok

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei

#### Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok mint technológia

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Tegyük fel, hogy

- a számítógépek sebessége végtelen
- és a memória ingyenes.

Van értelme ekkor az algoritmusokat tanulmányozni?

A válasz **igen**:

ha másért nem is, például azért, hogy megmutassuk, hogy

- algoritmusunk futása befejeződik,
- és helyes eredményt ad.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok mint technológia

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Tegyük fel, hogy

- a számítógépek sebessége végtelen
- és a memória ingyenes.

Van értelme ekkor az algoritmusokat tanulmányozni?

A válasz **igen**:

ha másért nem is, például azért, hogy megmutassuk, hogy

- algoritmusunk futása befejeződik,
- és helyes eredményt ad.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok mint technológia

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Ha a számítógépek végtelen gyorsak lennének, akkor bármely helyes megoldási módszer megfelelő lenne. Valószínűleg azt kívánnánk, hogy a megvalósítás megfeleljen a jó szoftvermérnöki gyakorlatnak (azaz jól tervezett és dokumentált legyen), és azt a módszert alkalmaznánk, amelyet a legkönnyebb megvalósítani.

Természetesen a számítógépek lehetnek gyorsak, de nem végtelenül gyorsak.

A memória pedig lehet olcsó, de nem ingyenes.

A számítási idő ezért korlátos erőforrás, és ugyanez vonatkozik a tárolási kapacitásra is.

Ezeket az erőforrásokat okosan kell felhasználni, és ebben segítenek a futási időt és/vagy memóriafelhasználást tekintve hatékony algoritmusok.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatékonyság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonysága
- Oszd meg és uralkodj

# Algoritmusok mint technológia

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Ha a számítógépek végtelen gyorsak lennének, akkor bármely helyes megoldási módszer megfelelő lenne. Valószínűleg azt kívánunk, hogy a megvalósítás megfeleljen a jó szoftvermérnöki gyakorlatnak (azaz jól tervezett és dokumentált legyen), és azt a módszert alkalmaznánk, amelyet a legkönnyebb megvalósítani.

Természetesen a számítógépek lehetnek gyorsak, de nem végtelenül gyorsak.

A memória pedig lehet olcsó, de nem ingyenes.

A számítási idő ezért korlátos erőforrás, és ugyanez vonatkozik a tárolási kapacitásra is.

Ezeket az erőforrásokat okosan kell felhasználni, és ebben segítenek a futási időt és/vagy memóriafelhasználást tekintve hatékony algoritmusok.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyág
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatékonyág elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonyiséga
- Oszd meg és uralkodj



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiá

### Hatékonyúság

Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyúság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonyisége  
Oszd meg és uralkodj

# Hatókonyság

Az ugyanannak a feladatnak a megoldására tervezett algoritmusok gyakran drámai módon különböző hatékonysságot mutatnak.

Ezek a különbségek jóval nagyobbak lehetnek, mint ami a hardver és a szoftver különbözőségből adódhat.

Ennek demonstrálására két rendező algoritmus hatékonysságát fogjuk összevetni.

Látni fogjuk, hogy egy véletlenszerű sorrendet mutató **n** elemű bemenet esetén

- a beszúró rendezésnek  $c_1 n^2$ -el arányos időre, míg
- az összefésülő rendezésnek  $c_2 n \log_2 n$ -el arányos időre

van szüksége a helyes kimenet meghatározásához.

(Tipikusan  $c_1 < c_2$ .)

## Hatókonysság: numerikus példa

Az algoritmusok  
szerepe a  
számításokban

Tegyük fel, hogy van két eltérő hardverünk:

- A-gép: másodpercenként  $10^9$  (egymilliárd) műveletet képes elvégezni
- B-gép: másodpercenként  $10^7$  (tízmillió) műveletet képes elvégezni

Ezen túlmenően tegyük fel, hogy

- a beszúró rendezést a világ legjobb programozója kódolja az A gép gépi kódjában, és ennek eredményeként  $c_1 = 2$ ,
- ezzel szemben az összefésülő rendezést rábízzuk egy tapasztalatlan kezdő programozóra, és végül  $c_2$  értéke 50 lesz.

Végezetül futtatjuk az elkészült –egyébként helyes működésű– algoritmusokat a fenti két gépen, az alábbi módon:

- Beszúró rendezést az A gépen
- Összefésülő rendezést a B gépen

Melyik gép fogja rövidebb idő alatt elvégezni a rendezést?

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológiák

#### Hatókonysság

Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatókonysság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés hatékonyssága  
Oszd meg és uralkodj

## Hatókonysság: numerikus példa

Az algoritmusok  
szerepe a  
számításokban

Tegyük fel, hogy van két eltérő hardverünk:

- A-gép: másodpercenként  $10^9$  (egymilliárd) műveletet képes elvégezni
- B-gép: másodpercenként  $10^7$  (tízmillió) műveletet képes elvégezni

Ezen túlmenően tegyük fel, hogy

- a beszúró rendezést a világ legjobb programozója kódolja az A gép gépi kódjában, és ennek eredményeként  $c_1 = 2$ ,
- ezzel szemben az összefésülő rendezést rábízzuk egy tapasztalatlan kezdő programozóra, és végül  $c_2$  értéke 50 lesz.

Végezetül futtatjuk az elkészült –egyébként helyes működésű– algoritmusokat a fenti két gépen, az alábbi módon:

- Beszúró rendezést az A gépen
- Összefésülő rendezést a B gépen

Melyik gép fogja rövidebb idő alatt elvégezni a rendezést?

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológiák

#### Hatókonysság

Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatókonysság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés hatékonyssága  
Oszd meg és uralkodj



## Hatókonysság: numerikus példa

A válasz függ a bemenettől, ezen belül leginkább annak méretétől, vagyis  $n$  értékétől.

Legyen  $n = 1000$  (ezer). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^3)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2ms (= 1s/500)$
- Összefésülő rend.:  $\frac{50 \cdot 10^3 \cdot \log_2 10^3 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 50ms (= 1s/20)$

Legyen  $n = 1000000$  (egymillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^6)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2000s$
- Összefésülő rendezés:  $\frac{50 \cdot 10^6 \cdot \log_2 10^6 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 100s$

Legyen  $n = 10000000$  (tízmillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^7)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 200000s \approx 2.3\text{nap}$
- Összefésülő rend.:  $\frac{50 \cdot 10^7 \cdot \log_2 10^7 \text{ művelet}}{10^7 \text{ művelet/s}} \approx 1160s \lesssim 20\text{perc}$

### Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

### Algoritmusok mint technológia

#### Hatókonysság

- Technológia
- Algoritmusok elemzése
- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonysság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatókonyssága
- Oszd meg és uralkodj



## Hatókonysság: numerikus példa

A válasz függ a bemenettől, ezen belül leginkább annak méretétől, vagyis  $n$  értékétől.

Legyen  $n = 1000$  (ezer). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^3)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2ms (= 1s/500)$
- Összefésülő rend.:  $\frac{50 \cdot 10^3 \cdot \log_2 10^3 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 50ms (= 1s/20)$

Legyen  $n = 1000000$  (egymillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^6)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2000s$
- Összefésülő rendezés:  $\frac{50 \cdot 10^6 \cdot \log_2 10^6 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 100s$

Legyen  $n = 10000000$  (tízmillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^7)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 200000s \approx 2.3\text{nap}$
- Összefésülő rend.:  $\frac{50 \cdot 10^7 \cdot \log_2 10^7 \text{ művelet}}{10^7 \text{ művelet/s}} \approx 1160s \lesssim 20\text{perc}$

### Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

### Algoritmusok mint technológia

#### Hatókonysság

- Technológia
- Algoritmusok elemzése
- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonysság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatókonyssága
- Oszd meg és uralkodj



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

### Hatékonysság

Technológia  
Algoritmusok elemzése  
Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonysság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűrő rendezés hatékonyssága  
Oszd meg és uralkodj

## Hatókonysság: numerikus példa

A válasz függ a bemenettől, ezen belül leginkább annak méretétől, vagyis  $n$  értékétől.

Legyen  $n = 1000$  (ezer). Ekkor a futási idők:

- Beszűró rendezés:  $\frac{2 \cdot (10^3)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2ms (= 1s/500)$
- Összefésülő rend.:  $\frac{50 \cdot 10^3 \cdot \log_2 10^3 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 50ms (= 1s/20)$

Legyen  $n = 1000000$  (egymillió). Ekkor a futási idők:

- Beszűró rendezés:  $\frac{2 \cdot (10^6)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2000s$
- Összefésülő rendezés:  $\frac{50 \cdot 10^6 \cdot \log_2 10^6 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 100s$

Legyen  $n = 10000000$  (tízmillió). Ekkor a futási idők:

- Beszűró rendezés:  $\frac{2 \cdot (10^7)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 200000s \approx 2.3\text{nap}$
- Összefésülő rend.:  $\frac{50 \cdot 10^7 \cdot \log_2 10^7 \text{ művelet}}{10^7 \text{ művelet/s}} \approx 1160s \lesssim 20\text{perc}$



## Hatókonysság: numerikus példa

A válasz függ a bemenettől, ezen belül leginkább annak méretétől, vagyis  $n$  értékétől.

Legyen  $n = 1000$  (ezer). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^3)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2ms (= 1s/500)$
- Összefésülő rend.:  $\frac{50 \cdot 10^3 \cdot \log_2 10^3 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 50ms (= 1s/20)$

Legyen  $n = 1000000$  (egymillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^6)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 2000s$
- Összefésülő rendezés:  $\frac{50 \cdot 10^6 \cdot \log_2 10^6 \text{ művelet}}{10^7 \text{ művelet/s}} \lesssim 100s$

Legyen  $n = 10000000$  (tízmillió). Ekkor a futási idők:

- Beszúró rendezés:  $\frac{2 \cdot (10^7)^2 \text{ művelet}}{10^9 \text{ művelet/s}} = 200000s \approx 2.3\text{nap}$
- Összefésülő rend.:  $\frac{50 \cdot 10^7 \cdot \log_2 10^7 \text{ művelet}}{10^7 \text{ művelet/s}} \approx 1160s \lesssim 20\text{perc}$

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

#### Hatókonysság

Technológia  
Algoritmusok elemzése  
Algoritmusok helyessége  
Ciklusinvariáns  
Hatókonysság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés hatókonyssága  
Oszd meg és uralkodj

# Algoritmusok mint technológia

- Az előző példák azt mutatják, hogy az algoritmusok – a számítógépek hardveréhez hasonlóan – **technológiák**.
- Egy rendszer teljesítménye éppúgy függ a hatékony algoritmusok kiválasztásától, mint a gyors hardver alkalmazásától.
- Amilyen gyors a haladás a többi számítógépes technológiában, éppoly gyors a fejlődés az algoritmusokkal kapcsolatban is.
- Az állandóan növekvő kapacitású számítógépeket nagyobb méretű feladatok megoldására használjuk, mint korábban bármikor.
  - Amint azt láttuk a beszúró rendezés és az összefésülő rendezés összehasonlításánál, nagyobb méretű feladatok esetén az algoritmusok hatékonyságának különbsége különösen fontossá válik.
- Az algoritmusok biztos ismerete olyan jellemző, amely elválasztja a jól képzett programozókat a kezdőktől.
  - A modern számítógépes technológia segítségével megoldhatunk bizonyos feladatokat anélkül, hogy sokat tudnánk az algoritmusokról, de az algoritmusokkal kapcsolatos jó alapok megléte esetén sokkal többet tudunk elérni.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság

## Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok elemzése

Az algoritmusok  
szerepe a  
számításokban

A rendezési feladat:

- **Bemenet:** n számot tartalmazó  $\langle a_1, a_2, \dots, a_n \rangle$  sorozat.
- **Kimenet:** a bemenet sorozat olyan  $\langle a'_1, a'_2, \dots, a'_n \rangle$  permutációja (újrarendezése), hogy  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

A rendezendő számokat **kulcsoknak** is szokás nevezni.

A félév folyamán rendszerint egy olyan pszeudokódban írt programként írjuk le az algoritmusokat, amely nagyon hasonlít a C-hez, a Pascalhoz vagy a Javához.

Ha ezek közül bármelyik nyelvet ismerik, nem lesz gondjuk az algoritmusok olvasása során.

Ami az „igazi” kódot megkülönbözteti a pszeudokódtól, az az, hogy pszeudokódban bármilyen kifejező eszközöt alkalmazhatunk, amellyel világosan és tömörén megadhatunk egy algoritmust.

- (Néha a legjobb eszköz a magyar nyelv.)

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok elemzése

Az algoritmusok  
szerepe a  
számításokban

A rendezési feladat:

- **Bemenet:** n számot tartalmazó  $\langle a_1, a_2, \dots, a_n \rangle$  sorozat.
- **Kimenet:** a bemenet sorozat olyan  $\langle a'_1, a'_2, \dots, a'_n \rangle$  permutációja (újrarendezése), hogy  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

A rendezendő számokat **kulcsoknak** is szokás nevezni.

A félév folyamán rendszerint egy olyan pszeudokódban írt programként írjuk le az algoritmusokat, amely nagyon hasonlít a C-hez, a Pascalhoz vagy a Javához.

Ha ezek közül bármelyik nyelvet ismerik, nem lesz gondjuk az algoritmusok olvasása során.

Ami az „igazi” kódot megkülönbözteti a pszeudokódtól, az az, hogy pszeudokódban bármilyen kifejező eszközöt alkalmazhatunk, amellyel világosan és tömören megadhatunk egy algoritmust.

- (Néha a legjobb eszköz a magyar nyelv.)

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Beszúró rendezés

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(a)

1	2	3	4	5	6
5	2	4	6	1	3



(b)

1	2	3	4	5	6
2	5	4	6	1	3



(c)

1	2	3	4	5	6
2	4	5	6	1	3



(d)

1	2	3	4	5	6
2	4	5	6	1	3



(e)

1	2	3	4	5	6
1	2	4	5	6	3



(f)

1	2	3	4	5	6
1	2	3	4	5	6



## Algoritmusok

Algoritmusok fogalma

Algoritmusok megadása

Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság

Technológia

## Algoritmusok elemzése

Algoritmusok helyessége

Ciklusinváriáns

Hatékonyság elemzés

A RAM modell

A bemenet mérete

A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Beszúró rendezés

A pseudo kód:

**procedure** BESZÚRÓ-RENDEZÉS(*A*)

- 1 **for** *j*  $\leftarrow$  2 **to** méret(*A*) **do**
- 2     kulcs  $\leftarrow$  *A[j]*  
     -- *A[j]* beszúrása az *A[1 . . j - 1]* rendezett sorozatba.
- 3     *i*  $\leftarrow$  *j* - 1
- 4     **while** *i*  $\geq$  1 és *A[i]* > kulcs **do**
- 5         *A[i + 1]*  $\leftarrow$  *A[i]*
- 6         *i*  $\leftarrow$  *i* - 1
- 7     **end while**
- 8     *A[i + 1]*  $\leftarrow$  kulcs
- 9 **end for**

**end procedure**

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj



# Algoritmusok helyessége

Egy algoritmust helyesnek mondunk,

- ha minden bemenetre megáll és helyes eredményt ad.

Azt mondjuk, hogy a helyes algoritmus megoldja az adott számítási feladatot.

Egy nem helyes algoritmus esetén előfordulhat, hogy

- nem minden bemenetre áll meg, vagy
- bizonyos bemenetekre nem a kívánt választ adja.

Várakozásunkkal ellentétben egy nem-helyes algoritmus is lehet hasznos, ha hibázási aránya elfogadható.

Ilyen algoritmusra példát láthatunk a **tankönyv<sup>1</sup>** 31. fejezetben, ahol a nagy prímszámokat előállító algoritmusokat tanulmányozzák.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei

Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonyság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonysága
- Oszd meg és uralkodj

<sup>1</sup>Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, Budapest, 2003.



# Algoritmusok helyessége

Egy algoritmust helyesnek mondunk,

- ha minden bemenetre megáll és helyes eredményt ad.

Azt mondjuk, hogy a helyes algoritmus megoldja az adott számítási feladatot.

Egy nem helyes algoritmus esetén előfordulhat, hogy

- nem minden bemenetre áll meg, vagy
- bizonyos bemenetekre nem a kívánt választ adja.

Várakozásunkkal ellentétben egy nem-helyes algoritmus is lehet hasznos, ha hibázási aránya elfogadható.

Ilyen algoritmusra példát láthatunk a **tankönyv<sup>1</sup>** 31. fejezetben, ahol a nagy prímszámokat előállító algoritmusokat tanulmányozzák.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonyság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonysága
- Oszd meg és uralkodj

<sup>1</sup>Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, Budapest, 2003.



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Algoritmusok helyessége

Egy algoritmust helyesnek mondunk,

- ha minden bemenetre megáll és helyes eredményt ad.

Azt mondjuk, hogy a helyes algoritmus megoldja az adott számítási feladatot.

Egy nem helyes algoritmus esetén előfordulhat, hogy

- nem minden bemenetre áll meg, vagy
- bizonyos bemenetekre nem a kívánt választ adja.

Várakozásunkkal ellentétben egy nem-helyes algoritmus is lehet hasznos, ha hibázási aránya elfogadható.

Ilyen algoritmusra példát láthatunk a **tankönyv<sup>1</sup>** 31. fejezetben, ahol a nagy prímszámokat előállító algoritmusokat tanulmányozzák.

<sup>1</sup>Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
*Új algoritmusok*, Scolar Informatika, Budapest, 2003.



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyág  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
**Ciklusinvariáns**  
Hatékonyág elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűró rendezés  
hatékonyiséga  
Oszd meg és uralkodj

# Ciklusinvariáns

## A helyesség bizonyításának egyik eszköze

A **ciklusinvariáns** egy állítás, amelyről három dolgot kell megmutatni:

- Teljesül:** Igaz közvetlenül a ciklus első iterációjának megkezdése előtt.
- Megmarad:** Ha igaz a ciklus egy iterációjának megkezdése előtt, akkor igaz marad a következő iteráció előtt is.
- Befejeződik:** Amikor a ciklus befejeződik, az invariáns olyan hasznos tulajdonságot ír le, amely segít abban, hogy az algoritmus helyességét bebizonyítsuk.

Ha az első két feltétel teljesül, akkor a ciklusinvariáns a ciklus minden iterációja előtt teljesül.

Vegyük észre a teljes indukcióhoz való hasonlóságot.

- Az invariánsnak az első iteráció előtt való teljesülése a kezdőértékhez hasonlít,
- az invariánsnak lépésről lépésre való teljesülése pedig az induktív lépéshoz.

A harmadik a tulajdonság a teljes indukciótól való eltérésre utal, itt megállítjuk az „indukciót”, amikor a ciklus véget ér.



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
**Ciklusinvariáns**  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Ciklusinvariáns

## A beszúró rendezés estére

A ciklusinvariáns:

- Az 1–9. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[1 \dots j - 1]$  résztömb azon elemekből áll, amelyek eredetileg is az  $A[1 \dots j - 1]$  résztömbben voltak, de most már rendezett sorrendben.
- $n + 2 - j \geq 0$  és a ciklus minden végrehajtásakor csökken.

---

### procedure BESZÚRÓ-RENDEZÉS(A)

```
1 for j ← 2 to méret(A) do
  2   kulcs ← A[j]
  3   i ← j - 1
  4   while i ≥ 1 és A[i] > kulcs do
    5     A[i + 1] ← A[i]
    6     i ← i - 1
  7   end while
  8   A[i + 1] ← kulcs
  9 end for
end procedure
```



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
**Ciklusinvariáns**  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Ciklusinvariáns

## A beszúró rendezés estére

A ciklusinvariáns:

- Az 1–9. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[1 \dots j - 1]$  résztömb azon elemekből áll, amelyek eredetileg is az  $A[1 \dots j - 1]$  résztömbben voltak, de most már rendezett sorrendben.
- $n + 2 - j \geq 0$  és a ciklus minden végrehajtásakor csökken.

---

A három feltétel vizsgálata:

**Teljesül:** Azt látjuk be először, hogy a ciklusinvariáns teljesül az első ciklusiteráció előtt, amikor is  $j = 2$ .

Az  $A[1 \dots j - 1]$  résztömb így pontosan az  $A[1]$  elemből áll, amely valóban az eredeti  $A[1]$  elem.

Továbbá, ez a résztömb rendezett (természetesen), azaz a ciklusinvariáns a ciklus első iterációja előtt teljesül.



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiá

Hatékonyág  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
**Ciklusinvariáns**  
Hatékonyág elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonyiséga  
Oszd meg és uralkodj

# Ciklusinvariáns

## A beszúró rendezés estére

### A ciklusinvariáns:

- Az 1–9. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[1 \dots j - 1]$  résztömb azon elemekből áll, amelyek eredetileg is az  $A[1 \dots j - 1]$  résztömbben voltak, de most már rendezett sorrendben.
- $n + 2 - j \geq 0$  és a ciklus minden végrehajtásakor csökken.

### A három feltétel vizsgálata:

**Megmarad:** Ezután a második tulajdonsággal foglalkozunk: megmutatjuk, hogy minden iteráció fenntartja a ciklusinvariánst. Informálisan a külső **for** ciklus magja az  $A[j - 1], A[j - 2]$  stb. elemeket mozgatja jobbra mindaddig, amíg  $A[j]$  a helyes pozíciót el nem éri (4–7. sorok, azaz a **while** ciklus), amikor is az „eredeti”  $A[i]$  (jelenleg a `kulcs`-ban) elemet beszúrja (8. sor).

A második tulajdonság formálisabb tárgyalása azt igényelné, hogy fogalmazzunk meg és bizonyítsunk egy ciklusinvariánst a belső **while** ciklusra is. Ebbe most inkább nem bonyolódunk bele, hanem az informális elemzésre hagyatkozunk, amely szerint a második tulajdonság teljesül a külső ciklusra nézve.

# Ciklusinvariáns

## A beszúró rendezés estére

A ciklusinvariáns:

- Az 1–9. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[1 \dots j - 1]$  résztömb azon elemekből áll, amelyek eredetileg is az  $A[1 \dots j - 1]$  résztömbben voltak, de most már rendezett sorrendben.
- $n + 2 - j \geq 0$  és a ciklus minden végrehajtásakor csökken.

---

A három feltétel vizsgálata:

**Befejeződik:** Végezetül megvizsgáljuk, hogy mi történik a ciklus befejeződésekor. A beszúró rendezés esetén a külső **for** ciklus akkor ér véget, amikor  $j$  meghaladja  $n$ -et, azaz, amikor  $j = n + 1$ . Ha a ciklusinvariáns megfogalmazásában  $(n+1)$ -et írunk  $j$  helyett, akkor azt kapjuk, hogy az  $A[1 \dots n]$  résztömb azokból az elemekből áll, amelyek eredetileg az  $A[1 \dots n]$  elemek voltak, de már rendezett formában.

Az  $A[1 \dots n]$  résztömb azonban az egész tömb!

Tehát az egész tömb rendezve van, vagyis az algoritmus helyes.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiá

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns

Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
**Ciklusinvariáns**  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága  
Oszd meg és uralkodj

# Ciklusinvariáns

## A beszúró rendezés estére

A ciklusinvariáns:

- Az 1–9. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[1 \dots j - 1]$  résztömb azon elemekből áll, amelyek eredetileg is az  $A[1 \dots j - 1]$  résztömbben voltak, de most már rendezett sorrendben.
- $n + 2 - j \geq 0$  és a ciklus minden végrehajtásakor csökken.

---

### procedure BESZÚRÓ-RENDEZÉS(A)

```
1 for j ← 2 to méret(A) do
  2   kulcs ← A[j]
  3   i ← j - 1
  4   while i ≥ 1 és A[i] > kulcs do
    5     A[i + 1] ← A[i]
    6     i ← i - 1
  7   end while
  8   A[i + 1] ← kulcs
  9 end for
end procedure
```

# Hatókonyság elemzés

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Egy algoritmus elemzése azt jelenti, hogy előre megmondjuk, milyen erőforrásokra lesz szüksége az algoritmusnak.

Alkalmanként az olyan erőforrások, mint

- memória,
- kommunikációs sávszélesség vagy
- logikai kapuk,

elsődleges fontosságúak, de leggyakrabban a

- számítási idő

az, amit mérni akarunk.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatókonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns

## Hatókonyság elemzés

A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatókonysága  
Oszd meg és uralkodj

# Hatékonyság elemzés

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mielőtt elemeznénk egy algoritmust, rendelkeznünk kell a felhasználó megvalósítási technológia modelljével, beleérte a technológia erőforrásainak modelljét és azok költségeit is.

Ezen tény keretében nagy részében számítási modellként egy általános feldolgozó egységet, az ún. közvetlen hozzáférésű gépet (random access machine = RAM) alkalmazzuk megvalósítási technológiaként, és algoritmusainkat úgy értelmezzük, hogy számítógépprogramként valósítjuk meg őket.

A RAM modellben az utasítások egymás után hajtódnak végre, egyidejű műveletek nélkül.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológiák

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonyság elemzés
  - A RAM modell
  - A bemenet mérete
  - A futási idő
- A beszúró rendezés hatékonysága
- Oszd meg és uralkodj

# A RAM modell

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A RAM modell olyan utasításokat tartalmaz, amelyek rendszerint megtalálhatók a valódi számítógépeken:

- aritmetikai
    - (összeadás, kivonás, szorzás, osztás, maradékképzés, alsó egész rész, felső egész rész),
  - adatmozgató
    - (betöltés, tárolás, másolás)
  - és vezérlésátadó
    - (feltételes és feltétel nélküli elágazás, eljáráshívás és visszatérés)
- utasítások.

Minden ilyen utasítás konstans hosszúságú időt vesz igénybe.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés

## A RAM modell

A bemenet mérete  
A futási idő  
A beszűrő rendezés  
hatékonysága  
Oszd meg és uralkodj

# A beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A Beszúró-rendezés eljárás által felhasznált idő a bemenettől függ:

- ezer szám rendezése tovább tart,
- mint három számé.

Sőt, a BESZÚRÓ-RENDEZÉS akár két ugyanolyan méretű bemenő sorozatot is rendezhet különböző idő alatt attól függően, hogy azok mennyire vannak már eleve rendezve.

Általában egy algoritmus által felhasznált idő a bemenet méretével nő, így hagyományosan egy program futási idejét bemenete méretének függvényével írjuk le.

Ehhez pontosabban kell definiálnunk a **futási idő** és a **bemenet mérete** kifejezéseket.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés

### A RAM modell

A bemenet mérete  
A futási idő

A beszúró rendezés hatékonysága

Oszd meg és uralkodj

## A bemenet mérete

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A **bemenet méretének** legjobb mértéke a vizsgált feladattól függ.

- Sok feladatra, mint például a rendezés, a legtermészetesebb mérték a bemenő elemek száma.
- Sok egyéb feladatra, mint például két egész szám szorzása, a bemenet méretére a legjobb mérték a bemenet közönséges bináris jelölésben való ábrázolásához szükséges bitek teljes száma.
- Néha megfelelőbb a bemenet méretét egy szám helyett inkább kettővel leírni.
  - Például, ha az algoritmus bemenete egy gráf, a bemenet méretét leírhatjuk a gráf éleinek és csúcsainak számával.

Minden vizsgált feladatnál megadjuk, hogy a bemenet méretének melyik mértékét használjuk.

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell

### A bemenet mérete

A futási idő  
A beszúró rendezés hatékonysága  
Oszd meg és uralkodj

# A futási idő

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az algoritmusok **futási ideje** egy bizonyos bemenetre a végrehajtott alapműveletek vagy lépések száma.

Kényelmes úgy definiálni a lépést, hogy minél inkább gépfüggetlen legyen.

Egyelőre fogadjuk el a következőt: pszeudokódunk minden egyik sorának végrehajtásához állandó mennyiségű idő szükséges.

Lehet, hogy az egyik sor tovább tart, mint a másik, de feltesszük, hogy az  $i$ -edik sor minden végrehajtása  $c_i$  ideig tart, ahol  $c_i$  állandó.

Ez a nézőpont megfelel a RAM modellnek, és tükrözi azt is, ahogyan a pszeudokódot végrehajtaná a legtöbb jelenlegi számítógép.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatókonyság elemzés
- A RAM modell
- A bemenet mérete

## A futási idő

- A beszúró rendezés hatékonysága
- Oszd meg és uralkodj

## A beszúró rendezés hatékonysága

A következő elemzésben a BESZÚRÓ-RENDEZÉS futási idejét a – minden egyes utasítás  $c_i$  végrehajtási idejét figyelembe vevő – bonyolult képletből egyszerűbb, tömörebb és könnyebben kezelhető alakra hozzuk. Ez az egyszerűbb alak annak elődöntését is megkönnyíti, hogy egy algoritmus hatékonyabb-e egy másiknál.

A BESZÚRÓ-RENDEZÉS eljárás bemutatását az utasítások időköltségével és az egyes utasítások végrehajtásainak számával kezdjük.

- Minden  $j = 2, 3, \dots, n$ -re, ahol  $n = \text{méret}[A]$ ,  $t_j$  legyen az a szám, ahányszor a **while** ciklus 4. sorban lévő tesztje végrehajtódik az adott  $j$  értékre.
- Ha egy **for** vagy **while** ciklusból a szokásos módon lépünk ki (azaz a ciklusfejben lévő teszt eredményeképpen), akkor a teszt eggyel többször hajtódik végre, mint a ciklusmag.
- Felte tesszük, hogy a megjegyzések nem végrehajtható utasítások, így nem vesznek igénybe időt.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

### A beszúró rendezés hatékonysága

Oszd meg és uralkodj!

# Beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**procedure** BESZÚRÓ-RENDEZÉS( $A$ )

- | ①                    | for | $j \leftarrow 2$ | to | méret( $A$ ) | do        | $C_1$                               | $n$                      |       |                    |
|----------------------|-----|------------------|----|--------------|-----------|-------------------------------------|--------------------------|-------|--------------------|
| ②                    |     |                  |    |              |           | $C_2$                               | $n-1$                    |       |                    |
| ③                    |     |                  |    |              |           | $C_3$                               | $n-1$                    |       |                    |
| ④                    |     |                  |    |              | while     | $i \geq 1$ és $A[i] > \text{kulcs}$ | do                       | $C_4$ | $\sum_{j=2}^n t_j$ |
| ⑤                    |     |                  |    |              |           | $C_5$                               | $\sum_{j=2}^n (t_j - 1)$ |       |                    |
| ⑥                    |     |                  |    |              |           | $C_6$                               | $\sum_{j=2}^n (t_j - 1)$ |       |                    |
| ⑦                    |     |                  |    |              | end while |                                     |                          |       |                    |
| ⑧                    |     |                  |    |              |           | $C_8$                               | $n-1$                    |       |                    |
| ⑨                    |     |                  |    |              | end for   |                                     |                          |       |                    |
| <b>end procedure</b> |     |                  |    |              |           |                                     |                          |       |                    |

(Itt feltételeztük, hogy  $n = \text{méret}(A) > 0$ .)

költség

végrehajtási szám

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj!

# Beszúró rendezés hatékonysága

A BESZÚRÓ-RENDEZÉS  $T(n)$  futási ideje tehát az egyes sorokhoz tartozó költségek és végrehajtási számok szorzatösszege:

$$\begin{aligned} T(n) &= c_1 n + c_2 (n - 1) + c_3 (n - 1) + c_4 \sum_{j=2}^n t_j \\ &\quad + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_8 (n - 1) \\ &= n \cdot (c_1 + c_2 + c_3 + c_8) - (c_2 + c_3 + c_8) \\ &\quad + \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6) - (n - 1) \cdot (c_5 + c_6) \\ T(n) &= n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8) \\ &\quad + \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6) \end{aligned}$$

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj!

# Beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6)$$



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj!

# Beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6)$$

„Szerencsés” esetben, azaz rendezett bemenetnél:

- $t_j = 1$  minden j-re

$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ (n - 1) \cdot (c_4 + c_5 + c_6)$$
$$= n \cdot (c_1 + c_2 + c_3 + c_4 + c_8) - (c_2 + c_3 + c_4 + c_8)$$
$$= a \cdot n + b$$

$T(n)$  lineáris függvénye  $n$ -nek:

$$T(n) \in \Theta(n)$$

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatókonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatókonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj

## Beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6)$$

Legrosszabb esetben, „fordítottan rendezett” bemenettel:

- $t_j = j$  minden  $j$ -re  $\left( \sum_{j=2}^n t_j = \sum_{j=2}^n j = \frac{n(n+1)}{2} - 1 \right)$

$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \left( \frac{n(n+1)}{2} - 1 \right) \cdot (c_4 + c_5 + c_6)$$
$$= n^2 \cdot \frac{c_4 + c_5 + c_6}{2} + n \cdot \left( c_1 + c_2 + c_3 + c_8 + \frac{c_4 - c_5 - c_6}{2} \right)$$
$$- (c_2 + c_3 + c_4 + c_8)$$
$$= a \cdot n^2 + b \cdot n + c$$

$T(n)$  négyzetes függvénye  $n$ -nek:  $T(n) \in \Theta(n^2)$

### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológia

Hatékonyság  
Technológia

### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

### A beszúró rendezés hatékonysága

Oszd meg és uralkodj

# Beszúró rendezés hatékonysága

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \left( \sum_{j=2}^n t_j \right) \cdot (c_4 + c_5 + c_6)$$

„Átlagos” esetben:

- $t_j \simeq j/2$  minden j-re  $\left( \sum_{j=2}^n t_j \simeq \sum_{j=2}^n \frac{j}{2} \simeq \frac{n^2}{4} \right)$

$$T(n) = n \cdot (c_1 + c_2 + c_3 - c_5 - c_6 + c_8) - (c_2 + c_3 - c_5 - c_6 + c_8)$$
$$+ \frac{n^2}{4} \cdot (c_4 + c_5 + c_6)$$
$$= a \cdot n^2 + b \cdot n + c$$

$T(n)$  négyzetes függvénye  $n$ -nek:  $T(n) \in \Theta(n^2)$

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Hatékonyság elemzés

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A későbbiekben általában a legrosszabb futási idő keresésére összpontosítunk, azaz a leghosszabb futási időre tetszőleges  $n$  méretű bemenetre.

Erre három indokot adunk.

- A legrosszabb futási idő garanciát jelent arra, hogy az algoritmus soha nem fog tovább tartani. Nem szükséges találhatásokba bocsátkoznunk a futási időről, azután reménykedni, hogy nem lesz ennél rosszabb.
- Bizonyos algoritmusokra gyakran fordul elő a legrosszabb eset. Például, ha egy bizonyos információt keresünk egy adatbázisban, a kereső-algoritmus legrosszabb esete gyakran előfordul, ha az információ nincs az adatbázisban.
- Az „átlagos eset” gyakran nagyából ugyanolyan rossz, mint a legrosszabb eset.
  - Lásd a BESZÚRÓ-RENDEZÉS fenti elemzését, ahol lényegében egy 2-es faktor volt csupán az eltérés.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj

# Hatókonyság elemzés

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Bizonyos esetekben az algoritmus átlagos vagy várható futási idejére van szükségünk;

Ilyenkor a valószínűségi elemzés módszerével határozható meg a várható futási idő.

Az átlagos eset elemzésének vizsgálata során probléma lehet, hogy nem feltétlenül nyilvánvaló, mi az „átlagos” bemenet.

Gyakran feltételezzük, hogy minden adott méretű bemenet ugyanolyan valószínű, de ez a gyakorlatban nem mindig jogos.

A valószínűségi elemzésre olyankor is szükség van, ha véletlenített algoritmust alkalmazunk.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatókonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatókonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő

## A beszúró rendezés hatékonysága

Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Sok hasznos algoritmus szerkezetében **rekurzív**:

- egy adott feladat megoldásához rekurzívan hívják magukat egyszer vagy többször,
- egymáshoz szorosan kötődő részfeladatok kezelésére.

Ezek az algoritmusok általában az **oszd-meg-és-uralkodj** megközelítést követik:

- a feladatot több részfeladatra osztják,
- amelyek hasonlóak az eredeti feladathoz,
- de méretük kisebb,
- rekurzív módon megoldják a részfeladatokat,
- majd „összevonják” ezeket a megoldásokat,

hogy az eredeti feladatra megoldást adjanak.

## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológia

- Hatékonyúság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatékonyúság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő

- A beszúró rendezés hatékonyisége

Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az oszd-meg-és-uralkodj paradigmája a rekurzió minden szintjén három lépésből áll.

**Felosztja** a feladatot több részfeladatra.

**Uralkodik** a részfeladatokon rekurzív módon való megoldásukkal.

- Ha a részfeladatok mérete elég kicsi, akkor közvetlenül megoldja a részfeladatokat.

**Összevonja** a részfeladatok megoldásait az eredeti feladat megoldásává.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiá

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az **összefésülő rendezés** algoritmusa szorosan követi az oszd-meg-és-uralkodj paradigmát.

Lényegét tekintve a következőképpen működik.

**Felosztás:** Az  $n$  elemű rendezendő sorozatot felosztja két  $n/2$  elemű részsorozatra.

**Uralkodás:** A két részsorozatot összefésülő rendezéssel rekurzív módon rendezi.

**Összevonás:** Összefésüli a két részsorozatot, létrehozva a rendezett választ.

A rekurzió akkor „áll le” (nem osztja tovább a feladatot két részre), amikor a rendezendő sorozat hossza 1: ekkor nincs mit csinálnia, mivel minden 1 hosszúságú sorozat már eleve sorba van rendezve.

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyág  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyág elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűrő rendezés hatékonyiséga

Oszd meg és uralkodj

# Oszd meg és uralkodj

## procedure

ÖSSZEFÉSÜL(A,p,q,r)

- ①  $n_1 \leftarrow q-p+1$
- ②  $n_2 \leftarrow r-q$
- ③ az  $L[1..n_1+1]$  és az  $R[1..n_2+1]$  tömbök létrehozása
- ④ **for**  $i \leftarrow 1$  **to**  $n_1$  **do**
- ⑤      $L[i] \leftarrow A[p+i-1]$
- ⑥ **end for**
- ⑦ **for**  $j \leftarrow 1$  **to**  $n_2$  **do**
- ⑧      $R[j] \leftarrow A[q+j]$
- ⑨ **end for**

- ⑩  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$
- ⑪  $i \leftarrow j \leftarrow 1$
- ⑫ **for**  $k \leftarrow p$  **to**  $r$  **do**
- ⑬     **if**  $L[i] \leq R[j]$  **then**
- ⑭          $A[k] \leftarrow L[i]$
- ⑮          $i \leftarrow i + 1$
- ⑯     **else**
- ⑰          $A[k] \leftarrow R[j]$
- ⑱          $j \leftarrow j + 1$
- ⑲     **end if**
- ⑳ **end for**

**end procedure**

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága

## Oszd meg és uralkodj

# Oszd meg és uralkodj

## procedure

ÖSSZEFÉSÜL(A,p,q,r)

- ①  $n_1 \leftarrow q-p+1$
- ②  $n_2 \leftarrow r-q$
- ③ az  $L[1..n_1+1]$  és az  $R[1..n_2+1]$  tömbök létrehozása
- ④ **for**  $i \leftarrow 1$  **to**  $n_1$  **do**
- ⑤      $L[i] \leftarrow A[p+i-1]$
- ⑥ **end for**
- ⑦ **for**  $j \leftarrow 1$  **to**  $n_2$  **do**
- ⑧      $R[j] \leftarrow A[q+j]$
- ⑨ **end for**

- ⑩  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$
- ⑪  $i \leftarrow j \leftarrow 1$
- ⑫ **for**  $k \leftarrow p$  **to**  $r$  **do**
- ⑬     **if**  $L[i] \leq R[j]$  **then**
- ⑭          $A[k] \leftarrow L[i]$
- ⑮          $i \leftarrow i + 1$
- ⑯     **else**
- ⑰          $A[k] \leftarrow R[j]$
- ⑱          $j \leftarrow j + 1$
- ⑲     **end if**
- ⑳ **end for**

**end procedure**

**Ciklusinvariáns:** A 12–19. sorokbeli **for** ciklus minden iterációjának kezdetén az  $A[p .. k-1]$  rész-tömb az  $L[1 .. n_1+1]$  és az  $R[1 .. n_2+1]$  tömbök  $k-p$  darab legkisebb elemét tartalmazzák, rendezetten. Továbbá  $L[i]$  és  $R[j]$  a megfelelő tömbök legkisebb olyan elemei, amelyek még nincsenek visszamásolva A-ba.

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága

## Oszd meg és uralkodj

# Oszd meg és uralkodj

## procedure

### ÖSSZEFÉSÜL(A,p,q,r)

- ①  $n_1 \leftarrow q-p+1$
- ②  $n_2 \leftarrow r-q$
- ③ az  $L[1..n_1+1]$  és az  $R[1..n_2+1]$  tömbök létrehozása
- ④ **for**  $i \leftarrow 1$  **to**  $n_1$  **do**
- ⑤      $L[i] \leftarrow A[p+i-1]$
- ⑥ **end for**
- ⑦ **for**  $j \leftarrow 1$  **to**  $n_2$  **do**
- ⑧      $R[j] \leftarrow A[q+j]$
- ⑨ **end for**

- ⑩  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$
- ⑪  $i \leftarrow j \leftarrow 1$
- ⑫ **for**  $k \leftarrow p$  **to**  $r$  **do**
- ⑬     **if**  $L[i] \leq R[j]$  **then**
- ⑭          $A[k] \leftarrow L[i]$
- ⑮          $i \leftarrow i + 1$
- ⑯     **else**
- ⑰          $A[k] \leftarrow R[j]$
- ⑱          $j \leftarrow j + 1$
- ⑲     **end if**
- ⑳ **end for**
- end procedure**

Annak belátásához, hogy ÖSSZEFÉSÜL  $\Theta(n)$  ideig fut, ahol  $n=r-p+1$ , figyeljük meg, hogy az 1–3. és a 8–11. sorok állandó időt vesznek igénybe, a 4–7. sorokban szereplő **for** ciklusok  $\Theta(n_1+n_2) = \Theta(n)$  időt, a 12–17. sorokban szereplő **for** ciklusnak pedig **n** iterációja van, és mindegyik állandó ideig tart.

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	2	4	5	7	1	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(a)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	5	7	1	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(c)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	2	3	1	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(e)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	2	3	4	5	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(g)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	2	3	4	5	6	7	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(h)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	4	5	7	1	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(b)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	2	4	5	7	$\infty$	$\dots$	$\dots$	$\dots$	$\dots$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(d)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	2	7	1	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(f)

$T$	8	9	10	11	12	13	14	15	16	17	$\dots$
$B$	...	1	2	2	3	4	2	3	6	$\infty$	$k$
$B$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$i$
$J$	1	2	3	4	5	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$j$

(h)



## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológiák

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinvariáns
- Hatékonyság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonysága

## Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

**procedure** ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, r$ )

1 **if**  $p < r$  **then**

2      $q \leftarrow \lfloor (p + r) / 2 \rfloor$

3     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, q$ )

4     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, q+1, r$ )

5     ÖSSZEFÉSÜL( $A, p, q, r$ )

6 **end if**

**end procedure**

---

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiá

Hatékonyúság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyúság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonyiséga

Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

**procedure** ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, r$ )

- 1 **if**  $p < r$  **then**
- 2      $q \leftarrow \lfloor (p + r) / 2 \rfloor$
- 3     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, q$ )
- 4     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, q+1, r$ )
- 5     ÖSSZEFÉSÜL( $A, p, q, r$ )
- 6 **end if**

**end procedure**

---

Amikor egy algoritmus rekurzívan hívja magát, futási idejét gyakran rekurzív egyenlőséggel vagy rekurzióval írhatjuk le, amely az  $n$  méretű feladat megoldásához szükséges teljes futási időt a kisebb bemenetekhez tartozó futási időkkel fejezi ki. A rekurzió megoldására matematikai eszközöket használunk, melyekkel az algoritmusra teljesítménykorlátokat tudunk adni.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológiák

Hatékonyág  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyág elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonyisége

Oszd meg és uralkodj

# Oszd meg és uralkodj

Az algoritmusok  
szerepe a  
számításokban

**procedure** ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, r$ )

- 1 **if**  $p < r$  **then**
- 2      $q \leftarrow \lfloor (p + r) / 2 \rfloor$
- 3     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, q$ )
- 4     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, q+1, r$ )
- 5     ÖSSZEFÉSÜL( $A, p, q, r$ )
- 6 **end if**

**end procedure**

$$T(n) = \begin{cases} \Theta(1) & \text{ha } n < c \\ 2 \cdot T(n/2) + D(n) + C(n) & \text{egyébként} \end{cases}$$

Esetünkben könnyű belátni, hogy:

$$D(n) \in \Theta(1)$$

és már láttuk, hogy:

$$C(n) \in \Theta(n)$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológiák

- Hatékonyság
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinváriáns
- Hatékonyság elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonysága

Oszd meg és uralkodj

# Oszd meg és uralkodj

**procedure** ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, r$ )

- 1 **if**  $p < r$  **then**
- 2      $q \leftarrow \lfloor (p + r) / 2 \rfloor$
- 3     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, p, q$ )
- 4     ÖSSZEFÉSÜLŐ-RENDEZÉS( $A, q+1, r$ )
- 5     ÖSSZEFÉSÜL( $A, p, q, r$ )
- 6 **end if**

**end procedure**

---

$$T(n) = \begin{cases} \Theta(1) & \text{ha } n = 1 \\ 2 \cdot T(n/2) + \Theta(n) & \text{egyébként} \end{cases}$$

Később látni fogjuk, hogy ebből az iteratív feltételből következik, hogy:

$$T(n) \in \Theta(n \cdot \log_2 n)$$

Az algoritmusok  
szerepe a  
számításokban

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Algoritmusok

- Algoritmusok fogalma
- Algoritmusok megadása
- Algoritmusok építőelemei
- Példa: rendezési feladat

## Algoritmusok mint technológiák

- Hatékonyág
- Technológia

## Algoritmusok elemzése

- Algoritmusok helyessége
- Ciklusinváriáns
- Hatékonyág elemzés
- A RAM modell
- A bemenet mérete
- A futási idő
- A beszúró rendezés hatékonyiséga

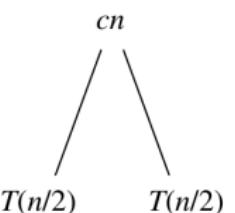
Oszd meg és uralkodj



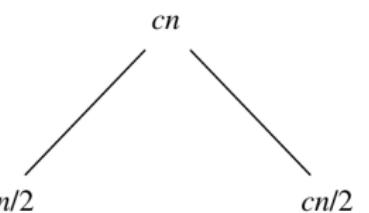
# Oszd meg és uralkodj

$T(n) \in \Theta(n \cdot \log_2 n)$ : „Bizonyítás” rekurziós fával

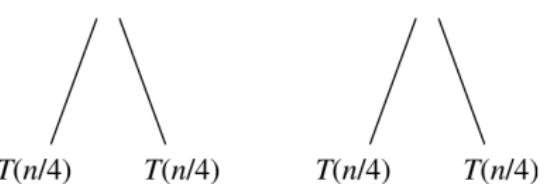
$T(n)$



(a)



(b)



(c)

## Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

## Algoritmusok mint technológia

Hatékonyság  
Technológia

## Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinváriáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszúró rendezés  
hatékonysága

Oszd meg és uralkodj

$$T(n) = \begin{cases} c & \text{ha } n = 1 \\ 2 \cdot T(n/2) + c \cdot n & \text{együbként} \end{cases}$$



### Algoritmusok

Algoritmusok fogalma  
Algoritmusok megadása  
Algoritmusok építőelemei  
Példa: rendezési feladat

### Algoritmusok mint technológiá

Hatékonyság  
Technológia

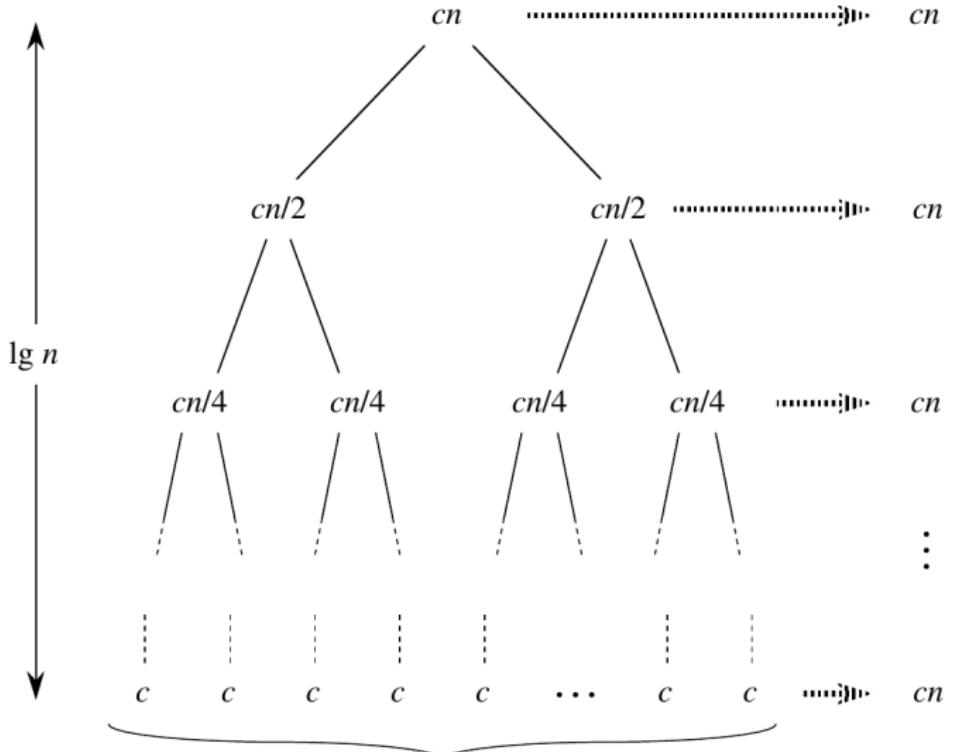
### Algoritmusok elemzése

Algoritmusok helyessége  
Ciklusinvariáns  
Hatékonyság elemzés  
A RAM modell  
A bemenet mérete  
A futási idő  
A beszűrő rendezés  
hatékonysága

### Oszd meg és uralkodj

# Oszd meg és uralkodj

$T(n) \in \Theta(n \cdot \log_2 n)$ : „Bizonyítás” rekurziós fával



$$T(n) = \begin{cases} c & \text{ha } n = 1 \\ 2 \cdot T(n/2) + c \cdot n & \text{egyébként} \end{cases}$$

$n$

(d)

Összesen:  $cn \lg n + cn$



# 2. előadás

## Függvények aszimptotikus viselkedése

Valamint további egyszerű algoritmusok

*Adatszerkezetek és algoritmusok előadás*

2018. február 13.

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

# Általános tudnivalók

Ajánlott irodalom:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- Gyakorlati aláírás
  - 2 ZH
- Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Függvények  
asziptomatikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények asziptomatikus  
viselkedése

Asziptomatikus jelölések  
képletekben

Asziptomatikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzív megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények növekedése

## Aszimptotikus jelölések

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Függvények növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

[Rekurzív megadott  
függvények](#)

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

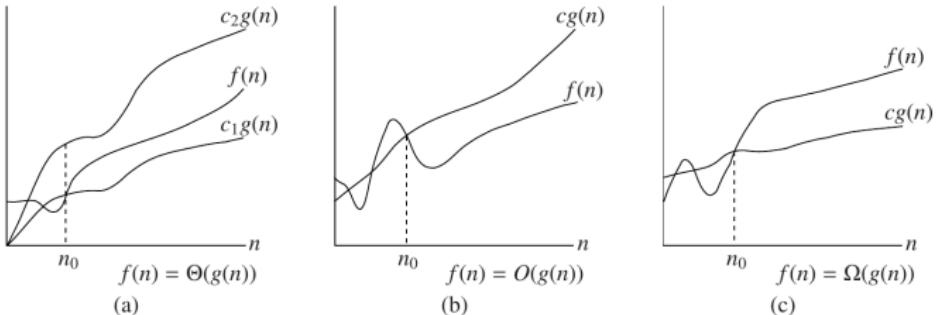
Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

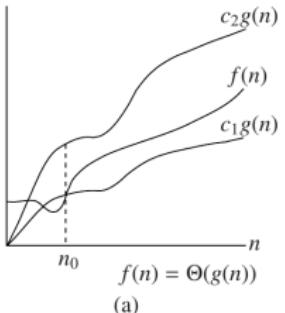


Az előzőekben megállapítottuk, hogy a beszúró rendezés futási ideje a legrosszabb esetben  $T(n) = \Theta(n^2)$ .

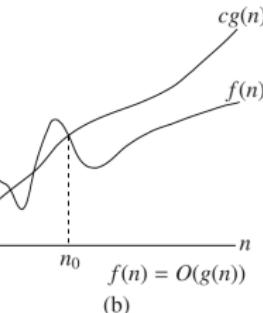
Most pontosan definiáljuk, hogy mit jelent ez a jelölés.

# Függvények növekedése

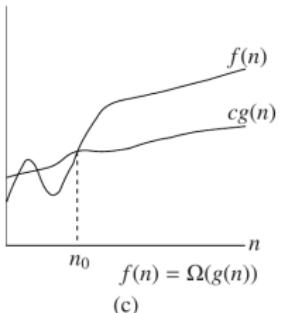
## Aszimptotikus jelölések



(a)



(b)



(c)

A  **$\Theta$**  jelölés (aszimptotikus éles korlát) definíciója:

Egy adott  $g(n)$  függvény esetén  $\Theta(g(n))$ -nel jelöljük a függvényeknek azt a halmazát, amelyre

$$\Theta(g(n)) = \left\{ f(n) : \begin{array}{l} \text{létezik } c_1, c_2 \text{ és } n_0 \text{ pozitív állandó, hogy} \\ 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \\ \text{teljesül minden } n \geq n_0 \text{ esetén} \end{array} \right\}$$

$$T(n) = \Theta(n^2) \quad \equiv \quad T(n) \in \Theta(n^2)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

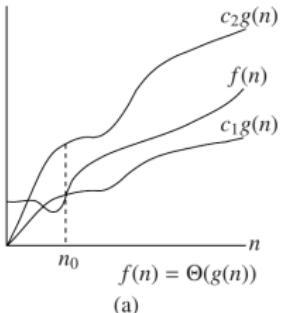
Amikor működik

Amikor nem működik

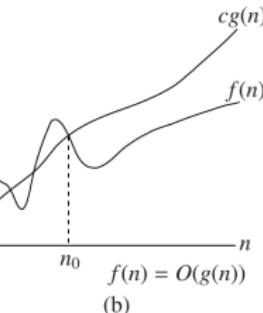
„Levezetés”

# Függvények növekedése

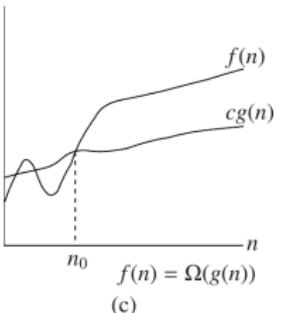
## Aszimptotikus jelölések



(a)



(b)



(c)

Az ***O*** jelölés (aszimptotikus felső korlát) definíciója:

Egy adott  $g(n)$  függvény esetén  $O(g(n))$ -nel („nagy ordó”  $g(n)$ ) jelöljük a függvényeknek azt a halmazát, amelyre

$$O(g(n)) = \left\{ \begin{array}{l} f(n) : \text{ létezik } c \text{ és } n_0 \text{ pozitív állandó, hogy} \\ \quad 0 \leq f(n) \leq c \cdot g(n) \\ \text{ teljesül minden } n \geq n_0 \text{ esetén} \end{array} \right\}$$

$$T(n) = O(n^2) \equiv T(n) \in O(n^2)$$

$$T(n) = \Theta(n^2) \implies T(n) \in O(n^2)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények növekedése

## Aszimptotikus jelölések

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Függvények növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

### Rekurzív megadott függvények

A helyettesítő módszer

Finomítások

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

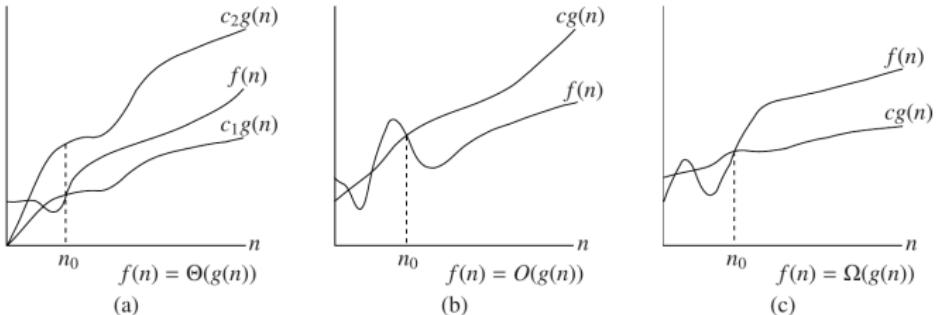
Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



Az  $\Omega$  jelölés (aszimptotikus alsó korlát) definíciója:

Egy adott  $g(n)$  függvény esetén  $\Omega(g(n))$ -nel jelöljük a függvényeknek azt a halmazát, amelyre

$$\Omega(g(n)) = \left\{ \begin{array}{l} f(n) : \text{ létezik } c \text{ és } n_0 \text{ pozitív állandó, hogy} \\ \quad 0 \leq c \cdot g(n) \leq f(n) \\ \text{ teljesül minden } n \geq n_0 \text{ esetén} \end{array} \right\}$$

$$T(n) = \Omega(n^2) \equiv T(n) \in \Omega(n^2)$$

$$T(n) = \Theta(n^2) \implies T(n) \in \Omega(n^2)$$



## Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

## Rekurzíván megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények növekedése

Korábban azt mondta, a  $T(n) = a \cdot n^2 + b \cdot n + c$  függvényre, hogy aszimptotikusan csak az  $n^2$  tagja a fontos:

$$T(n) = \Theta(n^2)$$

Most megnézhetjük, igaz-e ez?

Ehhez találnunk kell olyan pozitív  $c_1$ -et,  $c_2$ -öt és  $n_0$ -át, amivel

$$c_1 \cdot n^2 \leq a \cdot n^2 + b \cdot n + c \leq c_2 \cdot n^2 \quad \text{amikor } n \geq n_0.$$

Ehhez előbb osszuk el az egyenlőtlenséget  $n^2$ -tel ( $n > 0$ ):

$$c_1 \leq a + b/n + c/n^2 \leq c_2$$

majd vegyük észre, hogy a középső kifejezés elég nagy  $n$  esetén jól közelíthető  $a$ -val, vagyis bármely  $c_1$  és  $c_2$  esetén, amelyekre teljesül, hogy

$$0 < c_1 < a < c_2, \quad (\text{csak akkor van esélyünk, ha } 0 < a)$$

már csak annyi a dolgunk, hogy megfelelő pozitív  $n_0$ -t keressünk. Pl. amivel:

$$(|b| + |c|)/n_0 \leq \min(c_2 - a, a - c_1)$$

## Függvények növekedése

Korábban azt mondta, a  $T(n) = a \cdot n^2 + b \cdot n + c$  függvényre, hogy aszimptotikusan csak az  $n^2$  tagja a fontos:

$$T(n) = \Theta(n^2)$$

Most megnézhetjük, igaz-e ez?

Ehhez találnunk kell olyan pozitív  $c_1$ -et,  $c_2$ -öt és  $n_0$ -át, amivel

$$c_1 \cdot n^2 \leq a \cdot n^2 + b \cdot n + c \leq c_2 \cdot n^2 \quad \text{amikor } n \geq n_0.$$

Ehhez előbb osszuk el az egyenlőtlenséget  $n^2$ -tel ( $n > 0$ ):

$$c_1 \leq a + b/n + c/n^2 \leq c_2$$

majd vegyük észre, hogy a középső kifejezés elég nagy  $n$  esetén jól közelíthető  $a$ -val, vagyis bármely  $c_1$  és  $c_2$  esetén, amelyekre teljesül, hogy

$$0 < c_1 < a < c_2, \quad (\text{csak akkor van esélyünk, ha } 0 < a)$$

már csak annyi a dolgunk, hogy megfelelő pozitív  $n_0$ -t keressünk. Pl. amivel:

$$(|b| + |c|)/n_0 \leq \min(c_2 - a, a - c_1)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Függvények növekedése

Korábban azt mondta, a  $T(n) = a \cdot n^2 + b \cdot n + c$  függvényre, hogy aszimptotikusan csak az  $n^2$  tagja a fontos:

$$T(n) = \Theta(n^2)$$

Most megnézhetjük, igaz-e ez?

Ehhez találnunk kell olyan pozitív  $c_1$ -et,  $c_2$ -öt és  $n_0$ -át, amivel

$$c_1 \cdot n^2 \leq a \cdot n^2 + b \cdot n + c \leq c_2 \cdot n^2 \quad \text{amikor } n \geq n_0.$$

Ehhez előbb osszuk el az egyenlőtlenséget  $n^2$ -tel ( $n > 0$ ):

$$c_1 \leq a + b/n + c/n^2 \leq c_2$$

majd vegyük észre, hogy a középső kifejezés elég nagy  $n$  esetén jól közelíthető  $a$ -val, vagyis bármely  $c_1$  és  $c_2$  esetén, amelyekre teljesül, hogy

$$0 < c_1 < a < c_2, \quad (\text{csak akkor van esélyünk, ha } 0 < a)$$

már csak annyi a dolgunk, hogy megfelelő pozitív  $n_0$ -t keressünk. Pl. amivel:

$$(|b| + |c|)/n_0 \leq \min(c_2 - a, a - c_1)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények növekedése

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hasonlóan lehet belátni, hogy bármely

$$P_d(n) = \sum_{i=0}^d a_i \cdot n^i$$

polinom esetén (ha  $a_d > 0$ ) igaz, hogy

$$P_d(n) = \Theta(n^d)$$

Speciálisan, konstans függvények esetén írhatjuk, hogy benne vannak a  $\Theta(n^0)$  függvényhalmazban.

- Ehelyett –kicsit pontatlanul– azt szoktuk mondani, hogy benne vannak a  $\Theta(1)$  halmazban.
- Valójában gyakran használják a  $\Theta(1)$  jelölést akár állandóra, akár egy adott változóra nézve konstans függvényre is.

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása  
Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer  
Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer  
Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények növekedése

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Theorem

Bármely két  $f(n)$  és  $g(n)$  függvény esetén

$$f(n) = \Theta(g(n))$$

akkor és csak akkor, ha

$$f(n) = O(g(n)) \text{ és } f(n) = \Omega(g(n)).$$

A gyakorlatban ezt tételt általában arra használjuk, hogy az aszimptotikusan éles korlát létét mutassuk meg az aszimptotikus alsó és felső korlátok felhasználásával.

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Aszimptotikus jelölések képletekben

Ha az aszimptotikus jelölés egyedül van az egyenlet jobb oldalán, mint

$$n = O(n^2)$$

esetén, akkor úgy definiáltuk az egyenlőségjelet, mint a halmazhoz tartozás jelét:

$$n \in O(n^2)$$

Általában azonban, egy képletben előforduló aszimptotikus jelölést úgy tekintünk, mint egy olyan ismeretlen függvény jelölését, amit nem is akarunk megnevezni. Pl. a

$$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$$

képlet azt jelenti, hogy

$$2n^2 + 3n + 1 = 2n^2 + f(n)$$

ahol  $f(n) \in \Theta(n)$

Ebben az esetben  $f(n) = 3n + 1$ , ami tényleg benne van  $\Theta(n)$ -ben.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Aszimptotikus jelölések képletekben

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása  
Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

Ekképpen használva, az aszimptotikus jelölés segít a képletből eltávolítani a lényegtelen részleteket.

Pl. amikor rekurzívan kifejeztük az összefésülő rendezés legrosszabb futási idejét:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n).$$

Ha csak  $T(n)$  aszimptotikus viselkedése érdekel minket, akkor nincs értelme minden alacsonyabb rendű tagot pontosan megadni;

mindet beleértjük abba a névtelen függvénybe, amit a  $\Theta(n)$  taggal jelölünk.

## Aszimptotikus jelölések képletekben

Bizonyos esetekben aszimptotikus jelölés előfordulhat az egyenlet bal oldalán is, mint például

$$2n^2 + \Theta(n) = \Theta(n^2).$$

Ezeket az egyenleteket a következő szabály szerint értelmezzük:

- Mindegy, hogy miképpen választjuk meg az egyenlet bal oldalán lévő ismeretlen függvényt,
- van mód arra, hogy a jobb oldali ismeretlen függvényt úgy válasszuk meg,
- hogy az egyenlőség igaz legyen.

Így példánk jelentése az, hogy tetszőleges  $f(n) \in \Theta(n)$  függvényhez létezik  $g(n) \in \Theta(n^2)$  úgy, hogy

$$2n^2 + f(n) = g(n) \text{ minden } n\text{-re.}$$

Más szóval az egyenlet jobb oldala durvábban, kevésbé pontosan írja le a függvényt, mint az egyenlet bal oldala.

Egy még összetettebb példa:

$$\begin{aligned} 2n^2 + 3n + 1 &= 2n^2 + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása  
Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok  
Buktatók

Új változó bevezetése

A rekurziós fa módsze

Sejtés keresése  
Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL  
Amikor működik  
Amikor nem működik  
„Levezetés”



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
példákban

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása  
Alapfüggvények

Rekurzív megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A tételek

Amikor működik

Amikor nem működik

„Levezetés”

## Aszimptotikusan nem éles korlátok

Az O-jelölés által adott felső korlát aszimptotikusan

- vagy éles ( $2n^2 = O(n^2)$ ),
- vagy nem ( $2n = O(n^2)$ ).

Amikor hangsúlyozni szeretnénk, hogy az adott felső korlát nem éles, akkor a **o-jelölést** használjuk („kis ordó”):

$$o(g(n)) = \left\{ \begin{array}{l} f(n) : \text{ tetszőleges } c > 0 \text{-hoz található } n_0 > 0, \text{ hogy} \\ \quad 0 \leq f(n) < c \cdot g(n) \\ \text{teljesül minden } n \geq n_0 \text{ esetén} \end{array} \right\}$$

Vagy (nemnegatív aszimptotikus viselkedés mellett még):

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$$

Fenti függvényekre:

- $2n = O(n^2)$  és  $2n = o(n^2)$ , ill.
- $2n^2 = O(n^2)$  de  $2n^2 \neq o(n^2)$ .

## Aszimptotikusan nem éles korlátok

Hasonlóképpen, az  $\Omega$ -jelölés által adott alsó korlát is aszimptotikusan

- vagy éles ( $2n = \Omega(n)$ ),
- vagy nem ( $2n^2 = \Omega(n)$ ).

Amikor hangsúlyozni szeretnénk, hogy az adott alsó korlát nem éles, akkor a  **$\omega$ -jelölést** használjuk („kis omega”):

$$\omega(g(n)) = \left\{ \begin{array}{l} f(n) : \text{tetszőleges } c > 0 \text{-hoz található } n_0 > 0, \text{ hogy} \\ \quad 0 \leq c \cdot g(n) < f(n) \\ \text{teljesül minden } n \geq n_0 \text{ esetén} \end{array} \right.$$

Vagy (nemnegatív aszimptotikus viselkedés mellett még):

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$$

Fenti függvényekre:

- $2n^2 = \Omega(n)$  és  $2n^2 = \omega(n)$ , ill.
- $2n = \Omega(n)$  de  $2n \neq \omega(n)$ .

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

## Függvények összehasonlítása

Alapfüggvények

## Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő

módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

### Tranzitivitás:

ha  $f(n) = \Theta(g(n))$  és  $g(n) = \Theta(h(n))$ , akkor  $f(n) = \Theta(h(n))$

ha  $f(n) = O(g(n))$  és  $g(n) = O(h(n))$ , akkor  $f(n) = O(h(n))$

ha  $f(n) = \Omega(g(n))$  és  $g(n) = \Omega(h(n))$ , akkor  $f(n) = \Omega(h(n))$

ha  $f(n) = o(g(n))$  és  $g(n) = o(h(n))$ , akkor  $f(n) = o(h(n))$

ha  $f(n) = \omega(g(n))$  és  $g(n) = \omega(h(n))$ , akkor  $f(n) = \omega(h(n))$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

Reflexivitás:

$$f(n) = \Theta(f(n))$$

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

**Szimmetria:**

$$f(n) = \Theta(g(n)) \text{ akkor és csak akkor, ha } g(n) = \Theta(f(n))$$

**Felcserélt szimmetria:**

$$f(n) = O(g(n)) \text{ akkor és csak akkor, ha } g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \text{ akkor és csak akkor, ha } g(n) = \omega(f(n))$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

Mivel ezek a tulajdonságok érvényesek az aszimptotikus jelölésekre, ezért párhuzamot vonhatunk két függvény,  $f$  és  $g$ , valamint két valós szám,  $a$  és  $b$  összehasonlítása között:

$$f(n) = O(g(n)) \quad \approx \quad a \leq b$$

$$f(n) = \Omega(g(n)) \quad \approx \quad a \geq b$$

$$f(n) = \Theta(g(n)) \quad \approx \quad a = b$$

$$f(n) = o(g(n)) \quad \approx \quad a < b$$

$$f(n) = \omega(g(n)) \quad \approx \quad a > b$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

Vigyázat, a valós számok alábbi tulajdonsága már nem vihető át a függvények aszimptotikus viselkedésére:

**Trichotómia:** Bármely két valós  $a$  és  $b$  szám esetén az alábbi tulajdonságok közül pontosan egy teljesül:

- $a < b$
- $a = b$
- $a > b$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzív megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Függvények összehasonlítása

Számos – valós számokra vonatkozó – összefüggés alkalmazható aszimptotikus összehasonlításokra is.

A továbbiakban tegyük fel, hogy  $f(n)$  és  $g(n)$  aszimptotikusan pozitív.

---

Vigyázat, a valós számok alábbi tulajdonsága már nem vihető át a függvények aszimptotikus viselkedésére:

**Trichotómia:** Bármely két valós  $a$  és  $b$  szám esetén az alábbi tulajdonságok közül pontosan egy teljesül:

- $a < b$
- $a = b$
- $a > b$

Pl. az  $f(n) = n + n^{10} \sin^2 n$  függvényre igazak a következők:

- $f(n) = \Omega(n)$  és  $f(n) = \omega(1)$
- $f(n) = O(n^{10})$  és  $f(n) = o(n^{11})$
- de  $f(n)$  nem hasonítható semelyik  $P_d = n^d$  függvénnnyel, ha  $d = 2, 3, \dots, 9$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A térel

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Monotonitás

Az  $f(n)$  függvényre azt mondjuk, hogy  
monoton növekedő, ha

$$m < n \implies f(m) \leq f(n)$$

monoton csökkenő, ha

$$m < n \implies f(m) \geq f(n)$$

szigorúan monoton növekedő, ha

$$m < n \implies f(m) < f(n)$$

szigorúan monoton csökkenő, ha

$$m < n \implies f(m) > f(n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Egészrész függvények

**Alsó egészrész,**  $\lfloor x \rfloor$ : olyan egész, amire  $x \geq \lfloor x \rfloor > x - 1$

**Felső egészrész,**  $\lceil x \rceil$ : olyan egész, amire  $x \leq \lceil x \rceil < x + 1$

Minden n egész számra:

$$\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n,$$

és minden valós  $x \geq 0$  értékre és m, n > 0 egészre

$$\left\lfloor \frac{\lfloor x \rfloor}{n} \right\rfloor = \left\lfloor \frac{x}{m \cdot n} \right\rfloor$$

$$\left\lceil \frac{\lceil x \rceil}{n} \right\rceil = \left\lceil \frac{x}{m \cdot n} \right\rceil$$

$$\frac{m}{n} \geq \left\lfloor \frac{m}{n} \right\rfloor \geq \frac{m - (n - 1)}{n} = \frac{m}{n} - \frac{n - 1}{n}$$

$$\frac{m}{n} \leq \left\lceil \frac{m}{n} \right\rceil \leq \frac{m + (n - 1)}{n} = \frac{m}{n} + \frac{n - 1}{n}$$

Mindkét egészrész függvény monoton növekedő.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Egészrész függvények

**Alsó egészrész,  $\lfloor x \rfloor$ :** olyan egész, amire  $x \geq \lfloor x \rfloor > x - 1$

**Felső egészrész,  $\lceil x \rceil$ :** olyan egész, amire  $x \leq \lceil x \rceil < x + 1$

Minden n egész számra:

$$\left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil = n,$$

és minden valós  $x \geq 0$  értékre és  $m, n > 0$  egészre

$$\left\lfloor \frac{\lfloor x \rfloor}{m} \right\rfloor = \left\lfloor \frac{x}{m \cdot n} \right\rfloor$$

$$\left\lceil \frac{\lceil x \rceil}{m} \right\rceil = \left\lceil \frac{x}{m \cdot n} \right\rceil$$

$$\frac{m}{n} \geq \left\lfloor \frac{m}{n} \right\rfloor \geq \frac{m - (n - 1)}{n} = \frac{m}{n} - \frac{n - 1}{n}$$

$$\frac{m}{n} \leq \left\lceil \frac{m}{n} \right\rceil \leq \frac{m + (n - 1)}{n} = \frac{m}{n} + \frac{n - 1}{n}$$

Mindkét egészrész függvény monoton növekedő.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Osztási maradék

Tetszőleges egész  $a$  és pozitív egész  $n$  egész esetén jelölje  $a \bmod n$  az a szám  $n$ -nel vett **osztási maradékát**, vagyis

$$a \bmod n = a - n \cdot \left\lfloor \frac{a}{n} \right\rfloor$$

Hasznos lesz, ha arra is definiálunk egy külön jelölést, hogy az  $a$  és  $b$  egészek  $n$ -nel vett osztási maradéka megegyezik. Ha  $(a \bmod n) = (b \bmod n)$ , akkor ezt úgy jelöljük, hogy

$$a \equiv b \pmod{n},$$

és azt mondjuk, hogy  $a$  és  $b$  **kongruens** modulo  $n$ .

Ha  $a$  és  $b$  nem ekvivalens modulo  $n$ , akkor erre az

$$a \not\equiv b \pmod{n}$$

jelölést használjuk.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Polinomok

Adott  $d$  pozitív egész számra az  $n$  változó  $d$ -edfokú polinomján egy

$$P_d(n) = \sum_{i=0}^d a_i \cdot n^i$$

alakú függvényt értünk, ahol az  $a_0, a_1, \dots, a_d$  állandók a polinom együtthatói, és  $a_d \neq 0$ .

- Egy polinom aszimptotikusan pozitív akkor és csak akkor, ha  $a_d > 0$ .
- Az aszimptotikusan pozitív  $d$ -edfokú  $P_d(n)$  polinomra teljesül, hogy  $P_d(n) = \Theta(n^d)$ .
- minden valós  $a \geq 0$  állandó esetén  $n^a$  monoton növekedő,
- minden valós  $a \leq 0$  állandó esetén  $n^a$  monoton csökkenő.
- Az  $f(n)$  függvény **polinomiálisan korlátos**, ha  $f(n) = n^{O(1)}$ , ami ekvivalens azzal, hogy  $f(n) = O(n^k)$  valamely  $k$  állandóra.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Hatványok

A következő azonosságok teljesülnek minden  $a \neq 0$ ,  $m$ ,  $n$  valós számra:

$$a^0 = 1$$

$$a^1 = 1$$

$$a^{-1} = 1/a$$

$$(a^m)^n = a^{m \cdot n}$$

$$(a^m)^n = (a^n)^m$$

$$a^n \cdot a^m = a^{n+m}$$

Ha  $a \geq 1$ , akkor az  $n$  változótól függő  $a^n$  függvény monoton növekvő.

A **polinomok** és **hatványok** növekedése a következőképpen hasonlítható össze.

Minden  $a > 1$  és  $b$  valós állandó esetén

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

amiből

$$n^b = o(a^n)$$

Azaz minden **exponenciális** függvény, amelynek alapja nagyobb 1-nél, gyorsabban növekszik, mint bármely polinom függvény.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzív megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Hatványok

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A természetes logaritmus alapját  $e$  ( $= 2,71828 \dots$ ) jelöli.

Minden valós  $x$ -re

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$$

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$$

$$e^x \geq 1 + x \quad \text{egyenlőség csak, ha} \quad x = 0$$

$$1 + x \leq e^x \leq 1 + x + x^2 \quad \text{ha } |x| \leq 1$$

$$e^x = 1 + x + \Theta(x^2) \quad \text{ha } x \rightarrow 0$$

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Logaritmusok

A következő jelöléseket fogjuk használni:

$$\lg n = \log_2 n \quad (\text{kettes alapú logaritmus})$$

$$\ln n = \log_e n \quad (\text{természetes alapú logaritmus})$$

$$\lg^k n = (\lg n)^k \quad (\text{hatványozás})$$

$$\lg \lg n = \lg (\lg n) \quad (\text{kompozíció})$$

$$\lg n + k \equiv (\lg n) + k$$

$b > 1$  és  $n > 0$  esetén a  $\log_b n$  függvény sz. m. növekvő.

Minden valós  $a > 0$ ,  $b > 0$ ,  $c > 0$ ,  $x > -1$  és  $n$  számra igaz:

$$a = b^{\log_b a} \quad \log_c(a \cdot b) = \log_c a + \log_c b \quad \log_b a^n = n \cdot \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b} \quad \log_b \frac{1}{a} = -\log_b a \quad \log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a} \quad \frac{x}{1+x} \leq \ln(1+x) \leq x$$

ahol a logaritmus alapja soha nem 1.

Ha  $|x| \leq 1$ , akkor:  $\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

## Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Polilogaritmikusan korlátos függvények

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Korábban láttuk, hogy

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0$$

Írjuk ezt most át a következő módon:  $n \rightarrow \lg n$  ill.  $a \rightarrow 2^a$

$$\lim_{n \rightarrow \infty} \frac{(\lg n)^b}{(2^a)^{\lg n}} = \lim_{n \rightarrow \infty} \frac{\lg^b n}{n^a} = 0$$

innen pedig, bármely  $a > 0$ -ra

$$\lg^b n = o(n^a)$$

Azaz minden pozitív polinomfüggvény gyorsabban nő, mint bármelyik **polilogaritmikus** függvény.

Függvények  
növekedése

Függvények aszimptotikus viselkedése  
Aszimptotikus jelölések képletekben  
Aszimptotikusan nem éles korlátok  
Függvények összehasonlítása

### Alapfüggvények

Rekurzívan megadott függvények

A helyettesítő módszer  
Finomságok  
Buktatók  
Új változó bevezetése  
A rekurziós fa módszer  
Sejtés keresése  
Bizonyítás a helyettesítő módszerrel  
Mester módszer  
A téTEL  
Amikor működik  
Amikor nem működik  
„Levezetés”



### Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

### Alapfüggvények

### Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Faktoriálisok

$n \geq 0$  egészekre definiáljuk a faktoriális fogalmát:

$$n! = \begin{cases} 1, & \text{ha } n = 0, \\ n \cdot (n-1)!, & \text{ha } n > 0. \end{cases}$$

Egy „durva” korlát:  $n! \leq n^n$ .

Ennél sokkal jobb közelítés a Stirling-formula:

$$n! = \sqrt{2\pi n} \left( \frac{n}{e} \right)^n \left( 1 + \Theta \left( \frac{1}{n} \right) \right)$$

A Stirling-formula alapján belátható még, hogy

$$n! = o(n^n)$$

$$n! = \omega(2^n)$$

$$\lg(n!) = \Theta(n \lg n).$$

Ezekben túlmenően igaz még, hogy

$$n! = \sqrt{2\pi n} \left( \frac{n}{e} \right)^n e^{\alpha_n}$$

ahol

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n}.$$

# Szokásos jelölések és alapfüggvények

## Iterált függvények

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az  $f(n)$  függvény  $i$ -edik **iterált függvényének** definíciója  
( $i \geq 0$  egész):

$$f^{(i)}(n) = \begin{cases} n, & \text{ha } i = 0, \\ f(f^{(i-1)}(n)), & \text{ha } i > 0. \end{cases}$$

Pi.  $f(n) = 2n$  esetén  $f^{(i)}(n) = 2^i n$ .

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

**Alapfüggvények**

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Szokásos jelölések és alapfüggvények

## Fibonacci-számok

A Fibonacci-számokat az alábbi rekurzió definiálja:

$$F_0 = 0$$

$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \quad \text{ha } i > 1$$

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

A Fibonacci-számok kapcsolatban vannak az aranymetszés  $\phi$  arányával és annak  $\hat{\phi}$  konjugáltjával:

$$\phi = \frac{1 + \sqrt{5}}{2} = 1.61803 \dots$$

$$\hat{\phi} = \frac{1 - \sqrt{5}}{2} = -0.61803 \dots$$

Nevezetesen:

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}}$$

Mivel  $|\hat{\phi}| < 1$ , ezért  $|\hat{\phi}| / \sqrt{5} < 1 / \sqrt{5} < 1/2$ , így  $F_i$  meghatározható, mint a

$\phi^i / \sqrt{5}$ -höz legközelebbi egész szám.

# Rekurzívan megadott függvények aszimptotikus viselkedése

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Amint azt már korábban megemlítettük, ha egy algoritmus önmagát hívja többször egymás után, akkor futási ideje gyakran jellemző rekurzióval.

Például az ÖSSZEFÉSÜLŐ-RENDEZÉS  $T(n)$  legrosszabb futási idejére azt írtuk, hogy

$$T(n) = \begin{cases} \Theta(1) & \text{ha } n = 1 \\ 2 \cdot T(n/2) + \Theta(n) & \text{egyébként} \end{cases}$$

## Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

## Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Rekurzívan megadott függvények aszimptotikus viselkedése

Ez a fejezet három módszert mutat az ehhez hasonló rekurziók megoldására – azaz arra, hogyan kaphatunk aszimptotikus  $\Theta$  vagy  $O$  korlátokat a megoldásra.

- A helyettesítő módszernél
  - megsejtünk egy korlátot,
  - majd teljes indukcióval igazoljuk sejtésünk helyességét.
- A rekurziós fa módszerben
  - a rekurzív képlet alapján felépítünk egy fát, melyben egy adott szinten lévő csúcsok a rekurzió adott mélységében fellépő költségeknek felelnek meg;
  - a rekurziót ezután az összegek becslésére szolgáló módszerekkel oldjuk meg.
- A mester módszer az alábbi alakú rekurzív egyenletekre alkalmazható , ha  $a \geq 1$  és  $b > 1$ :

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- Ehhez a módszerhez három esetet kell megjegyezni, de ha ezt egyszer megtesszük, utána már könnyedén tudunk aszimptotikus korlátot megadni sok egyszerű rekurzív képletre vonatkozóan.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Rekurzívan megadott függvények aszimptotikus viselkedése

Ez a fejezet három módszert mutat az ehhez hasonló rekurziók megoldására – azaz arra, hogyan kaphatunk aszimptotikus  $\Theta$  vagy  $O$  korlátokat a megoldásra.

- A **helyettesítő módszernél**

- megsejtünk egy korlátot,
- majd teljes indukcióval igazoljuk sejtésünk helyességét.

- A **rekurziós fa módszerben**

- a rekurzív képlet alapján felépítünk egy fát, melyben egy adott szinten lévő csúcsok a rekurzió adott mélységében fellépő költségeknek felelnek meg;
- a rekurziót ezután az összegek becslésére szolgáló módszerekkel oldjuk meg.

- A **mester módszer** az alábbi alakú rekurzív egyenletekre alkalmazható , ha  $a \geq 1$  és  $b > 1$ :

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- Ehhez a módszerhez három esetet kell megjegyezni, de ha ezt egyszer megtesszük, utána már könnyedén tudunk aszimptotikus korlátot megadni sok egyszerű rekurzív képletre vonatkozóan.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



## Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

## Rekurzív megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Rekurzívan megadott függvények aszimptotikus viselkedése

Ez a fejezet három módszert mutat az ehhez hasonló rekurziók megoldására – azaz arra, hogyan kaphatunk aszimptotikus  $\Theta$  vagy  $O$  korlátokat a megoldásra.

### • A helyettesítő módszernél

- megsejtünk egy korlátot,
- majd teljes indukcióval igazoljuk sejtésünk helyességét.

### • A rekurziós fa módszerben

- a rekurzív képlet alapján felépítünk egy fát, melyben egy adott szinten lévő csúcsok a rekurzió adott mélységében fellépő költségeknek felelnek meg;
- a rekurziót ezután az összegek becslésére szolgáló módszerekkel oldjuk meg.

### • A mester módszer az alábbi alakú rekurzív egyenletekre alkalmazható , ha $a \geq 1$ és $b > 1$ :

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- Ehhez a módszerhez három esetet kell megjegyezni, de ha ezt egyszer megtesszük, utána már könnyedén tudunk aszimptotikus korlátot megadni sok egyszerű rekurzív képletre vonatkozóan.



## Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

## Rekurzív megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Rekurzív megadott függvények aszimptotikus viselkedése

Ez a fejezet három módszert mutat az ehhez hasonló rekurziók megoldására – azaz arra, hogyan kaphatunk aszimptotikus  $\Theta$  vagy  $O$  korlátokat a megoldásra.

- A **helyettesítő módszernél**

- megsejtünk egy korlátot,
- majd teljes indukcióval igazoljuk sejtésünk helyességét.

- A **rekurziós fa módszerben**

- a rekurzív képlet alapján felépítünk egy fát, melyben egy adott szinten lévő csúcsok a rekurzió adott mélységében fellépő költségeknek felelnek meg;
- a rekurziót ezután az összegek becslésére szolgáló módszerekkel oldjuk meg.

- A **mester módszer** az alábbi alakú rekurzív egyenletekre alkalmazható , ha  $a \geq 1$  és  $b > 1$ :

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

- Ehhez a módszerhez három esetet kell megjegyezni, de ha ezt egyszer megtesszük, utána már könnyedén tudunk aszimptotikus korlátot megadni sok egyszerű rekurzív képletre vonatkozóan.

## Technikai részletek

A gyakorlatban a rekurzív problémák megfogalmazása és megoldása során bizonyos technikai részleteket elhanyagolunk. Pl.:

- az ÖSSZEFÉSÜLŐ-RENDEZÉS legrosszabb futási idejét leíró rekurzió valójában

$$T(n) = \begin{cases} \Theta(1) & \text{ha } n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + \Theta(n) & \text{egyébként} \end{cases}$$

- A kezdeti feltételek a részleteknek egy másik olyan jellegzetes csoportját alkotják, amelyet általában elhanyagolunk.
- Mivel az algoritmus futási ideje állandó méretű bemenet esetén állandó, és így gyakran teljesül, hogy  $T(n) = \Theta(1)$ . Ezért ezt gyakran ki sem írjuk:

$$T(n) = 2 * T(n/2) + \Theta(n)$$

Ezek a részletek többnyire nem befolyásolják a megoldás aszimptotikus viselkedését ...

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A rekurziók **helyettesítő módszerrel** való megoldása két lépésből áll:

- ① Sejtsük meg a megoldást.
- ② Teljes indukcióval határozzuk meg az állandókat és igazoljuk a megoldás helyességét.

Az elnevezés onnan ered, hogy a helyesnek vélt megoldást be kell helyettesíteni a függvénybe, miközben az indukciós feltevést kisebb értékekre alkalmazzuk. Ez hatékony módszer, de nyilvánvalóan csak akkor alkalmazható, ha a helyes válasz könnyen megsejthető.

Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

**Rekurzív megadott  
függvények**

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A tételek

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A helyettesítő módszer rekurzióval megadott függvény felső vagy alsó korlátjának a meghatározására is használható.

Határozzuk meg a

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n$$

rekurzív képlettel jellemzett függvény egy felső korlátját!

- Sejtésünk az, hogy a megoldás:  $T(n) = O(n \lg n)$ .
- Módszerünk az, hogy bebizonyítjuk, megfelelően választott  $c > 0$  állandóra  $T(n) \leq cn \lg n$ .

Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- Először is feltesszük, hogy ezzel a korláttal  $k_0 \leq k < n$ -re érvényes az egyenlőtlenség, azaz  $T(k) \leq c \lfloor k \rfloor \lg(\lfloor k \rfloor)$ .
- Ezt a rekurzív egyenletbe helyettesítve kapjuk, hogy (kihasználjuk, hogy:  $n/2 < n$ )

$$\begin{aligned} T(n) &= 2T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \leq 2c \left\lfloor \frac{n}{2} \right\rfloor \lg\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + n \leq cn \lg \frac{n}{2} + n \\ &= cn \lg n - cn \lg 2 + n = cn \lg n - cn + n \leq cn \lg n \end{aligned}$$

ha  $c \geq 1$

- Most már csak egy  $n = 1$ -re kellene belátnunk az egyenlőtlenséget. Csakhogy:

$$c \cdot 1 \cdot \lg 1 = 0 \not\geq T(1)$$

- A megoldás:  $T(n) = O(n \lg n)$ -hez elég az egyenlőtlenséget valamely  $n_0$  után megkövetelni.

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A térel

Amikor működik

Amikor nem működik

„Levezetés”

## A helyettesítő módszer

Sajnos, nincs általános szabály a megoldás megsejtésére.

A jó sejtés kitalálásához

- gyakorlatra, és néha
- találékonyságra van szükség.

Szerencsére azonban van néhány heurisztikus módszer, amely segít kitalálni a jó sejtést.

Segíthet pl. a rekurziós fa módszer használata is.

Az is segíthet, ha hasonlóságot fedezünk fel már ismert és megoldott rekurzióhoz:

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + 3n$$

esetén pl. azonnal gondolnunk kell a

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

Sajnos, nincs általános szabály a megoldás megsejtésére.

A jó sejtés kitalálásához

- gyakorlatra, és néha
- találékonyságra van szükség.

Szerencsére azonban van néhány heurisztikus módszer, amely segít kitalálni a jó sejtést.

Segíthet pl. a rekurziós fa módszer használata is.

Az is segíthet, ha hasonlóságot fedezünk fel már ismert és megoldott rekurzióhoz:

$$T(n) = 2T\left(\left\lfloor \frac{n}{2} \right\rfloor + 17\right) + 3n$$

esetén pl. azonnal gondolnunk kell a

$$T(n) = O(n \lg n)$$

lehetőségre.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Finomságok

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Tekintsük a következő rekurzív egyenletet:

$$T(n) = 2T(n/2) + 1.$$

Erre az esetre „ mindenki ” első gondolata:  $T(n) = O(n)$ .

Próbáljuk meg bizonyítani:

- Tegyük fel, hogy  $1 \leq k < n$  esetén teljesül:  $T(k) \leq c \cdot k$ .
- Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot c \cdot n/2 + 1 = c \cdot n + 1 \not\leq c \cdot n$$

- Hogyan tovább?
  - Tegyük fel, hogy  $T(k) \leq c \cdot k - b$
  - Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot (c \cdot n/2 - b) + 1 = c \cdot n - 2b + 1 \leq c \cdot n - b,$$

legalábbis meg tudjuk  $b$ -t úgy választani, hogy teljesüljön az egyenlőtlenség.

Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Finomságok

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Tekintsük a következő rekurzív egyenletet:

$$T(n) = 2T(n/2) + 1.$$

Erre az esetre „ mindenki ” első gondolata:  $T(n) = O(n)$ .

Próbáljuk meg bizonyítani:

- Tegyük fel, hogy  $1 \leq k < n$  esetén teljesül:  $T(k) \leq c \cdot k$ .
- Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot c \cdot n/2 + 1 = c \cdot n + 1 \not\leq c \cdot n$$

- Hogyan tovább?

- Tegyük fel, hogy  $T(k) \leq c \cdot k - b$
- Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot (c \cdot n/2 - b) + 1 = c \cdot n - 2b + 1 \leq c \cdot n - b,$$

legalábbis meg tudjuk  $b$ -t úgy választani, hogy teljesüljön az egyenlőtlenség.

Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Finomságok

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Tekintsük a következő rekurzív egyenletet:

$$T(n) = 2T(n/2) + 1.$$

Erre az esetre „ mindenki ” első gondolata:  $T(n) = O(n)$ .

Próbáljuk meg bizonyítani:

- Tegyük fel, hogy  $1 \leq k < n$  esetén teljesül:  $T(k) \leq c \cdot k$ .
- Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot c \cdot n/2 + 1 = c \cdot n + 1 \not\leq c \cdot n$$

- Hogyan tovább?

- Tegyük fel, hogy  $T(k) \leq c \cdot k - b$
- Számoljuk ki  $T(n)$  értékét:

$$T(n) = 2T(n/2) + 1 \leq 2 \cdot (c \cdot n/2 - b) + 1 = c \cdot n - 2b + 1 \leq c \cdot n - b,$$

legalábbis meg tudjuk  $b$ -t úgy választani, hogy teljesüljön az egyenlőtlenség.

Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Buktatók

- Rekurzió:

$$T(n) = 2T(n/2) + n$$

- sejtés: kis  $k$  egészekre teljesül, hogy

$$T(k) \leq c \cdot (k)$$

- bizonyítás:

$$T(n) = 2T(n/2) + n \leq 2 \cdot c \cdot (n/2) + n = c \cdot n + n \in O(n)$$

Hol a hiba?

- Nem bizonyítottuk az induktíós feltevés pontos alakját:

$$T(n) \leq c \cdot n$$

ill. máshonnan közelítve:

$$c \cdot n + n \in O(n) \Rightarrow T(n) \in O(n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer  
Finomágok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Buktatók

- Rekurzió:

$$T(n) = 2T(n/2) + n$$

- sejtés: kis  $k$  egészekre teljesül, hogy

$$T(k) \leq c \cdot (k)$$

- bizonyítás:

$$T(n) = 2T(n/2) + n \leq 2 \cdot c \cdot (n/2) + n = c \cdot n + n \in O(n)$$

Hol a hiba?

- Nem bizonyítottuk az induktíós feltevés **pontos alakját**:

$$T(n) \leq c \cdot n$$

ill. máshonnan közelítve:

$$c \cdot n + n \in O(n) \Rightarrow T(n) \in O(n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer  
Finomsgák

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Új változó bevezetése

- Legyen

$$T(n) = 2T(\sqrt{n}) + \lg n.$$

- ???

- Vezessük be az  $m = \lg n$  új változót! Ekkor:

$$T(2^m) = 2T(2^{m/2}) + m.$$

- Írjuk fel az  $S(m) = T(2^m)$  függvényre vonatkozó rekurziós formulát:

$$S(m) = 2S(m/2) + m$$

Ismerőς valahonnan?

- Megoldás:

$$S(m) = O(m \lg m)$$

ill.

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg^{(2)} n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Új változó bevezetése

- Legyen

$$T(n) = 2T(\sqrt{n}) + \lg n.$$

- ???

- Vezessük be az  $m = \lg n$  új változót! Ekkor:

$$T(2^m) = 2T(2^{m/2}) + m.$$

- Írjuk fel az  $S(m) = T(2^m)$  függvényre vonatkozó rekurziós formulát:

$$S(m) = 2S(m/2) + m$$

Ismerős valahonnan?

- Megoldás:

$$S(m) = O(m \lg m)$$

ill.

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg^{(2)} n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# A helyettesítő módszer

## Új változó bevezetése

- Legyen

$$T(n) = 2T(\sqrt{n}) + \lg n.$$

- ???

- Vezessük be az  $m = \lg n$  új változót! Ekkor:

$$T(2^m) = 2T(2^{m/2}) + m.$$

- Írjuk fel az  $S(m) = T(2^m)$  függvényre vonatkozó rekurziós formulát:

$$S(m) = 2S(m/2) + m$$

Ismerős valahonnan?

- Megoldás:

$$S(m) = O(m \lg m)$$

ill.

$$T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg^{(2)} n)$$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## A rekurziós fa módszer

Egyszer már használtuk is a módszert.

- Az ÖSSZEFÉSÜLŐ-RENDEZÉS futási idejének számolására.

A rekurziós fa minden egyes csúcsa egy részfeladatnak felel meg:

- a függvény kiértékelésekor végrehajtódó minden rekurziós híváshoz tartozik egy csúcs.
- Szintenként összegezzük a csúcsok költségét,
- majd az így kapott szintenkénti költségeket összegezzük, hogy megkapjuk a teljes költséget.

Oszd-meg-és-uralkodj elvet használó algoritmusok elemzésénél a rekurziós fa nagyon kényelmes eszköznek bizonyul.

A rekurziós fa módszert leginkább egy jó sejtés megtalálására érdemes használni, amit aztán a helyettesítő módszerrel ellenőrzünk.

- Ilyenkor elegendő a számolást „nagyvonalúan” elvégezni.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus viselkedése  
Aszimptotikus jelölések képletekben  
Aszimptotikusan nem éles korlátok  
Függvények összehasonlítása  
Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer  
Finomságok  
Buktatók  
Új változó bevezetése

A rekurziós fa módszer

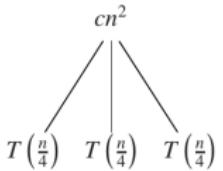
Sejtés keresése  
Bizonyítás a helyettesítő módszerrel  
Mester módszer  
A tételek  
Amikor működik  
Amikor nem működik  
„Levezetés”

# A rekurziós fa módszer

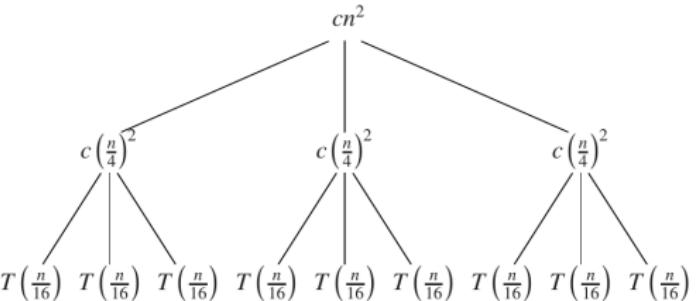
## Sejtés keresése

- Példaként nézzük meg a  $T(n) = 3T(\lfloor \frac{n}{4} \rfloor) + \Theta(n^2)$  rekurziót!
- A  $T(n) = 3T(n/4) + cn^2$  egyenletből az alábbi fát kapjuk:

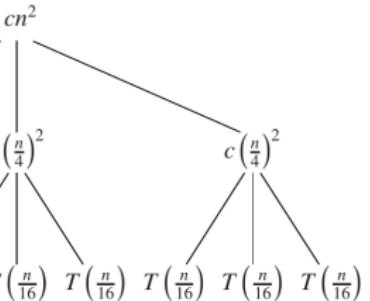
$T(n)$



(a)



(b)



(c)

- Majd végül ...

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer  
Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

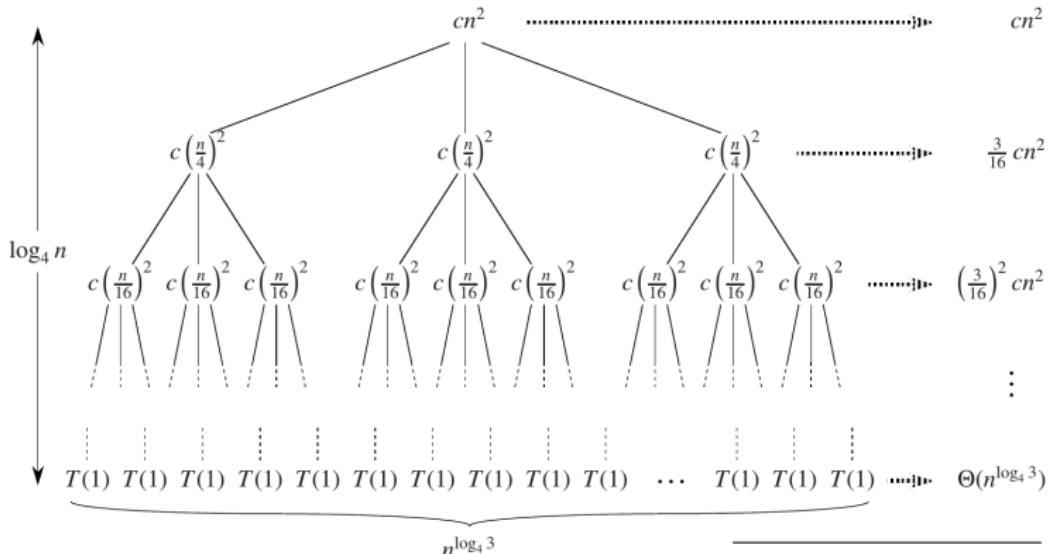
Amikor működik

Amikor nem működik

„Levezetés”

# A rekurziós fa módszer

## Sejtés keresése



$$\begin{aligned}
 T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{log_4 3}) \stackrel{(d)}{=} \frac{1 - \left(\frac{3}{16}\right)^{\log_4 n}}{1 - \frac{3}{16}} cn^2 + \Theta(n^{log_4 3}) \quad \text{Összesen: } O(n^2) \\
 &< \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta(n^{log_4 3}) = \frac{16}{13} cn^2 + \Theta(n^{log_4 3}) \\
 T(n) &= O(n^2) \qquad \quad \left(3^{\log_4 n} = (4^{\log_4 3})^{\log_4 n} = (4^{\log_4 n})^{\log_4 3} = n^{\log_4 3}\right)
 \end{aligned}$$



## Függvények növekedése

- Függvények aszimptotikus viselkedése
- Aszimptotikus jelölések képletekben
- Aszimptotikusan nem éles korlátok
- Függvények összehasonlítása
- Alapfüggvények

## Rekurzívan megadott függvények

- A helyettesítő módszer
- Finomságok
- Buktatók
- Új változó bevezetése
- A rekurziós fa módszer
- Sejtés keresése
- Bizonyítás a helyettesítő módszerrel
- Mester módszer
- A téTEL
- Amikor működik
- Amikor nem működik
- „Levezetés”

# A rekurziós fa módszer

## Bizonyítás a helyettesítő módszerrel

- Ha kis  $k$  egészekre már igaz, hogy

$$T(k) \leq d \cdot k^2$$

akkor

$$\begin{aligned} T(n) &= 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2) \leq 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + cn^2 \leq 3d\left(\left\lfloor \frac{n}{4} \right\rfloor\right)^2 + cn^2 \\ &\leq 3d\left(\frac{n}{4}\right)^2 + cn^2 = \frac{3}{16}dn^2 + cn^2 \leq dn^2 \end{aligned}$$

amennyiben teljesül, hogy:  $d \geq \frac{16}{13}c \iff c \leq \frac{13}{16}d$

amiből már következik, hogy  $T(n) = O(n^2)$

- Mivel  $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + \Theta(n^2)$ , nyilvánvalóan  $T(n) = \Omega(n^2)$ ,
- és így  $T(n) = \Theta(n^2)$ .

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer  
Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer  
Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



### Függvények növekedése

Függvények aszcimptotikus viselkedése

Aszcimptotikus jelölések  
képletekben

Aszcimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

### Rekurzívan megadott függvények

A helyettesítő módszer  
Finomsgák

Buktatók

Új változó bevezetése

A rekurziós fa módszer  
Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A térel

Amikor működik

Amikor nem működik

„Levezetés”

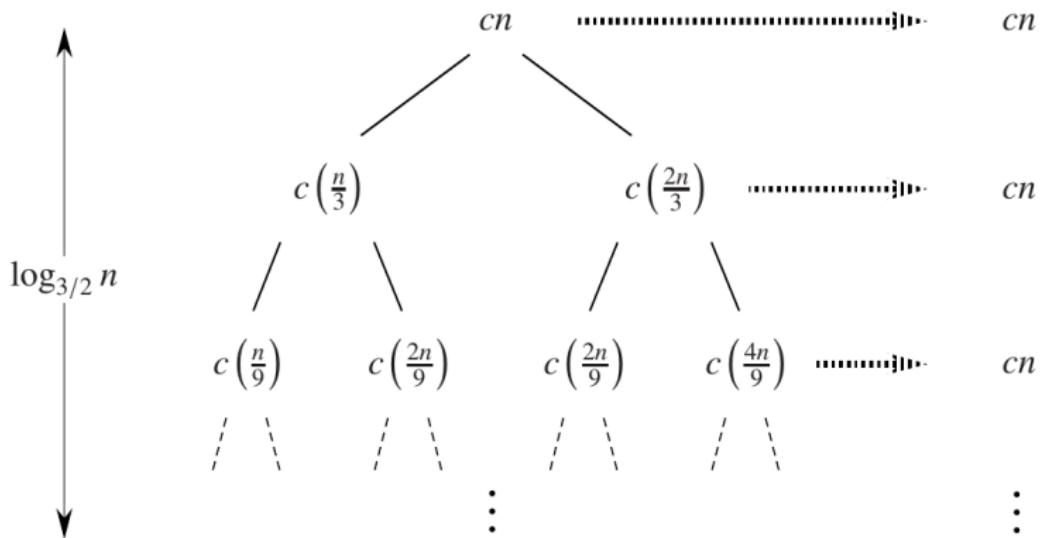
# A rekurziós fa módszer

## Egy picit bonyolultabb példa

- Tekintsük az alábbi rekurziót:

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

- A megfelelő rekurziós fa



$$\begin{aligned} n &\rightarrow \frac{2}{3}n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \dots \rightarrow 1, \\ \text{innen } \left(\frac{2}{3}\right)^k n &= 1, \text{ azaz } n = \left(\frac{3}{2}\right)^k \end{aligned}$$

Összesen:  $O(n \lg n)$   
„túloztunk”

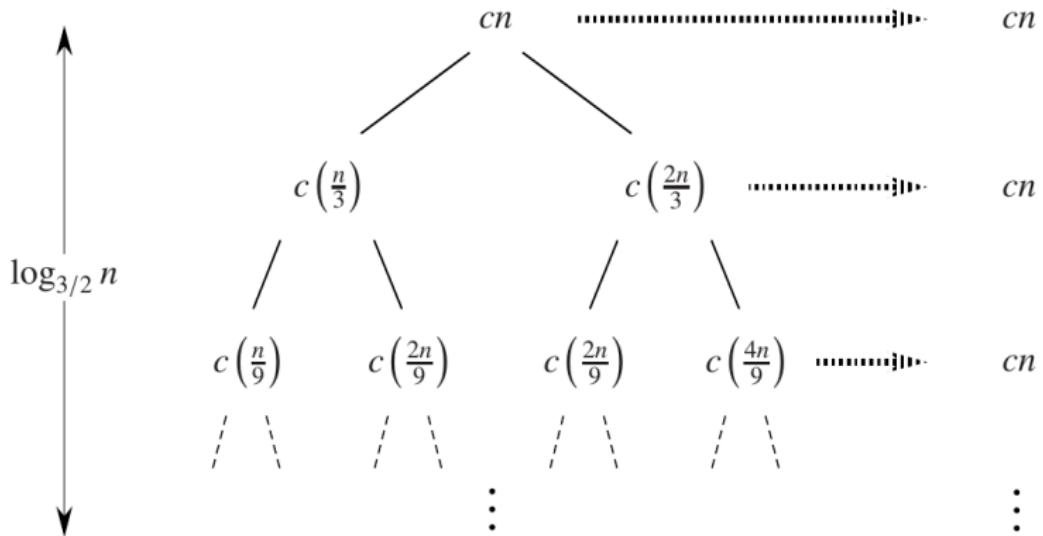
# A rekurziós fa módszer

## Egy picit bonyolultabb példa

- Tekintsük az alábbi rekurziót:

$$T(n) = T(n/3) + T(2n/3) + O(n)$$

- A megfelelő rekurziós fa



$$\begin{aligned} n &\rightarrow \frac{2}{3}n \rightarrow \left(\frac{2}{3}\right)^2 n \rightarrow \dots \rightarrow 1, \\ \text{innen } \left(\frac{2}{3}\right)^k n &= 1, \text{ azaz } n = \left(\frac{3}{2}\right)^k \end{aligned}$$

Összesen:  $O(n \lg n)$   
„túloztunk”

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

### Rekurzívan megadott függvények

A helyettesítő módszer  
Finomsgák

Buktatók

Új változó bevezetése

A rekurziós fa módszer  
Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A térel

Amikor működik

Amikor nem működik

„Levezetés”

# A rekurziós fa módszer

## Sejtés alapján helyettesítő módszer

- Induljunk ki a feltételezésből, hogy kis  $k$  egészekre alkalmas  $d$ -vel:  $T(k) \leq d \cdot k \lg k$

$$\begin{aligned} T(n) &= T(n/3) + T(2n/3) + O(n) \leq T(n/3) + T(2n/3) + cn \\ &\leq d \frac{n}{3} \lg \frac{n}{3} + d \frac{2n}{3} \lg \frac{2n}{3} + cn \\ &= d \frac{n}{3} (\lg n - \lg 3) + d \frac{2n}{3} \left( \lg n - \lg \frac{3}{2} \right) + cn \\ &= dn \lg n - d \left( \frac{n}{3} \lg 3 + \frac{2n}{3} \lg 3 - \frac{2n}{3} \lg 2 \right) + cn \\ &= dn \lg n - dn \left( \lg 3 - \frac{2}{3} \right) + cn \\ &= dn \lg n - \left( d \left( \lg 3 - \frac{2}{3} \right) - c \right) n \leq dn \lg n \\ \text{amennyiben: } d \left( \lg 3 - \frac{2}{3} \right) &> c \end{aligned}$$

Függvények  
asymptotikus  
viselkedése

Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzív megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

## Mester módszer

- A mester módszer receptet ad a

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

alakú rekurzív egyenletek megoldására, ahol

- $a \geq 1$  és
- $b > 1$  állandók,
- és az  $f(n)$  aszimptotikusa pozitív függvény.
- Szemléletesen, egy rekurzív eljárás az eredeti problémát
  - felosztja a darab, egyenként  $n/b$  méretű részproblémára,
  - a felosztás és az összevonás együttesen  $f(n)$  lépésben valósul meg.
- Belátható, hogy amennyiben  $n/b$  helyére (valamelyik) egészrészét írjuk ( $\lfloor \frac{n}{b} \rfloor$  vagy  $\lceil \frac{n}{b} \rceil$ ),
  - az nem érinti a „lényeget”
- A mester módszer alkalmazásához három esetet kell megjegyezni, ezután sok rekurzív egyenlet megoldása könnyen, sok esetben papír és ceruza nélkül, meghatározható.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



## Theorem (Mester téTEL)

Legyenek  $a \geq 1$ ,  $b > 1$  állandók,  $f(n)$  egy függvény,  $T(n)$  pedig a nemnegatív egészeken a

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

rekurzív egyenlettel definiált függvény, ahol  $n/b$  jelentheti akár az  $\lfloor n/b \rfloor$  egészrészét, akár az  $\lceil n/b \rceil$  egészrészét. Ekkor:

① Ha  $f(n) = O(n^{\log_b a - \epsilon})$ , egy  $\epsilon > 0$  állandóval, akkor

$$\longrightarrow T(n) = \Theta(n^{\log_b a})$$

② Ha  $f(n) = \Theta(n^{\log_b a})$ , akkor

$$\longrightarrow T(n) = \Theta(n^{\log_b a} \lg n) (= \Theta(f(n) \lg n))$$

③ Ha  $f(n) = \Omega(n^{\log_b a + \epsilon})$  és  $a \cdot f(n/b) \leq c \cdot f(n)$  valamely  $\epsilon > 0$  és  $c < 1$  állandóra és elég nagy  $n$ -re, akkor

$$\longrightarrow T(n) = \Theta(f(n))$$

### Függvények növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

### Rekurzívan megadott függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekursziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Mester módszer használata

## 1. eset

Ha  $f(n) = O(n^{\log_b a - \epsilon})$ , egy  $\epsilon > 0$  állandóval, akkor

$$\longrightarrow T(n) = \Theta(n^{\log_b a}).$$

- $T(n) = 9T(n/3) + n$

- $a = 9, b = 3$  és  $f(n) = n$

- $n^{\log_b a} = n^{\log_3 9} = n^2$

- $f(n) = O(n^{\log_3 9 - \epsilon})$ , ha  $0 < \epsilon \leq 1$

- $\implies T(n) = \Theta(n^2)$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Mester módszer használata

## 2. eset

**Ha  $f(n) = \Theta(n^{\log_b a})$ , akkor**

$$\rightarrow T(n) = \Theta(n^{\log_b a} \lg n) (= \Theta(f(n) \lg n)).$$

- $T(n) = T(2n/3) + 1$ 
  - $a = 1, b = 3/2$  és  $f(n) = n$
  - $n^{\log_b a} = n^{\log_{3/2} 1} = n^0$
  - $f(n) = O(n^{\log_{3/2} 1})$ ,
  - $\implies T(n) = \Theta(n^{\log_{3/2} 1} \lg n) = \Theta(\lg n)$

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

A szimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

**Rekurzíván megadott  
függvények**

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Mester módszer használata

## 3. eset

Ha  $f(n) = \Omega(n^{\log_b a + \epsilon})$  és  $a \cdot f(n/b) \leq c \cdot f(n)$  valamely  $\epsilon > 0$  és  $c < 1$  állandóra és elég nagy  $n$ -re, akkor

$$\rightarrow T(n) = \Theta(f(n)).$$

- $T(n) = 3T(n/4) + n \lg n$ 
  - $a = 3, b = 4$  és  $f(n) = n \lg n$
  - $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$   $(\log_4 3 < 0.793)$
  - $f(n) = \Omega(n^{\log_4 3 + \epsilon})$ , ha  $0 < \epsilon < 0.2$
  - Már csak azt kell megmutatnunk nagy  $n$ -ekre, hogy:  
$$a \cdot f(n/b) \leq c \cdot f(n)$$
  - $a \cdot f(n/b) = 3(n/4) \ln(n/4) \leq (3/4)n \lg n = 0.75 \cdot f(n)$
  - $\Rightarrow T(n) = \Theta(f(n)) = \Theta(n \ln n)$

Függvények  
aszimptotikus  
viselkedése  
Kósa Márk  
Páновics János  
Szathmáry László  
Halász Gábor



## Függvények növekedése

Függvények aszimptotikus viselkedése  
Aszimptotikus jelölések  
képletekben  
Aszimptotikusan nem éles korlátok  
Függvények összehasonlítása  
Alapfüggvények

## Rekurzíván megadott függvények

A helyettesítő módszer  
Finomságok  
Buktatók  
Új változó bevezetése  
A rekurziós fa módszer  
Sejtés keresése  
Bizonyítás a helyettesítő módszerrel  
Mester módszer

A téTEL  
Amikor működik  
Amikor nem működik  
„Levezetés”

# Mester módszer használata

Amikor egyik eset sem működik

- $T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$

- $a = 2, b = 2$  és  $f(n) = n \lg n$

- $n^{\log_b a} = n^{\log_2 2} = n$

- $f(n) = \Omega(n^{\log_b a}) \implies$  csak a 3. eset jöhét szóba,
  - de nincs olyan  $\epsilon > 0$ , amivel teljesülne, hogy

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

- $f(n)$  „nagyobb” ugyan  $n^{\log_b a}$ -nél, de nem „polinomiálisan nagyobb”.

- $\implies$  Nem használható a mester téTEL.

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

„Levezetés”

# Keresés rendezett adatok között

## Lineáris keresés

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**function** LINEÁRIS-KERES(A, érték)

- ① **i**  $\leftarrow$  1
  - ② **while**  $i \leq$  méret(A) és  $A[i] <$  érték **do**
  - ③       *i*  $\leftarrow$  *i* + 1
  - ④ **end while**
  - ⑤ **if**  $i >$  méret(A) vagy  $A[i] >$  érték **then**
  - ⑥       KIVÉTEL "nincs ilyen értékű elem"
  - ⑦ **else**
  - ⑧       **return** *i*
  - ⑨ **end if**
- end procedure**

Legrosszabb eset: **while** ciklus tesztelése lefut  $n + 1$ -szer.

$$T(n) = \Theta(n)$$

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Keresés rendezett adatok között

## Lineáris keresés

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**function** LINEÁRIS-KERES(A, érték)

- ① **i**  $\leftarrow$  1
  - ② **while**  $i \leq$  méret(A) és  $A[i] <$  érték **do**
  - ③       *i*  $\leftarrow$  *i* + 1
  - ④ **end while**
  - ⑤ **if**  $i >$  méret(A) vagy  $A[i] >$  érték **then**
  - ⑥       KIVÉTEL "nincs ilyen értékű elem"
  - ⑦ **else**
  - ⑧       **return** *i*
  - ⑨ **end if**
- end procedure**

Legrosszabb eset: **while** ciklus tesztelése lefut  $n + 1$ -szer.

$$T(n) = \Theta(n)$$

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzívan megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések képletekben

Aszimptotikusan nem éles korlátok

Függvények összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A térel

Amikor működik

Amikor nem működik

„Levezetés”

# Keresés rendezett adatok között

## Bináris keresés

**function** BINÁRIS\_KERES(A, érték, alsó, felső)

① **if** alsó > felső **then**

②        KIVÉTEL "nincs ilyen értékű elem"

③ **end if**

④ középső  $\leftarrow \lfloor (\text{alsó} + \text{felső}) / 2 \rfloor$

⑤ **if** A[középső] = érték **then**

⑥        **return** középső

⑦ **else if** A[középső] > érték **then**

⑧        **return** BINÁRIS\_KERES(A, érték, alsó, középső - 1)

⑨ **else**

⑩        **return** BINÁRIS\_KERES(A, érték, középső + 1, felső)

⑪ **end if**

**end function**

- $T(n) = T(n/2) + 1 \longrightarrow$  talán használható a mester téTEL

- $a = 1, b = 2$  és  $f(n) = 1$

- $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$

- $f(n) = \Theta(n^{\log_2 1})$

- 2. estet
- $\Rightarrow T(n) = \Theta(n^{\log_2 1} \ln n) = \Theta(\ln n)$



Függvények  
növekedése

Függvények aszimptotikus viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Keresés rendezett adatok között

## Bináris keresés

```
function BINÁRIS_KERES(A, érték, alsó, felső)
    1 if alsó > felső then
        2 KIVÉTEL "nincs ilyen értékű elem"
    3 end if
    4 középső ← ⌊(alsó + felső) / 2⌋
    5 if A[középső] = érték then
        6 return középső
    7 else if A[középső] > érték then
        8 return BINÁRIS_KERES(A, érték, alsó, középső - 1)
    9 else
        10 return BINÁRIS_KERES(A, érték, középső + 1, felső)
    11 end if
end function
```

- $T(n) = T(n/2) + 1 \longrightarrow$  talán használható a mester téTEL
  - $a = 1, b = 2$  és  $f(n) = 1$
  - $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$
  - $f(n) = \Theta(n^{\log_2 1}) \longrightarrow 2.$  estet
- $\Rightarrow T(n) = \Theta(n^{\log_2 1} \ln n) = \Theta(\ln n)$

# Beszúró rendezés

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## procedure BESZÚRÓ-RENDEZÉS(A)

```
1 for j ← 2 to méret(A) do
  2   kulcs ← A[j]
  3   i ← j - 1
  4   while i ≥ 1 és A[i] > kulcs do
    5     A[i + 1] ← A[i]
    6     i ← i - 1
  7   end while
  8   A[i + 1] ← kulcs
  9 end for
end procedure
```

- Vegyük észre: a **while** ciklus lineáris keresést használ kulcs helyének meghatározására az A „elejében”.
- Lehetne ezt bináris keresésre cserélni?

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Beszúró rendezés

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## procedure BESZÚRÓ-RENDEZÉS(A)

```
1 for j ← 2 to méret(A) do
  2   kulcs ← A[j]
  3   i ← j - 1
  4   while i ≥ 1 és A[i] > kulcs do
    5     A[i + 1] ← A[i]
    6     i ← i - 1
  7   end while
  8   A[i + 1] ← kulcs
  9 end for
end procedure
```

- Vegyük észre: a **while** ciklus lineáris keresést használ kulcs helyének meghatározására az A „elejében”.
- Lehetne ezt bináris keresésre cserélni?

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# Beszúró rendezés

Függvények  
aszimptotikus  
viselkedése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## procedure BESZÚRÓ-RENDEZÉS(A)

```
1 for j ← 2 to méret(A) do
  2   kulcs ← A[j]
  3   i ← j - 1
  4   while i ≥ 1 és A[i] > kulcs do
    5     A[i + 1] ← A[i]
    6     i ← i - 1
  7   end while
  8   A[i + 1] ← kulcs
  9 end for
end procedure
```

- Vegyük észre: a **while** ciklus lineáris keresést használ kulcs helyének meghatározására az A „elejében”.
- Lehetne ezt bináris keresésre cserélni?

Függvények  
növekedése

Függvények aszimptotikus  
viselkedése

Aszimptotikus jelölések  
képletekben

Aszimptotikusan nem éles  
korlátok

Függvények  
összehasonlítása

Alapfüggvények

Rekurzíván megadott  
függvények

A helyettesítő módszer

Finomságok

Buktatók

Új változó bevezetése

A rekurziós fa módszer

Sejtés keresése

Bizonyítás a helyettesítő  
módszerrel

Mester módszer

A téTEL

Amikor működik

Amikor nem működik

„Levezetés”

# 3. előadás

## Adatszerkezetekkel kapcsolatos alapfogalmak

Reprezentáció és implementáció, algoritmusok, a halmaz és a multihalmaz, tömbök.

*Adatszerkezetek és algoritmusok előadás*

2018. február 13.

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Absztrakt  
adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű)  
tárolás

Szétszóró (láncolt) tárolás

Reprezentáció és  
implementáció

Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz

A multihalmaz

Asszociatív  
adatszerkezetek

A tömb

Háromszögátrixok

Dinamikus tömb

# Rendszer

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- Elemek: egyedek
- Tulajdonságok (statikus rész)
- Viselkedés (dinamikus rész)
- Kölcsönhatás
- Komplex rendszer
- Nyílt rendszer
- Dinamikus rendszer

Absztrakt  
adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű) tárólás

Szélcsőrt (láncolt) tárólás

Reprezentáció és implementáció

Algoritmusok

Algoritmusok megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz

A multihalmaz

Asszociatív  
adatszerkezetek

A tömb

Háromszög mátrixok

Dinamikus tömb

# Absztrakció, modellezés

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- Modellalkotás, absztrakció
- Adatmodell, eljárásmódell
- Adat, információ

Az adatelemek lehetnek **egyszerűek** (atomiak) és **összetettek**.  
Minden adatelem rendelkezik valamilyen **értékkel**.

Az adatelemek között jól meghatározott kapcsolatrendszer van. Az adatelemek és a közöttük lévő kapcsolatok definiálják a **logikai (absztrakt) adatszerkezetet**. Független hardvertől, szoftvertől.

**Fizikai adatszerkezet (társzerkezet)**: adatszerkezet az operatív tárban vagy periférián (háttértáron).

Absztrakt  
adatszerkezetek

Rendszerelmélet

**Absztrakció, modellalkotás**

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű) tárás

Szétszóró (láncolt) tárás

Reprezentáció és implementáció

Algoritmusok

Algoritmusok megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

Struktúra nélküli adatszerkezetek

A halma

A multihalma

**Asszociatív adatszerkezetek**

A tömb

Háromszög mátrixok

Dinamikus tömb



## Absztrakt adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

## Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű)  
tárolás

Szétszórít (láncolt) tárolás

Reprezentáció és  
implementáció

Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz

A multihalmaz

### Asszociatív adatszerkezetek

A tömb

Háromszög mátrixok

Dinamikus tömb

# Absztrakt adatszerkezetek osztályozása

Lehetséges csoportosítási szempontok:

- ① Változhat-e az adatszerkezet elemeinek száma?
  - statikus
  - dinamikus
- ② Milyen az adatszerkezet elemeinek a típusa?
  - homogén
  - heterogén
- ③ Milyen kapcsolatban állnak egymással az adatelemek az adatszerkezetben?

Egy homogén adatszerkezet lehet

- struktúra nélküli
- asszociatív
- szekvenciális
- hierarchikus
- hálós

A heterogén adatszerkezeteket nem csoportosítjuk ilyen szempont alapján.

# Absztrakt adatszerkezetekkel végezhető műveletek (Alapvető algoritmusok)

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## 1 Létrehozás

## 2 Módosítás

- bővítés
- törlés (fizikai, logikai)
- csere

## 3 Rendezés

## 4 Keresés

## 5 Elérés

## 6 Bejárás

## 7 Feldolgozás

Absztrakt  
adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű)  
tárolás

Szétszóró (láncolt) tárolás

Reprezentáció és  
implementáció

Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz

A multihalmaz

Asszociatív  
adatszerkezetek

A tömb

Háromszög mátrixok

Dinamikus tömb

# Ábrázolási (tárolási) módok

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Ábrázolás alatt az adatszerkezet memóriában való megjelenési formáját értjük. Ez minden adatszerkezet esetén lehet

- folytonos (vektorszerű)
- szétszórt (láncolt)

Az adatalemek számára tárhelyeket foglalunk a memóriában. Egy **tárhely** mindig egy bájtcsoportot jelent, amely egy adatalem értékét tárolja, illetve szerkezetleíró információkat is hordozhat.

Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek

## Ábrázolási módok

Folytonos (vektorszerű)  
tárolás  
Szétszórt (láncolt) tárolás  
Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halma  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb



## Folytonos (vektorszerű) tárolás

Egy tárhelyen egy adatelem értékét tároljuk. A tárhelyek a memóriában folytonos, **összefüggő** tárterületet alkotnak, a tárhelyek **mérete** azonos.

Előnye:

- **közvetlen elérés**, a kezdőcím és az egy adatelemhez tartozó tárhely méretének ismeretében
- a csere művelete könnyen megvalósítható
- hatékony rendező algoritmusok (pl. gyorsrendezés)
- hatékony kereső algoritmusok (pl. bináris keresés)

Hátránya:

- nem segíti a bővítés és a fizikai törlés műveletének végrehajtását

### Absztrakt adatszerkezetek

- Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok
- Folytonos (vektorszerű)  
tárolás
- Szétszóró (láncolt) tárolás  
Representáció és  
implementáció
- Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

- A tömb  
Háromszög mátrixok  
Dinamikus tömb



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás

### Szétszórt (láncolt) tárolás

Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

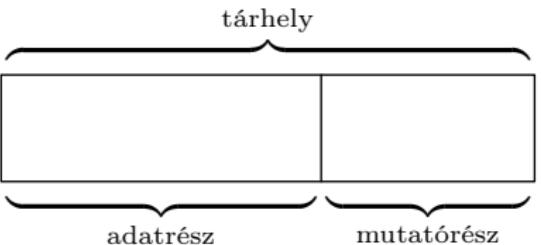
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

## Szétszórt (láncolt) tárolás

Egy tárhelyen egy adatelem értékét (**adatrész**) és legalább egy mutató értékét (**mutatórész**) tároljuk. A mutatók értékei memóriacímek lehetnek, amelyek megmondják az adatelem rákövetkezőinek tábeli helyét. A tárhelyek mérete nem szükségképpen azonos, elhelyezkedésük a memóriában tetszőleges.



### A szétszórt ábrázolási mód fajtái:

- egyirányban láncolt lista
- cirkuláris lista
- kétirányban láncolt lista
- multilista



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás

### Szétszóró (láncolt) tárolás

Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

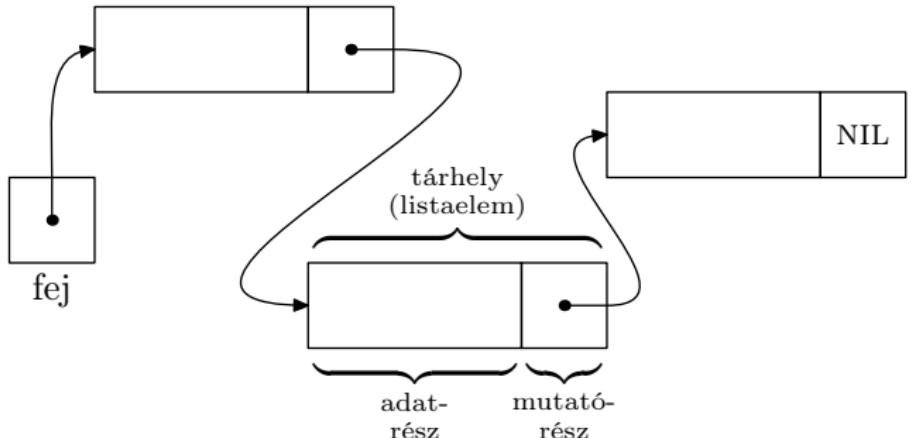
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

## Egyirányban láncolt lista

A tárhely (**listaelem**) az adatelem értékén kívül egy mutatót tartalmaz, amely a következő listaelem címét tartalmazza.



A láncolt lista első elemének tábeli címét egy mutató, a **fejmutató** tárolja.

A láncolt lista végét egy speciális érték, a **NIL** érték jelzi.  
Amennyiben a fejmutató tartalmazza ezt az értéket, akkor az  
egyirányban láncolt lista **üres**



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás

### Szétszóró (láncolt) tárolás

Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

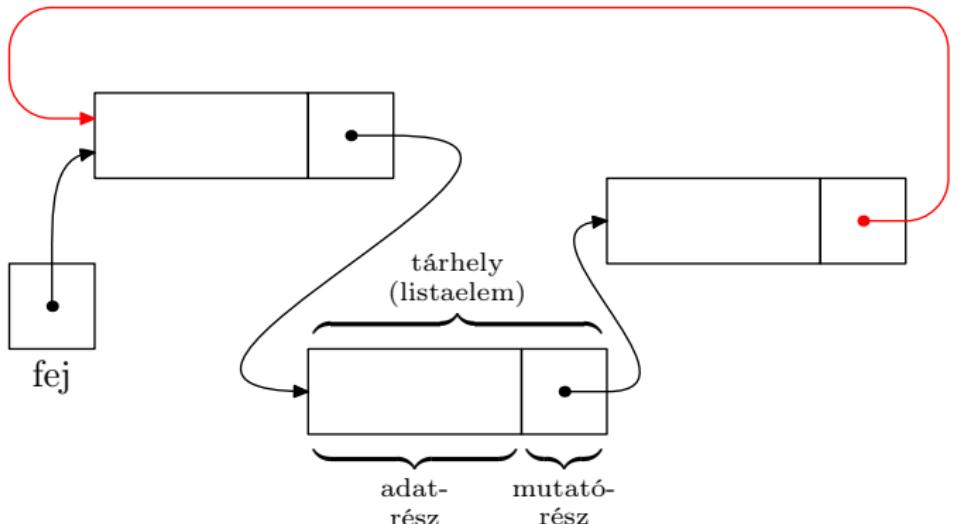
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Cirkuláris lista

Hasonló az egyirányban láncolt listához, ám itt egyik listaelem mutatórészére sem tartalmazhatja a NIL értéket: az „utolsó” listaelem mutatórészébe az „első” listaelem címe kerül.



A cirkuláris lista „első” elemének tábeli címét most is egy mutató, a **fejmutató** tárolja. Amennyiben a fejmutató a **NIL** értéket tartalmazza, akkor a cirkuláris lista **üres**.



## Absztrakt adatszerkezetek

- Rendszerelmélet
  - Absztrakció, modellalkotás
  - Absztrakt adatszerkezetek
  - Ábrázolási módok
  - Folytonos (vektorszerű) tárás
  - Szétzóró (láncolt) tárás
- Representáció és implementáció
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

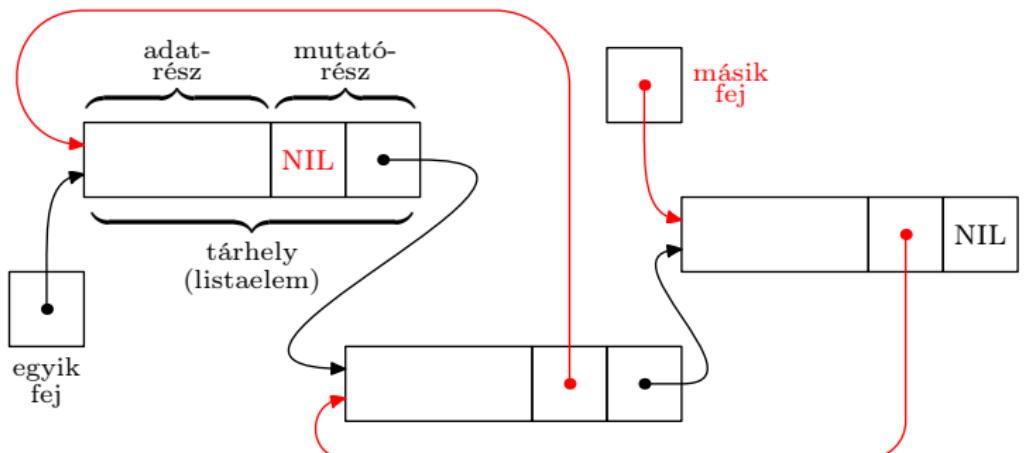
- A halmaz
- A multihalmaz

## Asszociatív adatszerkezetek

- A tömb
- Háromszögmátrixok
- Dinamikus tömb

# Kétirányban láncolt lista

Hasonló az egyirányban láncolt listához, ám itt minden listaelém mutatórészre **két részből** áll: az egyik mutató az adott listaelemet **megelőző**, a másik az adott listaelemet **követő** listaelémre mutat.



**Két lánc alakul ki, két fejmutatóval.** A fejmutatók a kétirányban láncolt lista **első** és **utolsó** elemére mutatnak. Ha minden fejmutató értéke **NIL**, akkor a kétirányban láncolt listának nincs egyetlen eleme sem, azaz **üres**.



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás

### Szötszörű (láncolt) tárolás

Representáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

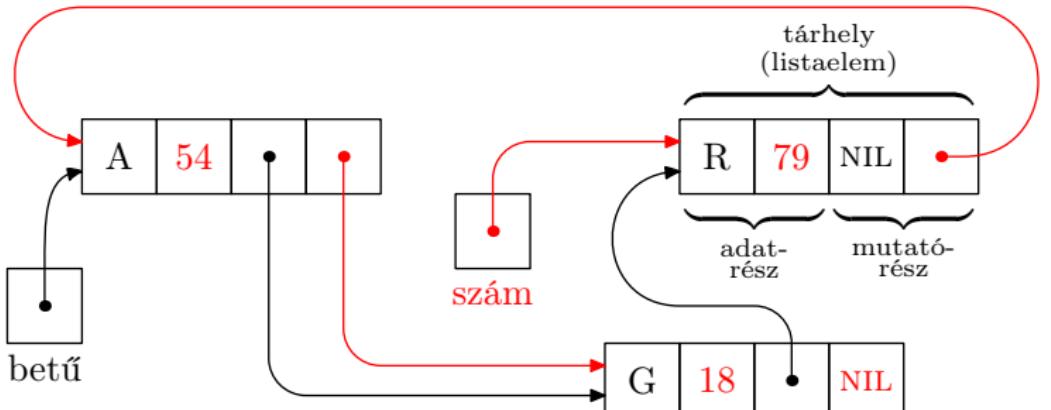
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

## Multilista (1)

Ebben a változatban a listaelemek adatrésze összetett. Az adatrész minden komponensére félépíthető egy egyirányban láncolt lista.



Annyi lánc alakítható ki, ahány komponensből áll az adatrész. minden lista külön fejmutatóval rendelkezik, és minden listaelem mindegyik láncban előfordul egyszer.



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás

### Szétszóró (láncolt) tárolás

Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

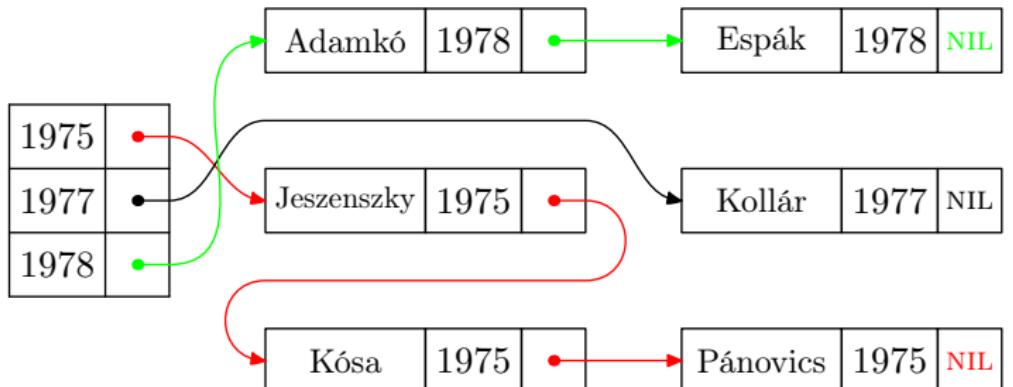
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

## Multilista (2)

Ebben a változatban a listaelemek adatrésze általában összetett. Az adatrész valamely komponensének értékeit figyelembe véve építjük föl az egyirányban láncolt listákat.

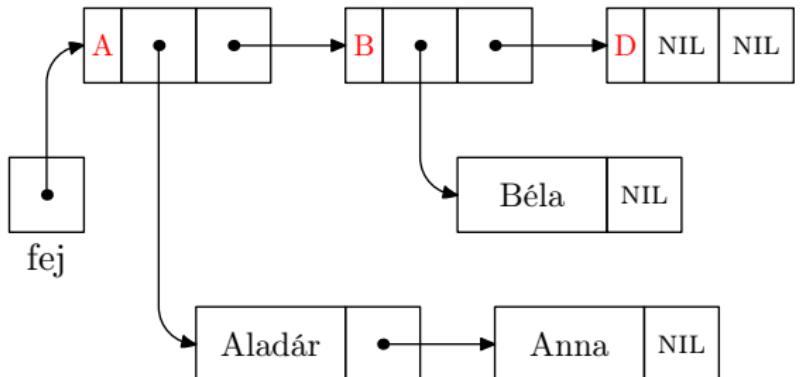


Annyi lánc alakul ki, ahány **különböző** értéket az adatrész adott komponense felvesz. minden lista külön fejmutatóval rendelkezik, és minden listaelem csak egy láncban szerepel, pontosan egyszer.



## Multilista (3)

Ebben a változatban a listaelemek tartalmaznak egy-egy **fejmutatót** is, melyek **újabb** láncolt listák első elemeire mutatnak.



Az allisták szerkezete **eltérhet** a főlista szerkezetétől.

### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás

### Szétszóró (láncolt) tárolás

- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszóró (láncolt) tárolás
- Reprezentáció és implementáció

- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

## Asszociatív adatszerkezetek

- A tömb
- Háromszögmátrixok
- Dinamikus tömb

# Reprezentáció és implementáció

Absztrakt adatszerkezetek grafikus (képi) megjelenítésénél használt jelölések:



: adatelem



: kapcsolat két adatelem között

Amikor egy absztrakt adatszerkezethez megadjuk a tárolási módját és a leképezését, akkor megadjuk az absztrakt adatszerkezet **reprezentációját**.

absztrakt adatszerkezet  $\xrightarrow{\text{leképezés}}$  ábrázolás

Ha a reprezentáció mellé megadjuk a műveletek megvalósítását (**algoritmusok**) is, akkor megadjuk az absztrakt adatszerkezet **implementációját**.

# Algoritmus fogalma

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Olyan eljárás (**elemi lépések sorozata**), melynek során a következők teljesülnek:

- jól meghatározott objektumokon jól meghatározott műveleteket végzünk
- minden lépés elvégzése után egyértelműen definiált helyzet áll elő
- véges sok lépés után végetér
- nem csak egy feladatra, hanem egy feladatosztály tagjaira érvényes

## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szélszörítő (láncolt) tárás  
Reprezentáció és implementáció  
**Algoritmusok**  
Algoritmusk megadásának módjai  
Algoritmusk építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Algoritmusok megadásának módjai

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Algoritmusokat a következő módokon lehet megadni:

- természetes (beszélt) emberi nyelven
- pontokba szedett természetes nyelvi „utasításokkal”
- folyamatábrával
- pszeudonyelvvel (lásd gyakorlaton)
- valamilyen programozási nyelven

Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszórít (láncolt) tárás  
Reprezentáció és implementáció  
Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei  
Gazdálkodás a memoriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb



## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Representáció és implementáció
- Algoritmusok

### Algoritmusok megadásának módjai

- Algoritmusok építőelemei
- Gazdálkodás a memóriával

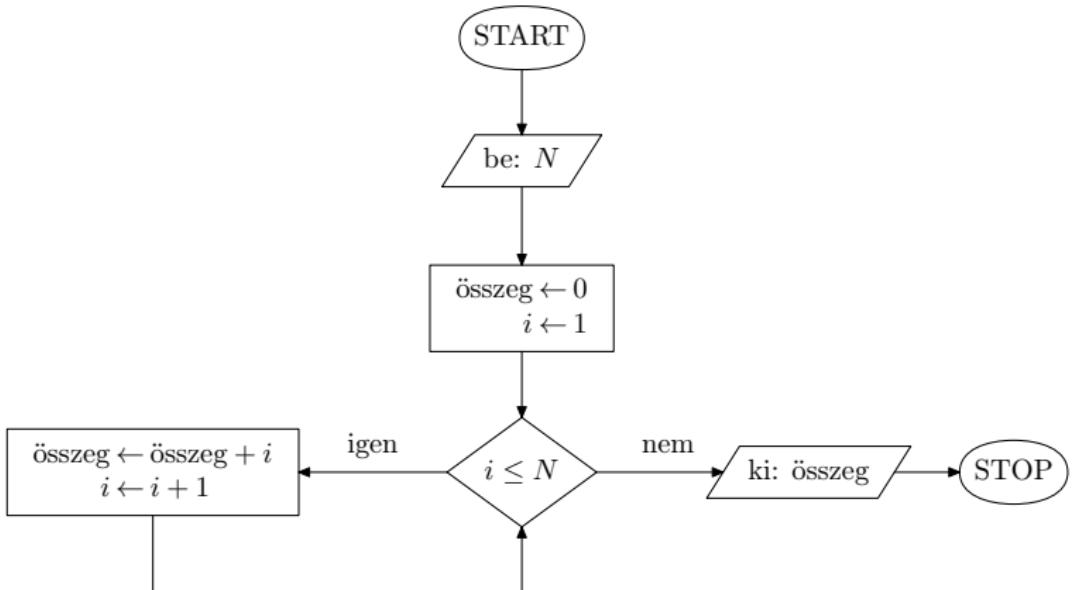
## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

## Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb

# Példa folyamatábrával





## Vezérlési szerkezetek

- ① szekvencia (utasítások végrehajtása a felírás sorrendjében)
- ② szelekció (elágazások)
- ③ iteráció (ciklusok)

Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai

Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

# Szabad helyek kezelése

## Adatszerkezetek szétszórt ábrázolás esetén

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az adatszerkezetek tárolásához **memória** kell, ami **véges**. A szabad helyekkel gázdálkodni kell! Lehetséges módszerek

- folytonos ábrázolás esetén:

- szabad tárhelyek összegyűjtése a lefoglalt tárterület végén (időigényes)
- szemetgyűjtögetés (garbage collection) elemmozgatással
- minden tárhelyhez hozzárendelünk egy bitet, ami a foglaltságot jelzi (nincs elemmozgatás)

- szétszórt ábrázolás esetén:

- szabad helyek láncolt lista (probléma: különböző méretű tárhelyek)
- szemetgyűjtögetés (garbage collection) a szabad helyek láncolásával
- a szabad helyek nyilvántartása bitvektor segítségével

Absztrakt  
adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű) tárás

Szétszórt (láncolt) tárás

Reprezentáció és implementáció

Algoritmusok

Algoritmusok megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memoriával

Struktúra nélküli  
adatszerkezetek

A halmasz

A multihalmasz

Asszociatív  
adatszerkezetek

A tömb

Háromszög mátrixok

Dinamikus tömb

## Absztrakt adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű)  
tárolás

Szétszórít (láncoolt) tárolás

Reprezentáció és

implementáció

Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz

A multihalmaz

## Asszociatív adatszerkezetek

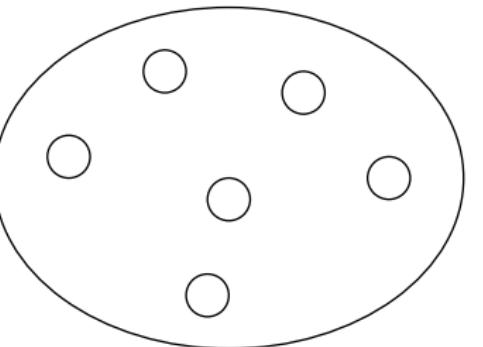
A tömb

Háromszögmátrixok

Dinamikus tömb

# Halmaz és multihalmaz

A halmaz és a multihalmaz **struktúra nélküli, homogén** és **dinamikus** adatszerkezetek.



A halmaz minden eleme különböző. A multihalmazban előfordulhatnak azonos elemek is.

Mindkét adatszerkezetre igaz, hogy az adatszerkezetben lévő elemek között **nincs kapcsolat** (ezért struktúra nélküli adatszerkezetek).



A **halmaz** adatszerkezet a matematikai halmaz fogalom megjelenése az adatszerkezetek szintjén. Mindig **véges** – ennyiben nem felel meg teljesen a matematikai halmaz fogalmának.

## A halmaz alapműveletei

- **eleme**,  $\in$ : megmondja, hogy egy adatelem benne van-e a halmazban vagy sem
- **unió**,  $\cup$ : két halmaz unióját adja
- **metszet**,  $\cap$ : két halmaz metszetét adja
- **különbség**,  $\setminus$ : két halmaz különbségét adja

### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárólás  
Szétszórít (láncolt) tárólás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszög mátrixok  
Dinamikus tömb



Az adatszerkezetekkel végezhető hagyományos műveletek megvalósítása halmazok esetén:

- **Létrehozás** kétféleképpen:
  - explicit módon, a halmaz elemeinek felsorolásával (esetleg üresen)
  - egy predikátum segítségével
- **Bővítés** unióképzéssel
- **Törlés** csak fizikai, különbségképzéssel
- **Csere** nincs
- **Rendezés, keresés, elérés, bejárás** nem értelmezettek
- **Feldolgozás** a halmaz alapműveleteinek a segítségével

## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

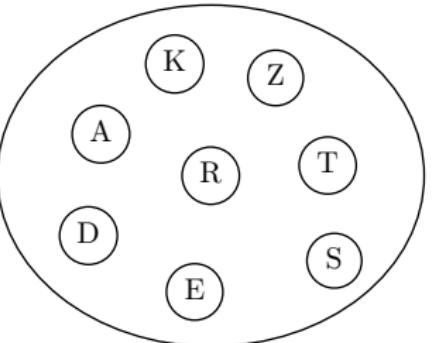
## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszög mátrixok  
Dinamikus tömb



# A halmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szélsőről (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

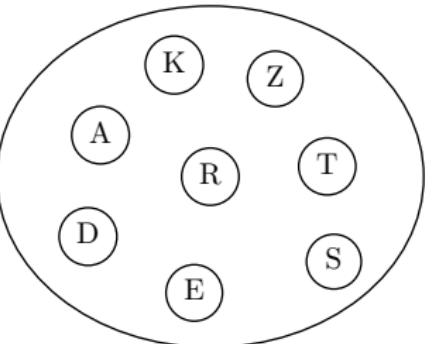
## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz
- Asszociatív adatszerkezetek
- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## A halmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A halmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy egy bit méretű tárterületet.

### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz
- Asszociatív adatszerkezetek
- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## Absztrakt adatszerkezetek

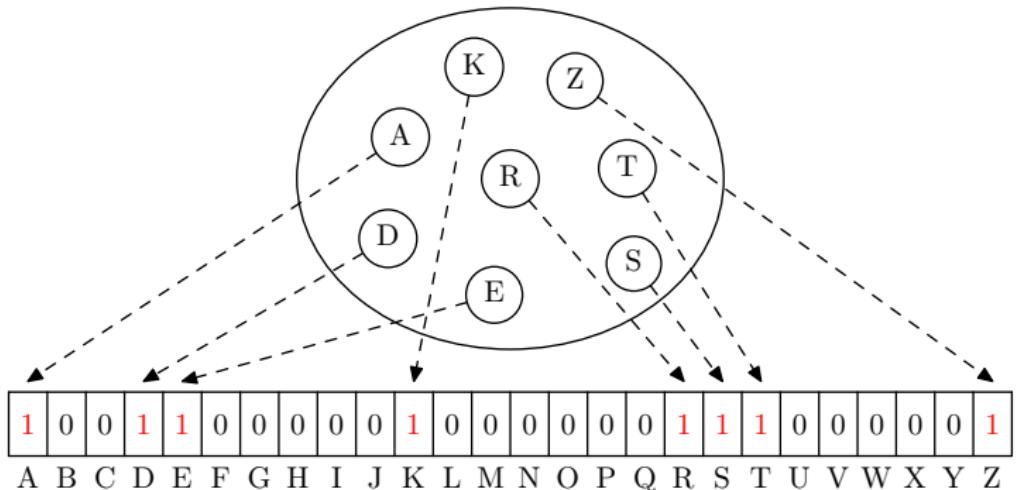
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szárszót (láncolt) tárás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A halmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



A halmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy egy bit méretű tárterületet. Az adott értékű adatelemhez tartozó bit fogja jelezni, hogy az adatelem benne van-e a halmazban (**1**) vagy sem (**0**).

# A halmaz adatszerkezet implementációja

Folytonos reprezentáció esetén a halmaz alapműveleteinek megvalósítása visszavezethető egyszerű bitműveletekre:

## Unióképzés

$$x \in A \cup B \Leftrightarrow x \in A \vee x \in B$$

## Metszetképzés

$$x \in A \cap B \Leftrightarrow x \in A \wedge x \in B$$

## Különbségképzés

$$x \in A \setminus B \Leftrightarrow x \in A \wedge x \notin B$$

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárólás  
Szétszórít (láncolt) tárólás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszög mátrixok  
Dinamikus tömb



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszóró (láncolt) tárás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A multihalmaz adatszerkezet

A **multihalmaz** abban különbözik a halmaztól, hogy megengedi az adatelemek ismétlődését, benne több azonos értékű elem is előfordulhat.

## A multihalmaz alapműveletei

- **eleme**,  $\in$ : megmondja, hogy egy adatelem benne van-e a multihalmazban vagy sem
- **unió**,  $\cup$ : két multihalmaz unióját adja
- **metszet**,  $\cap$ : két multihalmaz metszetét adja
- **különbség**,  $\setminus$ : két multihalmaz különbségét adja

Multihalmazoknál az adatszerkezetekkel végezhető hagyományos műveletek megvalósítása hasonló a halmazokéhoz (lásd ott). A multihalmaz **feldolgozása** a multihalmaz alapműveleteinek a segítségével történik.



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

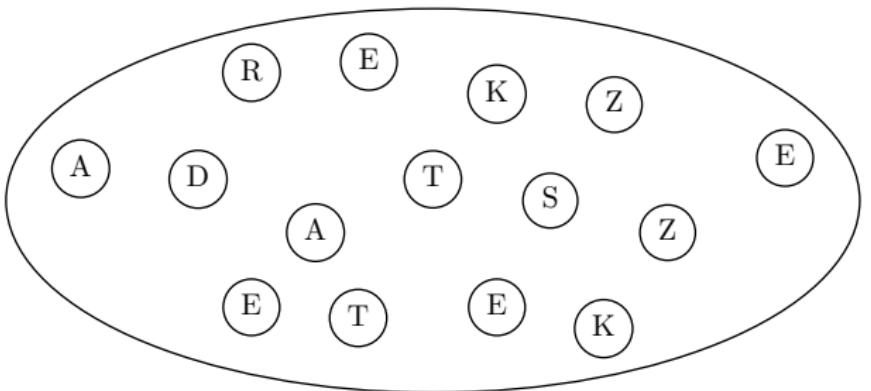
A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

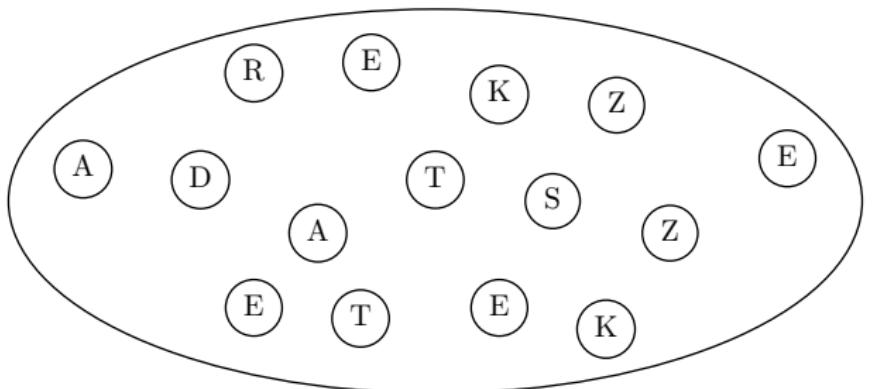
A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

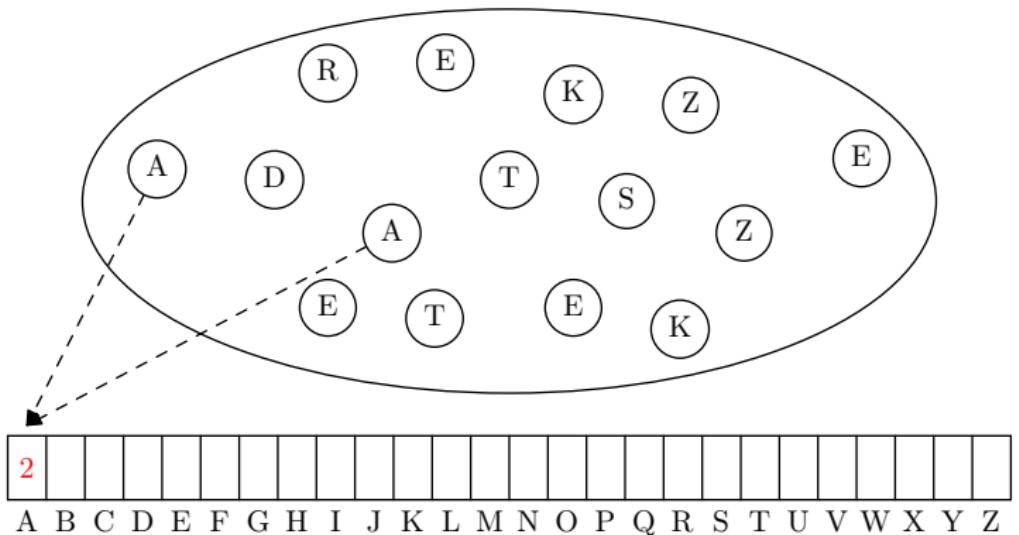
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

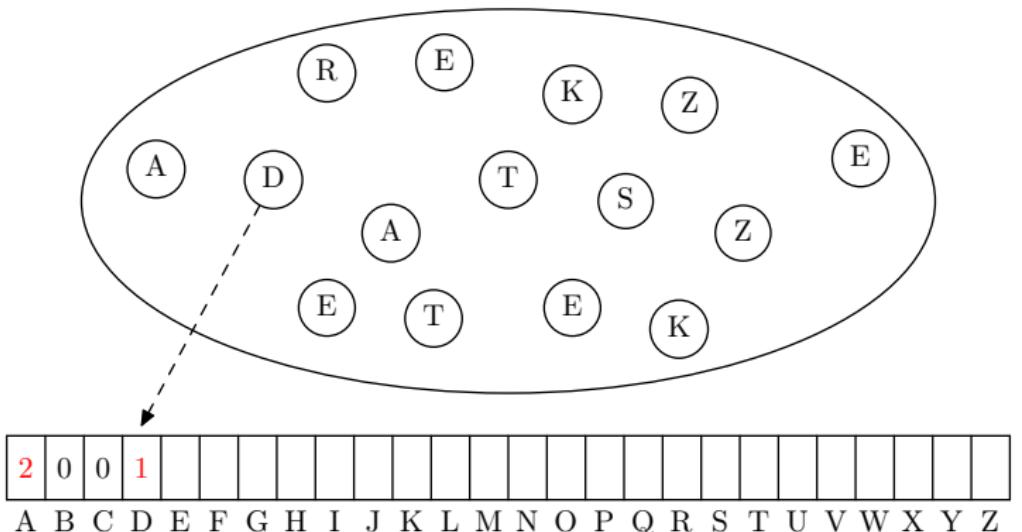
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszög mátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

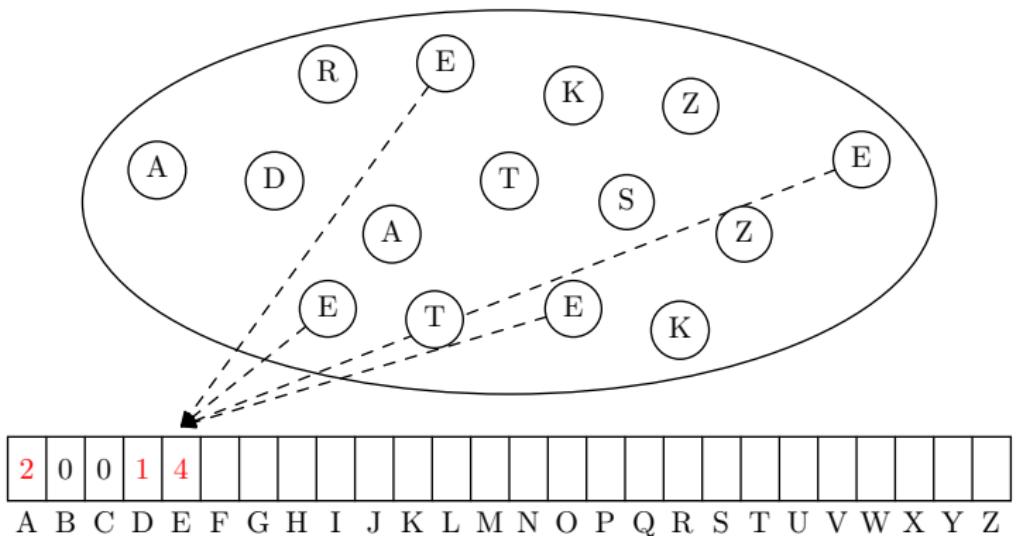
A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



## Absztrakt adatszerkezetek

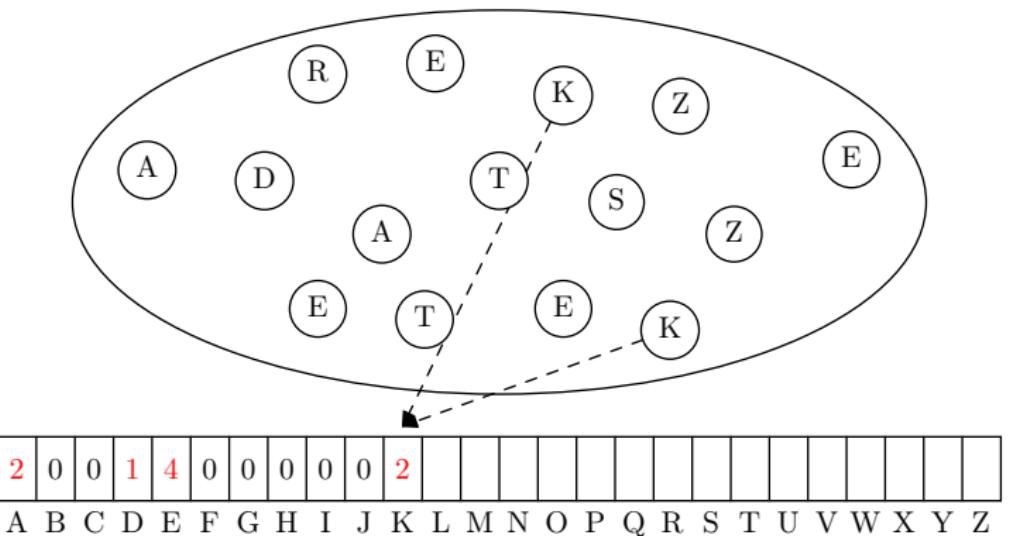
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

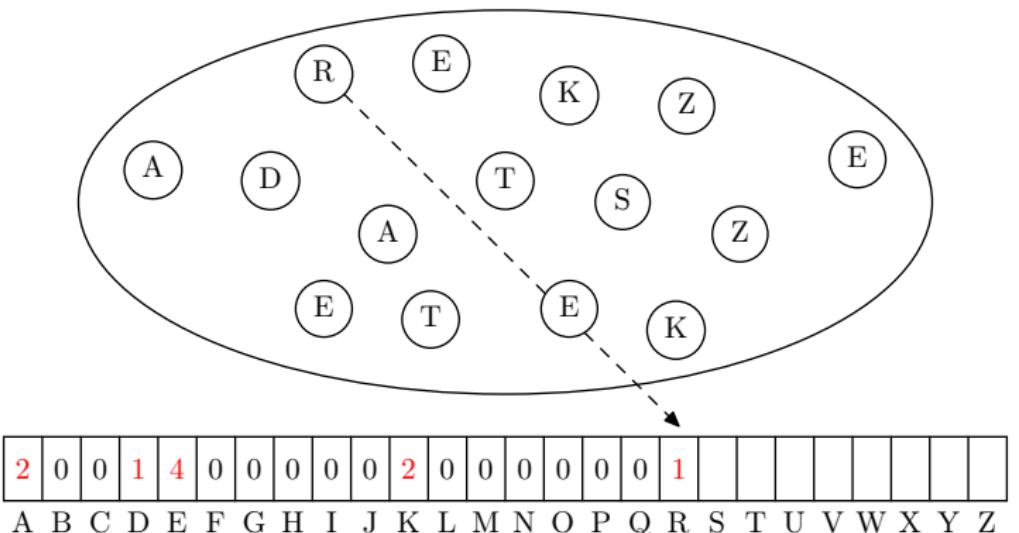
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



## Absztrakt adatszerkezetek

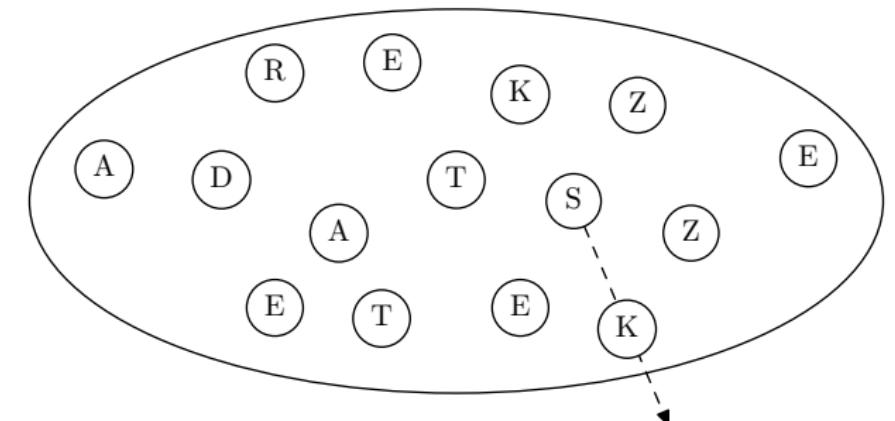
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

# A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



2	0	0	1	4	0	0	0	0	2	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

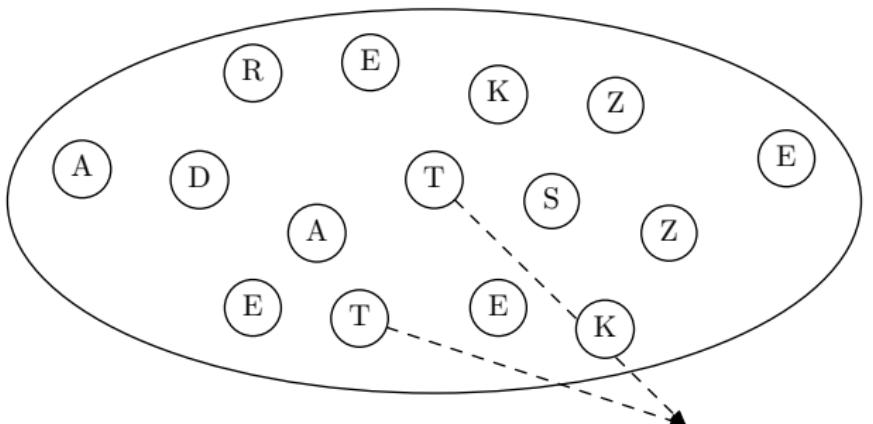
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



### Absztrakt adatszerkezetek

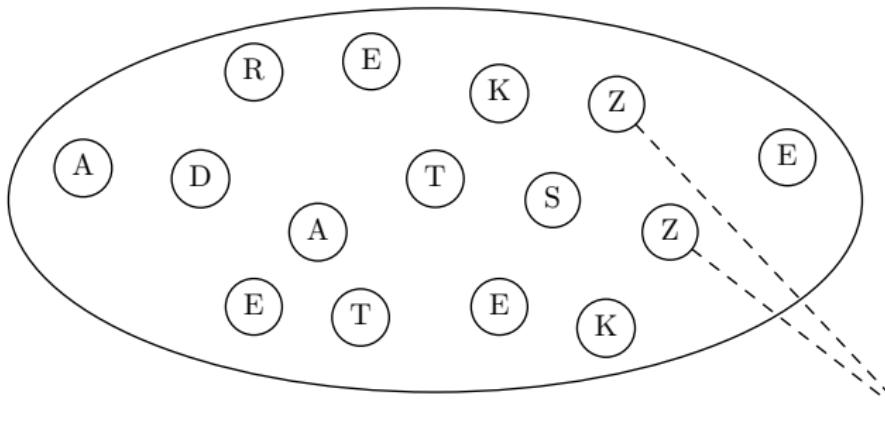
Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz  
Asszociatív  
adatszerkezetek  
A tömb  
Háromszögmátrixok  
Dinamikus tömb

## A multihalmaz adatszerkezet reprezentációja

Klasszikus reprezentációja folytonosan, **karakterisztikus függvény** segítségével történik.



2	0	0	1	4	0	0	0	0	2	0	0	0	0	0	1	1	2	0	0	0	0	2	
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Z

- A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.
  - általában 1 bájtot
- A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.



# A multihalmaz adatszerkezet implementációja

Folytonos reprezentáció esetén a multihalmaz alapműveleteinek megvalósítása visszavezethető egyszerű aritmetikai (számtani) műveletekre:

## Unióképzés

$$x A \cup B\text{-ben} = x A\text{-ban} + x B\text{-ben}$$

## Metszetképzés

$$x A \cap B\text{-ben} = \min\{x A\text{-ban}, x B\text{-ben}\}$$

## Különbséggépzés

$$x A \setminus B\text{-ben} = \max\{0, x A\text{-ban} - x B\text{-ben}\}$$



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszórít (láncolt) tárás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

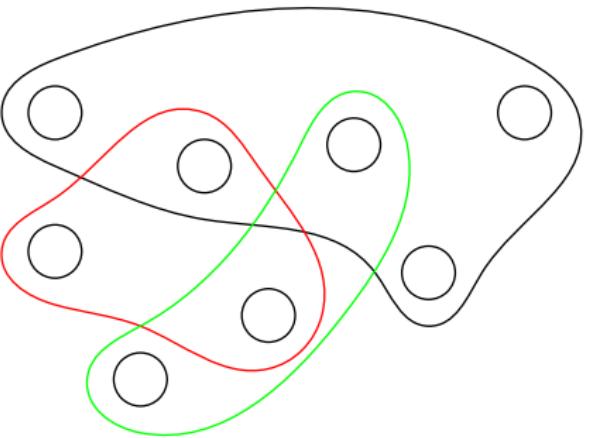
A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Asszociatív adatszerkezetek

Az asszociatív adatszerkezetek olyan adatszerkezetek, amelyekből bizonyos adott feltételeknek eleget tevő részhalmazokat választhatunk ki. A legfontosabb művelet tehát a részhalmaz kiválasztásának, a **részhalmazképzésnek** a művelete.



A részhalmazok – ahogy az ábrán is látható – **átfedhetik** egymást. Egyes esetekben a részhalmazok **egyeleműek**, máskor **akárhány** eleműek lehetnek.



## A tömb adatszerkezet

**Statikus, homogén** és **asszociatív** adatszerkezet. A felépítése definiálja: benne az adatalemek egymáshoz viszonyított helyzete a lényeges.

A tömb bármelyik eleme **egész számok sorozatán** keresztül érhető el. minden adatalemhez különböző egészszám-sorozat tartozik, így az asszociativitást biztosító részhalmazok **egyeleműek** és **diszjunktak**. A számsorozat számait **indexeknek** nevezzük, segítségükkel tudjuk az adatalemet kiválasztani. Az indexek darabszámát a tömb **dimenziójának** hívjuk.

Ha más nem mondunk, a tömb elemeinek az indexelése mindegyik dimenzióban 1-től indul.

### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszóró (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

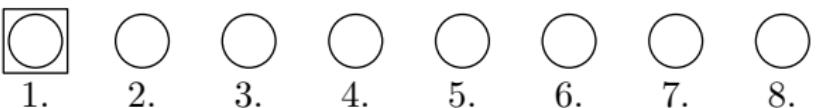
### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb

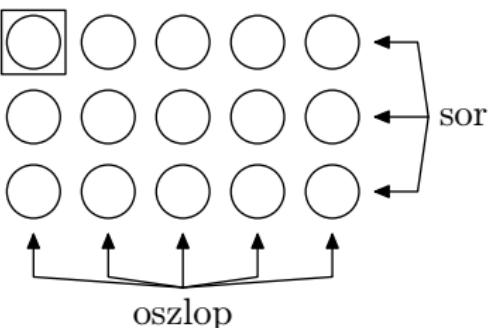


## A tömb adatszerkezet

A legegyszerűbb eset: egydimenziós tömb (**vektor**<sup>1</sup>).



Kétdimenziós tömb (**mátrix**).



Léteznek magasabb dimenziójú tömbök is. A dimenziók száma tetszőlegesen nagy lehet, de mindig **véges**.

<sup>1</sup> A vektor szó minden egyéb jelző nélküli használatakor statikus, egydimenziós tömbre gondolunk.

### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncoolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

### Asszociatív adatszerkezetek

- A tömb
- Háromszögmátrixok
- Dinamikus tömb

# Tömbökkel végezhető műveletek

- **Létrehozás:** rögzítjük a dimenziók számát és az indexartományokat. Ezzel egyben meghatározzuk a tömb elemszámát is. A szerkezet kialakításával párhuzamosan elemeket is elhelyezhetünk a tömbben.
- **Bővítés:** nincs, ugyanis a tömb **statikus**.
- **Csere:**
  - bármely (létező) elem értékét felülírhatjuk egy új értékkel
  - elhelyezhetünk elemet oda, ahová a létrehozáskor nem tettünk
- **Törlés:** csak logikai.
- **Elérés:** az adatelemek elérése **közvetlen**, az indexek segítségével.
- **Rendezés:** egydimenziós tömbök esetén értelmezhető, ott bármelyik rendezési algoritmus alkalmazható.
- **Keresés:** reprezentációfüggő művelet, egydimenziós tömbök esetén nagy a jelentősége, ott bármelyik keresési algoritmus alkalmazható.
- **Bejárás:** többdimenziós tömbök esetén reprezentációfüggő művelet (lásd később).
- A **feldolgozás** alapja a közvetlen elérés.

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szelezőírt (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb



## Tömbök folytonos reprezentációja

Az  $A[s..t]$  egydimenziós tömb leképezése:



- A tároláshoz szükséges tárterület mérete:  $\ell \cdot (t - s + 1)$  bájt, ahol  $\ell$  az egy adatelem tárolásához szükséges tárhely mérete.
- Ha ismerjük a tárterület kezdőcímét ( $K$ ), akkor a következő **címfüggvény** segítségével bármely elem tárbeli címe meghatározható:

$$\text{az } i \text{ indexű elem címe} = K + \ell \cdot (i - s)$$

### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszórít (láncoolt) tárás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

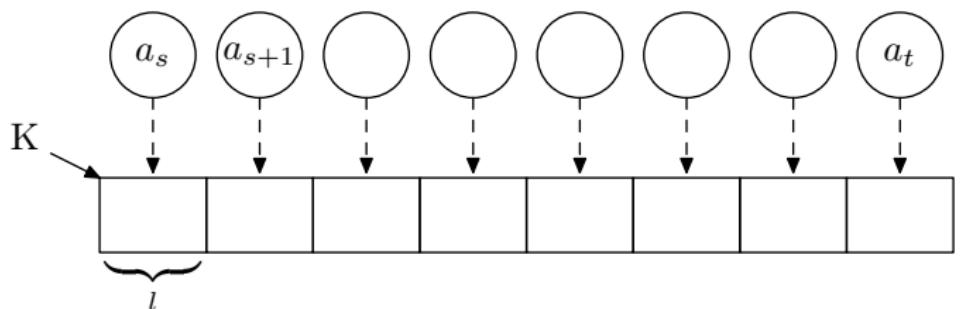


### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszóró (láncoolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával
- Struktúra nélküli adatszerkezetek
- A halmaz
- A multihalmaz
- Asszociatív adatszerkezetek
- A tömb
- Háromszögmátrixok
- Dinamikus tömb

## Tömbök folytonos reprezentációja

Az  $A[s..t]$  egydimenziós tömb leképezése:



- A tároláshoz szükséges tárterület mérete:  $\ell \cdot (t - s + 1)$  bájt, ahol  $\ell$  az egy adatelem tárolásához szükséges tárhely mérete.
- Ha ismerjük a tárterület kezdőcímét ( $K$ ), akkor a következő **címfüggvény** segítségével bármely elem tábeli címe meghatározható:

$$\text{az } i \text{ indexű elem címe} = K + \ell \cdot (i - s)$$

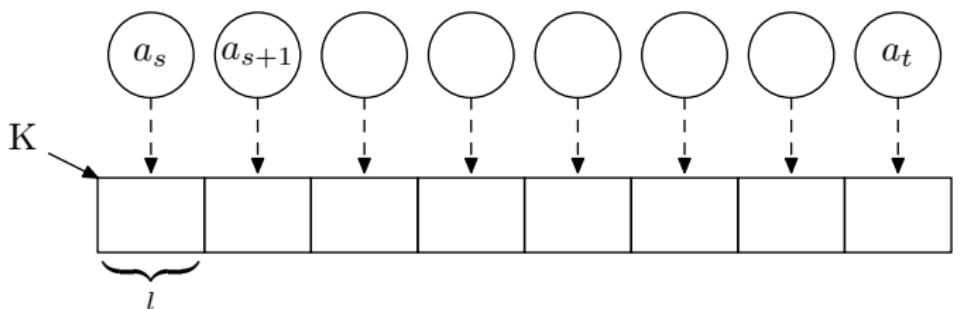


## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszóró (láncoolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával
- Struktúra nélküli adatszerkezetek
- A halmaz
- A multihalmaz
- Asszociatív adatszerkezetek
- A tömb
- Háromszögmátrixok
- Dinamikus tömb

# Tömbök folytonos reprezentációja

Az  $A[s..t]$  egydimenziós tömb leképezése:



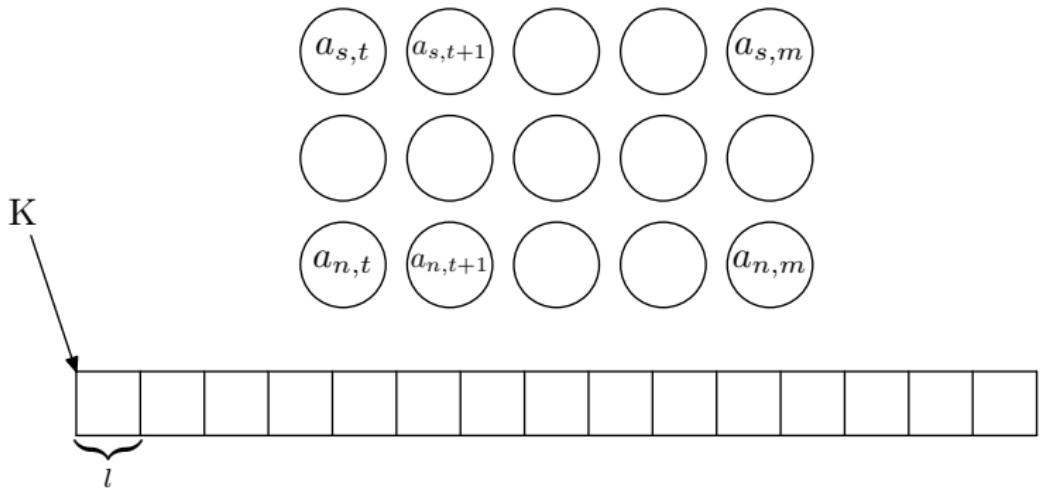
- A tároláshoz szükséges tárterület mérete:  $\ell \cdot (t - s + 1)$  bájt, ahol  $\ell$  az egy adatelem tárolásához szükséges tárhely mérete.
- Ha ismerjük a tárterület kezdőcímét ( $K$ ), akkor a következő **címfüggvény** segítségével bármely elem tárbeli címe meghatározható:

$$\text{az } i \text{ indexű elem címe} = K + \ell \cdot (i - s)$$



## Tömbök folytonos reprezentációja

Az  $A[s..n, t..m]$  kétdimenziós tömb leképezése történhet sorfolytonosan (lásd az ábrán) vagy oszlopfolytonosan.



### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

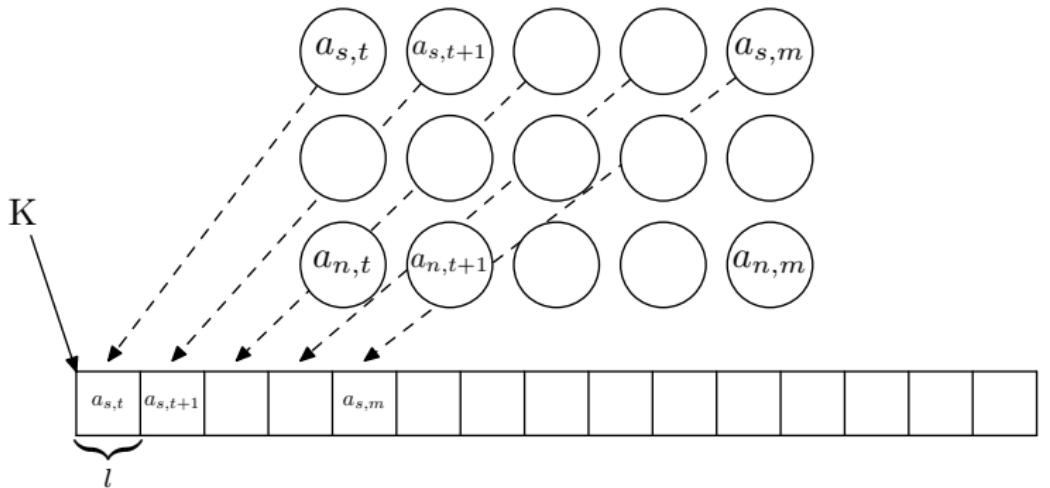
### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## Tömbök folytonos reprezentációja

Az  $A[s..n, t..m]$  kétdimenziós tömb leképezése történhet sorfolytonosan (lásd az ábrán) vagy oszlopfolytonosan.



### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

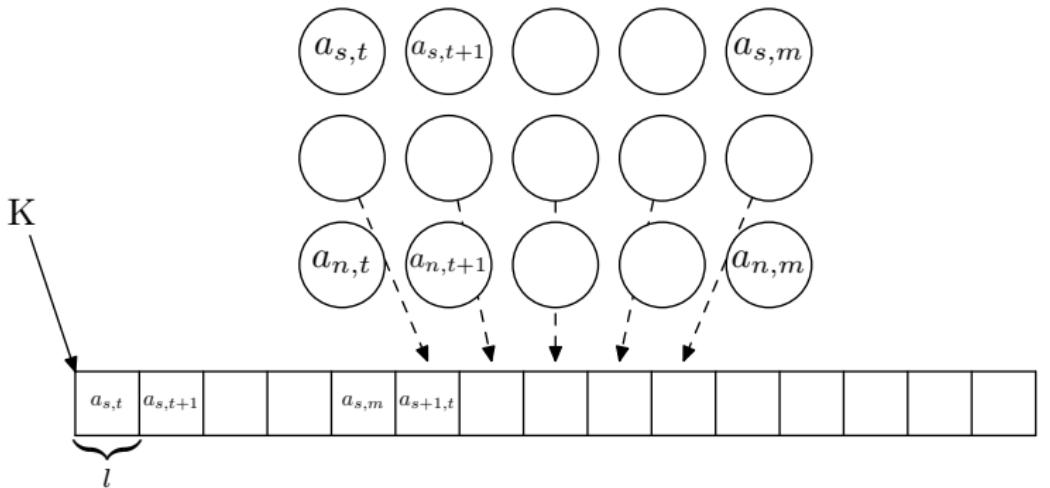
### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



# Tömbök folytonos reprezentációja

Az  $A[s..n, t..m]$  kétdimenziós tömb leképezése történhet sorfolytonosan (lásd az ábrán) vagy oszlopfolytonosan.



## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

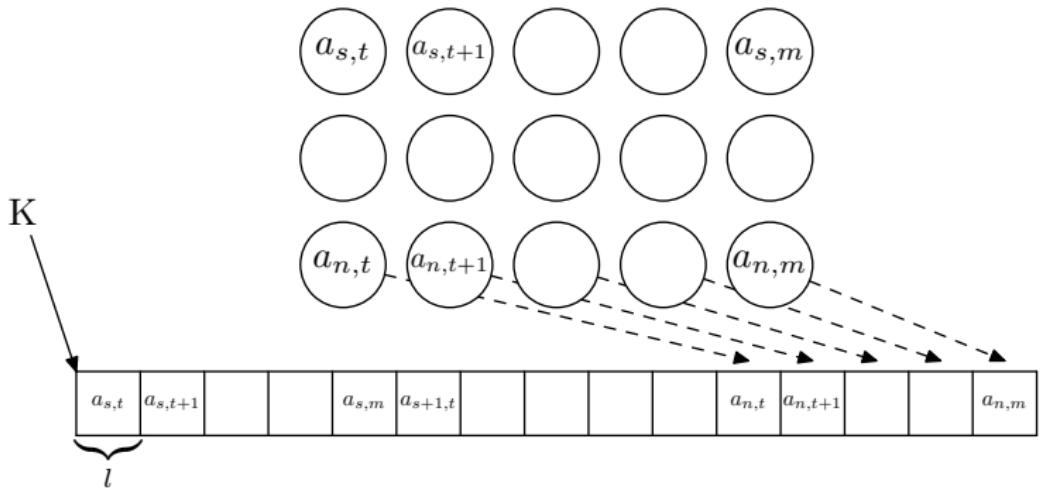
## Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## Tömbök folytonos reprezentációja

Az  $A[s..n, t..m]$  kétdimenziós tömb leképezése történhet sorfolytonosan (lásd az ábrán) vagy oszlopfolytonosan.



### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

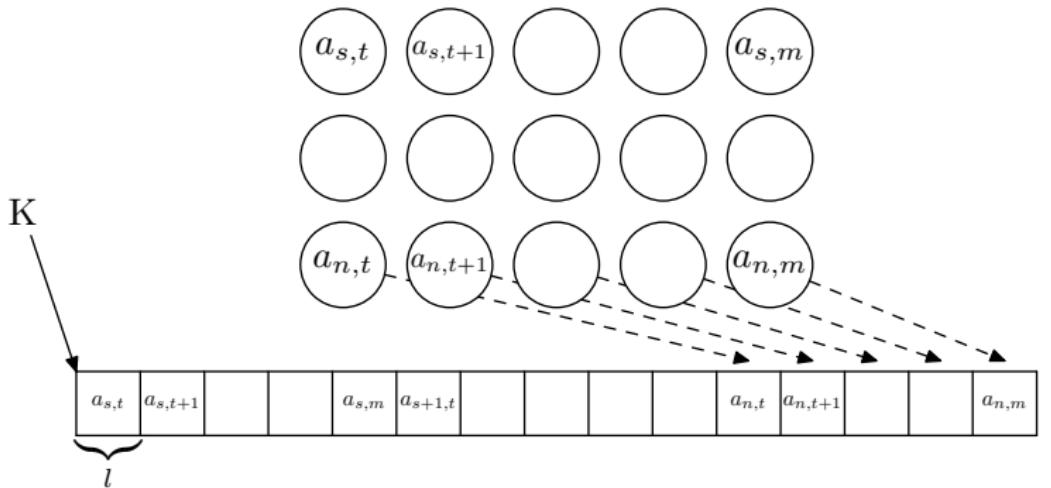
### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



## Tömbök folytonos reprezentációja

Az  $A[s..n, t..m]$  kétdimenziós tömb leképezése történhet **sorfolytonosan** (lásd az ábrán) vagy **oszlopfolytonosan**.



Sorfülytonos tárolás esetén ha ismerjük a tárterület kezdőcímét ( $K$ ), akkor a következő **címfüggvény** segítségével bármely elem tábeli címe meghatározható:

$$\text{az } (i, j) \text{ indexű elem címe} = K + \ell \cdot (i - s) \cdot (m - t + 1) + \ell \cdot (j - t)$$

### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb

# Tömbök folytonos reprezentációja

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Az  $A[s_1..n_1, s_2..n_2, \dots, s_d..n_d]$   $d$  dimenziós tömb sorfolytonos leképezése esetén a **címfüggvény** a következő ( $K$  továbbra is a tárterület kezdőcímét,  $\ell$  pedig az egy adatelem tárolásához szükséges tárhely méretét jelöli):

az  $(x_1, x_2, \dots, x_d)$  indexű elem címe =

$$= K + \ell \cdot \sum_{i=1}^d \left( (x_i - s_i) \cdot \prod_{j=i+1}^d (n_j - s_j + 1) \right)$$

## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórított (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

## Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárólás  
Szétszórít (láncolt) tárólás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

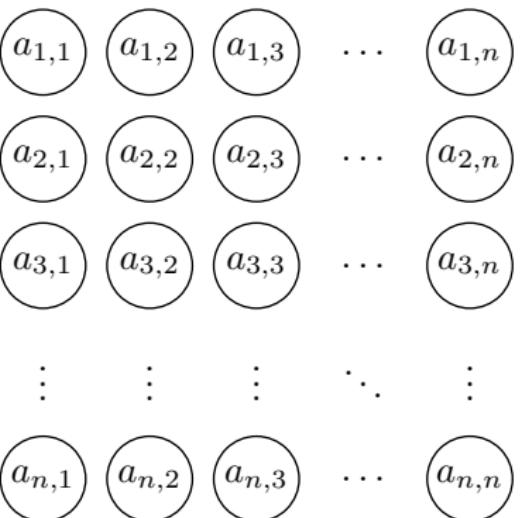
A tömb

Háromszögmátrixok

Dinamikus tömb

# Háromszögmátrixok

A háromszögmátrixok **négyzetes** (kvadratikus) mátrixok.



Kétfajta háromszögmátrixot szoktunk megkülönböztetni:

- a felsőháromszög-mátrixot és
- az alsóháromszög-mátrixot.

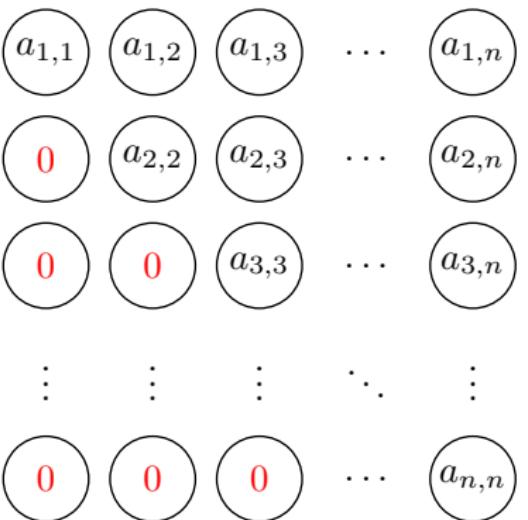
# Háromszögmátrixok

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A háromszögmátrixok **négyzetes** (kvadratikus) mátrixok.



Az olyan négyzetes mátrixot, amelynek **főátlója alatt** csupa 0 elem található, **felsőháromszög-mátrixnak** nevezzük.

Absztrakt  
adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű)  
tárolás

Szétszórít (láncoolt) tárolás

Reprezentáció és  
implementáció

Algoritmusok

Algoritmusok  
megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz

A multihalmaz

Asszociatív  
adatszerkezetek

A tömb

Háromszögmátrixok

Dinamikus tömb

# Háromszögmátrixok

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A háromszögmátrixok **négyzetes** (kvadratikus) mátrixok.

$a_{1,1}$	0	0	...	0
$a_{2,1}$	$a_{2,2}$	0	...	0
$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	...	0
⋮	⋮	⋮	⋮	⋮
$a_{n,1}$	$a_{n,2}$	$a_{n,3}$	...	$a_{n,n}$

Ha a négyzetes mátrix **főátlójá fölött** lévő elemek mindegyikének értéke 0, akkor **alsóháromszög-mátrixról** beszélünk.

## Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárólás
- Szétszórít (láncolt) tárólás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

## Asszociatív adatszerkezetek

- A tömb

## Háromszögmátrixok

- Dinamikus tömb

# Háromszögmátrixok folytonos reprezentációja

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A négyzetes mátrixokkal szemben, ahol az értékes elemek száma  $n^2$ , a háromszögmátrixoknál az értékes elemek száma csupán

$$\frac{n \cdot (n + 1)}{2}.$$

Az értékes elemeket emiatt – sor- vagy oszlopfolytonosan – egy  $\frac{n \cdot (n + 1)}{2}$  elemű  $V$  vektorra szoktuk leképezni.

## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncoolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memoriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötte elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:

$$\begin{matrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ 0 & 0 & a_{3,3} & \dots & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{n,n} \end{matrix}$$



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszórít (láncoolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

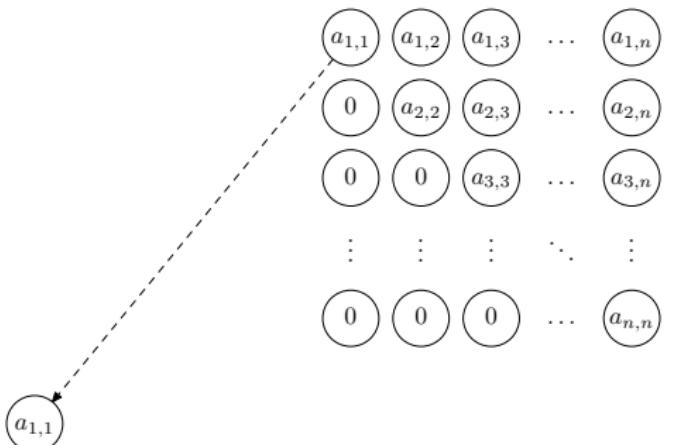
A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Felsőháromszög-mátrixok folytonos reprezentációja

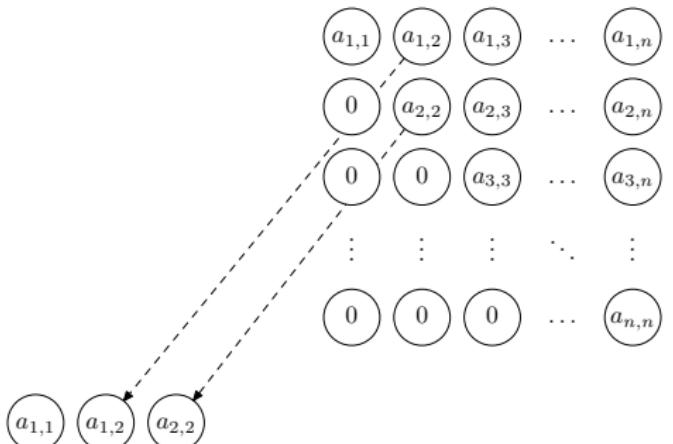
A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötte elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:





## Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötté elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:



### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

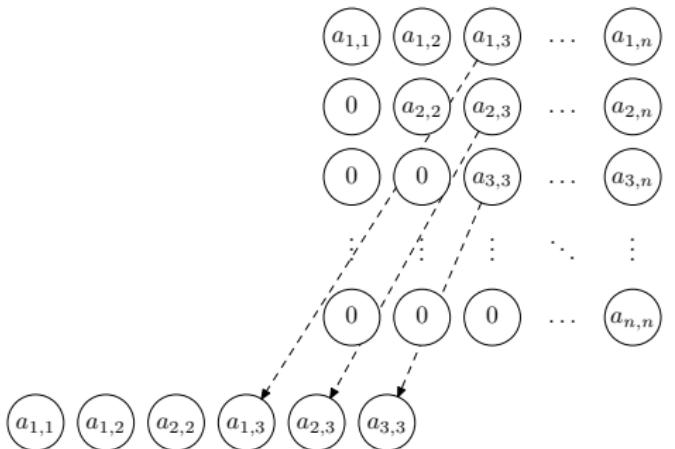
### Asszociatív adatszerkezetek

- A tömb
- Háromszögmátrixok
- Dinamikus tömb



## Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötté elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:



### Absztrakt adatszerkezetek

- Rendszerelmélet
- Absztrakció, modellalkotás
- Absztrakt adatszerkezetek
- Ábrázolási módok
- Folytonos (vektorszerű) tárolás
- Szétszórít (láncolt) tárolás
- Reprezentáció és implementáció
- Algoritmusok
- Algoritmusok megadásának módjai
- Algoritmusok építőelemei
- Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

- A halmaz
- A multihalmaz

### Asszociatív adatszerkezetek

- A tömb
- Háromszög mátrixok
- Dinamikus tömb



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszóró (láncoolt) tárás  
Representáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

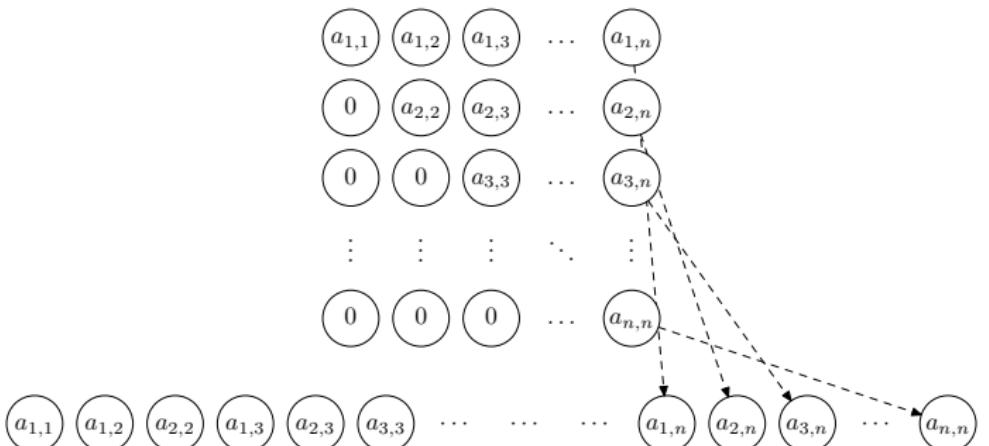
A tömb

### Háromszögmátrixok

Dinamikus tömb

# Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötte elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:





## Absztrakt adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű) tárás

Szétszóró (láncolt) tárás

Reprezentáció és implementáció

Algoritmusok

Algoritmusok megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz

A multihalmaz

## Asszociatív adatszerkezetek

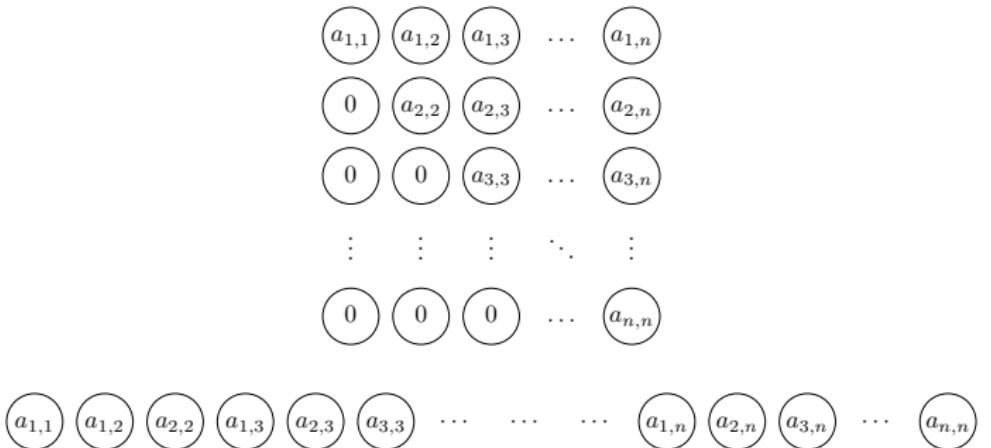
A tömb

## Háromszögmátrixok

Dinamikus tömb

# Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölött elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:





### Absztrakt adatszerkezetek

Rendszerelmélet

Absztrakció, modellalkotás

Absztrakt adatszerkezetek

Ábrázolási módok

Folytonos (vektorszerű) tárás

Szétszóró (láncoolt) tárás

Reprezentáció és implementáció

Algoritmusok

Algoritmusok megadásának módjai

Algoritmusok építőelemei

Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz

A multihalmaz

### Asszociatív adatszerkezetek

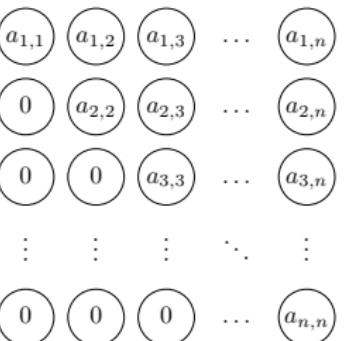
A tömb

Háromszögmátrixok

Dinamikus tömb

## Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötte elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:



$V:$





### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szélszörű (láncolt) tárolás  
Representáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

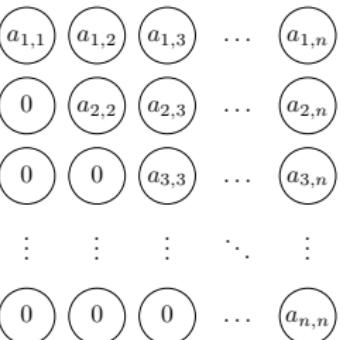
A tömb

### Háromszögmátrixok

Dinamikus tömb

# Felsőháromszög-mátrixok folytonos reprezentációja

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötté elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  $\frac{n \cdot (n+1)}{2}$  elemű  $V$  vektorra:



1.	2.	3.	4.	5.	6.		$\frac{(n-1) \cdot n}{2} + 1$		$\frac{(n-1) \cdot n}{2} + n$
$a_{1,1}$	$a_{1,2}$	$a_{2,2}$	$a_{1,3}$	$a_{2,3}$	$a_{3,3}$	$\cdots$	$\cdots$	$\cdots$	$a_{1,n}$ $a_{2,n}$ $a_{3,n}$ $\cdots$ $a_{n,n}$

# Háromszög-mátrixok folytonos reprezentációja

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Felsőháromszög mátrixok esetén ennek megfelelően

- A  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti felsőháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i > j \\ V_t, & \text{egyébként, ahol } t = \frac{j \cdot (j-1)}{2} + i \end{cases}$$

Alsóháromszög-mátrixok esetén viszont sorfolytonos leképezést célszerű használni.

- Ennek megfelelően, a  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti alsóháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i < j \\ V_t, & \text{egyébként, ahol } t = \frac{i \cdot (i-1)}{2} + j \end{cases}$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb

## Háromszögmátrixok

Dinamikus tömb

# Háromszög-mátrixok folytonos reprezentációja

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Felsőháromszög mátrixok esetén ennek megfelelően

- A  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti felsőháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i > j \\ V_t, & \text{egyébként, ahol } t = \frac{j \cdot (j-1)}{2} + i \end{cases}$$

Alsóháromszög-mátrixok esetén viszont sorfolytonos leképezést célszerű használni.

- Ennek megfelelően, a  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti alsóháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i < j \\ V_t, & \text{egyébként, ahol } t = \frac{i \cdot (i-1)}{2} + j \end{cases}$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb

## Háromszögmátrixok

Dinamikus tömb

# Háromszög-mátrixok folytonos reprezentációja

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

Felsőháromszög mátrixok esetén ennek megfelelően

- A  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti felsőháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i > j \\ V_t, & \text{egyébként, ahol } t = \frac{j \cdot (j-1)}{2} + i \end{cases}$$

Alsóháromszög-mátrixok esetén viszont sorfolytonos leképezést célszerű használni.

- Ennek megfelelően, a  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti alsóháromszög-mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = \begin{cases} 0, & \text{ha } i < j \\ V_t, & \text{egyébként, ahol } t = \frac{i \cdot (i-1)}{2} + j \end{cases}$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárolás  
Szétszóró (láncolt) tárolás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

## Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

## Asszociatív adatszerkezetek

A tömb

## Háromszög-mátrixok

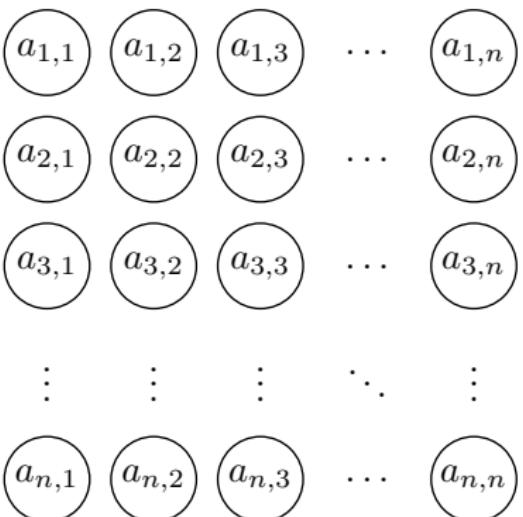
Dinamikus tömb

# Szimmetrikus mátrixok

Adatszerkezetekkel  
kapcsolatos  
alapfogalmak

A szimmetrikus mátrixok **négyzetes** (kvadratikus) mátrixok.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- Melyekre teljesül az

$$a_{i,j} = a_{j,i}$$

egyenlőség bármely  $i$ -re és  $j$ -re ( $1 \leq i, j \leq n$ ).

Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű)  
tárolás  
Szétszórít (láncolt) tárolás  
Reprezentáció és  
implementáció  
Algoritmusok  
Algoritmusok  
megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszög mátrixok  
Dinamikus tömb



### Absztrakt adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárás  
Szétszórít (láncoolt) tárás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

### Struktúra nélküli adatszerkezetek

A halmaz  
A multihalmaz

### Asszociatív adatszerkezetek

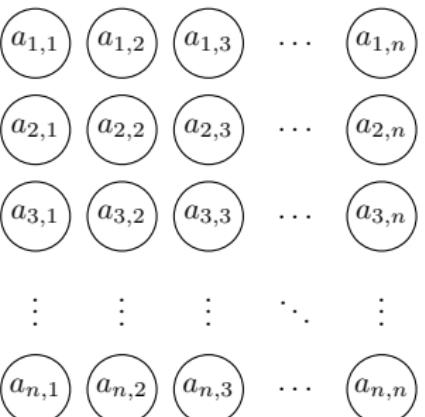
A tömb  
Háromszög mátrixok  
Dinamikus tömb

# Szimmetrikus mátrixok folytonos reprezentációja

Szimmetrikus mátrixok esetén

- vagy a felső háromszöget képezzük le oszlopfolytonosan,
- vagy pedig az alsóháromszöget sorfolytonosan
- A  $V$  vektorból a következő képlet segítségével kaphatjuk vissza az eredeti szimmetrikus mátrix  $(i, j)$  indexű elemének az értékét:

$$a_{i,j} = V_t \text{ ahol } t = \begin{cases} \frac{j \cdot (j-1)}{2} + i, & \text{ha } i \leq j \\ \frac{i \cdot (i-1)}{2} + j & \text{egyébként,} \end{cases}$$





Absztrakt  
adatszerkezetek

Rendszerelmélet  
Absztrakció, modellalkotás  
Absztrakt adatszerkezetek  
Ábrázolási módok  
Folytonos (vektorszerű) tárólás  
Szárszót (láncolt) tárólás  
Reprezentáció és implementáció  
Algoritmusok  
Algoritmusok megadásának módjai  
Algoritmusok építőelemei  
Gazdálkodás a memóriával

Struktúra nélküli  
adatszerkezetek

A halmaz  
A multihalmaz

Asszociatív  
adatszerkezetek

A tömb  
Háromszögmátrixok  
Dinamikus tömb

## Dinamikus tömb

Általában **egydimenziós tömböt** értünk alatta, ekkor más szavakkal (**dinamikus vektornak**) is nevezzük.

- A dinamikus tömb **mérete** szűkebb értelemben a feldolgozás során tetszőlegesen (dinamikusan) változik. Ebben az esetben gyakorlatilag egy **szekvenciális lista** adatszerkezetet kapunk (lásd később).
- Tágabb értelemben fizikailag továbbra is statikus tömbről beszélünk, a logikai adatszerkezet létrehozáskor megadott elemszámát viszont később bizonyos határok között – a lefoglalt tárterület méretétől függően – módosíthatjuk. Ilyenkor a tömb végén lehetnek fel nem használt adatelemek.
- **Bővítés** a dinamikus tömb tetszőleges helyén végrehajtható.
- **Fizikai törlés** bármely elem esetén értelmezhető.
- A dinamikus tömb egyéb műveletei megegyeznek a (statikus) tömb műveleteivel.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



# 4. előadás

## [Elemi adatszerkezetek](#)

[Szekvenciális adatszerkezetek](#)

*Adatszerkezetek és algoritmusok előadás*

2018. február 27.

[Lista](#)

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

[Verem](#)

[Sor](#)

[Kétféle sorok](#)

[Prioritásos sor](#)

[Tömör ábrázolás](#)

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor

**Listá**

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

# Általános tudnivalók

Ajánlott irodalom:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- Gyakorlati aláírás
  - 2 ZH
- Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció  
 szétszórt reprezentáció

**Verem****Sor****Kétféle sorok****Prioritásos sor****Tömör ábrázolás**

# Beszúró rendezés

## **procedure BESZÚRÓ-RENDEZÉS(A)**

```

1 for  $j \leftarrow 2$  to méret(A) do
2     kulcs  $\leftarrow A[j]$ 
3      $i \leftarrow j - 1$ 
4 while  $i \geq 1$  és  $A[i] > \text{kulcs}$  do
5          $A[i + 1] \leftarrow A[i]$ 
6          $i \leftarrow i - 1$ 
7 end while
8      $A[i + 1] \leftarrow \text{kulcs}$ 
9 end for
end procedure

```

- Vegyük észre: a **while** ciklus lineáris keresést használ kulcs helyének meghatározására az A „elejében”.
- **Lehetne ezt bináris keresésre cserélni?**



# A lista adatszerkezet

Dinamikus, homogén, szekvenciális adatszerkezet.

Jelölések:

lista:  $q = [x_1, x_2, \dots, x_n]$

üres lista:  $q = []$

a lista feje:  $x_1$

a lista farka:  $[x_2, \dots, x_n]$

a lista vége:  $x_n$

a lista hossza, mérete:  $n$  vagy  $|q|$

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem**

**Sor**

**Kétféle sorok**

**Prioritásos sor**

**Tömör ábrázolás**



# A lista adatszerkezet

## Alapműveletek

- Hozzáférés, elérés: közvetlen.

$$q[i] = x_i$$

- Részlistaképzés, allistaképzés:

$$q[i \dots j] = [x_i, x_{i+1}, \dots, x_{j-1}, x_j]$$

- Konkatenáció, egyesítés, összefűzés:

$$r = [y_1, \dots, y_m]$$

$$q \& r = [x_1, x_2, \dots, x_n, y_1, \dots, y_m]$$

Lista

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

Verem

Sor

Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

# A lista adatszerkezet

## Listával végezhető műveletek

- Bővítés:** bárhol bővíthető. Bővítéskor részlistákat képzünk, majd azokat konkatenáljuk az új elemből/elemekből álló részlistával. A  $k$ -adik elem mögé történő bővítés:

$$q[1 \dots k] \& [\text{elem}] \& q[(k+1) \dots n]$$

- Törlés:** megvalósítható a fizikai törlés, melynek során részlistákat képzünk (melyekben már nem szerepel(nek) a törlni kívánt elem(ek)), majd konkatenáljuk ezeket a részlistákat. A  $k$ -adik elem törlése:

$$q[1 \dots (k-1)] \& q[(k+1) \dots n]$$

**Lista**

Az absztrakt adatszerkezet

folytonos reprezentáció

szétszórt reprezentáció

**Verem****Sor****Kétféléző sorok****Prioritásos sor****Tömör ábrázolás**

# A lista adatszerkezet

## Listával végezhető műveletek

- Csere:** bármelyik elem cserélhető. Részlistákat képzünk, majd azokat koncatenáljuk az új értékből álló részlistával.  
A  $k$ -adik elem cseréje:

$$q[1 \dots (k-1)] \& [\text{elem}] \& q[(k+1) \dots n]$$

- Rendezés:** értelmezhető, bármelyik rendezési algoritmus használható.
- Keresés:** értelmezhető, bármelyik keresési algoritmus használható.
- Elérés:** soros vagy közvetlen.
- Bejárás:** értelmezhető.
- Feldolgozás:** a lista alapműveletei segítségével.



## A szélső elemekkel végezhető speciális műveletek

- **ACCESS HEAD**: az első elem elérése:

**return**  $x_1$

- **PUSH**: bővítés az első elem előtt:

$q \leftarrow [\text{elem}] \& q$

- **POP**: az első elem törlése:

$q \leftarrow q[2 \dots n]$  és **return**  $x_1$

- **ACCESS END**: az utolsó elem elérése:

**return**  $x_n$

- **INJECT**: bővítés az utolsó elem után:

$q \leftarrow q \& [\text{elem}]$

- **EJECT**: az utolsó elem törlése:

$q \leftarrow q[1 \dots n - 1]$  és **return**  $x_n$

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem**

**Sor**

Kétvégű sorok

Prioritásos sor

Tömör ábrázolás



### Listá

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

### Verem

### Sor

### Kétféle sorok

### Prioritásos sor

### Tömör ábrázolás

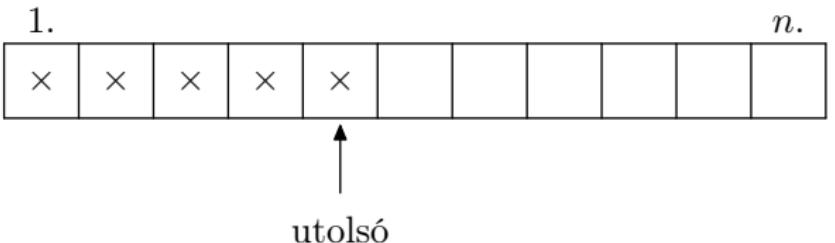
- Folytonos reprezentáció: vektorral.
- Szétszórt reprezentáció: láncolt listával.
  - Egyirányban láncolt listával.
  - **Kétirányban** láncolt listával.
  - Cirkulárisan láncolt listával.



# A lista adatszerkezet folytonos reprezentációja

Mutatók nélkül

Folytonos reprezentáció esetén a lista elemei egy tömbben vannak letárolva egymás után. A lista könnyen bejárható, illetve egy új elemet azonnal be lehet tenni a lista végére.



Egy új elem beszúrása a lista közepére viszont azzal jár, hogy az őt követő elemeket eggyel jobbra kell mozgatni. Hasonlóképpen, egy elem törlése (kivéve ha az utolsó elemet töröljük) szintén elemmozgatással jár.

[Lista](#)

[Az absztrakt adatszerkezet](#)

[folytonos reprezentáció](#)

[szétszórt reprezentáció](#)

[Verem](#)

[Sor](#)

[Kétvégű sorok](#)

[Prioritásos sor](#)

[Tömör ábrázolás](#)

**Lista**

Az absztrakt adatszerkezet

folytonos reprezentáció

szétszórt reprezentáció

**Verem****Sor**

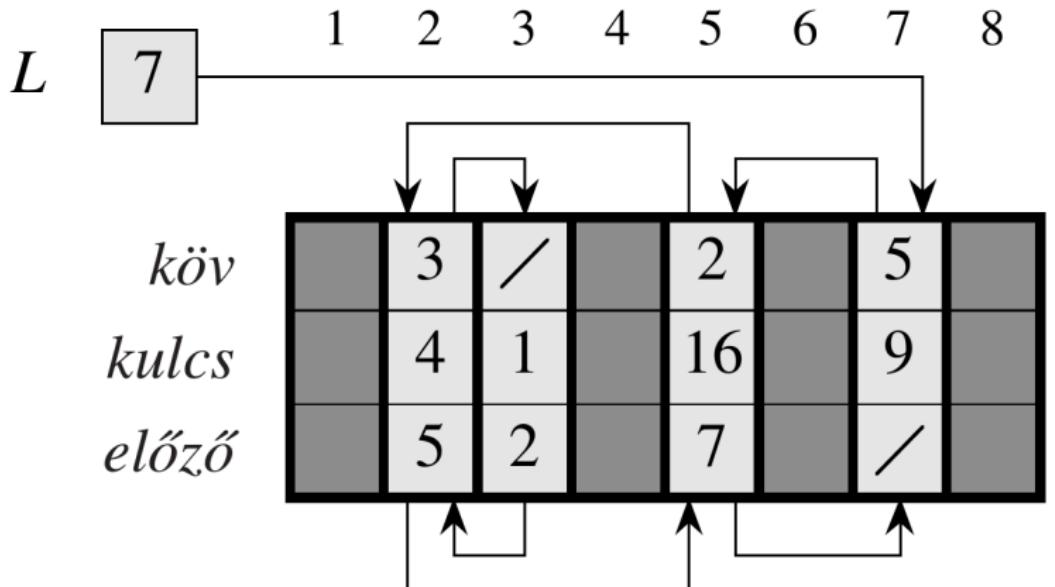
Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

# A lista adatszerkezet folytonos reprezentációja

Mutatókkal





# A lista adatszerkezet szétszórt reprezentációja

Láncolt listák esetén az egymás utáni elemeket valamilyen módon meg kell címezni. Erre valók a **mutatók**.

## Mutató

A mutató (pointer) egy olyan változó, amelynek az **értéke** egy **memóriacím**. Magas szintű programozási nyelvekben ehhez hozzárendelődik még az a típus is, amelyre a mutató mutat.

## Műveletek mutatókkal

- Mutató hozzárendelése egy objektumhoz:

```
// A pszeudonyelvben:
//      p <- lefoglal
// C-ben:
tipus *p;
p = (tipus *)malloc(sizeof(tipus));
```

- A mutató nincs semmilyen objektumhoz sem rendelve:

```
p = NULL;
```

Lista

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

Verem

Sor

Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétféléző sorok****Prioritásos sor****Tömör ábrázolás**

# A lista adatszerkezet szétszórt reprezentációja

## Műveletek mutatókkal (folyt.)

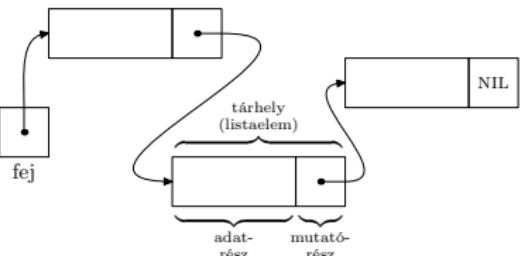
- A mutató által címzett objektum megszüntetése:  
`free(p);`
- A mutató átállítása egy másik mutató által címzett objektumra:  
`q = p;`
  - 1 Mi történik a `q`-val korábban címzett objektummal?
  - 2 Legyen a `p` mutató egy lokális változó egy eljárásban. Mi történik az általa címzett objektummal?
- Értékadás mutatóval:  
`valtozo = *p;`  
`*p = ertekek;`
- Összehasonlítás:  
`if (p == q) ... /* Ugyanoda mutatnak? */`  
`if (*p == *q) ... /* A mutatott objektumok  
azonos ertekek? */`

**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

# Egyszerű láncolt lista



A listaelem egy **rekord**. Egy üres láncolt listát a következőképpen tudunk létrehozni C-ben:

hosszabb változat:	rövidebb változat:
<pre>struct listaelem {     típus adat;     struct listaelem *kov; };  typedef struct listaelem LISTAELEM; LISTAELEM *fej; fej = NULL;</pre>	<pre>typedef struct listaelem {     típus adat;     struct listaelem *kov; } LISTAELEM;  LISTAELEM *fej = NULL;</pre>

# A lista adatszerkezet szétszórt reprezentációja

Elemi adatszerkezetek

## Műveletek egyszerű láncolt listákon

- Üres lánc inicializálása:

`fej = NULL;`

- Elem beszúrása:

- lista elejére
- lista végére
- aktuális elem után
- aktuális elem elő

- Elem törlése:

- lista elejéről
- lista végéről
- aktuális elem

- Pozicionálás:

- lista elejére
- lista végére
- aktuális elem után

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



[Lista](#)

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

[Verem](#)

[Sor](#)

[Kétvégű sorok](#)

[Prioritásos sor](#)

[Tömör ábrázolás](#)

## Műveletek egyszerű láncolt listákon (folyt.)

- aktuális elem értékének olvasása
- jelezzük, ha a lánc üres
- jelezzük, ha az aktuális elem a lánc utolsó eleme

A továbbiakban nézzük meg néhány művelet implementációját. Az egyszerűség kedvéért a fejmutatót globális változóként kezeljük.

### beszúrás a lánc elejére

```
void elejere_beszur(tipus adat)
{
    LISTAELEM *uj = (LISTAELEM *)malloc(sizeof(LISTAELEM));

    uj->adat = adat;
    uj->kov = fej;
    fej = uj;
}
```

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



#### Lista

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

#### Verem

#### Sor

#### Kétféléző sorok

#### Prioritásos sor

#### Tömör ábrázolás

## beszúrás a lánc végére

Ha alkalmazunk egy „utolsó” mutatót is (mely minden esetben a lánc utolsó elemére mutat), akkor könnyű dolgunk van. Ellenkező esetben meg kell keresni a lánc végét:

```
void vegere_beszur(tipus adat)
{
    LISTAELEM *akt = fej;

    LISTAELEM *uj = (LISTAELEM *)malloc(sizeof(LISTAELEM));
    uj->adat = adat;
    uj->kov = NULL;

    if (fej == NULL) {
        fej = uj;
    }
    else
    {
        while (akt->kov != NULL) {
            akt = akt->kov;
        }
        akt->kov = uj;
    }
}
```

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Lista

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

### Sor

### Kétvégű sorok

### Prioritásos sor

### Tömör ábrázolás

# A lista adatszerkezet szétszórt reprezentációja

Elemi adatszerkezetek

Kósá Márk

Pánovics János

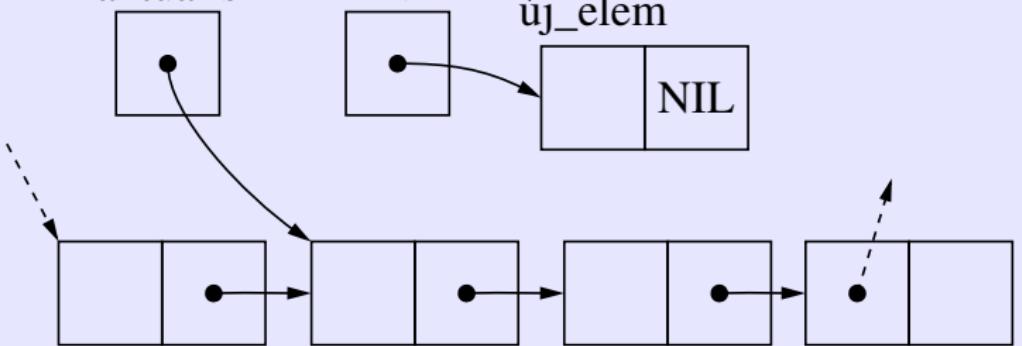
Szathmáry László

Halász Gábor



## aktuális elem után való beszúrás

aktuális              új              új\_elem



## [Lista](#)

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

## [Verem](#)

## [Sor](#)

## [Kétvégű sorok](#)

## [Prioritásos sor](#)

## [Tömör ábrázolás](#)

# A lista adatszerkezet szétszórt reprezentációja

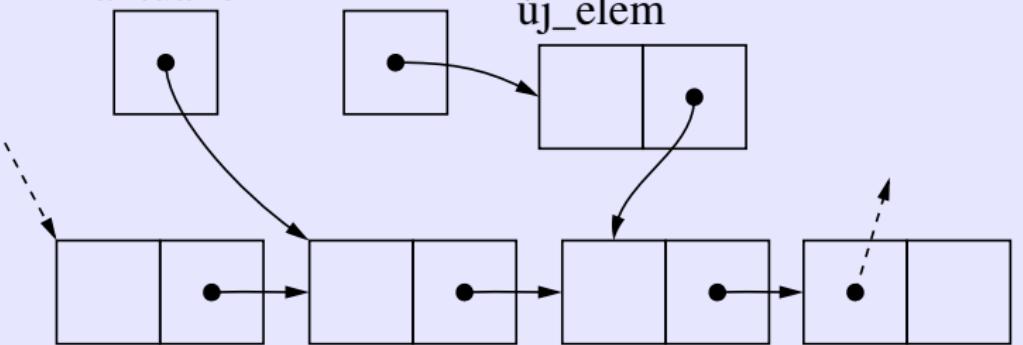
Elemi adatszerkezetek

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## aktuális elem után való beszúrás

aktuális      új      új\_elem



## Listá

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

## Verem

## Sor

## Kétvégű sorok

## Prioritásos sor

## Tömör ábrázolás

# A lista adatszerkezet szétszórt reprezentációja

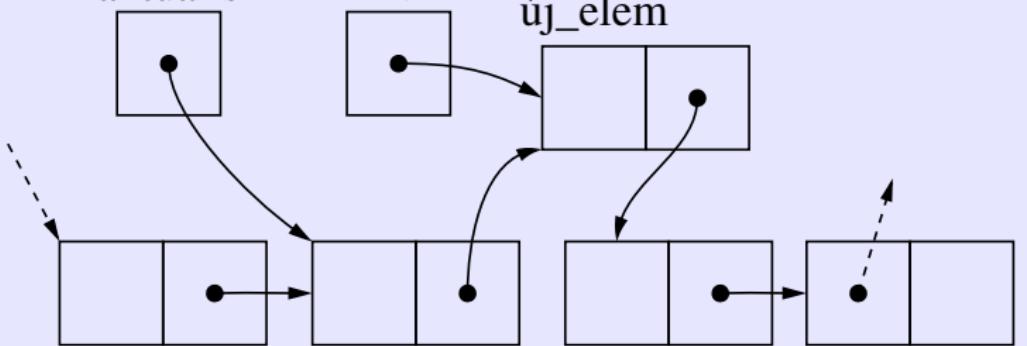
Elemi adatszerkezetek

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## aktuális elem után való beszúrás

aktuális      új      új\_elem



## Listá

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

## Verem

## Sor

## Kétféle sorok

## Prioritásos sor

## Tömör ábrázolás

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

# A lista adatszerkezet szétszórt reprezentációja

## aktuális elem elé való beszúrás

- Két mutatót használunk: „előző” és „aktuális”. Az „előző” mutató az „aktuális” elem előtti elemre mutat. A beszúrás az „előző” elem után történik.
- Egy mutató használata esetén: beszúrás az aktuális elem után, majd adatrészek cseréje, vagy
- „mutató lánc” használata:  
**while** m→következő→kulcs≠keresett\_érték **do** ...

# A lista adatszerkezet szétszórt reprezentációja

## Elem törlése

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### [Lista](#)

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

### [Verem](#)

### [Sor](#)

### [Kétvégű sorok](#)

### [Prioritásos sor](#)

### [Tömör ábrázolás](#)

## első elem törlése

Lásd ÁBRA



# A lista adatszerkezet szétszórt reprezentációja

## Elem törlése

### utolsó elem törlése

Meg kell jegyezni, hogy melyik az utolsó előtti elem, s annak a következő mutatóját NIL-re állítjuk.

```
void utolso_torles()
{
    LISTAELEM *elozo = NULL;
    LISTAELEM *akt = fej;

    if (fej == NULL) {           hiba("a lanc ures");
        return;
    }
    /* különben, ha a lanc nem ures */
    while (akt->kov != NULL) {
        elozo = akt;
        akt = akt->kov;
    }
    if (elozo == NULL) { /* egyelemű */
        free(akt);
        fej = NULL;
    }
    else { /* egynél több elemű */
        free(akt);
        elozo->kov = NULL;
    }
}
```

### Lista

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

### Verem

#### Sor

#### Kétféle sorok

#### Prioritásos sor

#### Tömör ábrázolás



# A lista adatszerkezet szétszórt reprezentációja

## Keresés

### adott elem keresése a láncban

```
int benne_van(LISTAELEM *fej, tipus keresett_ertek)
{
    LISTAELEM *akt = fej;

    while ((akt != NULL) && (akt->adat != keresett_ertek))
    {
        akt = akt->kov;
    }

    if (akt != NULL) return 1;
    else return 0;
}
```

#### [Lista](#)

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

#### [Verem](#)

#### [Sor](#)

#### [Kétvégű sorok](#)

#### [Prioritásos sor](#)

#### [Tömör ábrázolás](#)

**Listá**

Az absztrakt adatszerkezet folytonos reprezentáció  
 szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

## A verem adatszerkezet

Speciális lista adatszerkezet. Csak a verem **tetejére** lehet betenni, illetve csak onnan lehet kivenni.

- Az utoljára betett elem a verem tetejére kerül.
- Az elsőnek betett elem a verem aljára kerül.

### Veremmel végezhető műveletek

- **Létrehozás:** üres verem inicializálása.
- **Bővítés:** új elem bevitelé az utoljára betett elem fölé (PUSH).
- **Csere:** nincs.
- **Törlés:** fizikai, a verem tetején lévő elemet (POP).
- **Rendezés, keresés és bejárás:** nem értelmezett.
- **Elérés:** a fölső elemet közvetlenül, a többöt sehogyan sem (TOP).
- **Feldolgozás:** Last In First Out (LIFO) adatszerkezet, az utolsóként érkező elemet dolgozzuk fel először.

Az utolsóként érkezett elemhez történő hozzáférést illetve ezen elem törlésének a műveletét egy műveletként is definiálhatjuk.

**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

**Verem****Sor**

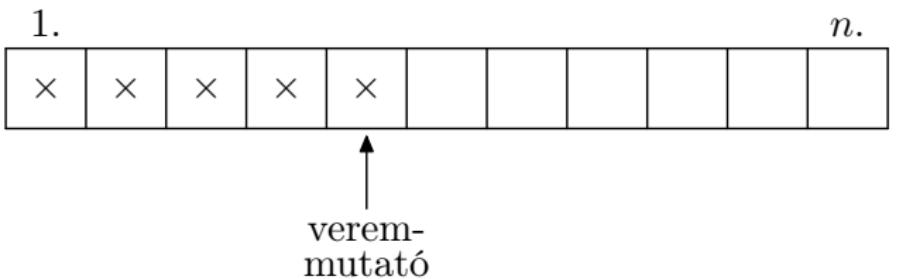
Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

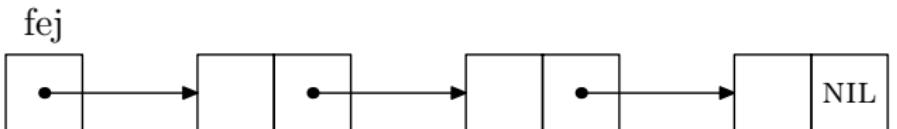
# A verem adatszerkezet reprezentációi

Folytonos reprezentáció:



A veremmutató mindenkor a verem tetején lévő elemet indexeli. Ha a veremmutató értéke **0**, a verem **üres**. Ha a veremmutató értéke ***n***, a verem **tele** van.

Szétszórt reprezentáció: egyirányban láncolt listával.



A **fej** mutató mindenkor a verem tetején lévő elemre mutat. Ha a fejnek NIL az értéke, a verem **üres**.

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor

**Listá**

Az absztrakt adatszerkezet folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

## A sor adatszerkezet

Speciális lista adatszerkezet, melynek alapműveletei a speciális listaműveletek közül a következők:

- az első elemhez történő hozzáférés: ACCESS HEAD
- bővítés az utolsó elem mögé: PUT (INJECT)
- az első elem törlése: GET (POP)

Az első elemhez történő hozzáférés és az első elem törlésének műveletét gyakran egy műveletként is definiáljuk.

### Sorral végezhető műveletek

- **Létrehozás**: üres sor.
- **Bővítés**: az utolsó elem mögé.
- **Csere**: nincs.
- **Törlés**: fizikai, az első elemet.
- **Rendezés, keresés és bejárás**: nem értelmezett.
- **Elérés**: az első elemet közvetlenül, a többöt seholgyan sem.
- **Feldolgozás**: First In First Out (FIFO) adatszerkezet, az elsőként érkező elemet dolgozzuk fel először.

Kósá Márk

Pánovics János

Szathmáry László

Halász Gábor

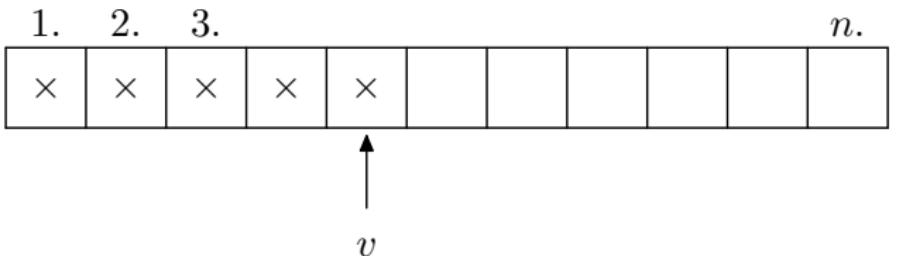
**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

## A sor adatszerkezet folytonos reprezentációi

Fix kezdetű sor:



A sor első elemének a helye rögzített, minden az 1. indexű tárhely a vektorban. A  $v$  (vége) mutató a sor utolsó elemét indexeli. **Üres** a sor, ha  $v = 0$ . **Tele** van a sor, ha  $v = n$ .

Az új elemet a  $(v + 1)$ -edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az 1. pozíciót lévő elemet tudjuk, ha a sor nem üres. Törléskor a sor megmaradó elemeit egy tárhellyel előrébb csúsztatjuk.

**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

**Verem****Sor**

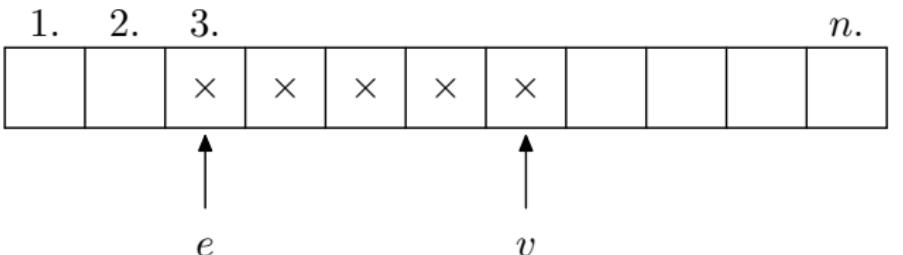
Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

# A sor adatszerkezet folytonos reprezentációi

Vándorló sor:



A sor első elemét az  $e$  (eleje) mutató, az utolsót a  $v$  (vége) mutató indexeli. **Üres** a sor, ha  $e = 0$  és  $v = 0$ . **Tele** van a sor, ha  $e = 1$  és  $v = n$ .

Ha bővítéskor  $v = n$ , de a sor nincs tele, akkor először a sor minden elemét  $e - 1$  pozícióval előrébb csúsztatjuk, majd végrehajtjuk a bővítést az új elemmel. Az új elemet a  $(v + 1)$ -edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az  $e$ -edik pozícion lévő elemet tudjuk, ha a sor nem üres.

**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

**Verem****Sor**

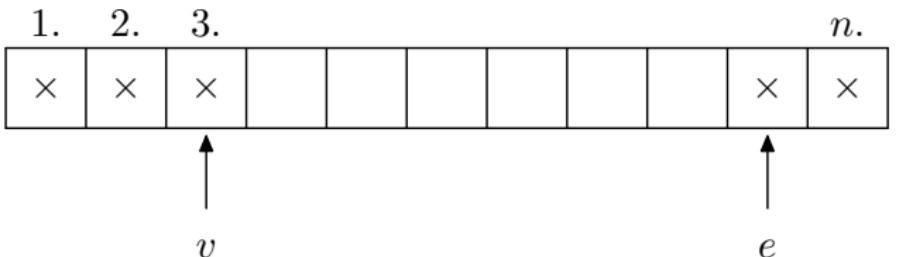
Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

# A sor adatszerkezet folytonos reprezentációi

Ciklikus sor:



A sor első elemét az  $e$  (eleje) mutató, az utolsót a  $v$  (vége) mutató indexeli. **Üres** a sor, ha  $e = 0$  és  $v = 0$ . **Tele** van a sor, ha  $e = v \bmod n + 1$ .

Az új elemet a  $(v \bmod n + 1)$ -edik pozícióra helyezzük el, ha a sor nincs tele. Törölni az  $e$ -edik pozíciót lévő elemet tudjuk, ha a sor nem üres.

# A sor adatszerkezet szétszórt reprezentációi

Elemi adatszerkezetek

Kósá Márk

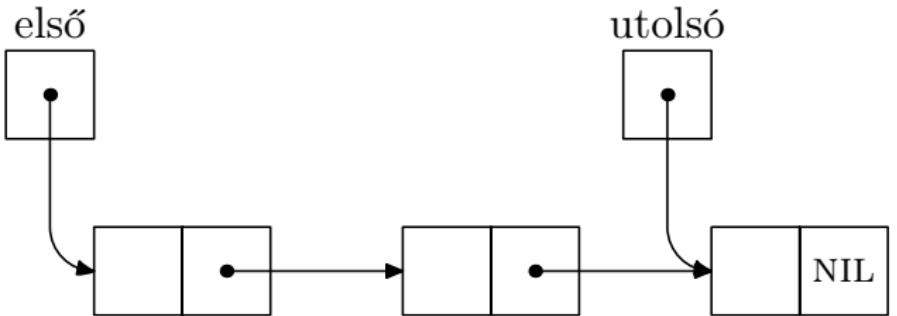
Pánovics János

Szathmáry László

Halász Gábor



Szétszórt reprezentáció egyirányban láncolt listával, két segédmutatóval:



**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem**

**Sor**

Kétvégű sorok

Prioritásos sor

Tömör ábrázolás

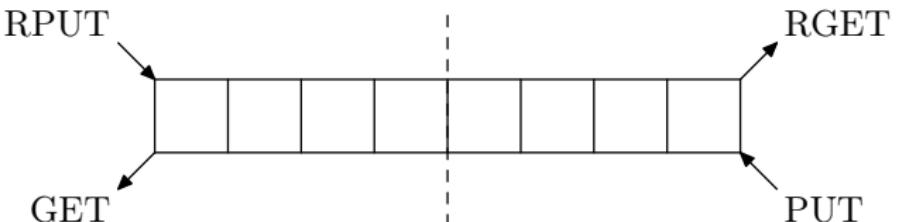
- Elérni és **feldolgozni** az „első” mutató által hivatkozott elemet tudjuk,
- **bővíteni** pedig az „utolsó” mutató által hivatkozott elem mögé tudunk.
- A sor **üres**, ha minden segédmutató értéke **NIL**.

**Lista**

Az absztrakt adatszerkezet  
folytonos reprezentáció  
szétszórt reprezentáció

**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

- Kétvégű sor:** a sor műveletei mellett további két művelet jelenik meg: RGET és RPUT (reverse GET és reverse PUT).



Tekinthető két, az aljuknál összeragasztott veremnek.

- Inputkorlátozott kétvégű sor:** nincs RPUT művelet.
- Outputkorlátozott kétvégű sor:** nincs RGET művelet.

Kósa Márk  
 Pánovics János  
 Szathmáry László  
 Halász Gábor

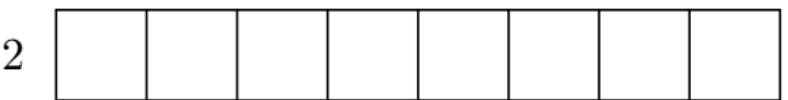
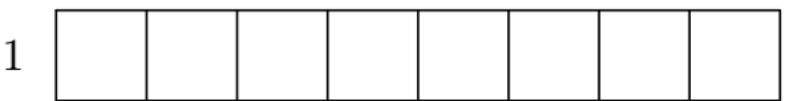
**Lista**

Az absztrakt adatszerkezet folytonos reprezentáció  
 szétszórt reprezentáció

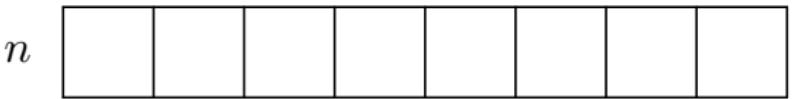
**Verem****Sor****Kétvégű sorok****Prioritásos sor****Tömör ábrázolás**

## Prioritásos sor

Olyan sor, amelyben az adatelemekhez **prioritásértéket** rendelünk, majd ezen értékek sorrendjében (azonos prioritású elemek esetén pedig továbbra is a bekerülés sorrendjében) dolgozzuk fel őket. Megvalósítása  $n$  különböző prioritásérték esetén  $n + 1$  (hagyományos) sorral történhet: minden prioritásértékhez tartozik egy-egy sor, a prioritás nélküli elemeket pedig külön sorban tároljuk:



⋮





## Lista

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

## Verem

## Sor

## Kétvégű sorok

## Prioritásos sor

## Tömör ábrázolás

# Lista tömör folytonos ábrázolása

Tömör ábrázolás alatt azt értjük, hogy

- ① Az **n** elemű lista folytonosan van tárolva
- ② a vektor első **n** elemében.

fej=6; veg=4    1    2    3    4    5    6    7    8    9    10    11

<b>Kulcs:</b>	43	21	23	87	45	12	54	67			
<b>Előző:</b>	3	6	2	8	1	0	5	7			
<b>Következő:</b>	5	3	1	0	7	2	8	4			

(**n**=8 elemű lista: 12, 21, 23, 43, 45, 54, 67, 87)

Hogyan lehet ilyen esetben hatékonyan

- Törölni, vagy
- Bővíteni?
  - „rendezetlen” listában
  - rendezett listában



Lista

Az absztrakt adatszerkezet folytonos reprezentáció szétszórt reprezentáció

Verem

Sor

Kétféle sorok

Prioritásos sor

Tömör ábrázolás

## Lista tömör folytonos ábrázolása

```
function TÖMÖR_LISTÁBAN_KERES(L,k)
```

```

1   i ← L.fej
2   while i ≠ 0 és L.kulcs[i] < k do
3       j ← VÉLETLEN(1, L.méret)
4       if L.kulcs[i] < L.kulcs[j] és L.kulcs[j] ≤ k then
5           i ← j
6           if L.kulcs[i] = k then
7               return i
8           end if
9       end if
10      i ← L.következő[i]
11   end while
12   if i = 0 vagy L.kulcs[i] > k then
13       return 0
14   else
15       return i
16   end if
end function
```



# 5. előadás

## Hierarchikus adatszerkezetek

Fák, bináris fák, bejárások

*Adatszerkezetek és algoritmusok előadás*

2018. március 6.

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

# Általános tudnivalók

Ajánlott irodalom:

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- Gyakorlati aláírás
  - 2 ZH
- Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## **Lista tömör folytonos ábrázolása**

**function** TÖMÖR\_LISTÁBAN\_KERES(L,k)

```
1: i ← L.fej
2: while i ≠ 0 és L.kulcs[i] < k do
3:     j ← VÉLETLEN(1, méret(L.kulcs))
4:     if L.kulcs[i] < L.kulcs[j] és L.kulcs[j] ≤ k then
5:         if L.kulcs[j] = k then
6:             return j
7:         end if
8:         i ← j
9:     end if
10:    i ← L.következő[i]
11: end while
12: if i = 0 vagy L.kulcs[i] > k then
13:     return 0
14: else
15:     return i
16: end if
end function
```

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



## Hierarchikus adatszerkezet

A szekvenciális adatszerkezet általánosítása:

- minden adatelemének – egyet kivéve –
- pontosan egy megelőzője,
- és tetszőleges számú (akár 0) rákövetkezője lehet.

Hierarchikus adatszerkezetek:

- fa
- hierarchikus lista

Hierarchikus  
adatszerkezetek

A fa adatszerkezet

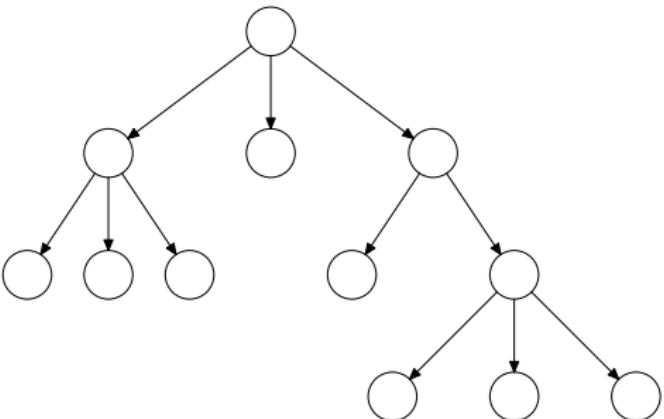
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

# A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

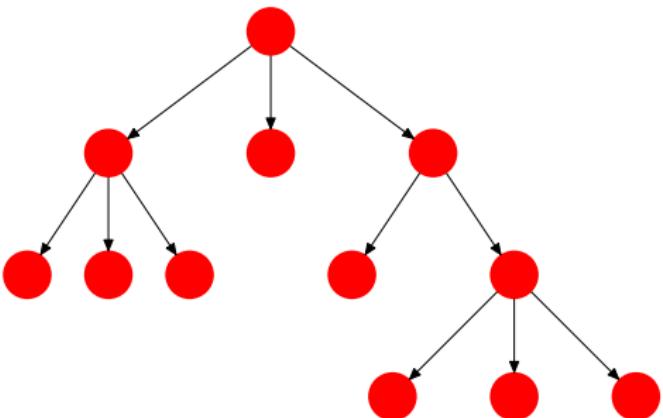
- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság



## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

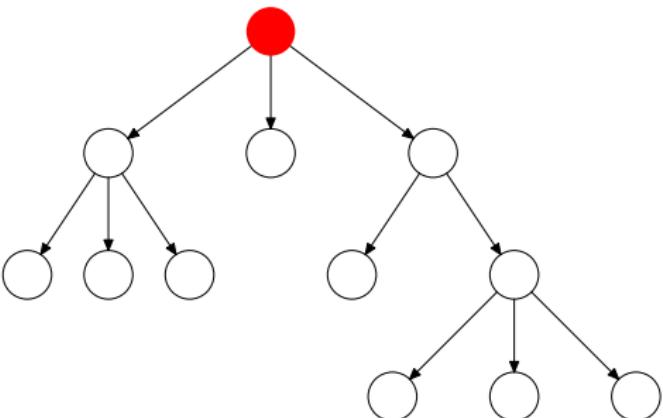
- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság



## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

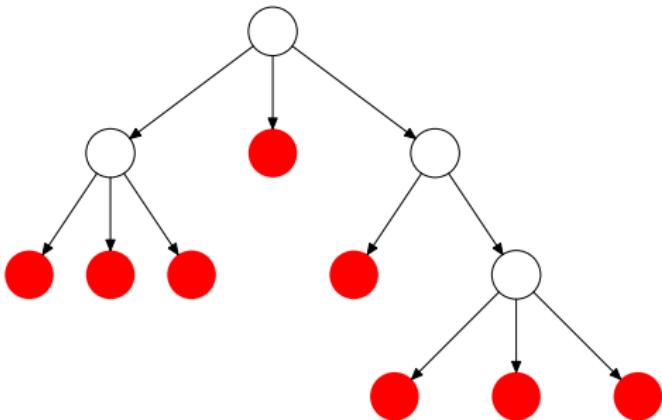
- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság



## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság

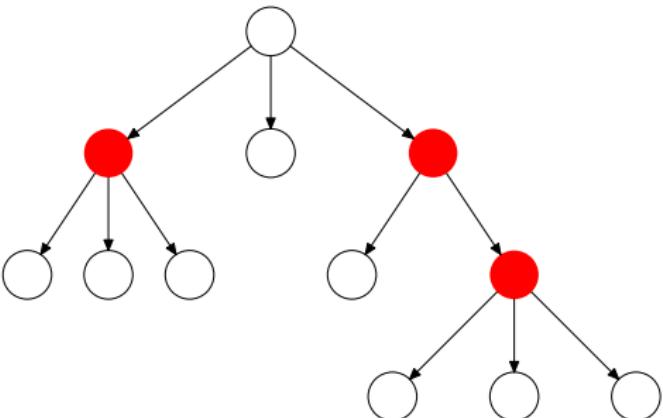




## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

- csúcs, csomópont
  - gyökérelem
  - levélelem
  - közbenső elem
- él
  - út
  - részfa
  - szint
  - magasság

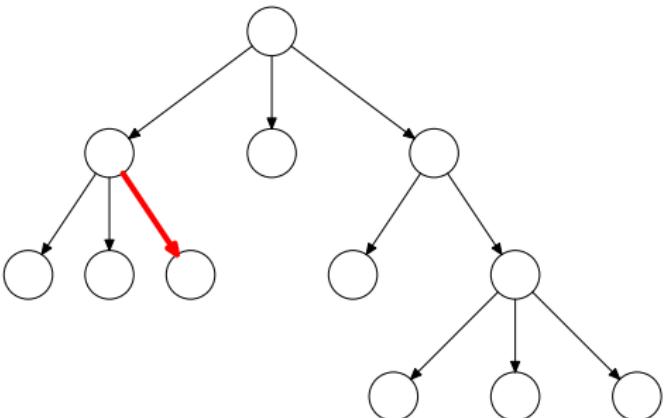




## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság

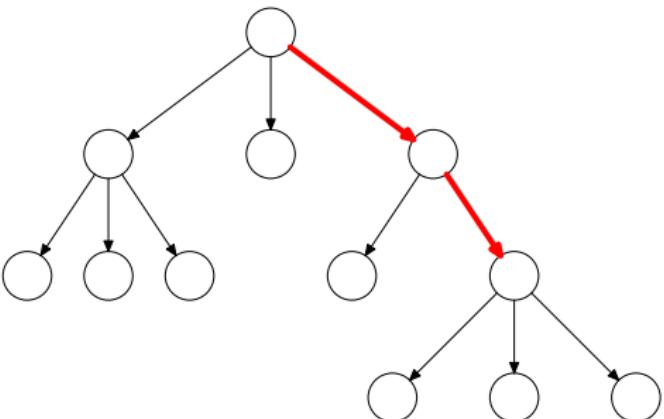




## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

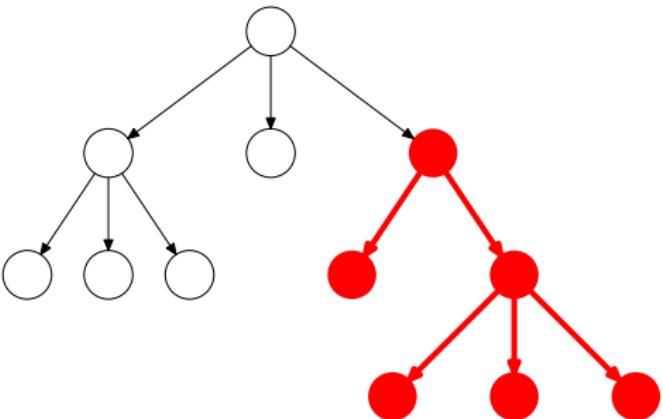
- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság



## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

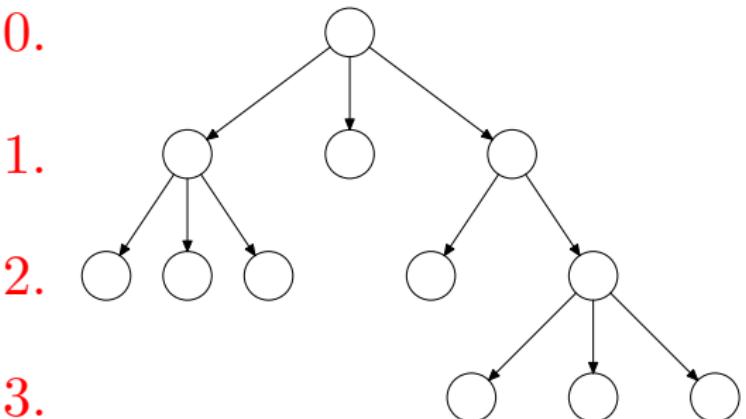
- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság



# A fa adatszerkezet

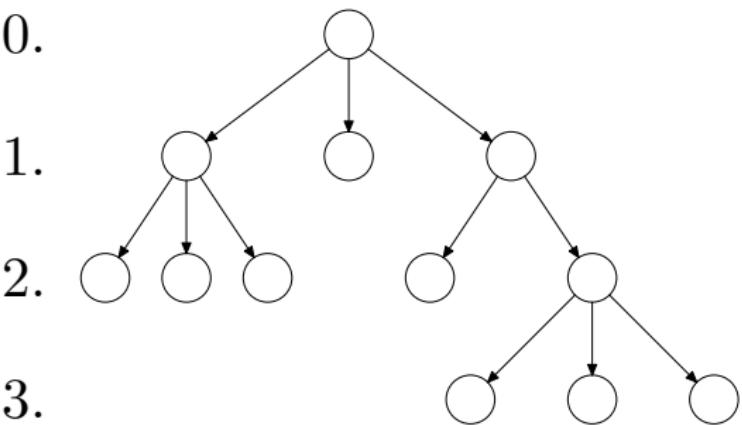
Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság





4



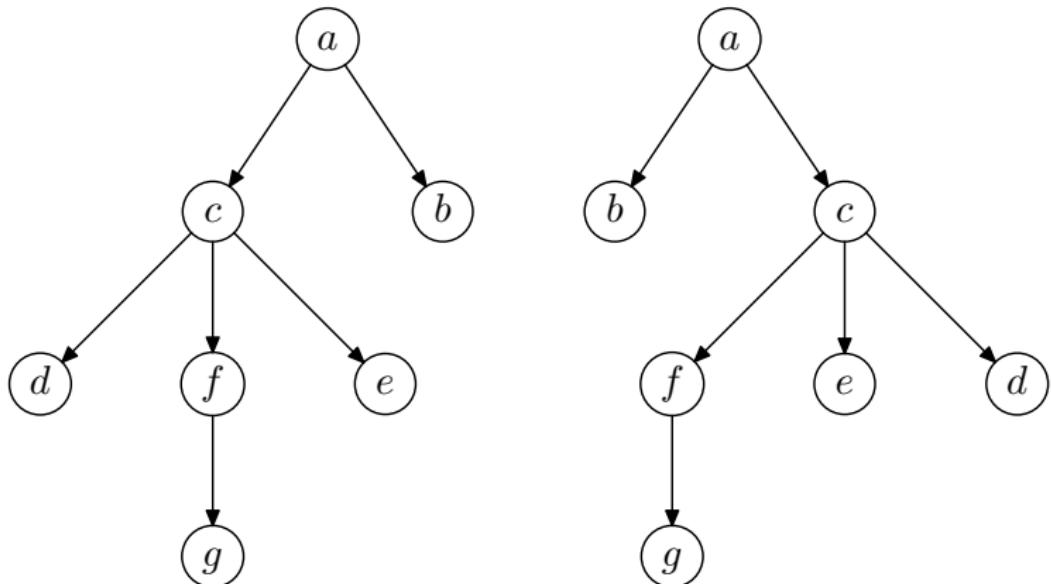
## A fa adatszerkezet

Homogén, dinamikus, hierarchikus adatszerkezet. Fa adatszerkezetekkel kapcsolatos fogalmak:

- csúcs, csomópont
- gyökérelem
- levélelem
- közbenső elem
- él
- út
- részfa
- szint
- magasság

## Rendezetlen és rendezett fák

Rendezetlen fáknál nem lényeges az ugyanazon csúcsból kiinduló élek sorrendje, rendezett fáknál viszont igen.



Az ábrán látható két fa ekvivalens egymással, ha eltekintünk az ugyanazon csúcsokból kiinduló élek sorrendjétől (azaz ha rendezetlen fáknak tekintjük őket).

Mivel az informatikában a rendezett adatszerkezetek játszanak fontos szerepet, a továbbiakban rendezett fákkal foglalkozunk.





## Bináris fa

### Bináris fa

Olyan fa, melyben minden adatelemnek legfeljebb két rákövetkezője van.

### Szigorú értelemben vett bináris fa

Szigorú értelemben vett bináris fáról beszélünk, ha a bináris fában minden adatelemnek 0 vagy 2 rákövetkezője van.

### Rendezett bináris fa

Rendezett bináris fa elemeire értelmezhetők a következő fogalmak:

- bal/jobb oldali rákövetkező
- bal/jobb oldali részfa

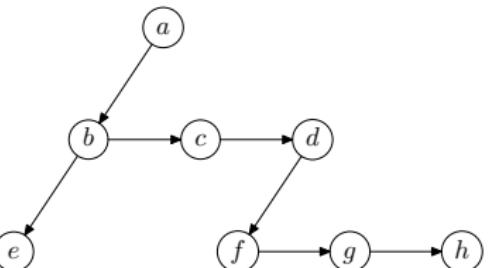
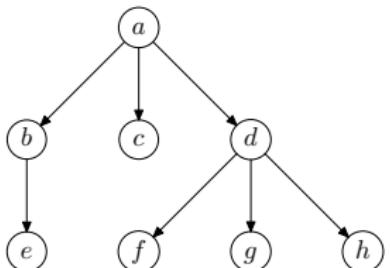
A továbbiakban bináris fa alatt – hacsak másat nem mondunk – rendezett bináris fát értünk.

# Nem bináris fa rendezett bináris fává alakítása

Minden nem bináris fa reprezentálható rendezett bináris fával.

## Tetszőleges fa binarizálásának algoritmusa

- 1 Legyen a bináris fa gyökere a nem bináris fa gyökere.
- 2 A bináris fa egy tetszőleges elemének bal oldali rákövetkezője legyen a nem bináris fa megfelelő elemének bal oldali (első) rákövetkezője.
- 3 A bináris fa egy tetszőleges elemének jobb oldali rákövetkezője legyen a nem bináris fa megfelelő elemének következő (azonos szülőhöz tartozó) testvércsúcsa.



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

### A fa adatszerkezet

- Bináris fa
- Bejárási algoritmusok
  - Preorder bejárás
  - Inorder bejárás
  - Postorder bejárás
- Reprezentáció
  - Kifejezésfák
  - Implementáció

### Kupac

Kupacrendezés

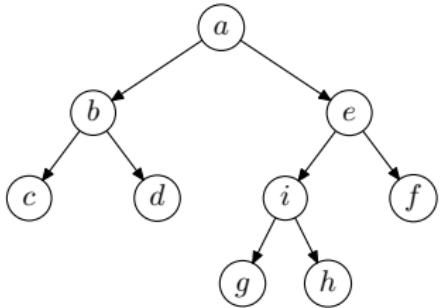
## Bináris fával végezhető műveletek

- **Létrehozás:** üres fa.
- **Bővítés:** egy elemmel vagy egy részfával, általában levélelemnél.
- **Törlés:** részfát vagy egy elemet, utóbbi esetben a fát a legtöbb esetben újra kell szervezni (hogy továbbra is fa maradjon).
- **Csere:** megengedett.
- **Rendezés:** nincs.
- **Keresés, elérés** és **feldolgozás:** a bejárás algoritmusa alapján.
- **Bejárás:** szokás szerint olyan algoritmus, amelynek segítségével a bináris fa elemeit leképezzük egy sorra (preorder, inorder vagy postorder módon).



## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

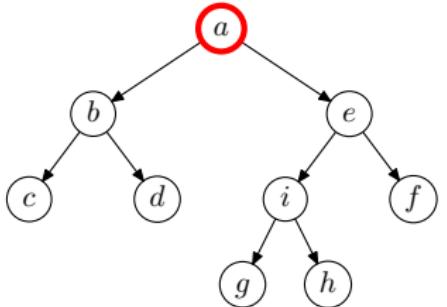
Bináris fa  
Bejárási algoritmusok  
**Preorder bejárás**  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

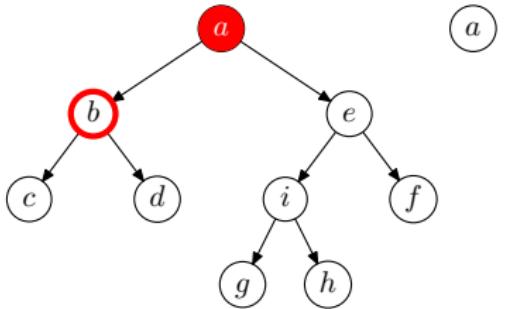
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa

Bejárási algoritmusok

Preorder bejárás

Inorder bejárás

Postorder bejárás

Reprezentáció

Kifejezésfák

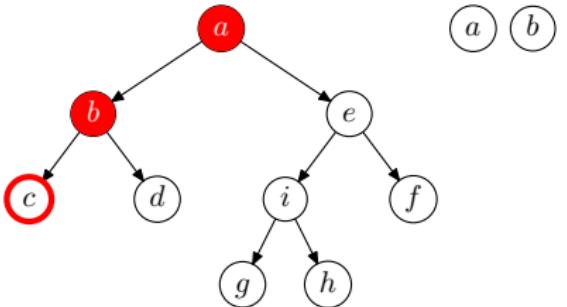
Implementáció

Kupac

Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok

Preorder bejárás

Inorder bejárás  
Postorder bejárás

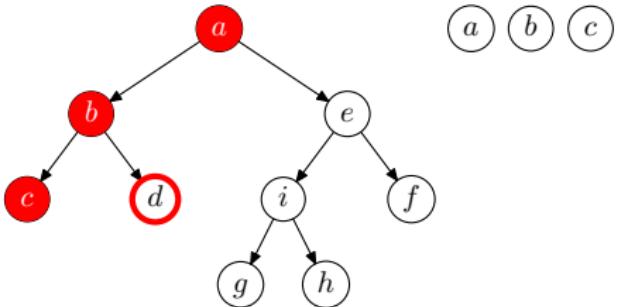
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Hierarchikus  
adatszerkezetek

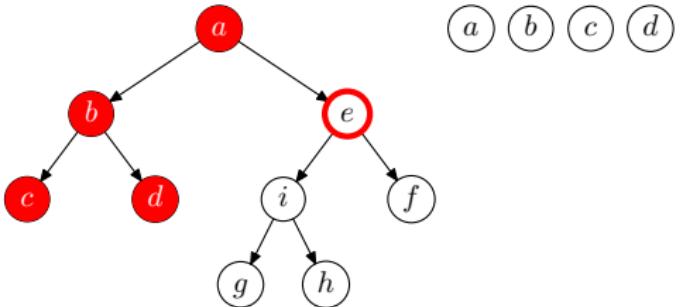
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

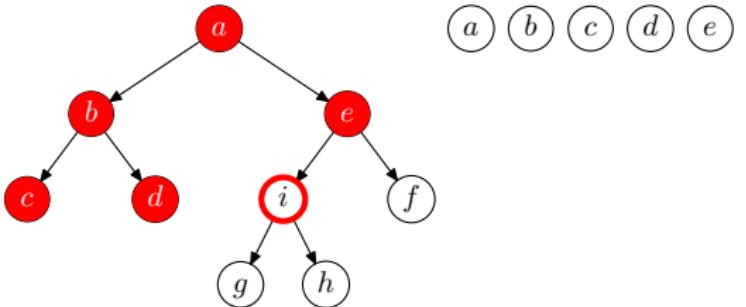
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok

Preorder bejárás

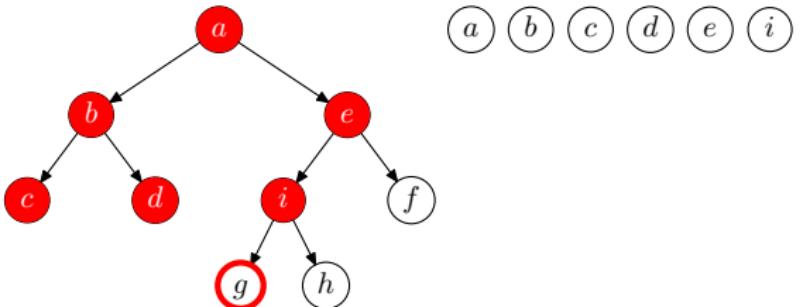
Inorder bejárás  
Postorder bejárás

Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok

Preorder bejárás

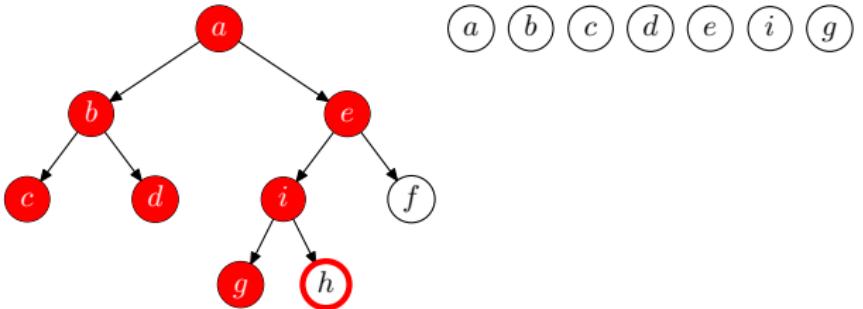
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok

Preorder bejárás

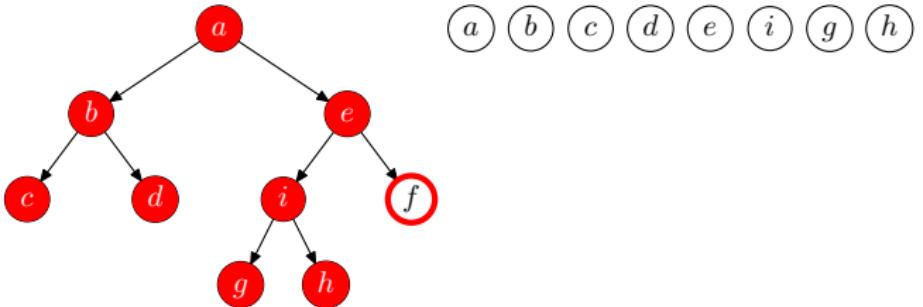
Inorder bejárás  
Postorder bejárás

Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Hierarchikus  
adatszerkezetek

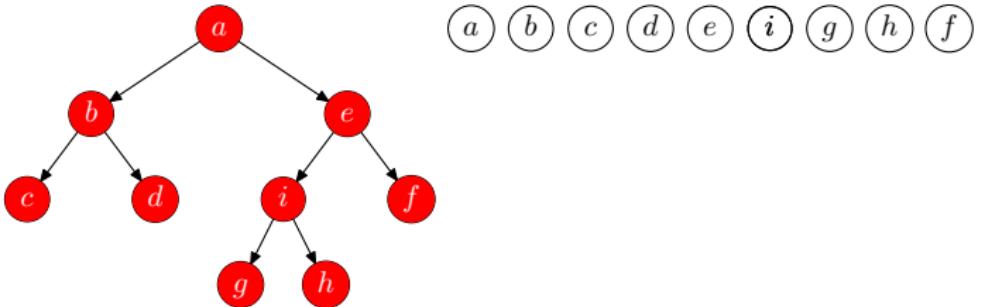
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Preorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ③ Járjuk be a gyökérelem bal oldali részfáját preorder módon.
- ④ Járjuk be a gyökérelem jobb oldali részfáját preorder módon.



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok

Preorder bejárás

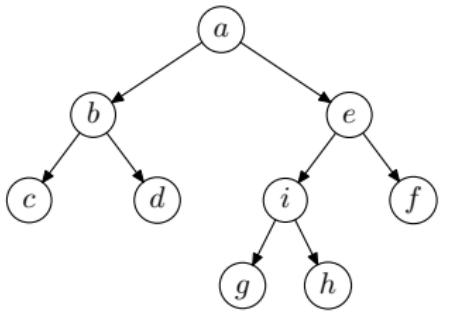
Inorder bejárás  
Postorder bejárás

Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

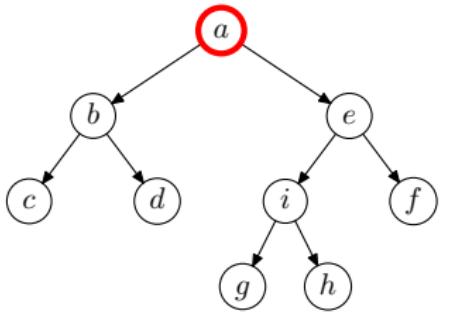
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

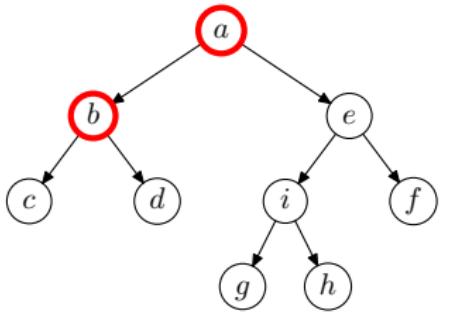
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

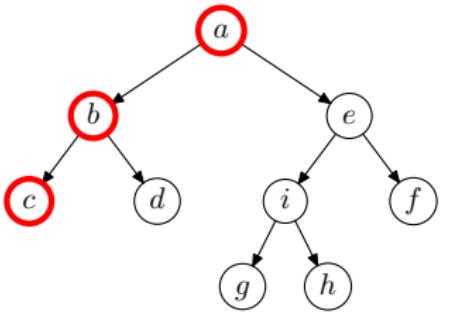
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Hierarchikus  
adatszerkezetek

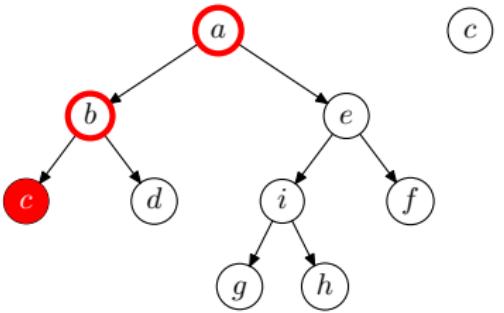
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

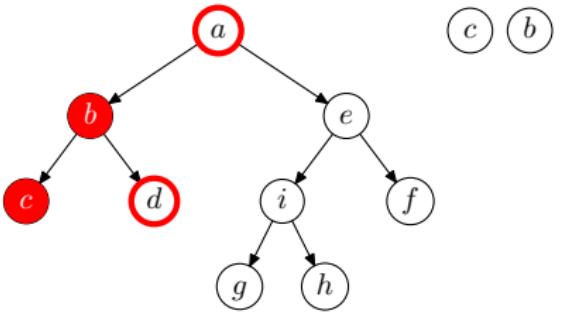
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

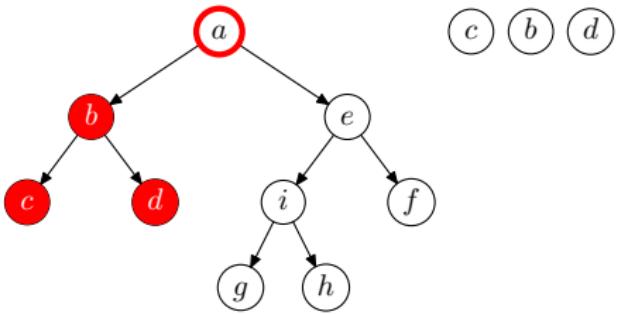
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

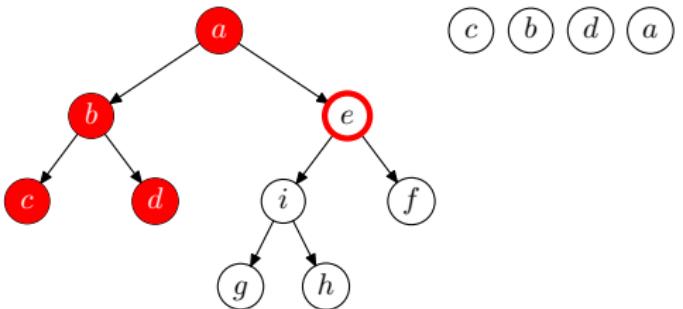
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

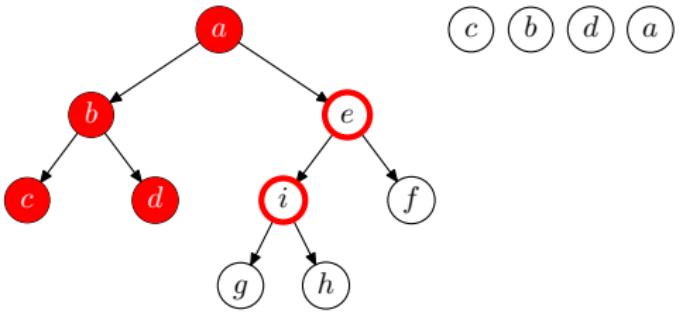
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

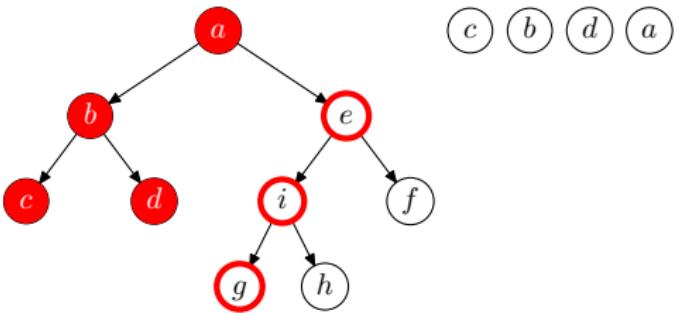
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

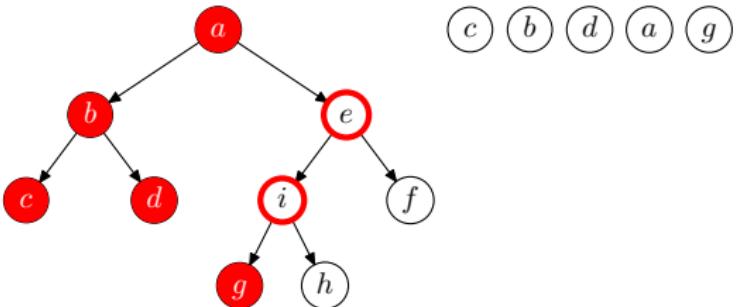
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

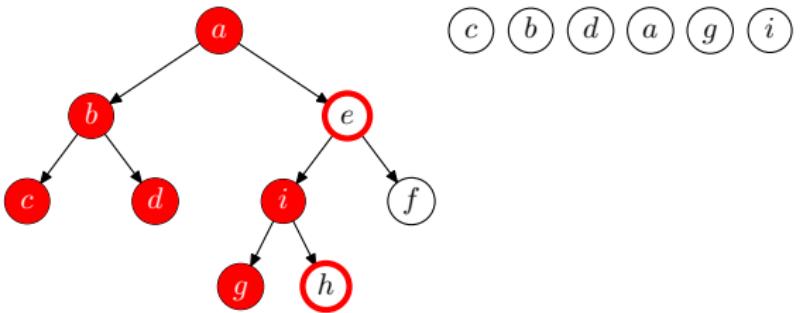
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

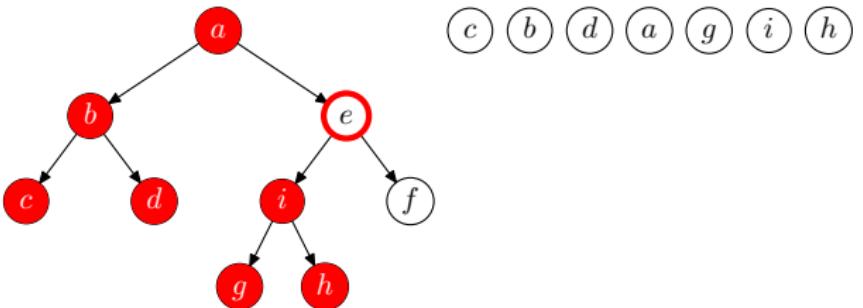
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

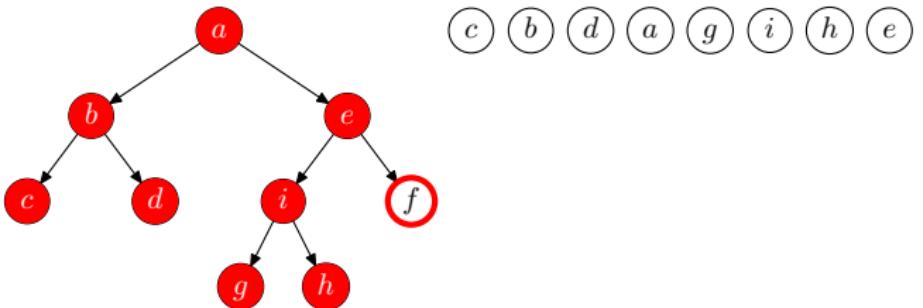
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- ③ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- ④ Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

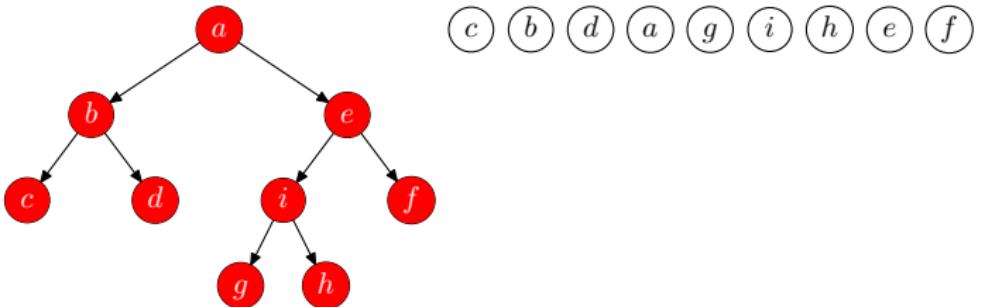
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Inorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelelem bal oldali részfáját inorder módon.
- 3 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).
- 4 Járjuk be a gyökérelelem jobb oldali részfáját inorder módon.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

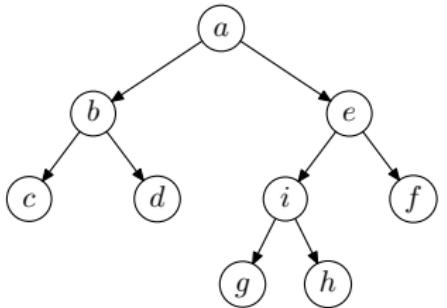
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
**Inorder bejárás**  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

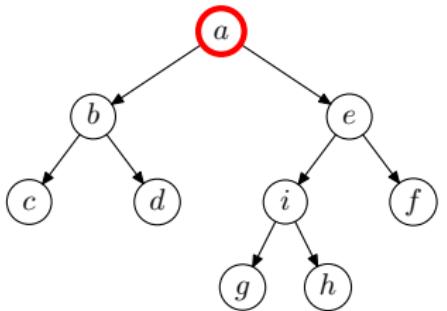
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

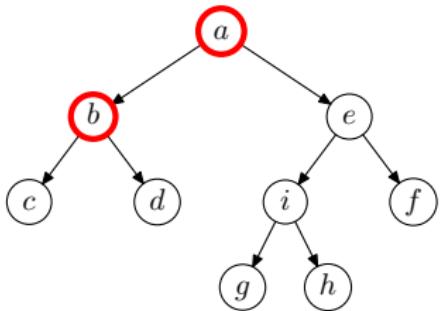
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

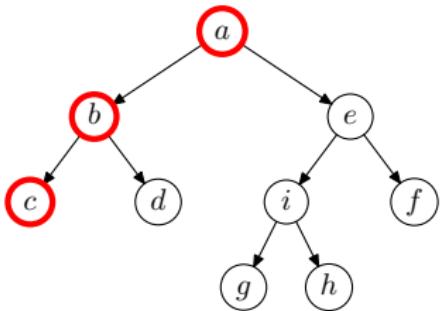
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

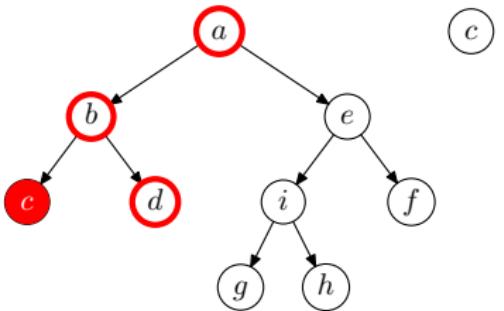
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

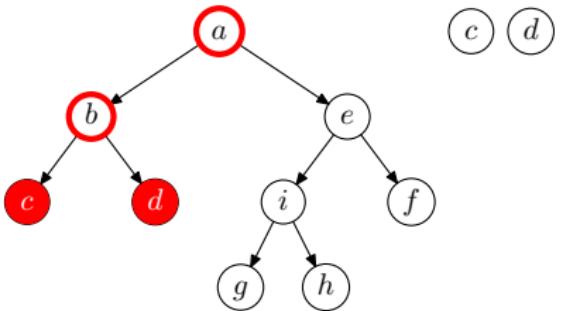
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

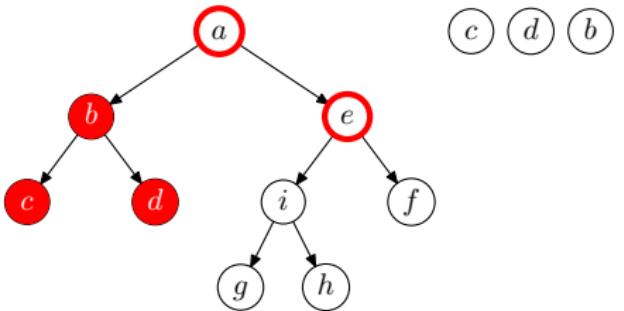
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

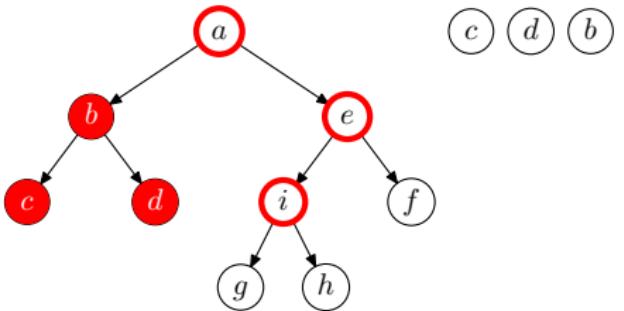
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

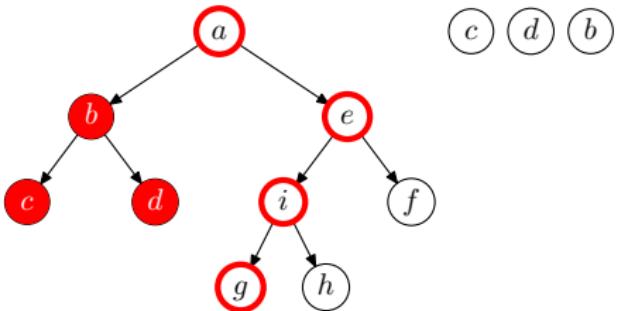
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

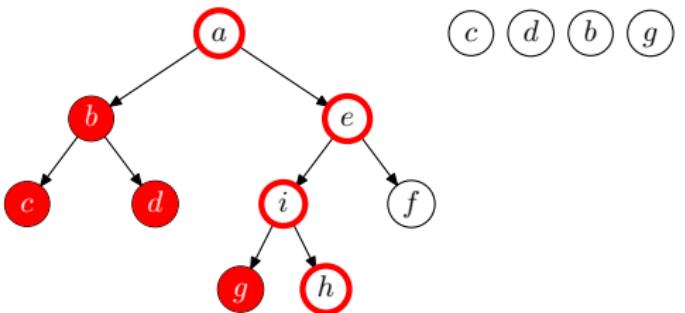
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- 1 Ha a bejárandó fa üres, az algoritmus véget ér.
- 2 Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- 3 Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- 4 Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

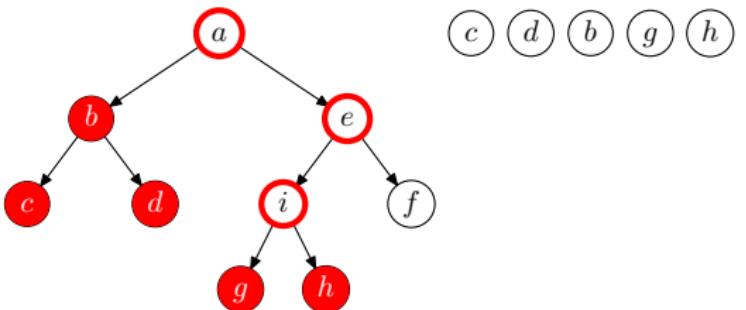
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

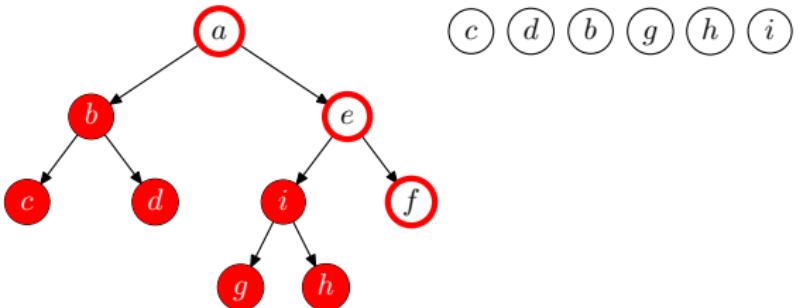
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

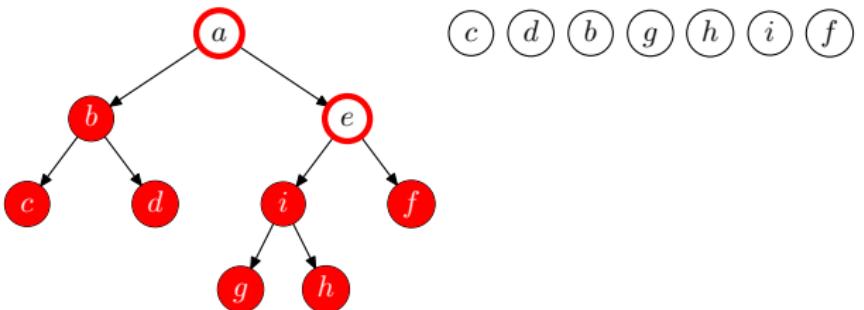
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

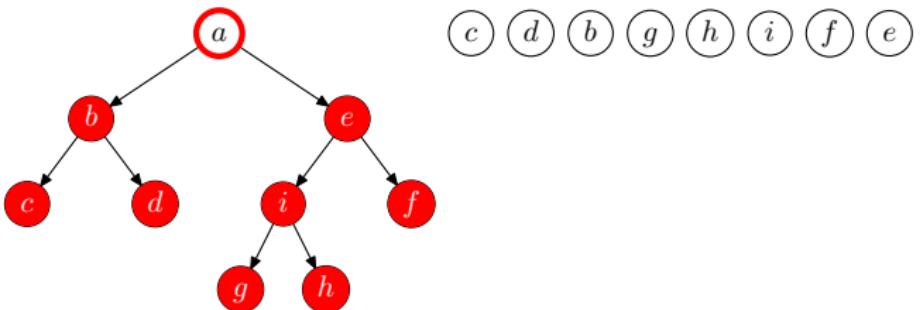
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

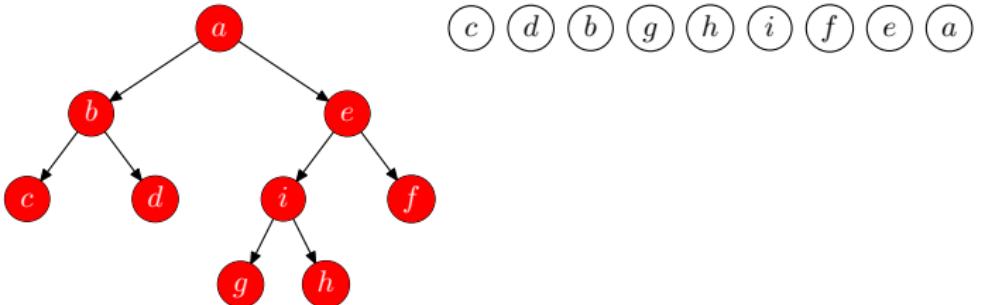
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Postorder bejárás algoritmusa

- ① Ha a bejárandó fa üres, az algoritmus véget ér.
- ② Járjuk be a gyökérelem bal oldali részfáját postorder módon.
- ③ Járjuk be a gyökérelem jobb oldali részfáját postorder módon.
- ④ Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük a gyökérelemet a sor végére).



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

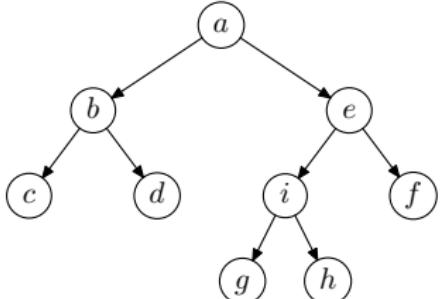
# Bináris fa folytonos reprezentációja

Hierarchikus  
adatszerkezetek

Három vektor segítségével, ahol a vektorok azonos indexű elemei a következő információkat tárolják:

- az ADAT vektorban az adatelem értékét,
- a BAL vektorban a bal oldali rákövetkező vektorbeli indexét,
- a JOBB vektorban a jobb oldali rákövetkező vektorbeli indexét.

Általában a fa gyökérelemét e vektorok első eleme írja le.



ADAT    BAL    JOBB

1.	a	2	5
2.	b	3	4
3.	c	0	0
4.	d	0	0
5.	e	6	9
6.	i	7	8
7.	g	0	0
8.	h	0	0
9.	f	0	0

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás

Reprezentáció

Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

# Bináris fa szétszort reprezentációja

Hierarchikus  
adatszerkezetek

A listaelemek **adatréssze** az adatelem értékét tartalmazza, a **mutatórész** pedig két mutatót: egyet, amely a bal oldali rákövetkezőt leíró listaelemet címzi, és egy másikat, amely a jobb oldali rákövetkezőt leíró listaelemet címzi. A gyökérelemhez (és rajta keresztül az adatszerkezet többi eleméhez) a „**gyökér**” mutató segítségével tudunk hozzáférni.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

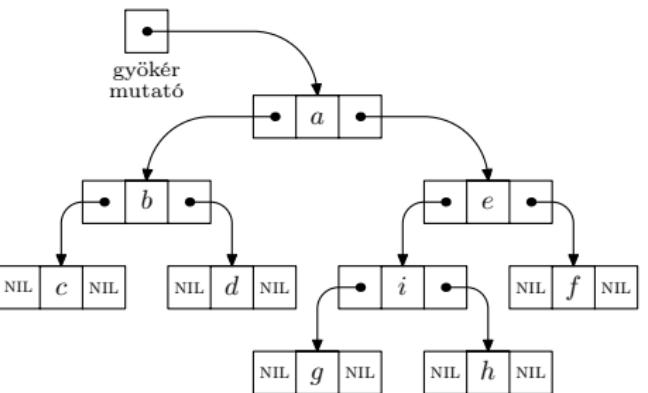
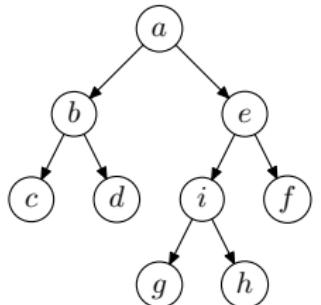
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás

Reprezentáció

Kifejezésfák  
Implementáció

Kupac

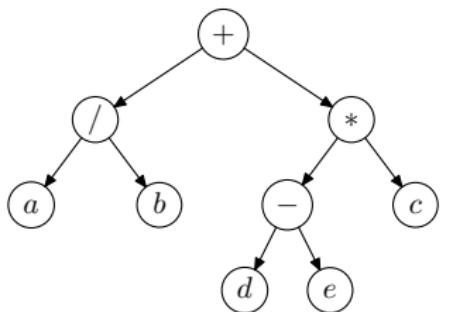
Kupacrendezés





## Kifejezésfa

A kifejezésfa olyan fa, melyben a levélelemek egy kifejezés operandusait, a nem levél elemek pedig ugyanazon kifejezés operátorait tartalmazzák.



prefix:  $+ / a b * - d e c$

infix:  $a / b + d - e * c$

postfix:  $a b / d e - c * +$

Aszerint, hogy a kifejezésfát – a korábban említettek közül – melyik bejárási algoritmussal járjuk be, kapjuk a kifejezés prefix, infix és postfix alakját.

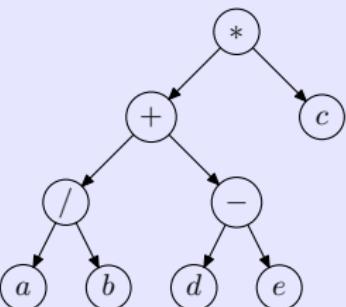
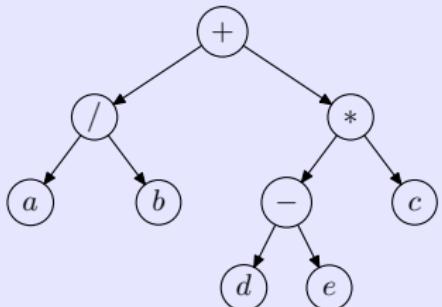
# Kifejezésfák és a bejárások kapcsolata

A prefix és a postfix alak egyértelmű, az infix nem az (de zárójelek használatával azzá tehető).

## Példa

Az alábbi két kifejezésfát inorder módon bejárva az infix kifejezést kapjuk:

$$a / b + d - e * c$$



Zárójelezést alkalmazva:

$$(a / b) + ((d - e) * c)$$

$$((a / b) + (d - e)) * c$$



- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák**
- Implementáció

# Bináris fa implementálása

## Üres fa létrehozása

```
typedef struct faelem {  
    tipus adat;  
    struct faelem *bal;  
    struct faelem *jobb;  
} FAELEM;
```

```
FAELEM *gyoker = NULL;
```

## Bináris fa bejárása inorder stratégiával

```
void inorder_bejaras(FAELEM *csucs)  
{  
    if (csucs != NULL)  
    {  
        inorder_bejaras(csucs->bal);  
        feldolgoz(csucs);  
        inorder_bejaras(csucs->jobb);  
    }  
}
```

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

### A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

# A kupac definíciója

## Kupac

A kupac olyan fa, amely rendelkezik a **kupac tulajdonság**gal: a gyökérelemet kivéve bármely adatelemének a kulcsa kisebb vagy egyenlő az adatelem szülőjének a kulcsánál. Az ilyen fában a legnagyobb kulcsú elem mindig a gyökérelem, ezért **max-kupacnak** is nevezzük. Ha megfordítjuk a relációt, akkor a gyökérelem lesz a legkisebb kulcsú elem, ekkor **min-kupacot** kapunk.

## Megjegyzés

A kupac egyes elemeiben a gyermek csomópontok számára nézve általában nincs megszorítás. A kupac adatszerkezetnek rengeteg változata létezik attól függően, hogy hány gyermek csomópontja lehet az egyes elemeknek.

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

- Kupacrendezés

# Bináris kupac

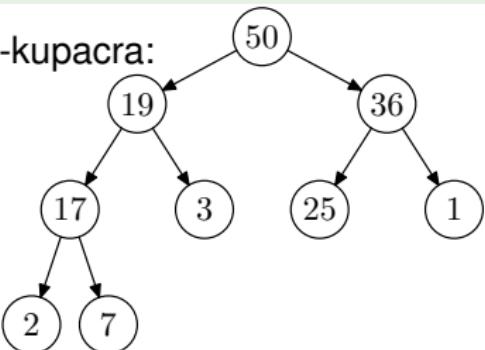
## Bináris kupac

A bináris kupac olyan bináris fa, amely a **kupac tulajdonság**on kívül az **alak tulajdonság**gal is rendelkezik: a fa **teljes bináris fa**, azaz minimális magasságú, és ha a legalsó szint nincs teljesen kitöltve, akkor azon a szinten a csomópontok balról jobbra kerülnek feltöltésre.

## Megjegyzés

A kupac tulajdonság nem határozza meg a gyermek csomópontok sorrendjét, ezért azok tetszőlegesen felcserélhetők, hacsak meg nem sértik az alak tulajdonságot.

Példa bináris max-kupacra:

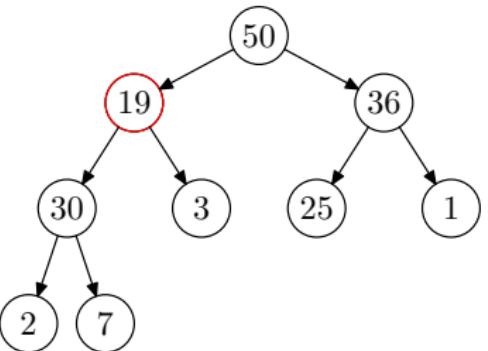


- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

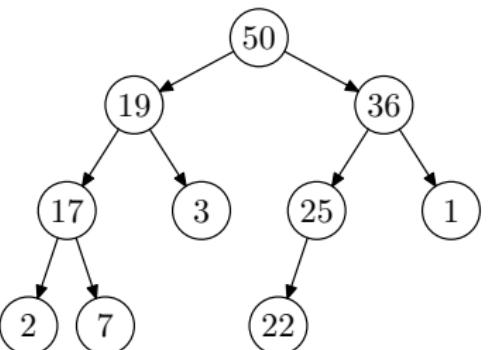
- Kupacrendezés

## Ellenpéldák

Sérül a kupac tulajdonság:



Sérül az alak tulajdonság:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

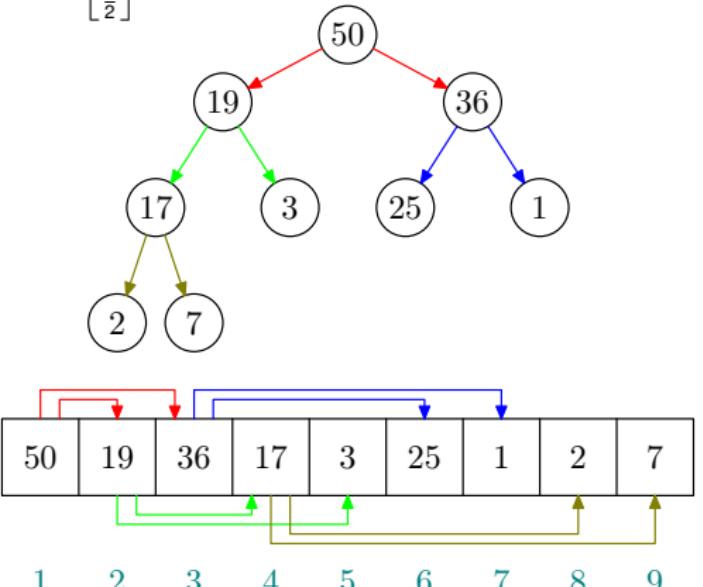
Kupacrendezés



## A bináris kupac reprezentációja

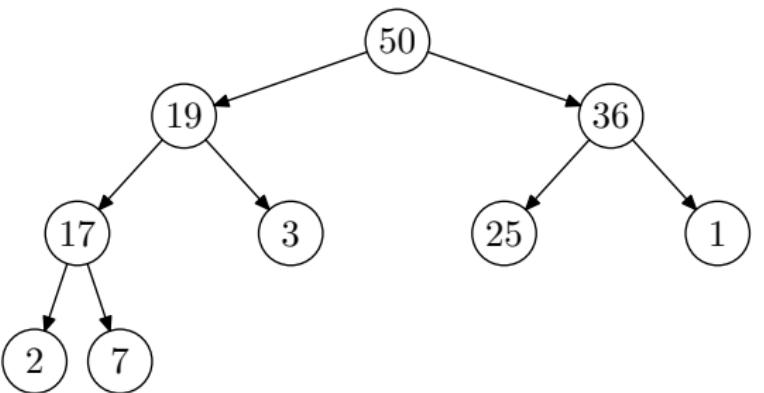
Az alak tulajdonság miatt a bináris kupacot leggyakrabban egy tömbbel reprezentáljuk. Nincs szükség mutatókra, mivel bármely adatelem szülőjének és gyermekeinek az indexe egyszerű számtani műveletekkel meghatározható az adatelem indexéből. Ha a tömb indexelése 1-ről indul, akkor az  $A_i$  elem

- gyermekei az  $A_{2i}$  és az  $A_{2i+1}$ ,
- szülője az  $A_{\lfloor \frac{i}{2} \rfloor}$  elem lesz.



# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

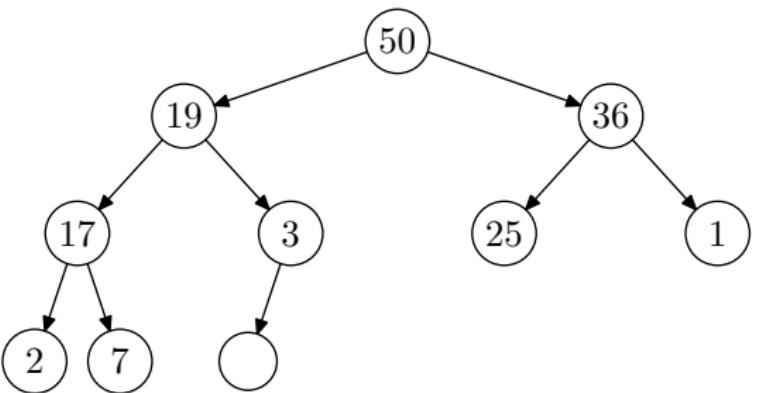
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

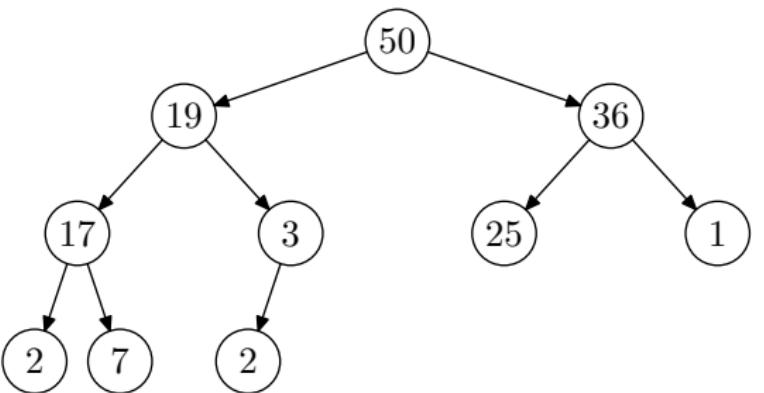
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

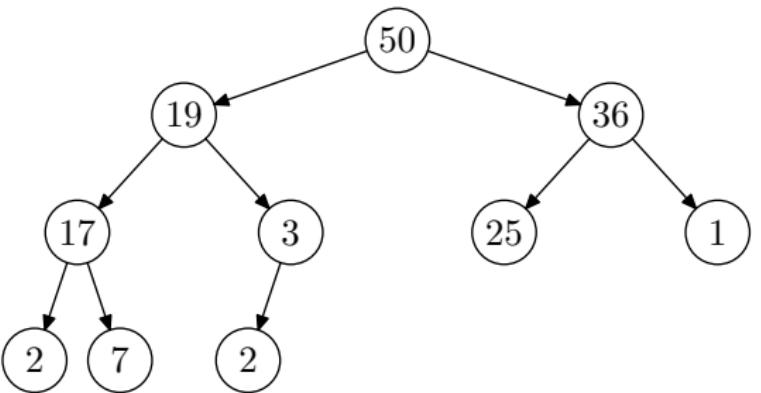
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

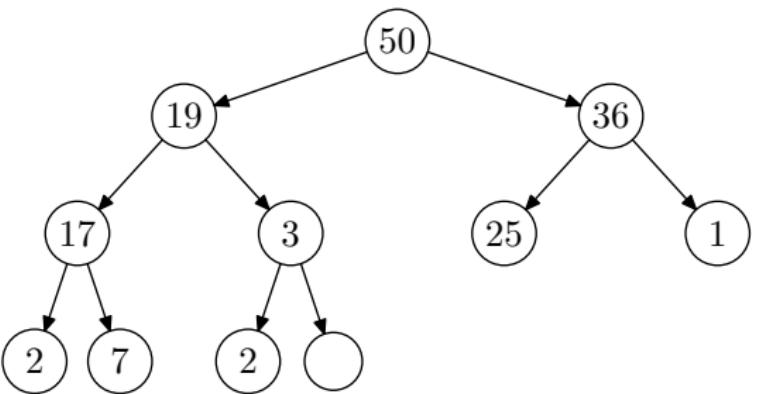
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

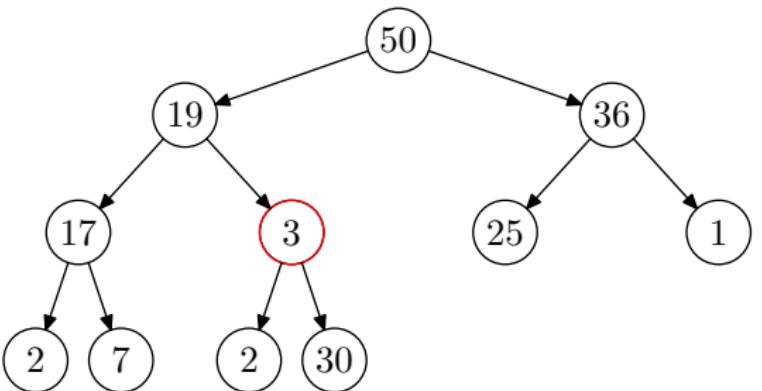
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

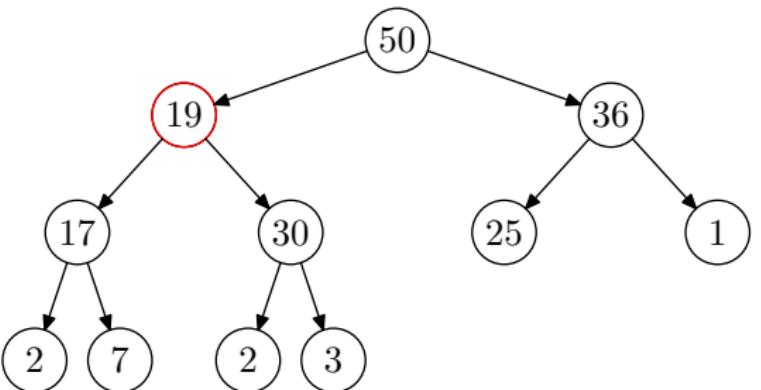
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

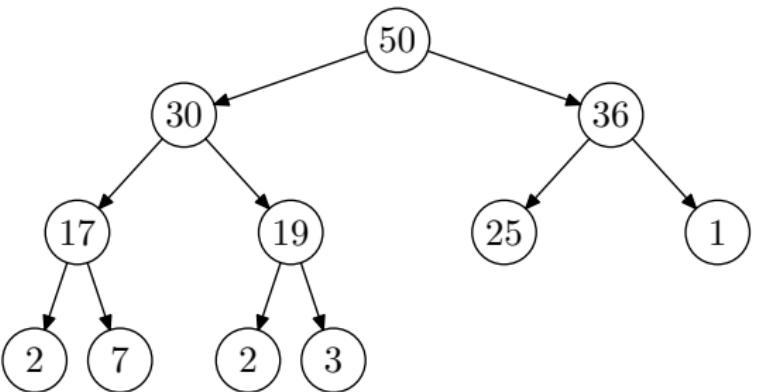
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

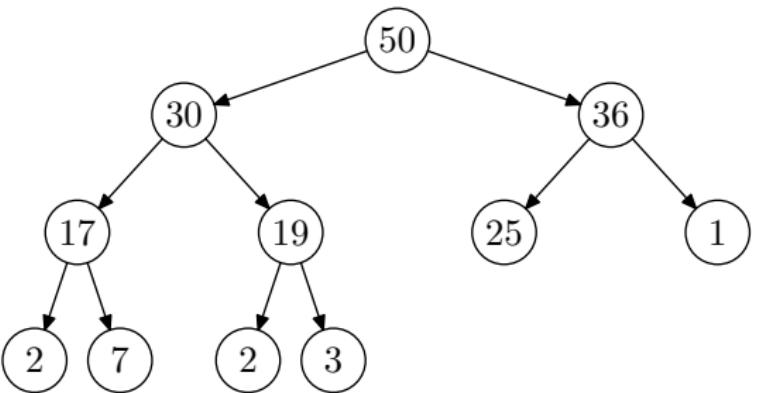
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

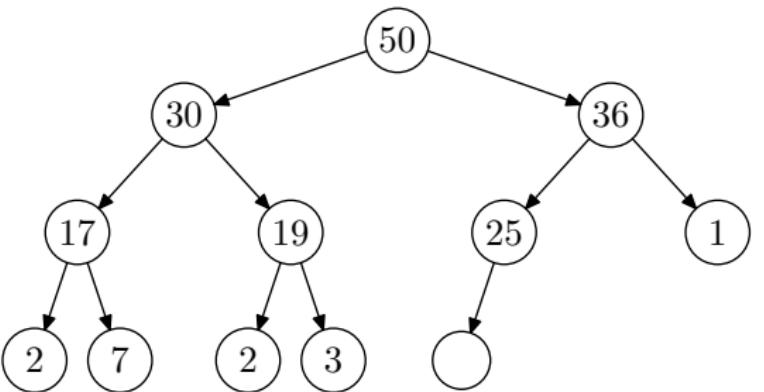
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

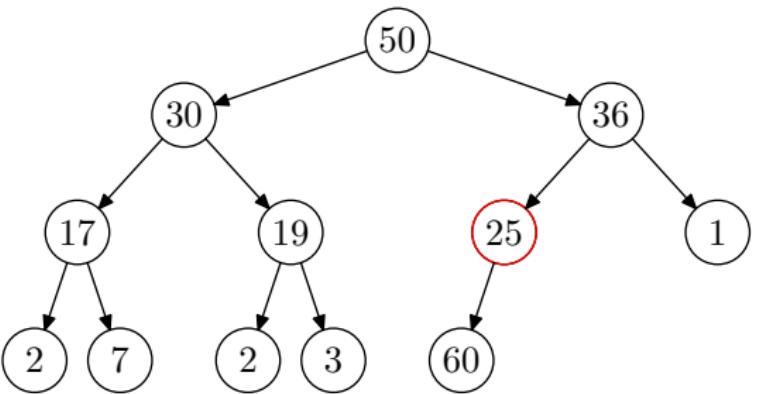
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

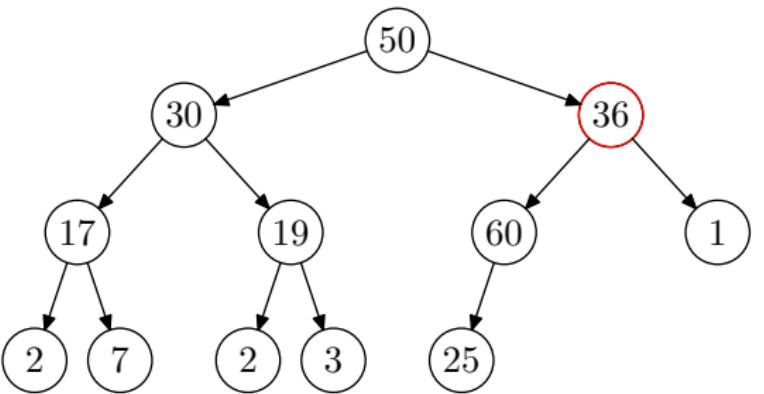
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

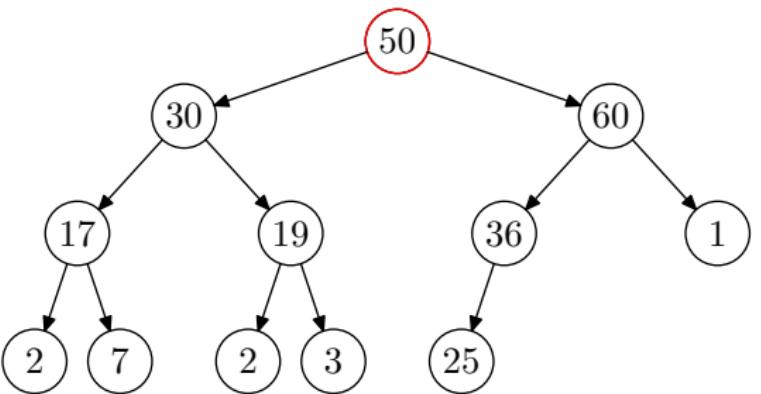
# Beszúrás bináris kupacba

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

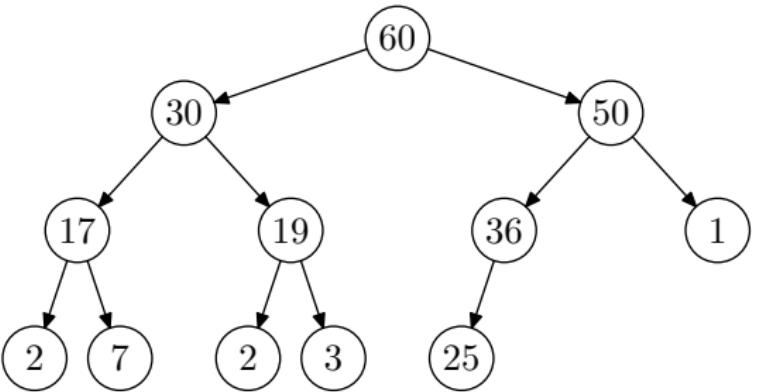
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Beszúrás bináris kupacba

- beszúr: 2
- beszúr: 30
- beszúr: 60



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- Legyen  $i$  az  $A$  tömb egy olyan elmének az indexe, amelyre:
  - Az  $A[i]$  elem baloldali és jobboldali részfái kupacok.
  - Elképzelhető viszont, hogy  $A[i]$  kisebb mint a gyerekei, vagyis sérül a kupac tulajdonság.
- A kupacosítás során az  $i$ . pozíión lévő bináris részfából kupacot csinálunk úgy, hogy az  $A[i]$  elemet lefelé mozgatjuk a kupacban.

Hierarchikus  
adatszerkezetek

A fa adatszerkezet

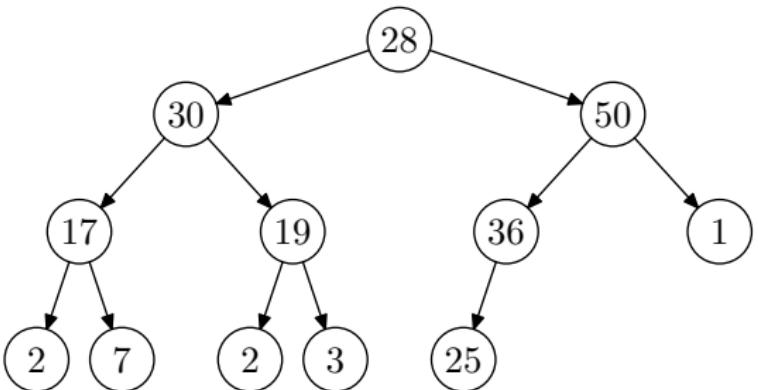
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- **kupacosít(1)**



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

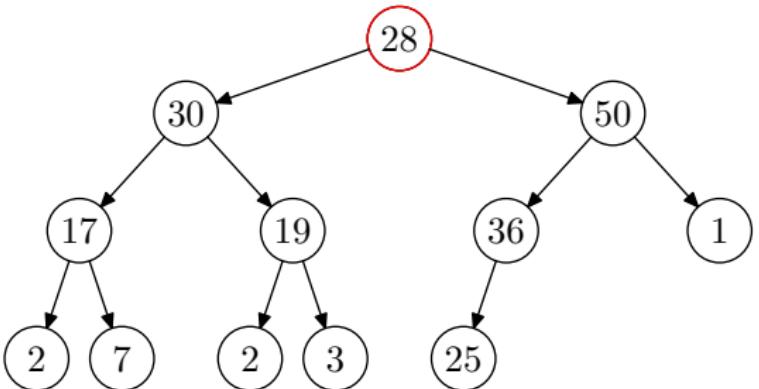
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- **kupacosít(1)**



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

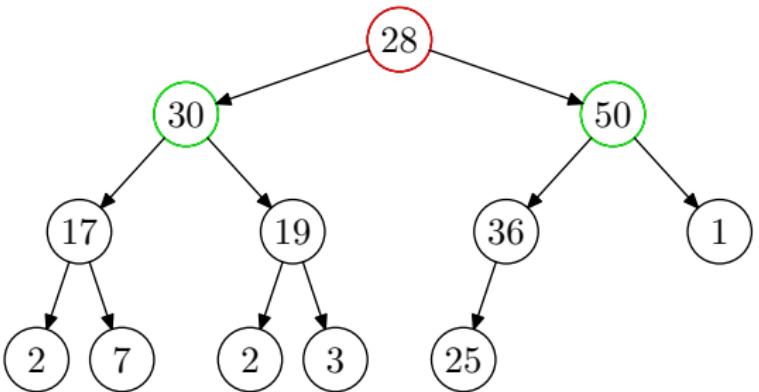
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- **kupacosít(1)**



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

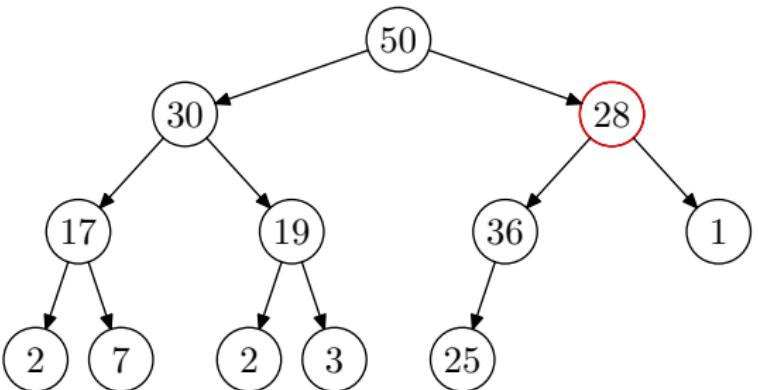
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- kupacosít(1)



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

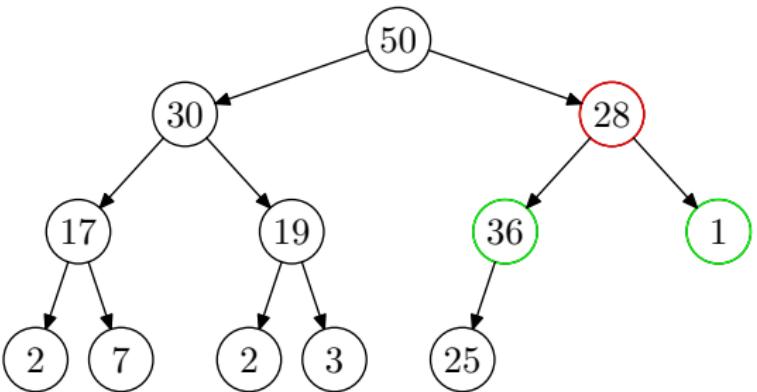
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- kupacosít(1)



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

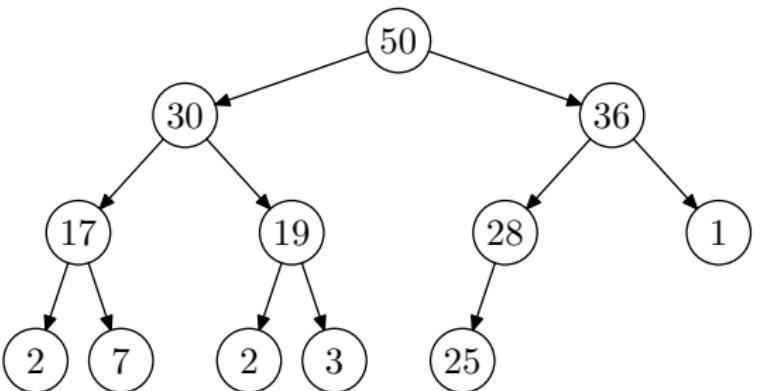
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Kupacosítás

- A kupac tulajdonság sérül az 1-es indexű elem esetében.  
A 2-es és 3-as indexű részfák viszont kupacok.
- kupacosít(1)



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

## Maximális elem törlése (#1)

- A legnagyobb elem a kupac tetején található. Ez a kupac gyökere.
- Ezt kitörölhetjük, s felhozzuk a helyére az egyik gyerekét.
- Az üres hely lefelé mozog a fában.
- Az üres hely bárhová kerülhet az utolsó szinten.
- Az így létrejövő fa utolsó szintjén az elemek jó eséllyel nem lesznek balra rendezettek, vagyis sérül az alak tulajdonság.

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



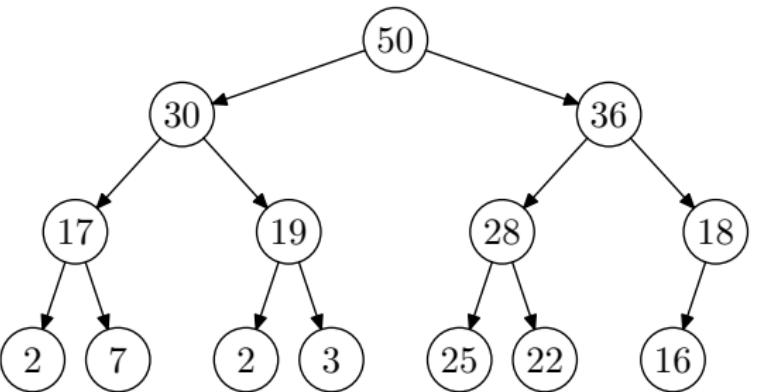
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



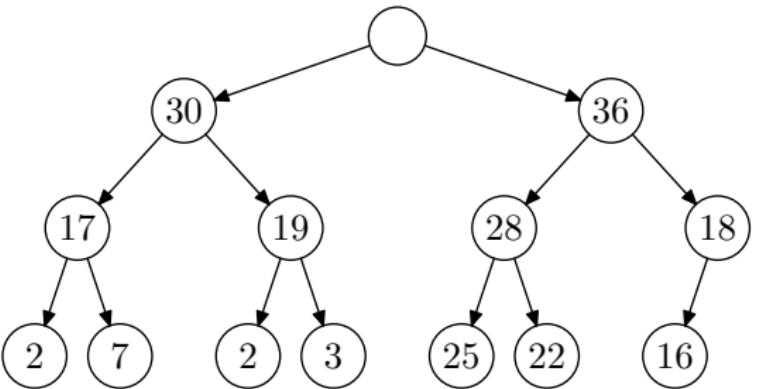
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



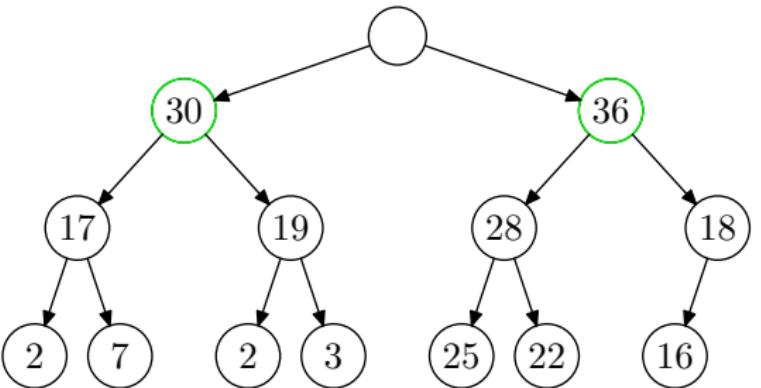
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



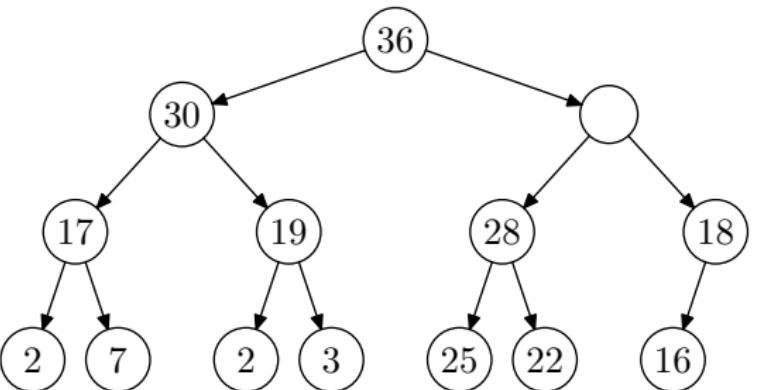
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



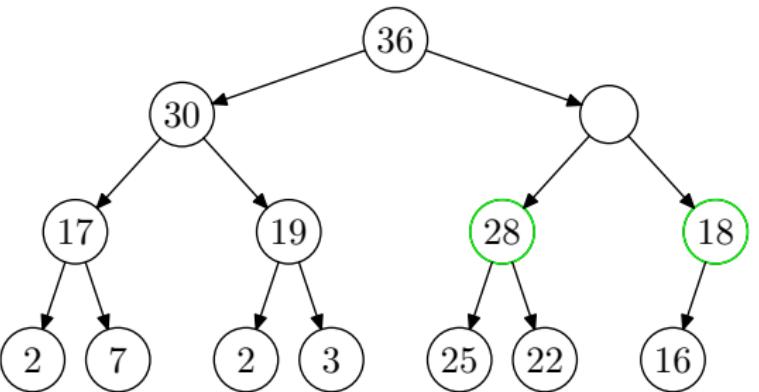
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



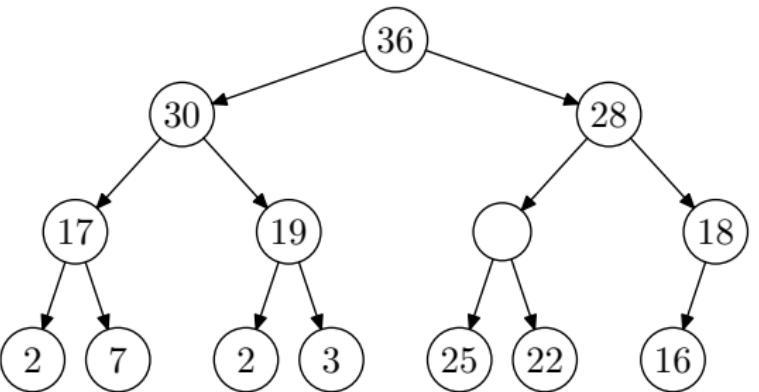
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



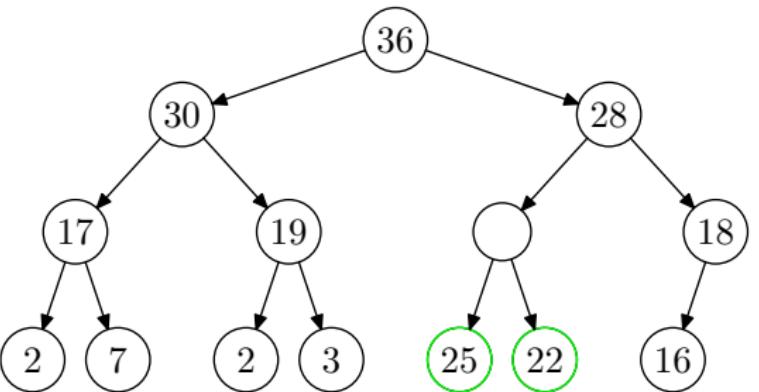
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



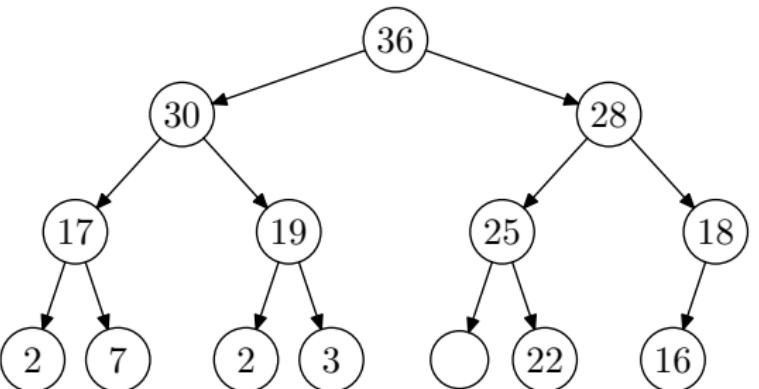
Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés



# Maximális elem törlése (#1)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

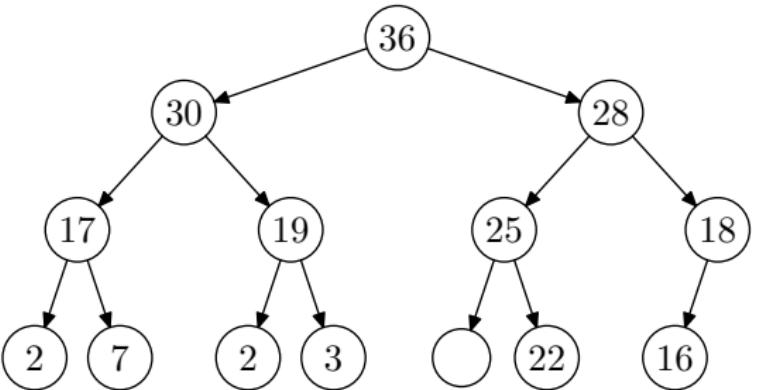


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



Nem teljesül az alak tulajdonság.

## Maximális elem törlése (#2)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

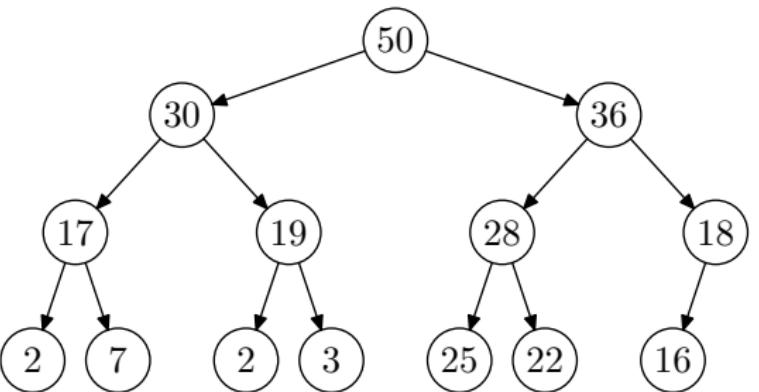


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

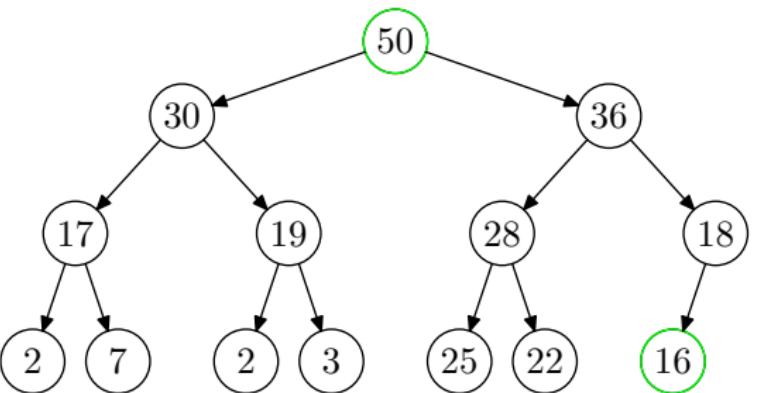


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- `kupacosít(1)`

## Maximális elem törlése (#2)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

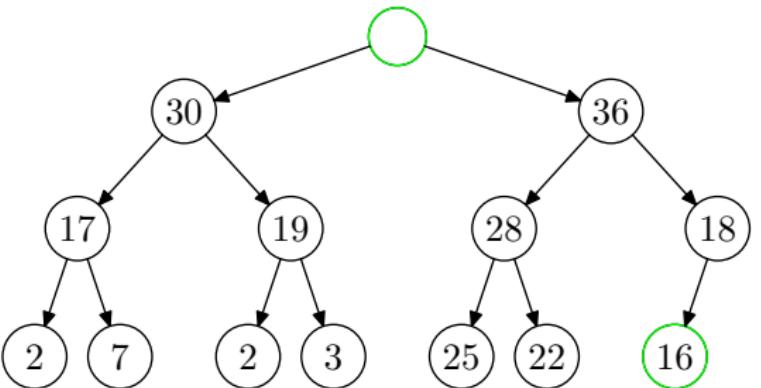


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- `kupacosít(1)`

## Maximális elem törlése (#2)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

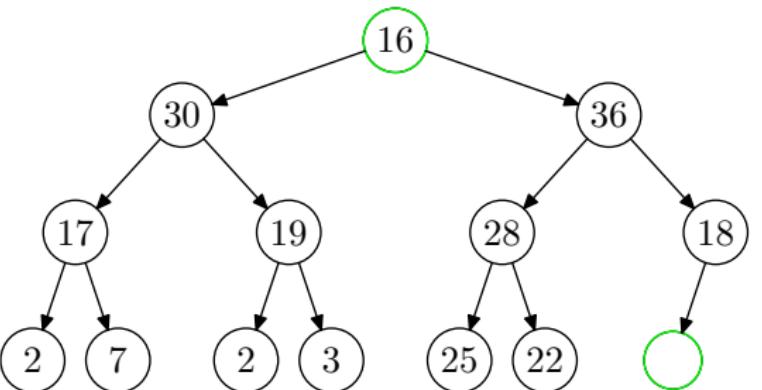


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

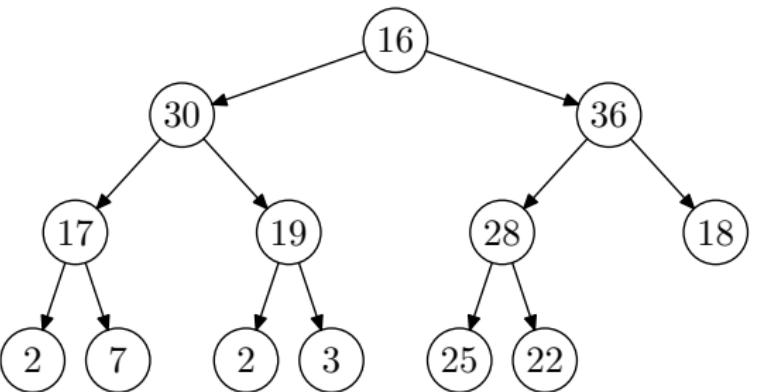
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés



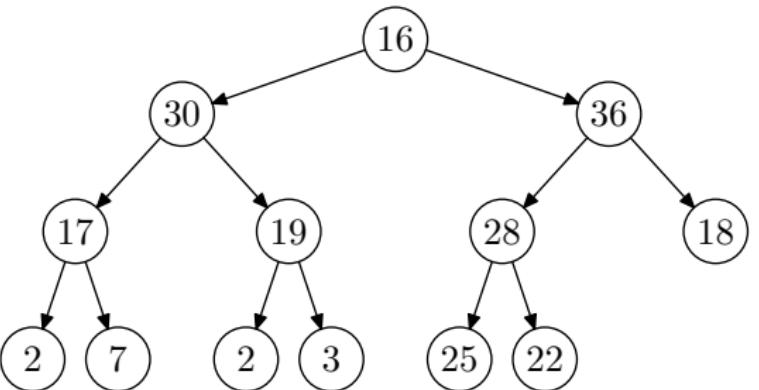
- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- `kupacosít(1)`

## Maximális elem törlése (#2)



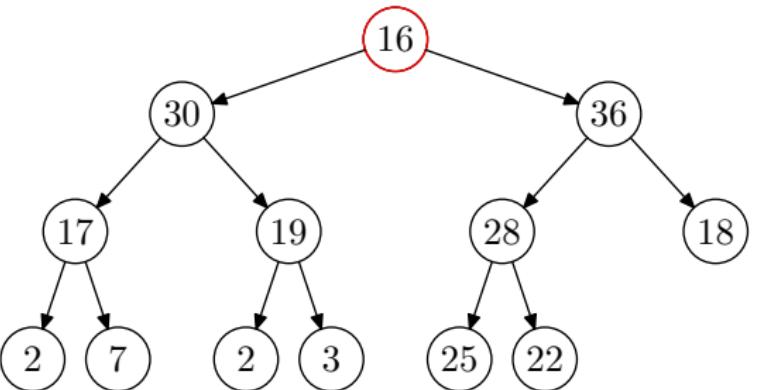
- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- `kupacosít(1)`

## Maximális elem törlése (#2)



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor

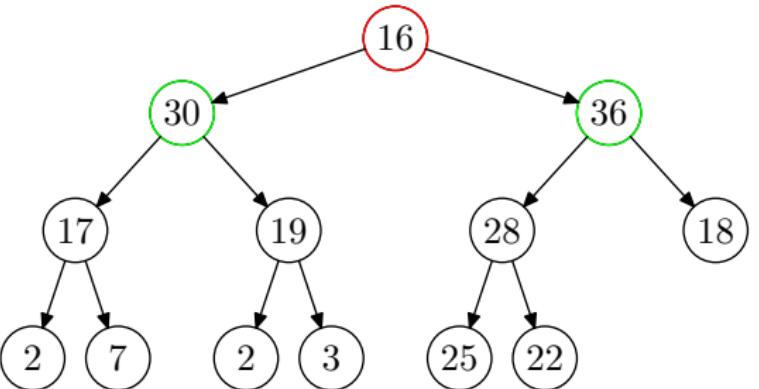


Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

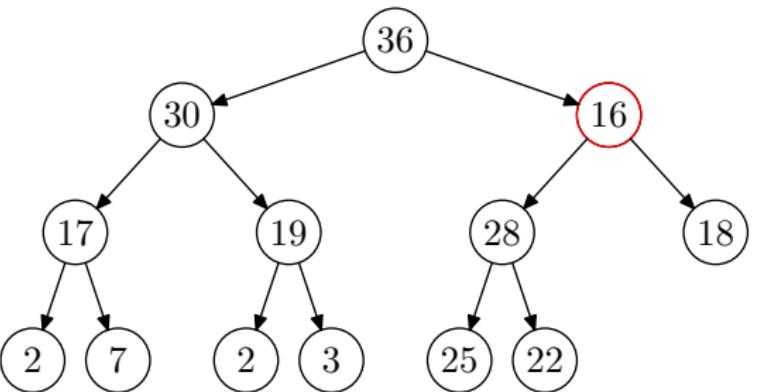


- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)

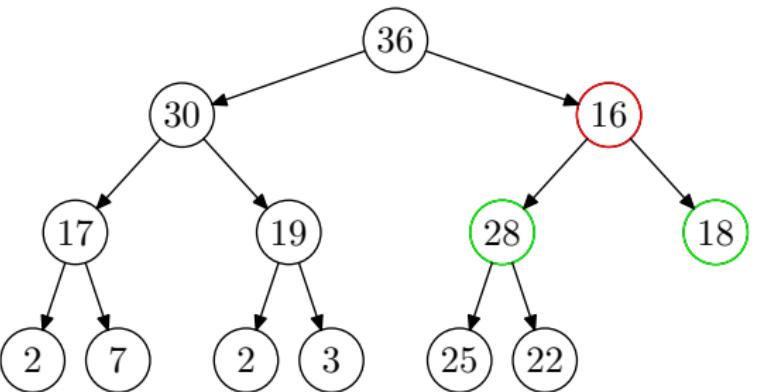


Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció



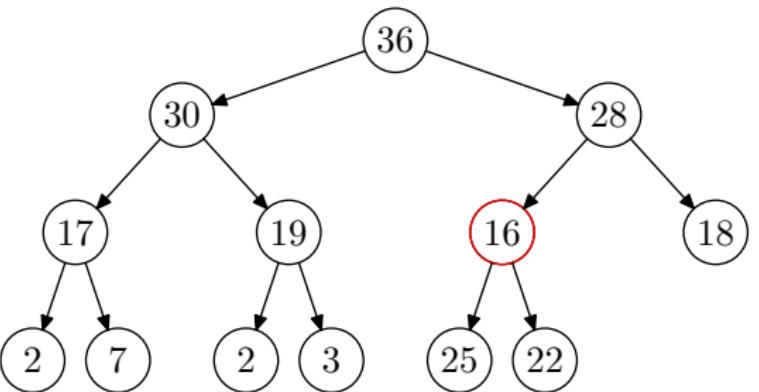
- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)



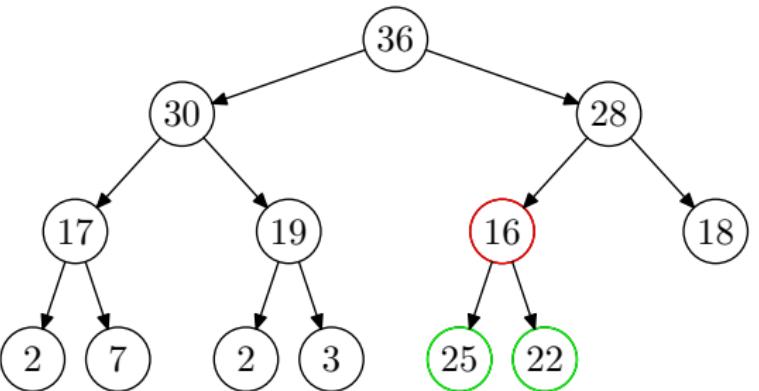
- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)



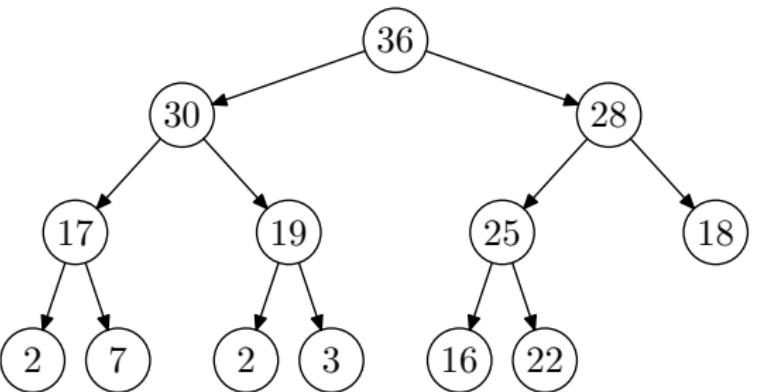
- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)



- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

## Maximális elem törlése (#2)

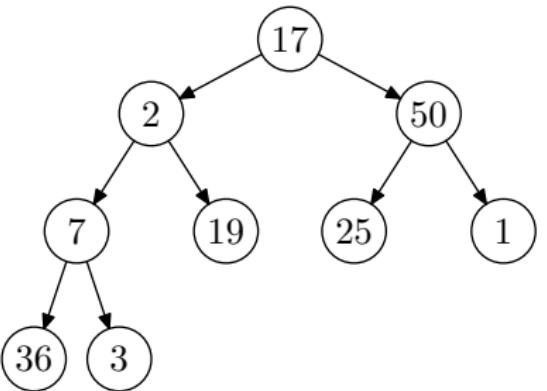


- Cseréljük ki a kupac tetején lévő elemet a kupac utolsó elemével.
- kupacosít(1)

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



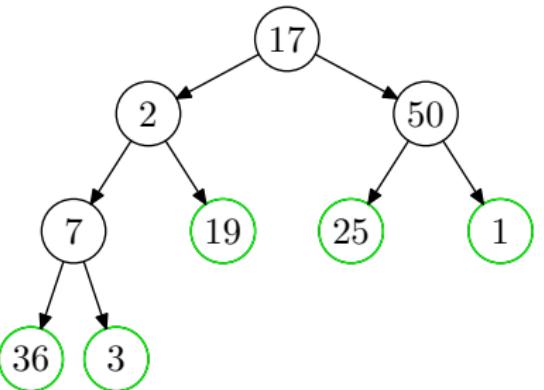
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupacrendezés

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```

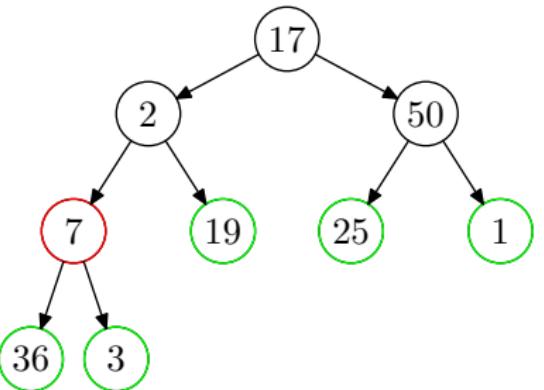


Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```

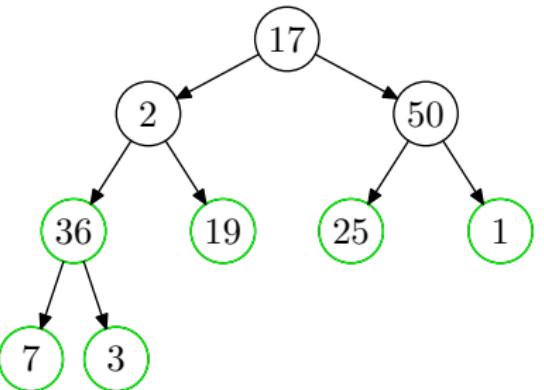


Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```

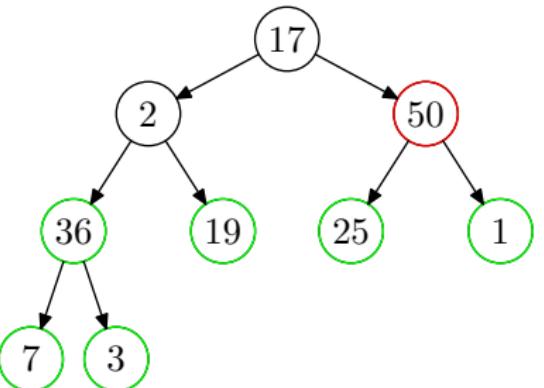


Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```

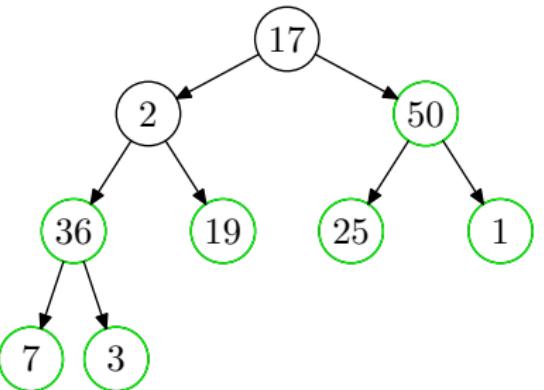


# Kupac építése

Hierarchikus  
adatszerkezetek

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



Kóska Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

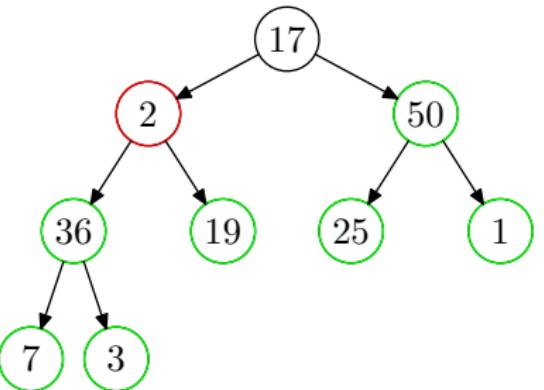
Kupac

Kupacrendezés

# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

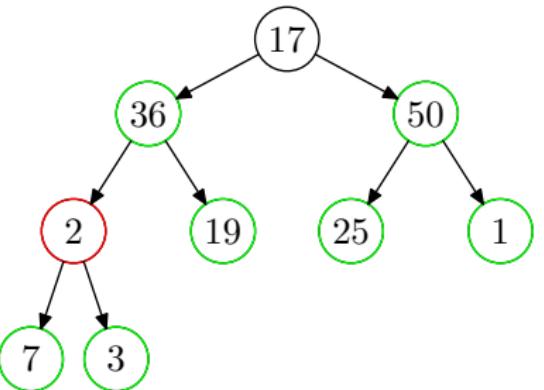
```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

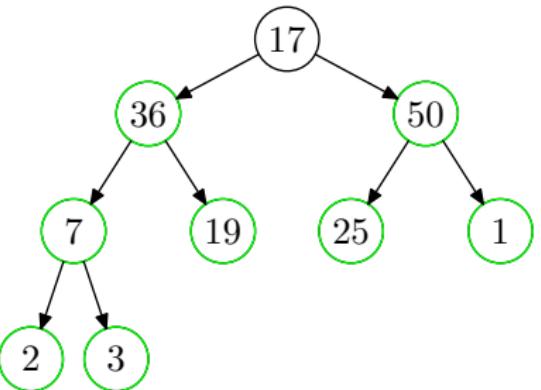
```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

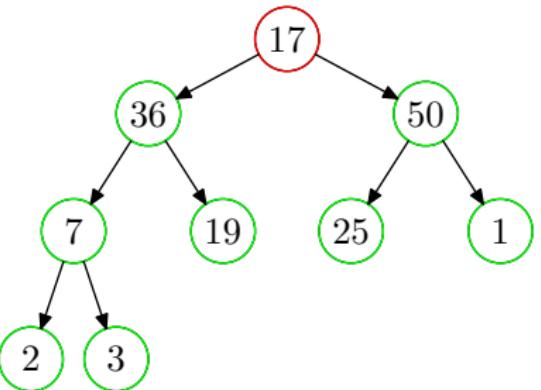
```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

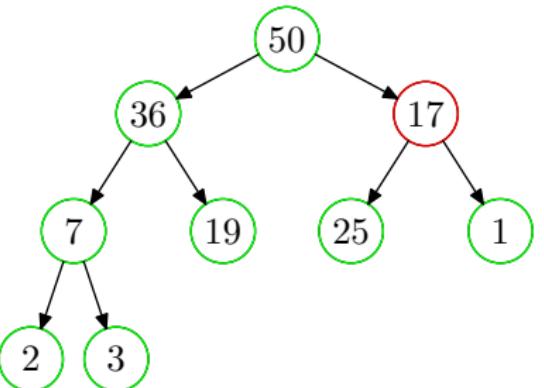
```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

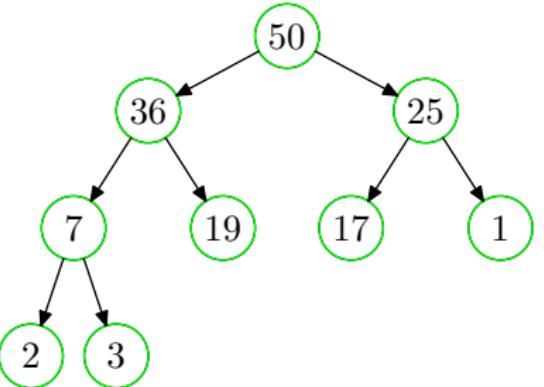
```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac építése

- A legalsó szinten kezdünk s onnan haladunk felfelé, jobbról balra haladva.
- Kezdetben minden levélelem kupac.

```
1: procedure KUPAC-ÉPÍTÉSE(A)
2:   for i  $\leftarrow$  [n/2] downto 1 do
3:     KUPACOSÍT(A, i)
4:   end for
5: end procedure
```



# Kupac rendezés

A kupac segítségével viszonylag egyszerűen implementálható egy olyan helyben rendező algoritmus, amely általános esetben majdnem olyan gyors, mint a gyorsrendezés, a legrosszabb esetben viszont gyorsabb annál.

## Az ötlet

Először is bináris max-kupaccá alakítjuk a rendezendő tömböt. Ezután kicseréljük a tömb első (legnagyobb) elemét az utolsóval, amely így a helyére kerül. Helyreállítjuk a bináris max-kupacot az utolsó elem elhagyásával kapott résztömbben, majd kicseréljük az első elemet az utolsó előttivel, és így tovább...

## Megjegyzés

Az algoritmus alatt a tömb eleje tartalmazza a kupacot, a vége pedig a már rendezett résztömböt.

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

## A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

**Kupac**  
Kupac rendezés



## A kupacrendezés algoritmusa

Az algoritmus bemeneteként adott adatszerkezetet  $A$ -val, elemeinek a számát  $n$ -nel jelöljük.

```
1: procedure KUPACRENDEZÉS( $A$ )
2:   KUPACOSÍT( $A$ )
3:   vég  $\leftarrow n$ 
4:   while vég  $> 1$  do
5:     CSERÉL( $A$ , 1, vég)
6:     SZITÁL( $A$ , 1, vég - 1)
7:     vég  $\leftarrow$  vég - 1
8:   end while
9: end procedure
```

```
1: procedure KUPACOSÍT( $A$ )
2:   start  $\leftarrow [n/2]$ 
       $\triangleright$  start kezdetben az utolsó nem levél elem indexe
3:   while start  $\geq 1$  do
4:     SZITÁL( $A$ , start, n)
5:     start  $\leftarrow$  start - 1
6:   end while
7: end procedure
```

# A szitálás algoritmusa

```
1: procedure SZITÁL(A, bal, jobb)
2:   gyökér  $\leftarrow bal
3:   while gyökér * 2  $\leq$  jobb do
4:     gyerek  $\leftarrow$  gyökér * 2
          ▷ gyerek a gyökér bal oldali gyermeke
5:     csere  $\leftarrow$  gyökér
          ▷ csere a gyökér azon gyermeke, amelyikkel ki kell őt cserélni
6:     if A[csere] < A[gyerek] then
7:       csere  $\leftarrow$  gyerek
8:     end if
9:     if gyerek < jobb and A[csere] < A[gyerek + 1] then
10:      csere  $\leftarrow$  gyerek + 1
11:    end if
12:    if csere  $\neq$  gyökér then
13:      CSERÉL(A, gyökér, csere)
14:      gyökér  $\leftarrow$  csere
15:    else
16:      return
17:    end if
18:  end while
19: end procedure$ 
```

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

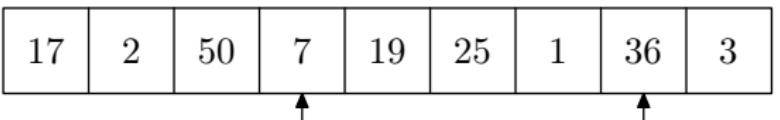
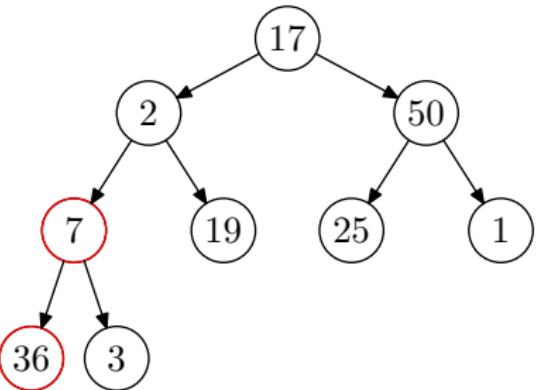
A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

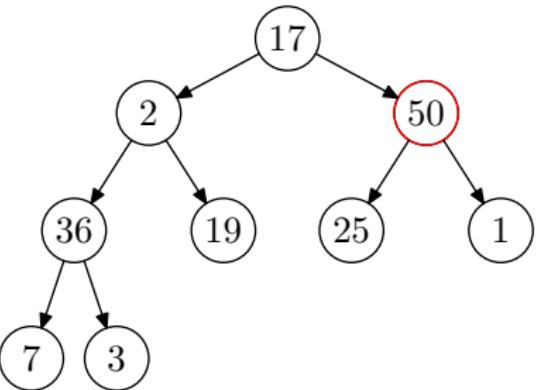
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



17	2	50	36	19	25	1	7	3
----	---	----	----	----	----	---	---	---

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

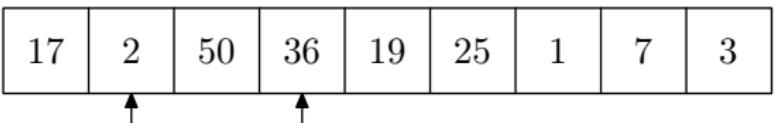
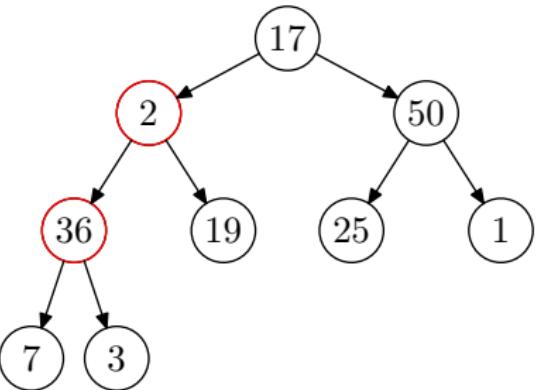
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

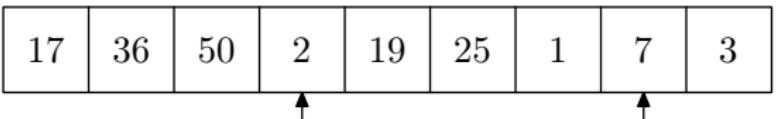
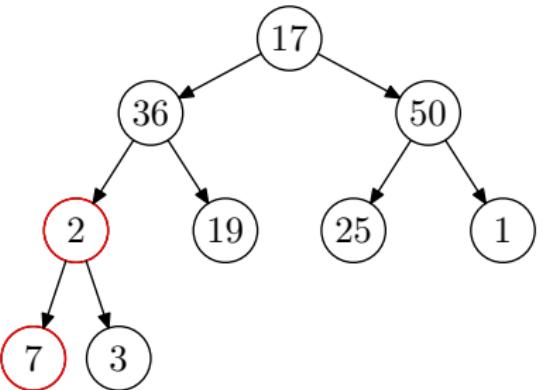
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

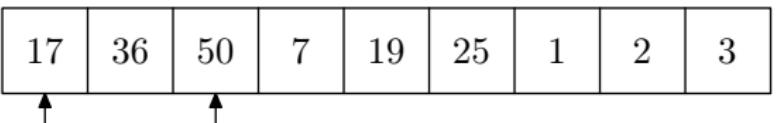
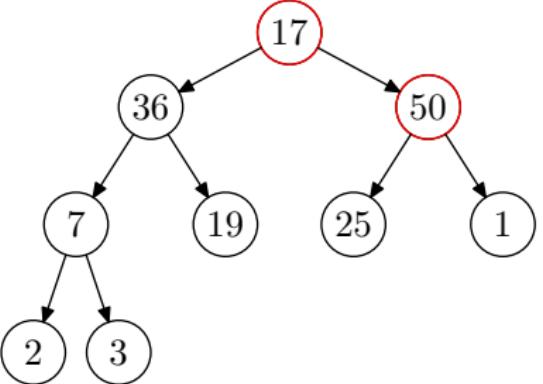
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

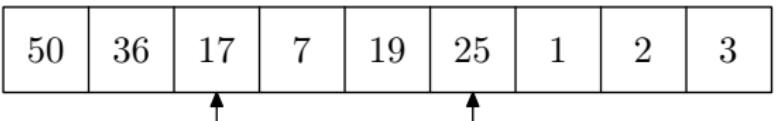
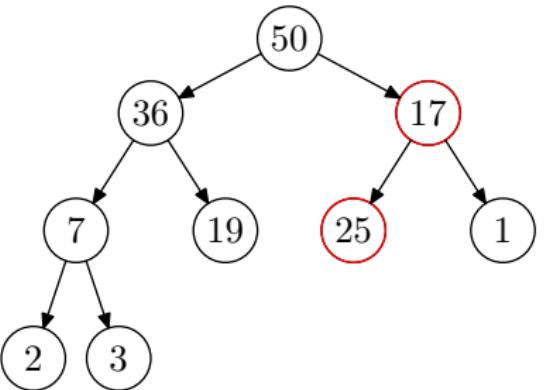
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

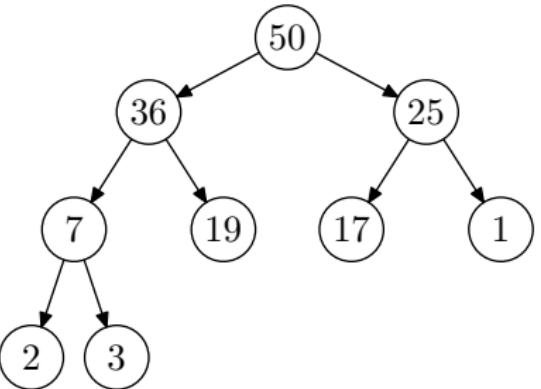
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Kupacosítás:



50	36	25	7	19	17	1	2	3
----	----	----	---	----	----	---	---	---

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

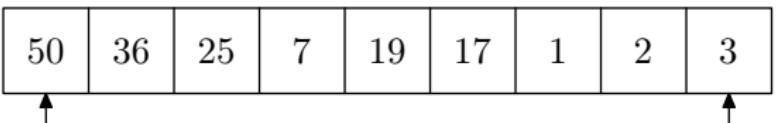
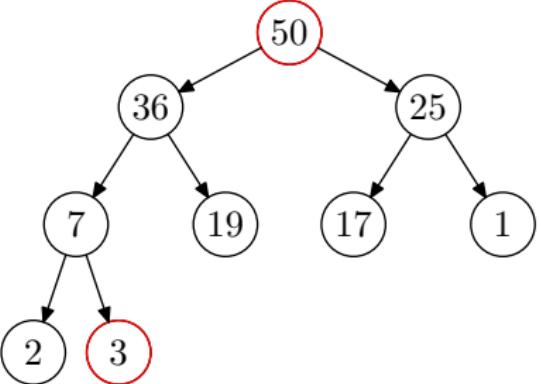
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

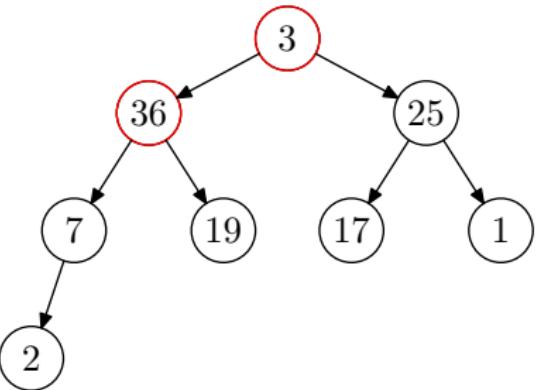
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



3	36	25	7	19	17	1	2	50
↑↑								

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

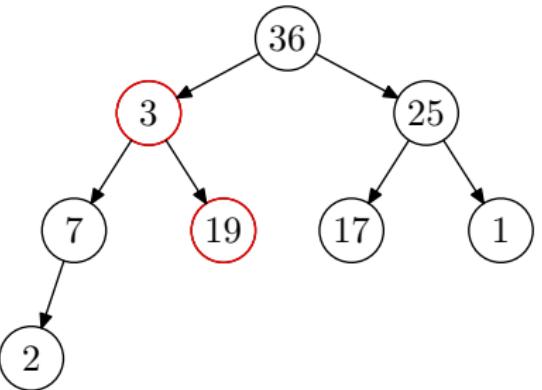
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



36	3	25	7	19	17	1	2	50
							↑	↑

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

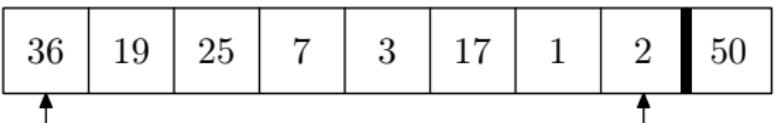
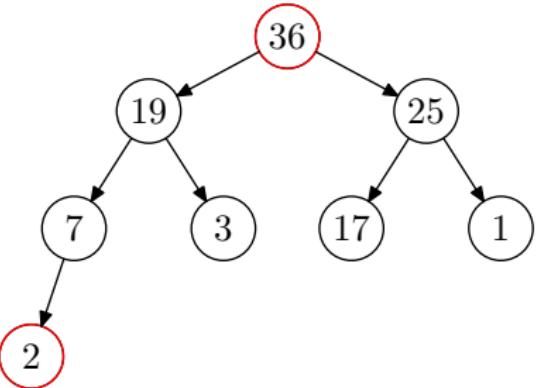
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

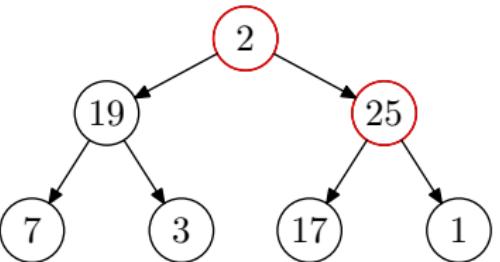
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



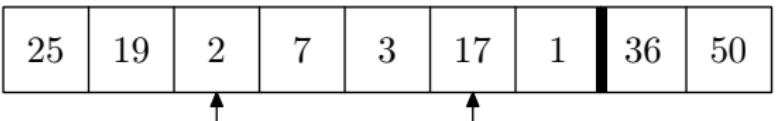
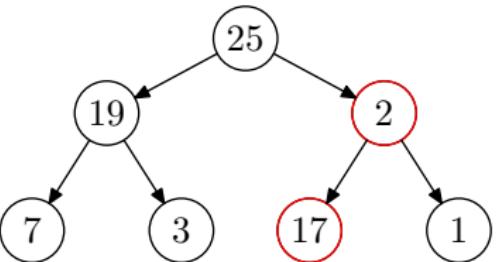
2	19	25	7	3	17	1	36	50
↑      ↑								



- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

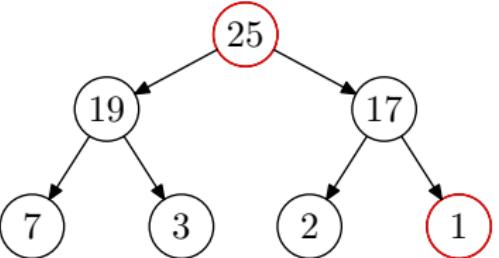
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

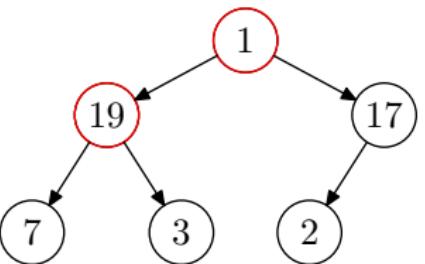
Kupac

Kupacrendezés

25	19	17	7	3	2	1	36	50
							↑	↑

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

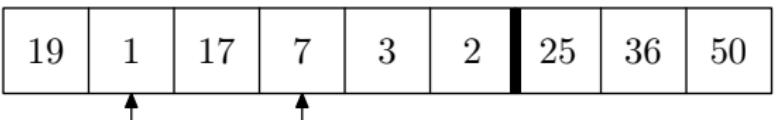
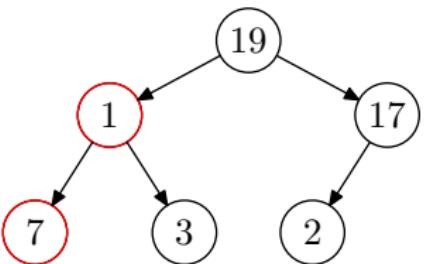
Kupac

Kupacrendezés

1	19	17	7	3	2	25	36	50

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

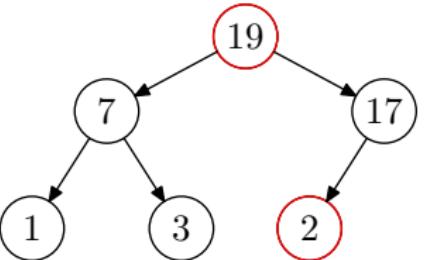
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



19	7	17	1	3	2	25	36	50
↑                      ↑								

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

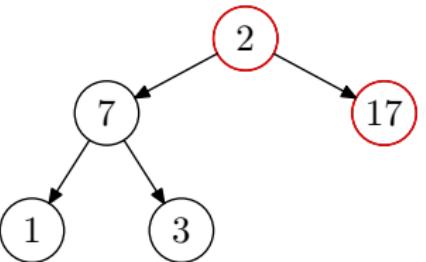
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

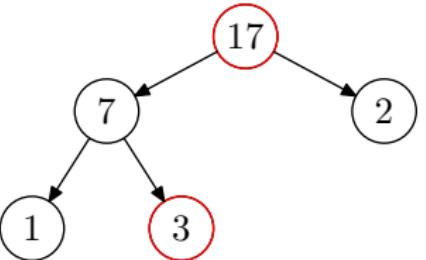
Kupac

Kupacrendezés

2	7	17	1	3	19	25	36	50
↑ ↑								

# Példa kupacrendezésre

Rendezés:



17	7	2	1	3	19	25	36	50
↑      ↑								

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

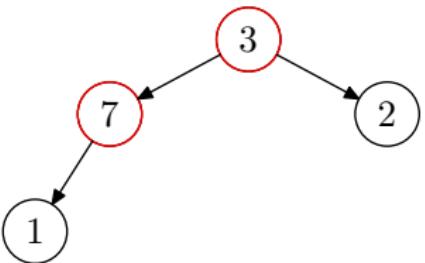
- Bináris fa
- Bejárási algoritmusok
- Preorder bejárás
- Inorder bejárás
- Postorder bejárás
- Reprezentáció
- Kifejezésfák
- Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

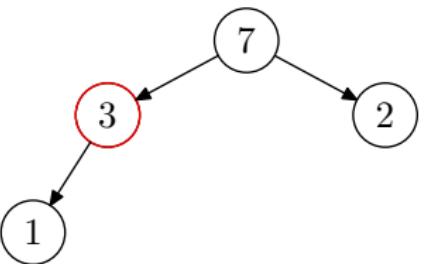
Kupac

Kupacrendezés

3	7	2	1	17	19	25	36	50
↑	↑							

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

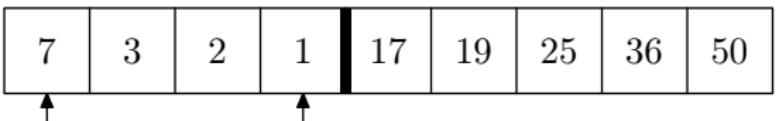
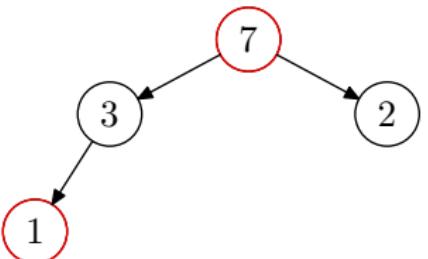
Kupac

Kupacrendezés

7	3	2	1	17	19	25	36	50
---	---	---	---	----	----	----	----	----

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

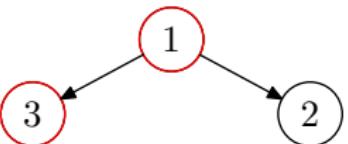
Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

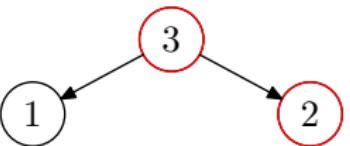
Kupac

Kupacrendezés

1	3	2	7	17	19	25	36	50
↑	↑							

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

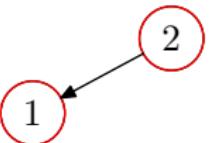
Kupac

Kupacrendezés

3	1	2	7	17	19	25	36	50
↑	↑							

# Példa kupacrendezésre

Rendezés:



Hierarchikus  
adatszerkezetek  
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac  
Kupacrendezés

2	1	3	7	17	19	25	36	50
↑	↑							

# Példa kupacrendezésre

Rendezés:

1

Hierarchikus  
adatszerkezetek

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hierarchikus  
adatszerkezetek

A fa adatszerkezet

Bináris fa  
Bejárási algoritmusok  
Preorder bejárás  
Inorder bejárás  
Postorder bejárás  
Reprezentáció  
Kifejezésfák  
Implementáció

Kupac

Kupacrendezés

1	2	3	7	17	19	25	36	50
---	---	---	---	----	----	----	----	----



# 6. előadás

## Bináris keresőfák

Kiegyensúlyozottság, AVL-fa, piros-fekete fa

*Adatszerkezetek és algoritmusok előadás*

2018. március 6.

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>



## Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Tökéletesen kiegyensúlyozott bináris fa

## Minimális magasságú bináris fa

Azt mondjuk, hogy egy bináris fa **minimális magasságú**, ha adott számú elemet nem lehetne kisebb magasságú bináris fában elhelyezni.

## Például

Minimális magasságú lesz egy olyan bináris fa, amelynek minden szintje telített – esetleg a legalsó szint kivételével.

## Tökéletesen kiegyensúlyozott bináris fa

Azt mondjuk, hogy egy bináris fa **tökéletesen kiegyensúlyozott**, ha bármely elemének bal és jobb oldali részfájában az elemek darabszáma legfeljebb 1-gyel tér el.

## Megjegyzés

Minden tökéletesen kiegyensúlyozott fa minimális magasságú.



## Adott elemszámú ( $n$ adatelemt tartalmazó) tökéletesen kiegyensúlyozott bináris fa építésének (egy lehetséges) algoritmusa

- ① Ha az adatelemek száma 0, az eredmény egy üres fa, és ezzel az algoritmus véget ér.
- ② Az első adatelem legyen a tökéletesen kiegyensúlyozott bináris fa gyökéreleme.
- ③ Osszuk két részre a megmaradt  $n - 1$  elemet, és
  - az első  $nb = \left\lfloor \frac{n}{2} \right\rfloor$  elemből építsük fel a gyökérelem bal oldali tökéletesen kiegyensúlyozott részfáját ugyanazzel az algoritmussal, majd
  - a megmaradt  $n_j = n - 1 - nb$  elemből építsük fel a gyökérelem jobb oldali tökéletesen kiegyensúlyozott részfáját ugyanazzel az algoritmussal.

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

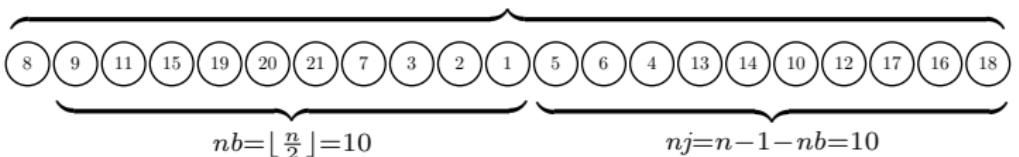
Bővítés

Törlés



# Tökéletesen kiegyensúlyozott bináris fa

$n=21$



8



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

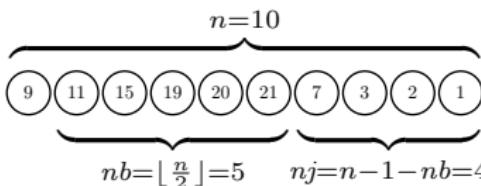
# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



8



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

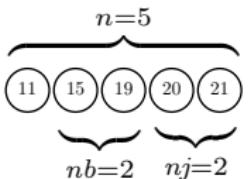
Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

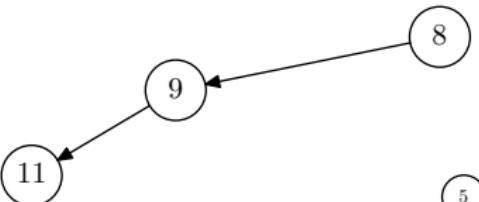
Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

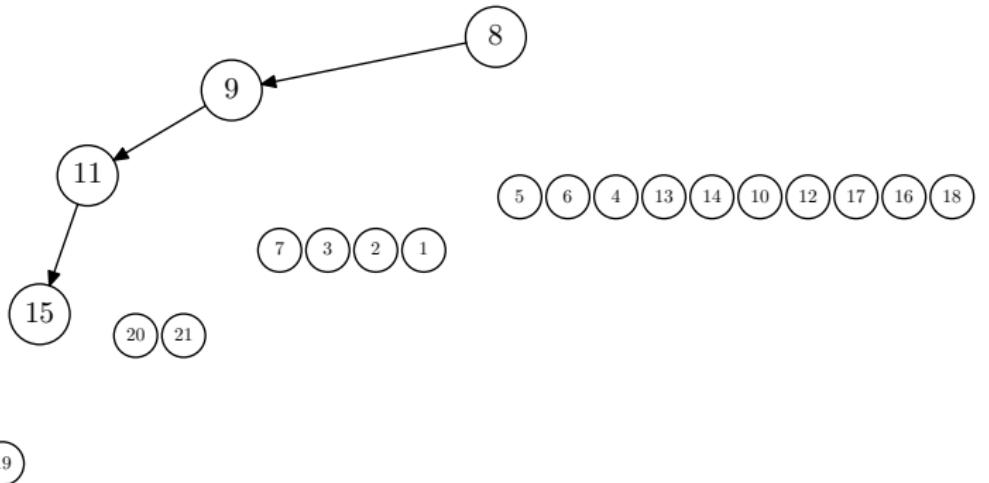
Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

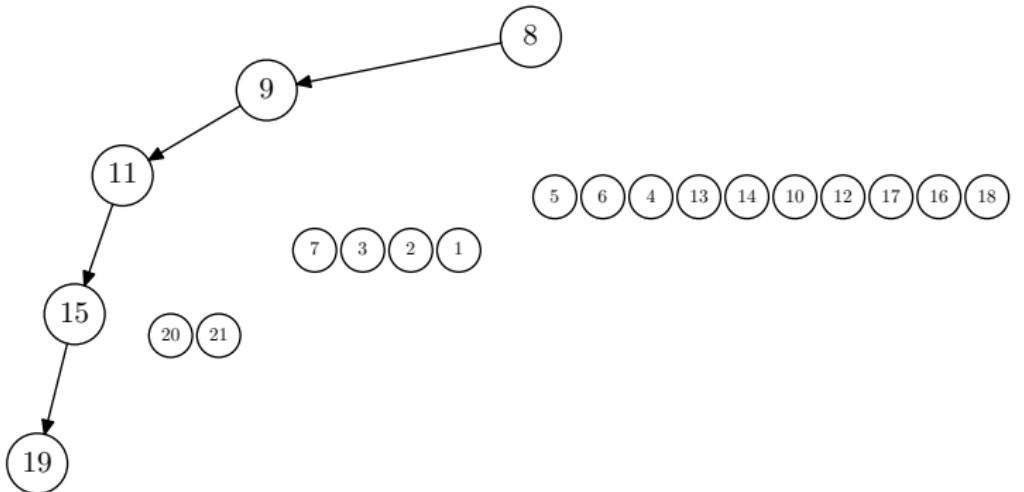
Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

8 9 11 15 19 20 21 7 3 2 1 5 6 4 13 14 10 12 17 16 18



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

Többágú fák

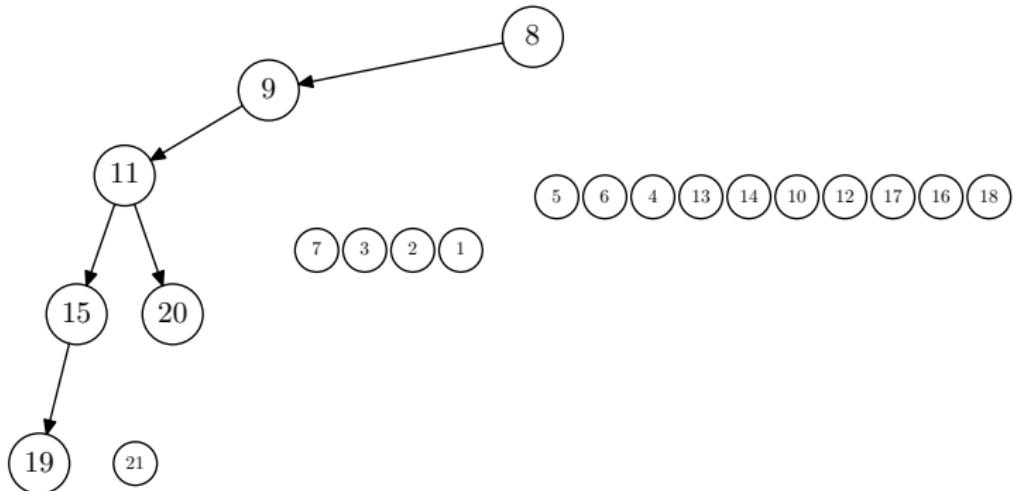
B-fa

Keresés  
Bővítés  
Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

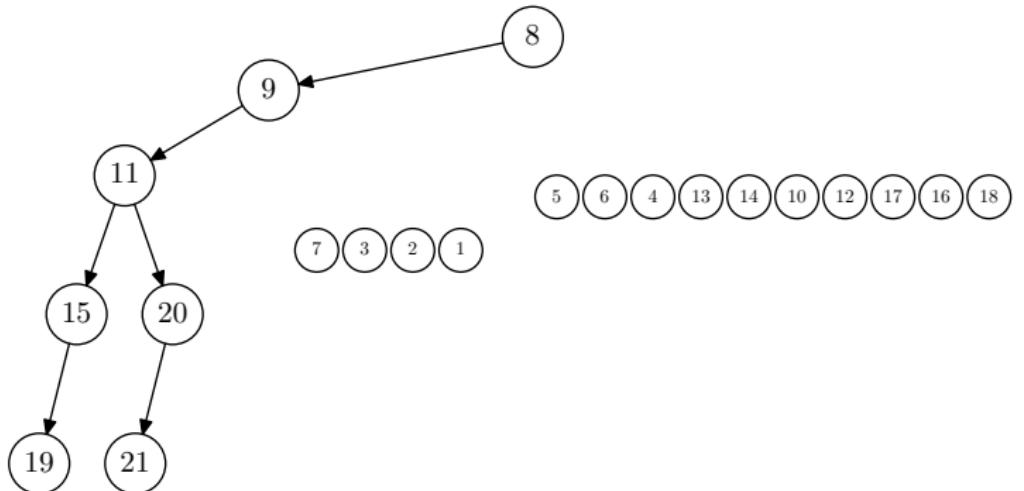
Többágú fák

B-fa

Keresés

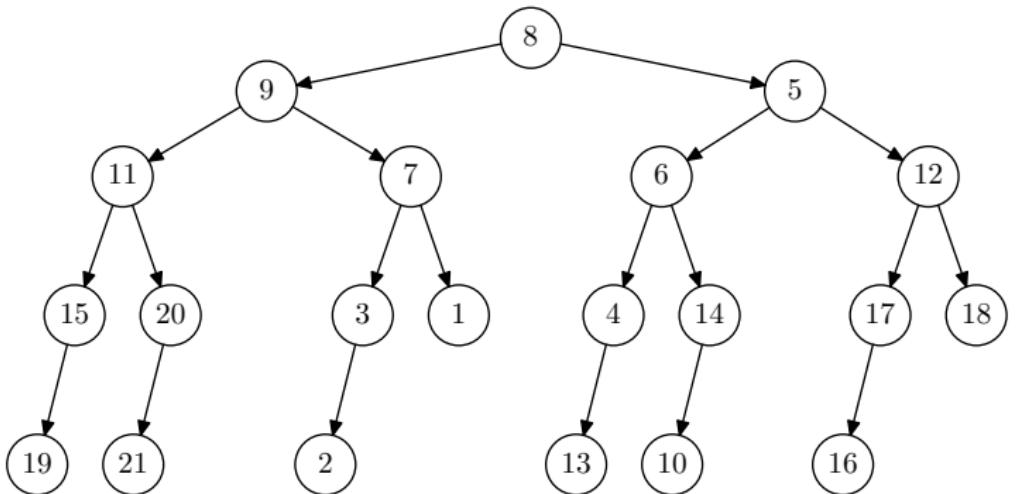
Bővítés

Törlés



# Tökéletesen kiegyensúlyozott bináris fa

8 9 11 15 19 20 21 7 3 2 1 5 6 4 13 14 10 12 17 16 18



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Tetszőleges elemszámú tökéletesen kiegyensúlyozott bináris fa építésének (egy másik lehetséges) algoritmusá

Hozzunk létre egy üres fát, majd alkalmazzuk az alábbi eljárást az egy-egy elemmel való bővítésre.

### Tökéletesen kiegyensúlyozott fát bővíti egy elemmel

- ① Ha a fa üres, akkor a beszúrandó adatelem legyen a tökéletesen kiegyensúlyozott bináris fa gyökéreleme.
- ② Egyébként:
  - Ha a gyökérelem jobboldali részfájának kisebb az elemszáma, mint a baloldalinak, akkor alkalmazzuk ugyanezt az algoritmust a jobboldali részfa bővítésére a beszúrandó adatelemmel,
  - egyébként pedig, alkalmazzuk ugyanezt az algoritmust a baloldali részfa bővítésére a beszúrandó adatelemmel.

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- ⑧ 9 11 15 19 20 21 7 3 2 1 5 6 4 13 14 10 12 17 16 18



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

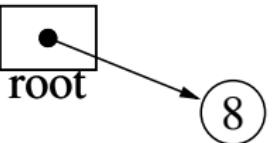
# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- (8)
- (9)**
- (11)
- (15)
- (19)
- (20)
- (21)
- (7)
- (3)
- (2)
- (1)
- (5)
- (6)
- (4)
- (13)
- (14)
- (10)
- (12)
- (17)
- (16)
- (18)



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

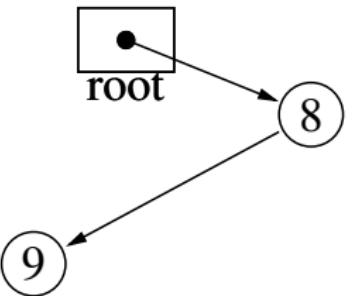
# Tökéletesen kiegyensúlyozott bináris fa

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- (8)
- (9)
- (11)**
- (15)
- (19)
- (20)
- (21)
- (7)
- (3)
- (2)
- (1)
- (5)
- (6)
- (4)
- (13)
- (14)
- (10)
- (12)
- (17)
- (16)
- (18)



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

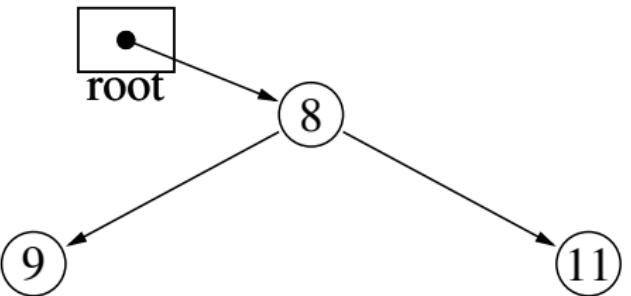
Többágú fák

B-fa

Keresés  
Bővítés  
Törlés

# Tökéletesen kiegyensúlyozott bináris fa

(8) (9) (11) (15) (19) (20) (21) (7) (3) (2) (1) (5) (6) (4) (13) (14) (10) (12) (17) (16) (18)



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

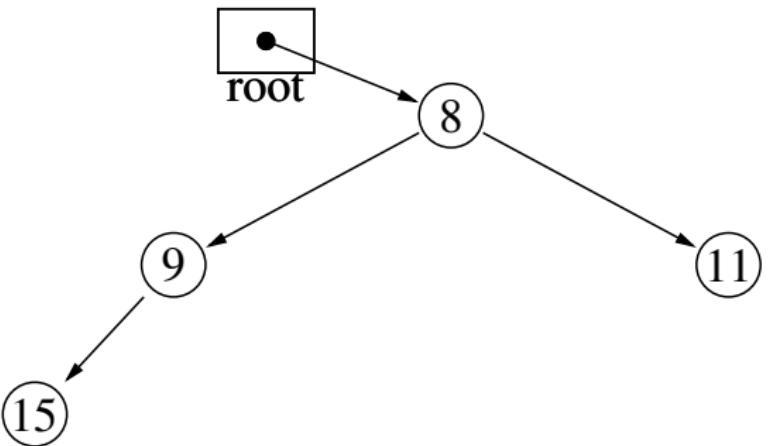
Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

(8) (9) (11) (15) (19) (20) (21) (7) (3) (2) (1) (5) (6) (4) (13) (14) (10) (12) (17) (16) (18)



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

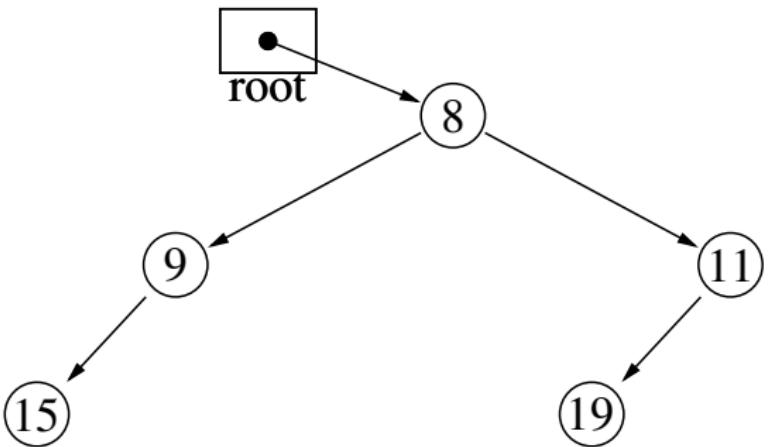
Többágú fák

B-fa

Keresés  
Bővítés  
Törlés

# Tökéletesen kiegyensúlyozott bináris fa

8 9 11 15 19 20 21 7 3 2 1 5 6 4 13 14 10 12 17 16 18



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

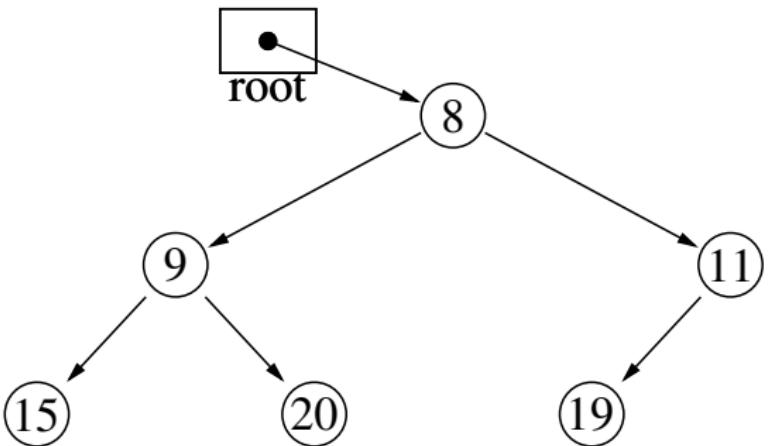
Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

(8) (9) (11) (15) (19) (20) (21) (7) (3) (2) (1) (5) (6) (4) (13) (14) (10) (12) (17) (16) (18)



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

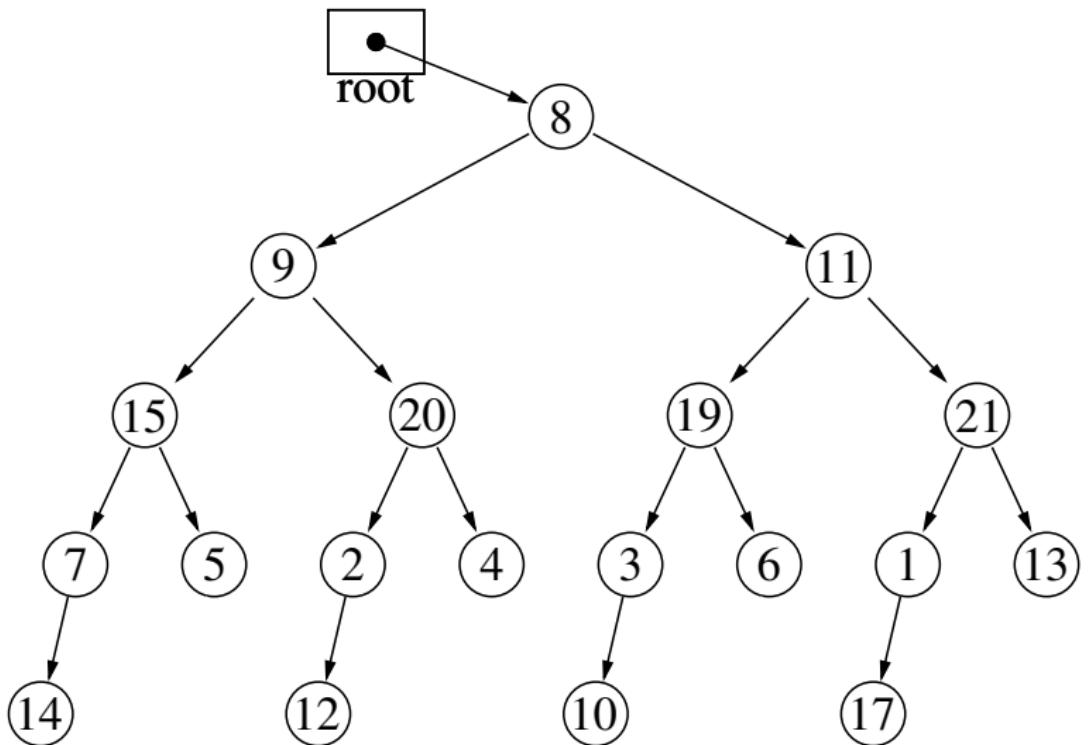
Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

(8) (9) (11) (15) (19) (20) (21) (7) (3) (2) (1) (5) (6) (4) (13) (14) (10) (12) (17) (16) (18)



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

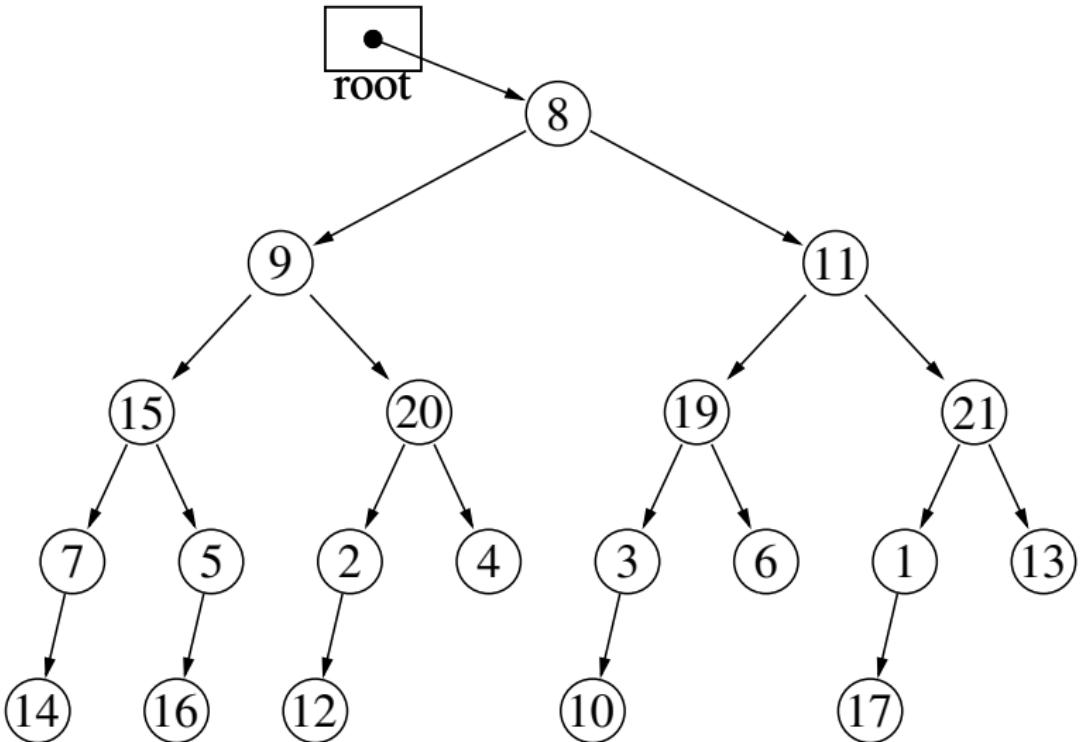
Keresés

Bővítés

Törlés

# Tökéletesen kiegyensúlyozott bináris fa

⑧ ⑨ ⑪ ⑯ ⑮ ⑲ ⑳ ㉑ ⑦ ③ ② ① ⑤ ⑥ ④ ⑬ ⑭ ⑩ ⑫ ⑰ ⑯ ⑯ ⑯



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

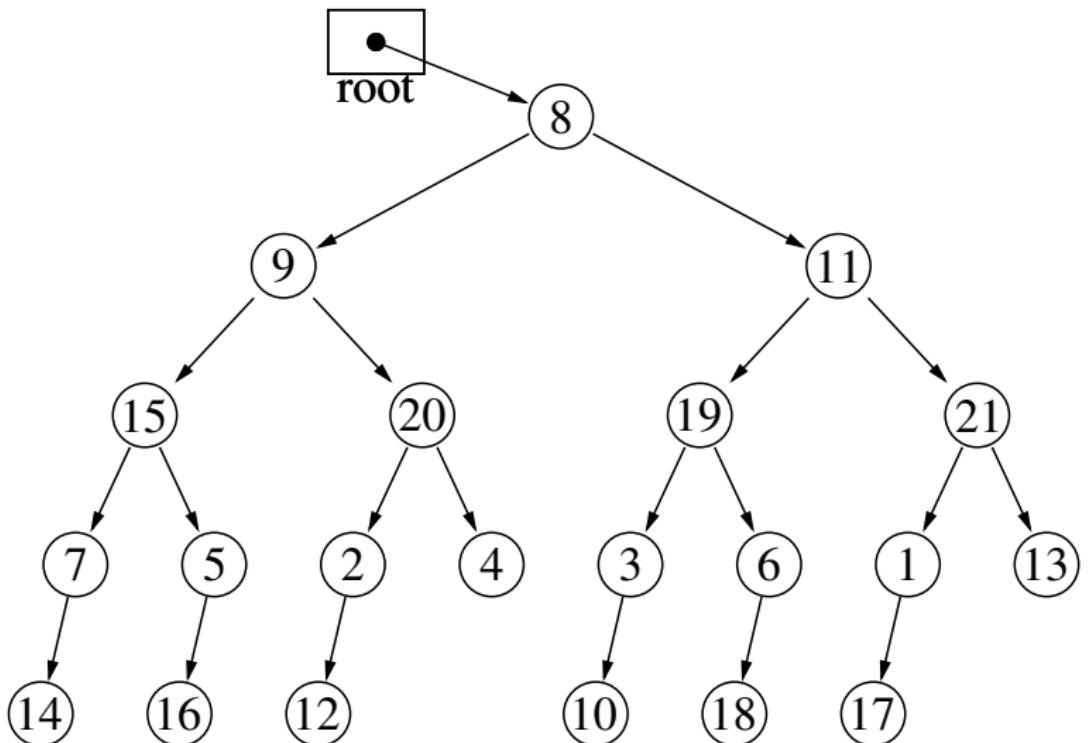
Többágú fák

B-fa

Keresés  
Bővítés  
Törlés

# Tökéletesen kiegyensúlyozott bináris fa

(8) (9) (11) (15) (19) (20) (21) (7) (3) (2) (1) (5) (6) (4) (13) (14) (10) (12) (17) (16) (18)



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

Többágú fák

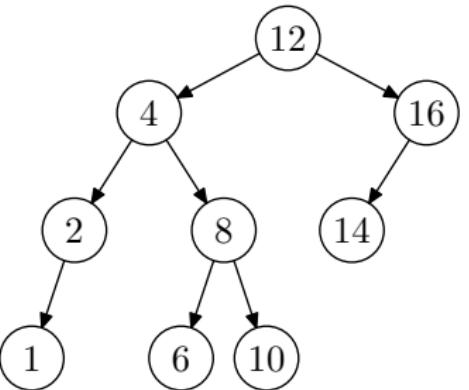
B-fa

Keresés  
Bővítés  
Törlés

## Bináris keresőfa

A bináris keresőfa olyan rendezett bináris fa, melyben az adatelemek minden egyike rendelkezik egy kulccsal, és minden adatelemre igaz az, hogy az adatelem bal oldali részfájában lévő elemek kulcsai kisebbek, a jobb oldali részfájában lévő elemek kulcsai pedig nagyobbak az elem kulcsánál.

Példa bináris keresőfára:



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Bináris keresőfa bővítése rekurzívan

- ① Ha üres a fa, akkor a beszúrandó elem lesz a fa egyetlen eleme (levéleleme), és ezzel az algoritmus sikeresen véget ér.
- ② Összehasonlítjuk a gyökérelem értékét a beszúrandó elemmel.
  1. Ha a két elem egyenlő, akkor a beszúrandó elemet nem helyezhetjük el a fában (mert nem szerepelhet benne két azonos értékű elem), és ezzel az algoritmus sikertelenül véget ér.
  2. Ha a beszúrandó elem kisebb a gyökérelemnél, akkor a gyökérelem bal oldali részfáját bővítjük a beszúrandó elemmel.
  3. Egyébként a gyökérelem jobb oldali részfáját bővítjük a beszúrandó elemmel.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Bináris keresőfa bővítése rekurzívan

- ① Ha üres a fa, akkor a beszúrandó elem lesz a fa egyetlen eleme (levéleleme), és ezzel az algoritmus sikeresen véget ér.
- ② Összehasonlítjuk a gyökérelem értékét a beszúrandó elemmel.
  1. Ha a két elem egyenlő, akkor a beszúrandó elemet nem helyezhetjük el a fában (mert nem szerepelhet benne két azonos értékű elem), és ezzel az algoritmus sikertelenül véget ér.
  2. Ha a beszúrandó elem kisebb a gyökérelemnél, akkor a gyökérelem bal oldali részfáját bővítjük a beszúrandó elemmel.
  3. Egyébként a gyökérelem jobb oldali részfáját bővítjük a beszúrandó elemmel.

# Bináris keresőfa bővítése

Bináris keresőfák

## Bináris keresőfa bővítése rekurzívan (implementáció)

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



```
typedef struct faelem {
    int adat;
    struct faelem *bal;
    struct faelem *jobb;
} FAELEM;

FAELEM *gyoker = NULL;

void beszur(int ertek)
{
    if (gyoker == NULL)
    {
        FAELEM *uj = (FAELEM *)malloc(sizeof(FAELEM));
        uj->adat = ertek;
        uj->bal = uj->jobb = NULL;
        gyoker = uj;
    }
    else
    {
        beszur_reszfaba(gyoker, ertek);
    }
}
```

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Bináris keresőfa bővítése

Bináris keresőfák

## Bináris keresőfa bővítése rekurzívan (implementáció) [folyt.]

```
void beszur_reszfaba(FAELEM *akt, int ertek)
{
    if (ertek == akt->adat) {
        /* már letezik ilyen kulcsú elem */
        return;
    }
    else
    {
        if (ertek < akt->adat)
        {
            if (akt->bal != NULL) {
                beszur_reszfaba(akt->bal, ertek);
            }
            else {
                FAELEM *uj = (FAELEM *)malloc(sizeof(FAELEM));
                uj->adat = ertek;
                uj->bal = uj->jobb = NULL;
                akt->bal = uj;
            }
        }
        else /* if ertek > akt->adat */
        {
            if (akt->jobb != NULL) {
                beszur_reszfaba(akt->jobb, ertek);
            }
            else {
                FAELEM *uj = (FAELEM *)malloc(sizeof(FAELEM));
                uj->adat = ertek;
                uj->bal = uj->jobb = NULL;
                akt->jobb = uj;
            }
        }
    }
}
```

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Bináris keresőfa bővítése iteratívan

- ① Aktuális részfának a teljes fát tekintjük.
- ② Ha üres az aktuális részfa, akkor a beszúrandó elem lesz az aktuális részfa egyetlen eleme (levéleleme), és ezzel az algoritmus sikeresen véget ér.
- ③ Összehasonlítjuk az aktuális részfa gyökérelemének értékét a beszúrandó elemmel.
  1. Ha a két elem egyenlő, akkor a beszúrandó elemet nem helyezhetjük el a fában (mert nem szerepelhet benne két azonos értékű elem), és ezzel az algoritmus sikertelenül véget ér.
  2. Ha a beszúrandó elem kisebb az aktuális elemnél, akkor aktuális részfának tekintsük az aktuális részfa gyökérelemének bal oldali részfáját.
  3. Egyébként aktuális részfának tekintsük az aktuális részfa gyökérelemének jobb oldali részfáját.
- ④ Folytassuk az algoritmust a 2. lépéssel.

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszúrás

CLRS-féle beszúrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Bináris keresőfa bővítése

Bináris keresőfák

## Bináris keresőfa bővítése iteratívan (implementáció)

```
void beszur(int ertek)
{
    FAELEM *uj = (FAELEM *)malloc(sizeof(FAELEM));
    uj->adat = ertek;
    uj->bal = uj->jobb = NULL;

    if (gyoker == NULL) {
        gyoker = uj;
    }
    else
    {
        FAELEM *akt = gyoker;

        while (akt != NULL)
        {
            if (ertek < akt->adat)
            {
                if (akt->bal == NULL) {
                    akt->bal = uj;
                    return;
                }
                akt = akt->bal;
            }
            else if (ertek > akt->adat)
            {
                if (akt->jobb == NULL) {
                    akt->jobb = uj;
                    return;
                }
                akt = akt->jobb;
            }
            else { /* már letezik ilyen kulcsú elem */
                return;
            }
        }
    }
}
```

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Törlés bináris keresőfából rekurzívan

- ① Ha üres a fa, akkor nem tudunk törölni, és ezzel az algoritmus sikertelenül véget ér.
- ② Összehasonlítjuk a gyökérelem értékét a törlendő elemmel.
  1. Ha a törlendő elem kisebb a gyökérelemnél, akkor a gyökérelem bal oldali részfájából töröljük a törlendő elemet.
  2. Ha a törlendő elem nagyobb a gyökérelemnél, akkor a gyökérelem jobb oldali részfájából töröljük a törlendő elemet.
  3. Ha a két elem egyenlő, akkor megnézzük, hogy a gyökérelemnek hány rákövetkezője van.
    - i. Ha a gyökérelemnek egy rákövetkezője sincs (azaz levélelem), akkor egyszerűen törölhető.
    - ii. Ha a gyökérelemnek egy rákövetkezője van, akkor felülírjuk a gyökérelemet azzal a rákövetkező elemmel (azaz egy szinttel feljebb csúsztatjuk a gyökérelem nem üres részfáját).
    - iii. Ha a gyökérelemnek két rákövetkezője van, akkor a gyökérelem értékét felülírjuk a gyökérelem bal oldali részfája legjobboldalibb elemének az értékével, majd a gyökérelem bal oldali részfájából töröljük ezt a legjobboldalibb elemet.

Ezzel az algoritmus sikeresen véget ér.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# Kiegyensúlyozott fa

## Kiegyensúlyozott bináris fa

Azt mondjuk, hogy egy bináris fa **kiegyensúlyozott**, ha bármely elemére igaz, hogy az elem bal oldali és jobb oldali részfájának magasságkülönbsége legfeljebb 1.

### Megjegyzés

Minden tökéletesen kiegyensúlyozott fa egyben kiegyensúlyozott is.

### Kiegyensúlyozott keresőfa (AVL-fa)

Akkor nevezünk egy bináris fát **kiegyensúlyozott keresőfának** vagy **AVL-fának**, ha kiegyensúlyozott is és keresőfa is egyben.

### Megjegyzés

Az AVL-fa elnevezés Georgij Makszimovics Adelszon-Velszkij és Jevgenij Mihajlovics Landisz nevéből származik. Az AVL-fáról először egy 1962-es cikkükben írtak.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

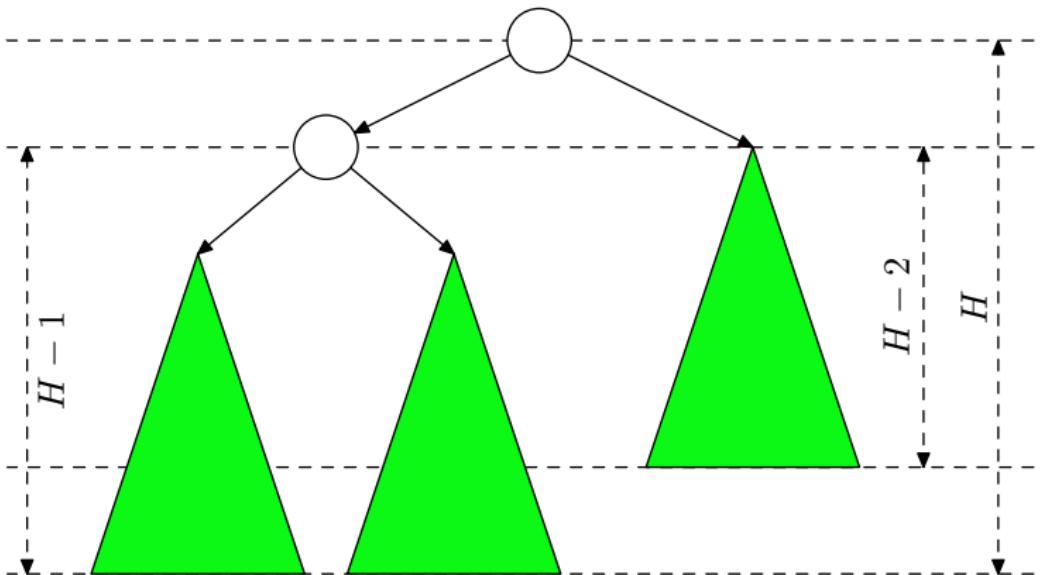
# AVL-fák bővítése

Egy AVL-fát úgy bővítünk, mint egy keresőfát: minden levélelemmel. A levélemmel történő bővítést követően a következő esetek fordulhatnak elő:

- 1 A fa továbbra is kiegysúlyozott. Ekkor nincs teendőnk, készen vagyunk.
- 2 A fa elveszti kiegysúlyozottságát. Ekkor egy vagy két **forgatással** újra kiegysúlyozottá kell tennünk a fát.

## Amikor elromlik a kiegyensúlyozottság: LL és LR bővítés

**Alaphelyzet:** a gyökérelem bal oldali részfája egy szinttel magasabb, mint a jobb oldali részfája - a fa még **kiegyensúlyozott**.



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

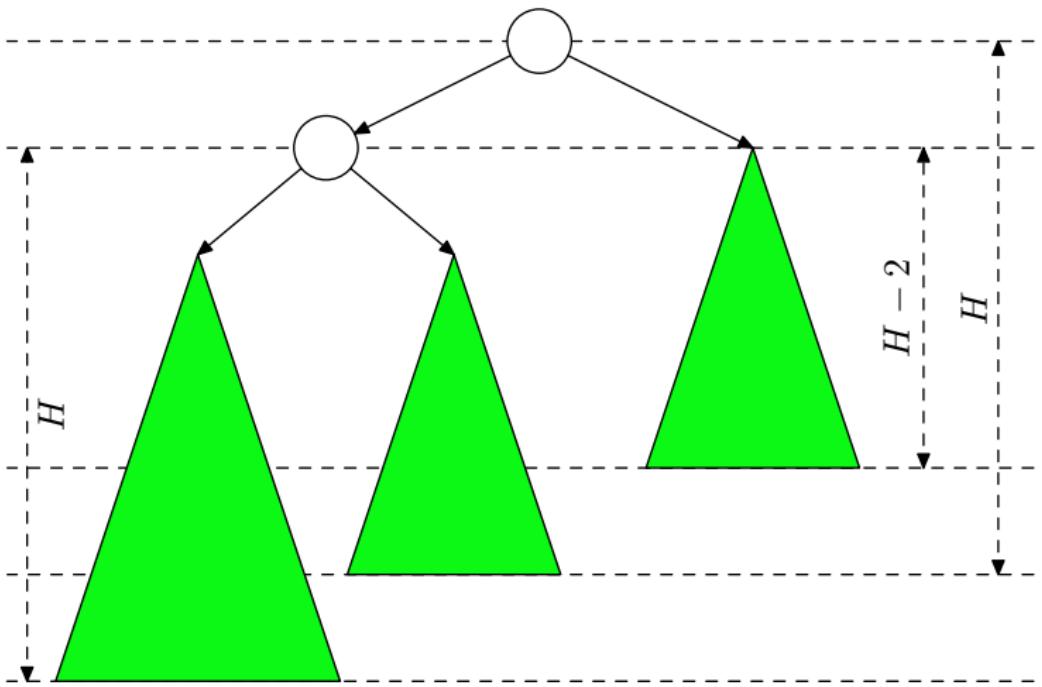
Többágú fák

B-fa

Keresés  
Bővítés  
Törlés

## Amikor elromlik a kiegyensúlyozottság: LL és LR bővítés

**LL bővítés:** a gyökérelem bal oldali részfájának bal oldali részfájába kerül az új elem - a fa **elveszti kiegyensúlyozottságát**.



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

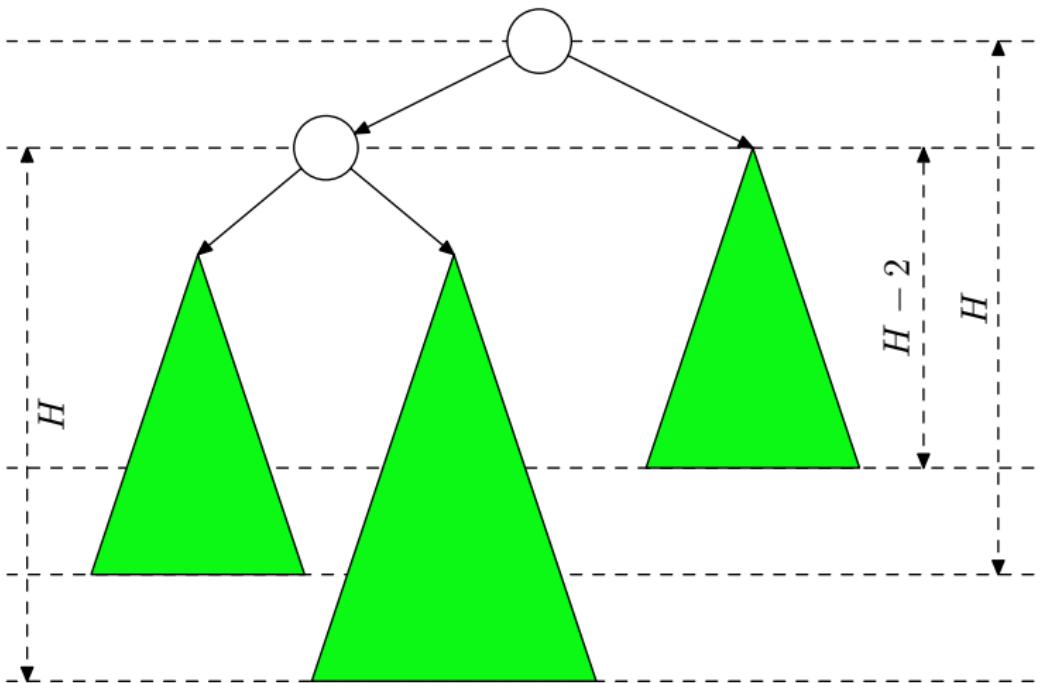
Keresés

Bővítés

Törlés

## Amikor elromlik a kiegyensúlyozottság: LL és LR bővítés

**LR bővítés:** a gyökérelem bal oldali részfájának jobb oldali részfájába kerül az új elem - a fa **elveszti kiegyensúlyozottságát**.



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

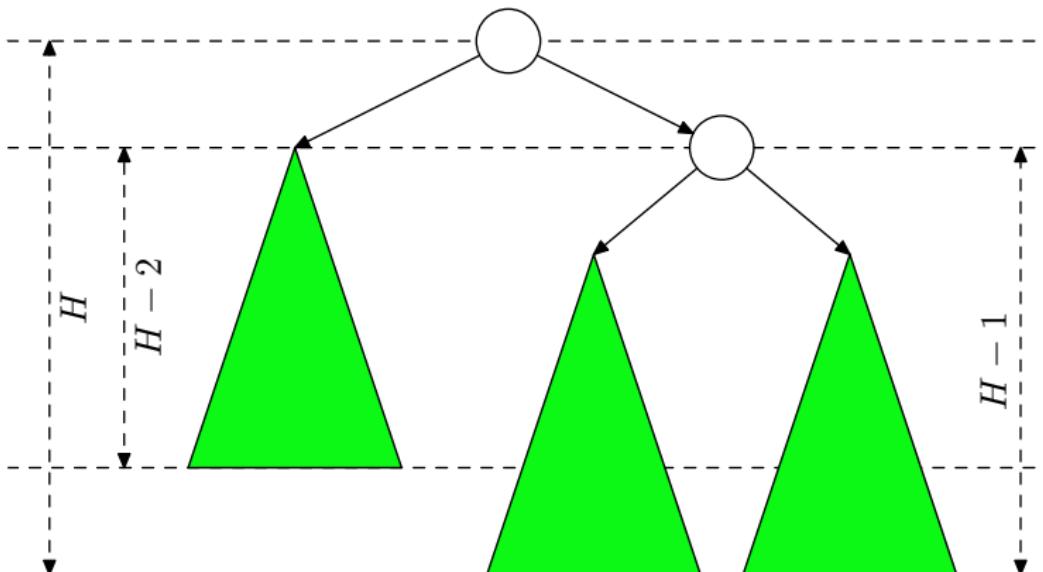
Keresés

Bővítés

Törlés

## Amikor elromlik a kiegyensúlyozottság: RL és RR bővítés

**Alaphelyzet:** a gyökérelem bal oldali részfája egy szinttel alacsonyabb, mint a jobb oldali részfája - a fa még **kiegyensúlyozott**.





Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

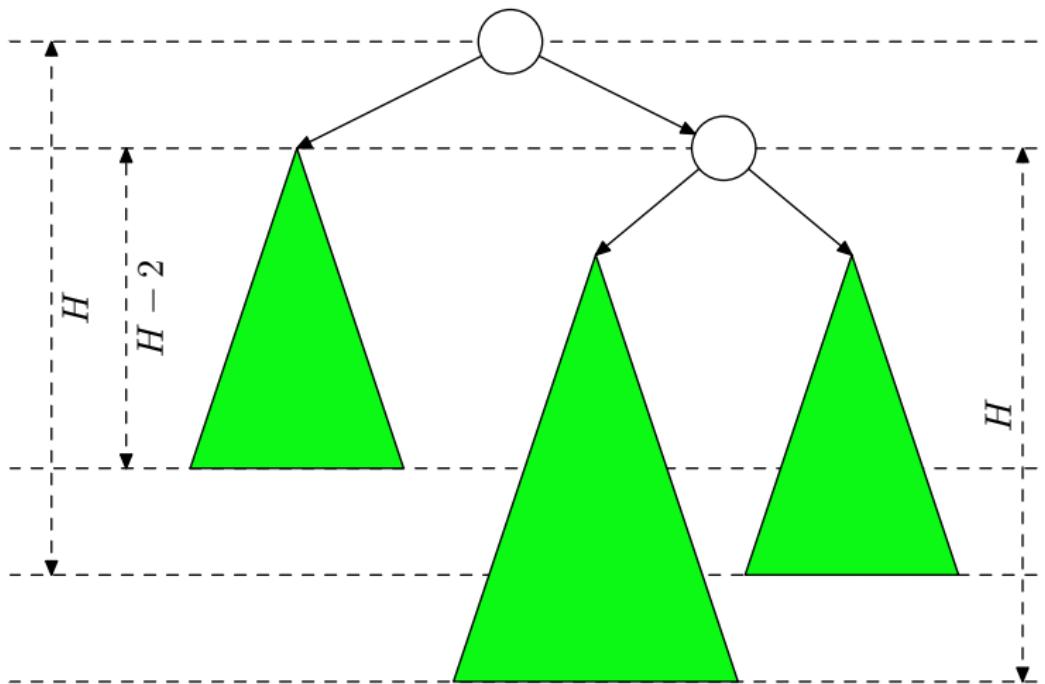
Keresés

Bővítés

Törlés

## Amikor elromlik a kiegyensúlyozottság: RL és RR bővítés

**RL bővítés:** a gyökérelem jobb oldali részfájának bal oldali részfájába kerül az új elem - a fa elveszti kiegyensúlyozottságát.





Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

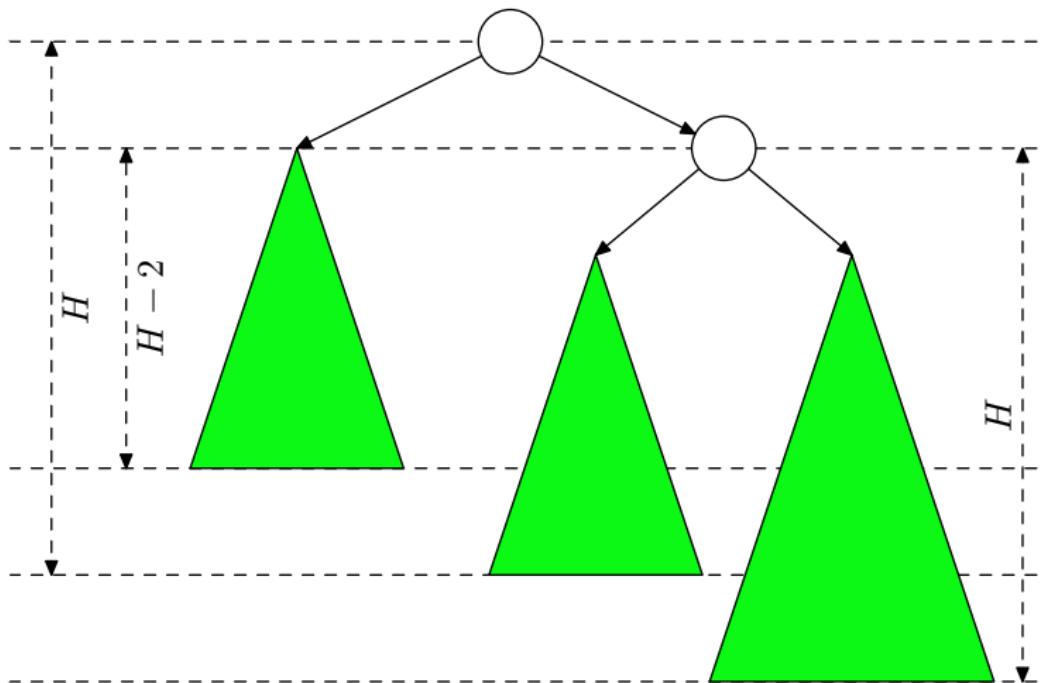
Keresés

Bővítés

Törlés

## Amikor elromlik a kiegyensúlyozottság: RL és RR bővítés

**RR bővítés:** a gyökerelem jobb oldali részfájának jobb oldali részfájába kerül az új elem - a fa elveszti kiegyensúlyozottságát.



# Az LL bővítés megoldása

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

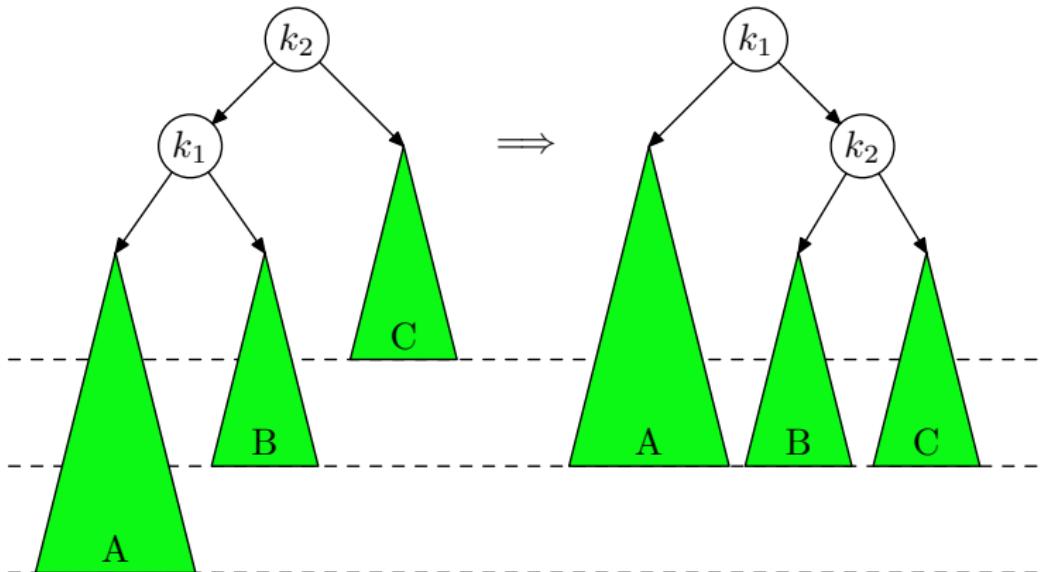
Többágú fák

B-fa

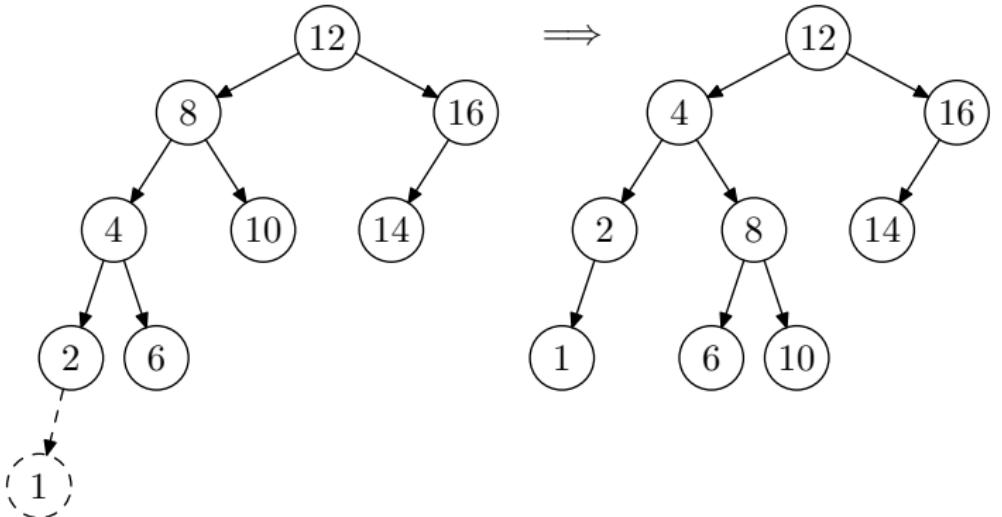
Keresés

Bővítés

Törlés



## Példa LL bővítésre



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

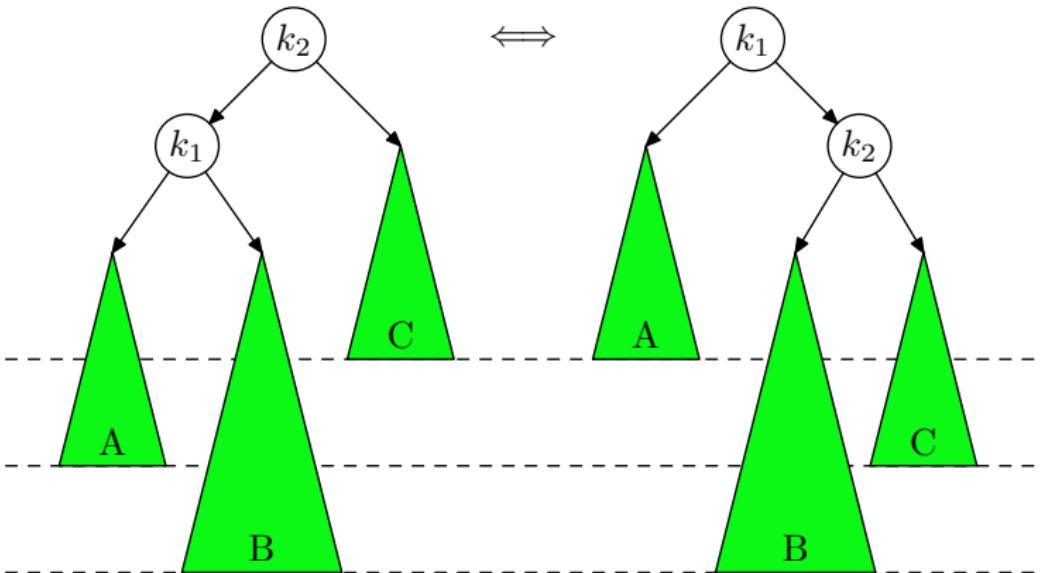
Keresés

Bővítés

Törlés

## Az LR bővítés megoldása

Az LR bővítés nem oldható meg egyetlen forgatással: sem a kiinduló, sem az egyszer elforgatott fa nem kiegyensúlyozott.



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

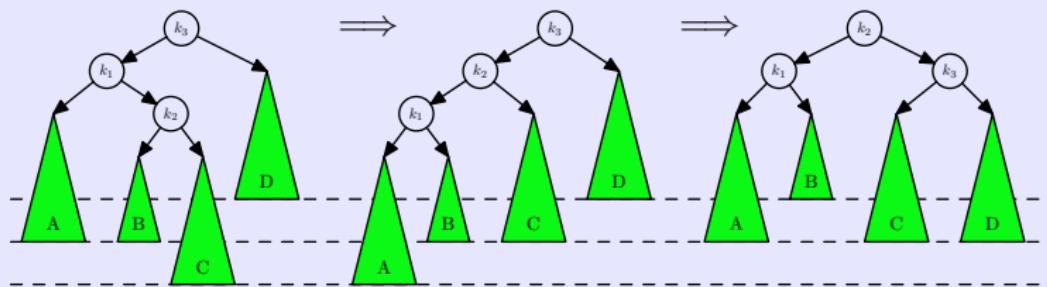
Bővítés

Törlés

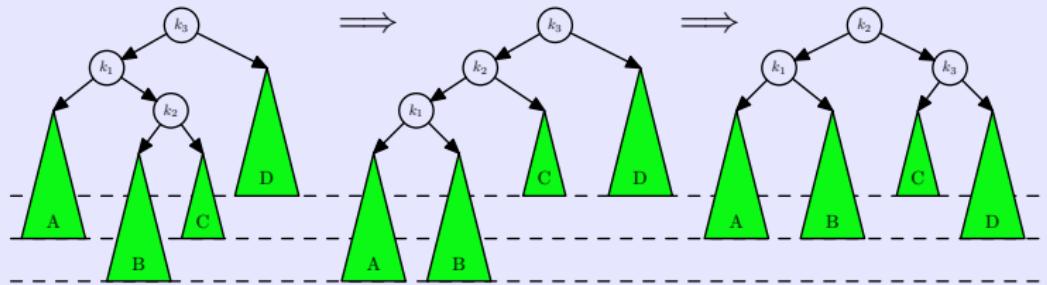
# Az LR bővítés megoldása

A megoldás: két forgatás!

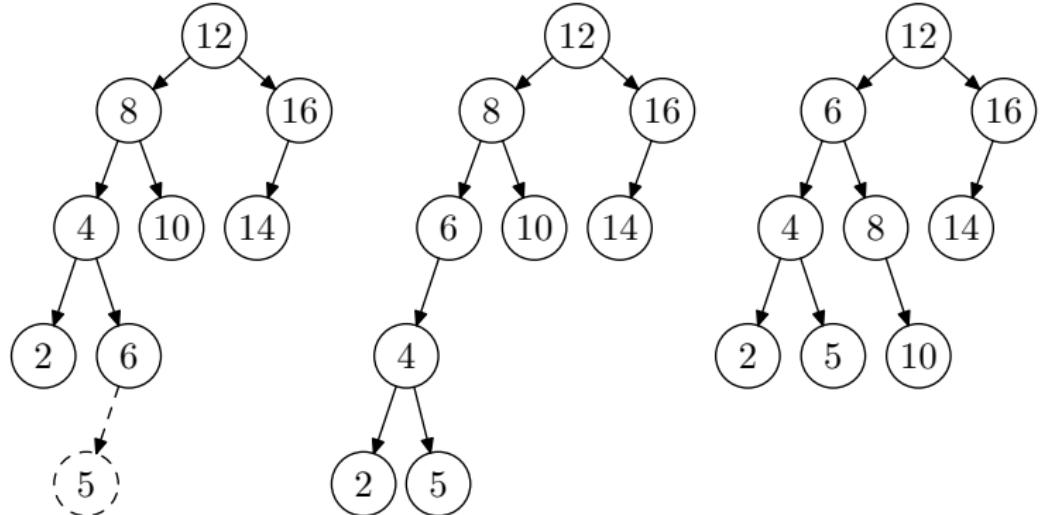
## Az egyik eset:



## A másik eset:



## Példa LR bővítésre



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

Egy AVL-fából ugyanúgy törlünk, mint egy keresőfából. A törlést követően a következő esetek fordulhatnak elő:

- ① A fa továbbra is kiegysúlyozott. Ekkor nincs teendőnk, készen vagyunk.
- ② A fa elveszti kiegysúlyozottságát. Ekkor a törlés helyétől a gyökér felé haladva megkeressük az első olyan elemet, amelyhez mint gyökérelemhez tartozó részfa már nem kiegysúlyozott, és a bővítésnél ismertetett négy eset közül a megfelelőt alkalmazva kiegysúlyozzuk ezt a részfát. Ezt a lépést a gyökér felé haladva addig ismételjük, amíg a teljes fa kiegysúlyozott nem lesz.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa**Piros-fekete fa**
 Okasaki-féle beszűrás  
 CLRS-féle beszűrás  
 Törlés
**Többágú fák**
 B-fa  
 Keresés  
 Bővítés  
 Törlés

# A piros-fekete fa definíciója

## Piros-fekete fa

A piros-fekete fa olyan bináris keresőfa, amely a következő tulajdonságokkal rendelkezik:

- ① minden csomópontja piros vagy fekete;
- ② a gyökere fekete;
- ③ minden (NIL értékű) levele fekete;
- ④ ha egy csomópont piros, akkor minden rácövetkezője fekete. (Más szavakkal kifejezve: nincs benne két egymást követő piros csomópont.)
- ⑤ minden csomópont esetén az összes olyan úton, amely az adott csomópontból indul ki és levélig vezet, ugyanannyi a fekete csomópontok száma.

## Megjegyzés

Egy  $n$  adatalemet tartalmazó piros-fekete fa magassága legfeljebb  $2 \log(n + 1)$ .



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

**Piros-fekete fa**

Okasaki-féle beszűrás

CLRS-féle beszűrás

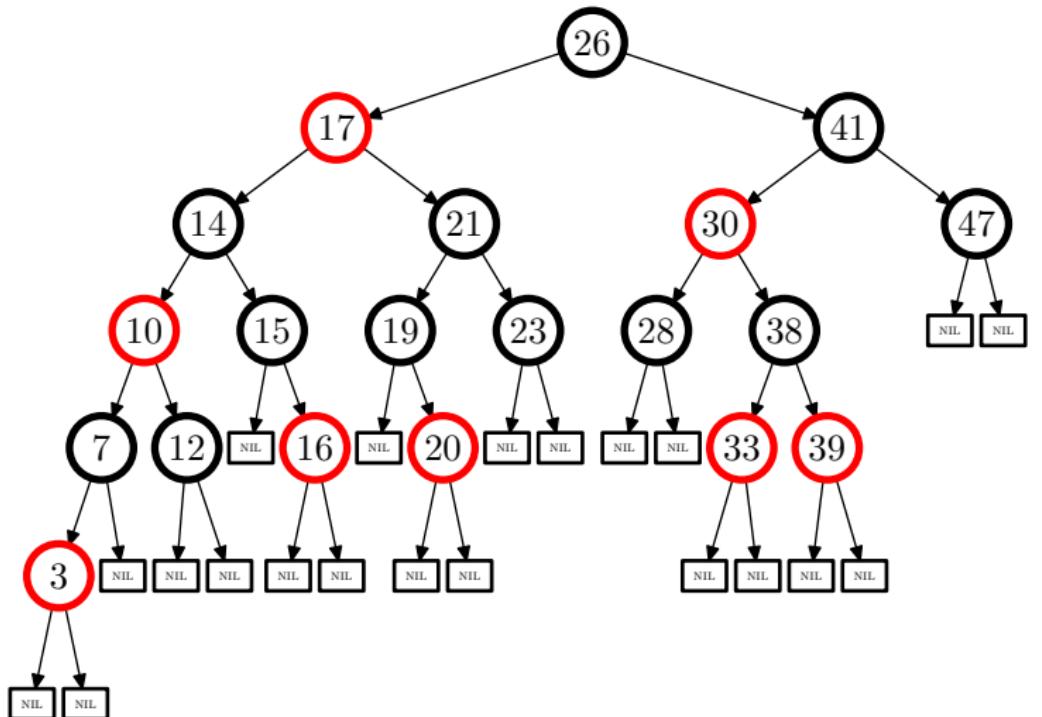
Törlés

**Többágú fák****B-fa**

Keresés

Bővítés

Törlés





Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Beszúrás piros-fekete fába (Okasaki-módszer)

Egy piros-fekete fát úgy bővítenünk, mint egy keresőfát: minden levélelemmel, amelyet pirosra színezünk. A levélelemmel történő bővítést követően a következő esetek fordulhatnak elő:

- ① A fa továbbra is rendelkezik a piros-fekete tulajdonságokkal. Ekkor nincs teendőnk, készen vagyunk.
- ② Nem teljesül a 2-es tulajdonság, miszerint a gyökérelem fekete. Ez csak akkor fordulhat elő, ha éppen a gyökeret szúrtuk be, azaz a fa előzőleg üres volt. Ekkor átszínezzük a beszúrt (gyökér)elemet feketére, és készen vagyunk.
- ③ Nem teljesül a 4-es tulajdonság, miszerint nincs a fában két egymást követő piros csomópont. Ez csak akkor fordulhat elő, ha a beszúrt elem szülője is piros. Mivel a gyökér fekete, a beszúrt elemnek biztosan létezik nagyszülője, amelynek a 4-es tulajdonság miatt feketének kell lennie. Ekkor **forgatásokat** és **átszínezéseket** kell végrehajtanunk.

# Beszúrás piros-fekete fába (Okasaki-módszer)

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

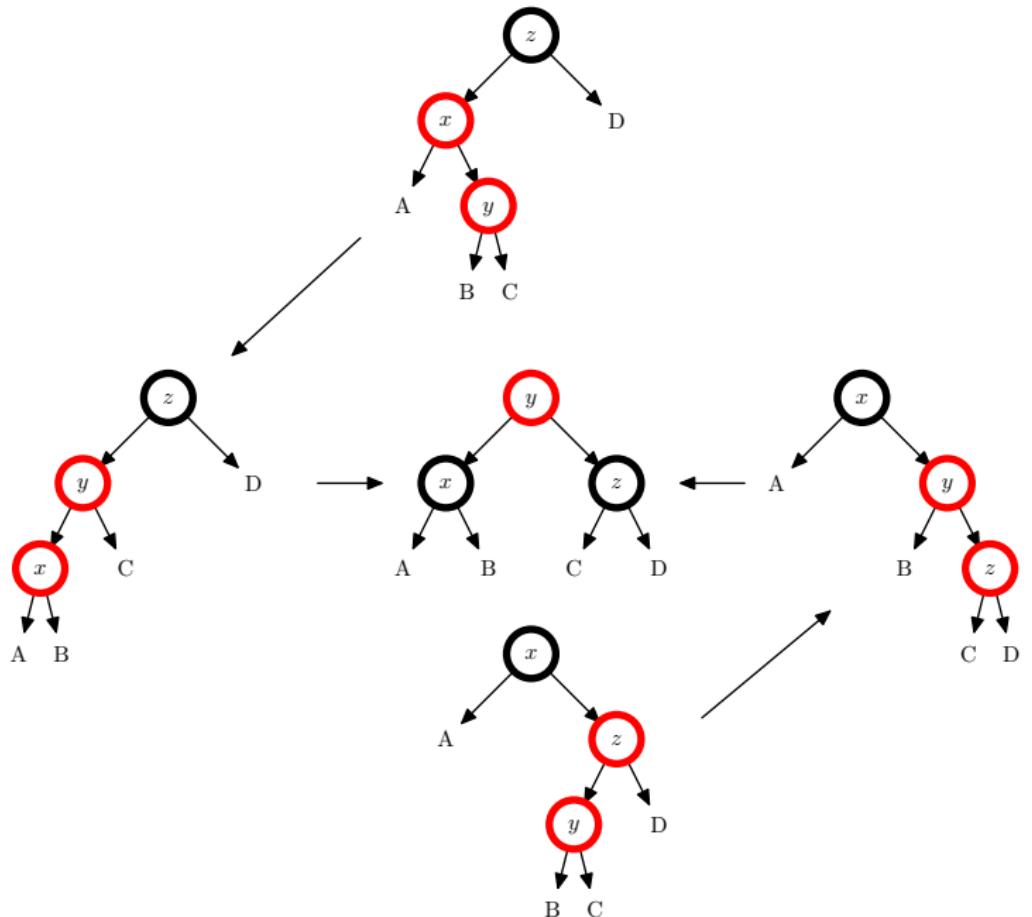
Okasaki-féle beszúrás

CLRS-féle beszúrás  
Törlés

Többágú fák

B-fa

Keresés  
Bővítés  
Törlés





Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Beszűrás piros-fekete fába (Okasaki-módszer)

A fenti transzformáció (egy vagy két forgatás, valamint egy átsínezés) után az  $y$  szülőjéből (ha létezik) bármelyik levélbe vezető úton ugyanannyi fekete elem lesz, mint amennyi a transzformáció előtt volt. Az így kapott fa már vagy piros-fekete fa, vagy nem teljesül a 2-es tulajdonság (ha  $y$  a gyökérelem), vagy nem teljesül a 4-es tulajdonság (ha  $y$  szülője piros).

A fenti transzformációt tehát addig kell ismételnünk, amíg  $y$  szülője fekete nem lesz (akkor nincs további teendőnk), vagy  $y$  a gyökér nem lesz. Utóbbi esetben átsínezzük  $y$ -t feketére, és készen vagyunk. (A gyökérelem feketére színezésével a gyökérből az egyes levelekbe vezető utak mindegyikén ugyanannyival nő a fekete csomópontok száma, tehát az 5-ös tulajdonság továbbra is fennáll.)

[Kiegysúlyozottság](#)[Bináris keresőfa](#)[Kiegysúlyozott fa](#)[Kiegysúlyozott keresőfa](#)[Piros-fekete fa](#)

Okasaki-féle beszúrás

[CLRS-féle beszúrás](#)

Törlés

[Többágú fák](#)[B-fa](#)

Keresés

Bővítés

Törlés

## Beszúrás piros-fekete fába (CLRS-módszer)

A CLRS-módszer abban különbözik az Okasaki-módszertől, hogy hogyan kezeli azt az esetet, amikor a beszúrás után nem teljesül a 4-es tulajdonság. Ekkor két esetet különböztetünk meg attól függően, hogy a beszúrt elem nagybácsija (a szülőjének a testvére) piros-e vagy fekete. Tételezzük fel először, hogy fekete! Ekkor hasonló forgatásokat hajtunk végre, mint az Okasaki-módszer esetén, viszont utána az  $y$  csomópont lesz fekete, míg a két gyermeké ( $x$  és  $z$ ) piros. Ezáltal biztosan teljesülni fog a 4-es tulajdonság, így nincs szükség további forgatásokra és átszínezésekre (természetesen a 2-es tulajdonság is teljesül).



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszúrás

CLRS-féle beszúrás

Törlés

Többágú fák

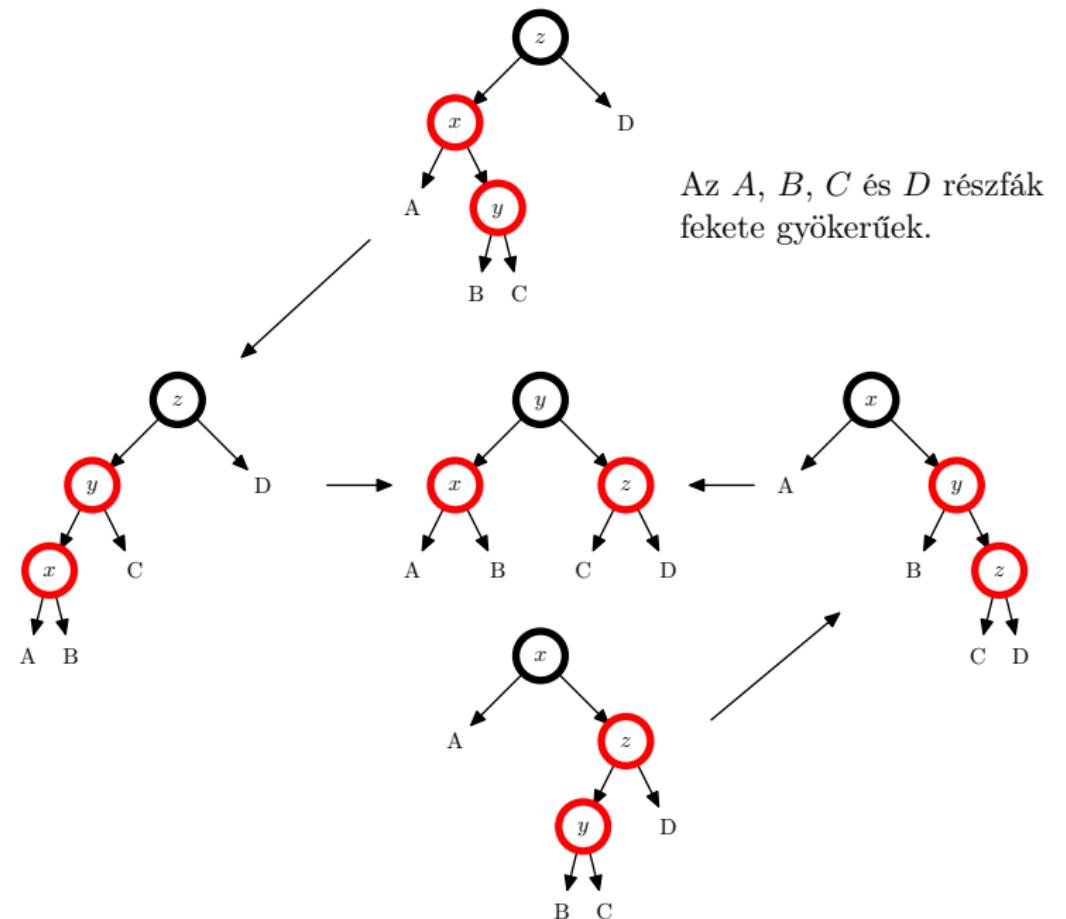
B-fa

Keresés

Bővítés

Törlés

# Beszúrás piros-fekete fába (CLRS-módszer)





Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszúrás

CLRS-féle beszúrás

Törlés

Többágú fák

B-fa

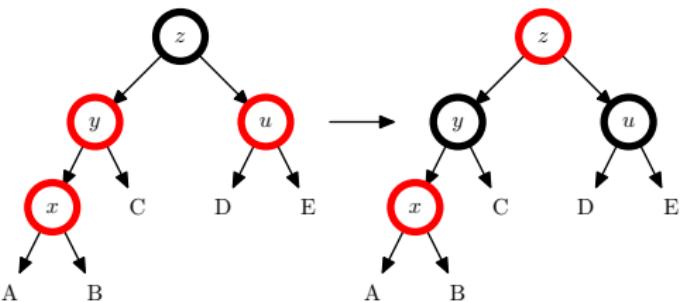
Keresés

Bővítés

Törlés

## Beszúrás piros-fekete fába (CLRS-módszer)

Mi történik akkor, ha a beszúrt elem nagybácsija piros? Ebben az esetben a beszúrt elem szülőjét és annak testvérét (a nagybácsit) feketére színezzük, a szülőjüket pedig pirosra. Forgatást ilyenkor nem kell végrehajtani. Az egyik lehetséges esetet szemlélteti a következő ábra:



Könnyen látható, hogy az átszínezés után nem változik a gyökérből a levelekbe vezető utakon a fekete elemek száma. Előfordulhat viszont, hogy nem teljesül a 2-es vagy a 4-es tulajdonság. Az eljárást tehát mindaddig kell ismételnünk, amíg (i) z szülője fekete nem lesz (akkor készen vagyunk), (ii) z a gyökér nem lesz (amit átszínezünk feketére, és készen vagyunk), vagy (iii) z nagybácsija fekete nem lesz (akkor végrehajtunk egy vagy két forgatást, és készen vagyunk).

## Törlés piros-fekete fából

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

Egy piros-fekete fából ugyanúgy törlünk, mint egy keresőfából. Az a csomópont, amelyet eltávolítottunk a fából, nem feltétlenül az a csomópont, amely a törölt adatelemet tartalmazta. A piros-fekete tulajdonságok helyreállításához az eltávolított csomópontot kell figyelembe vennünk. Legyen ez a csomópont  $v$ , a szülője pedig  $p(v)$ !

Az eltávolított csomópont ( $v$ ) legalább egyik gyermekének levélnek kell lennie. Ha  $v$ -nek van egy nem levél gyermekje, akkor a helyét az a bizonyos gyermek, különben pedig egy levélelem veszi át. Legyen  $u$  az a gyermek, amelyik  $v$  helyére kerül a törlés után! Ha  $u$  levél, akkor tudjuk, hogy fekete.

Ha  $v$  piros, akkor készen vagyunk, mivel egyetlenegy piros-fekete tulajdonságot sem sérthettünk. Tehát tegyük fel, hogy  $v$  fekete!



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törles

Többágú fák

B-fa

Keresés

Bővítés

Törles

## Törlés piros-fekete fából

A gyökérből a levelekbe vezető azon utak, amelyek keresztülmennek  $v$ -n, eggyel kevesebb fekete csomópontot fognak tartalmazni, mint a többi gyökér-levél út a fában, és ez megsérti az 5-ös tulajdonságot. Ha  $p(v)$  és  $u$  is piros, akkor a 4-es tulajdonságot is megsértjük, de látni fogjuk, hogy az 5-ös tulajdonság helyreállítása a 4-es tulajdonságot is helyreállítja további teendők nélkül, ezért mi most az 5-ös tulajdonság helyreállítására koncentrálunk.

Képzeljük el, hogy egy fekete **tokent** rendelünk  $u$ -hoz! Ez a token azt jelzi, hogy az ezen a csomóponton átmenő, levélig vezető utak eggyel kevesebb fekete csomópontot tartalmaznak, mint kellene. (Kezdetben ez azért van, mert  $v$ -t kitöröltük.) A tokent a fában egyre feljebb visszük, amíg ki nem alakul egy olyan helyzet, amelyben az 5-ös tulajdonságot helyreállíthatjuk. Ezt a tokent egy kis fekete négyzettel jelöljük az ábrákon. Ha a tokennel rendelkező csomópont fekete, akkor azt **duplán fekete csomópontnak** nevezzük. (A token csak egy fogalmi eszköz, fizikailag nem jelenik meg az adatszerkezetben.)



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törles

Többágú fák

B-fa

Keresés

Bővítés

Törles

## Törlés piros-fekete fából (1. eset)

**1. eset:** Ha a tokennel rendelkező csomópont piros, vagy a fa gyökere (vagy mindenki), akkor egyszerűen színezzük át feketére, és készen vagyunk. Vegyük észre, hogy ez a 4-es tulajdonságot (nincs két egymást követő piros csomópont) helyreállítja. Az 5-ös tulajdonságot is helyreállítja a következő okból. A token azt jelezte, hogy a gyökérből az ezen a csomópontron áthaladó, levél vezető utaknak egy újabb fekete csomópontra lenne szükségük ahhoz, hogy a többi gyökér-levél útnak megfeleljenek. Azáltal, hogy ezt a piros csomópontot feketére színeztük, pontosan azon utakhoz adtunk egy újabb fekete csomópontot, amelyekben eggyel kevesebb volt a kelleténél.

Ha a token a gyökérben van, és a gyökér fekete, a tokent egyszerűen eldobjuk. Ebben az esetben minden gyökér-levél úton eggyel csökkentettük a fekete csomópontok számát, így az 5-ös tulajdonság továbbra sem sérül.

A további esetekben feltételezhetjük, hogy a tokennel rendelkező csomópont fekete, és nem a gyökér.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

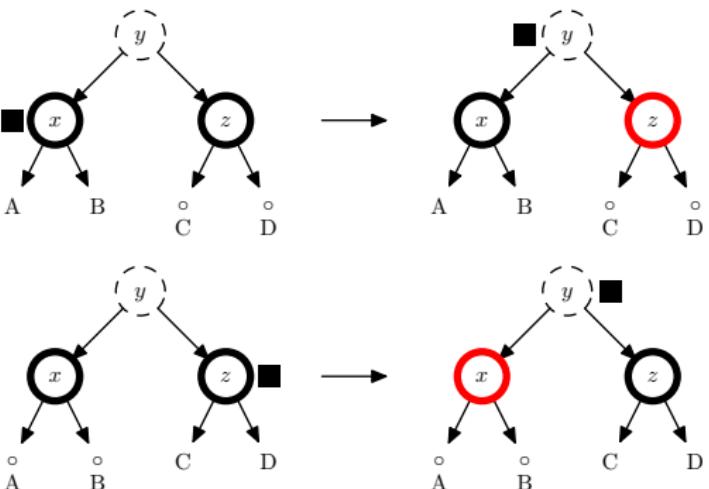
Bővítés

Törlés

## Törlés piros-fekete fából (2. eset)

**2. eset:** Ha a duplán fekete csomópont testvére és minden fekete unokaöccse fekete, akkor a testvért pirosra színezzük, a tokent pedig egy csomóponttal feljebb visszük a gyökér irányába.

Az alábbi ábrán, amely a két lehetséges alesetet mutatja, az  $y$  körül szaggatott vonal jelzi, hogy ezen a ponton nem érdekel minket  $y$  színe, az  $A$ , a  $B$ , a  $C$  és a  $D$  fölötti kis karikák pedig azt jelzik, hogy az adott részfa gyökere fekete.





Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Törlés piros-fekete fából (2. eset)

A testvér pirosra színezése kitöröl egy fekete csomópontot a belőle elérhető levelekhez vezető utakból, így azokon az utakon ugyanannyi fekete csomópont lesz, mint amennyi a duplán fekete csomópontból elérhető levelekhez vezető utakon. A tokent felvisszük az  $y$  szülőbe, jelezve, hogy *minden*  $y$  alatti út most eggyel kevesebb fekete csomópontot tartalmaz, mint kellene. Nem oldottuk meg a problémát, csak egy szinttel feljebb tolunk a gyökér felé.

Ezt a műveletet nyilván csak akkor hajthatjuk végre, ha minden unokaöcs fekete, hiszen különben egymást követő piros csomópontokat kapnánk.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

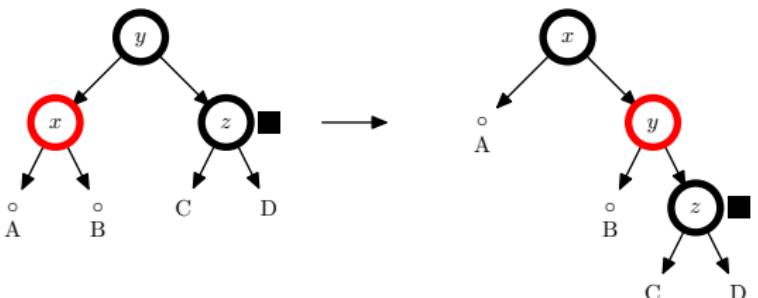
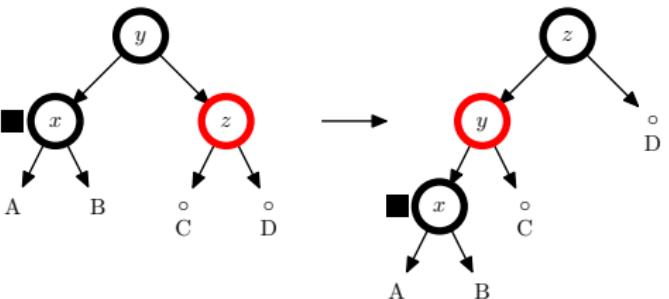
Keresés

Bővítés

Törlés

## Törlés piros-fekete fából (3. eset)

**3. eset:** Ha a duplán fekete csomópont testvére piros, akkor egy forgatást és egy színcserét kell végrehajtani. A két lehetséges esetet a következő ábra mutatja:



## Törlés piros-fekete fából (3. eset)

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

Ez a lépés nem változtatja meg a gyökérből a levelekhez vezető utakon a fekete csomópontok számát, de garantálja, hogy a duplán fekete csomópont testvére fekete lesz, amelynek következtében vagy a 2., vagy a 4. eset fog előállni.

Úgy tűnhet, hogy rontottunk a helyzenet, mivel a token távolabb került a gyökértől, mint korábban volt. Mivel azonban a duplán fekete csomópont szülője már piros, ezért ha a 2. eset állt elő, akkor a tokent egy piros csomópontba fogjuk továbbítani, amely aztán feketévé alakul, és készen leszünk. Ha pedig a 4. eset áll elő, akkor – ahogy minden járt látni fogjuk – minden eltűnik a token, és befejeződik a művelet. Ez a „visszalépés” tehát annak a jele, hogy már majdnem készen vagyunk.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törles

Többágú fák

B-fa

Keresés

Bővítés

Törles

## Törlés piros-fekete fából (4. eset)

**4. eset:** Végül maradt az az eset, ahol a duplán fekete csomópontnak fekete a testvére, és legalább egy piros unokaöccse van. Legyen egy  $x$  csomópont **közeli unokaöccse**  $x$  testvérének a bal oldali gyermeké, ha  $x$  bal oldali gyermek, és  $x$  testvérének a jobb oldali gyermeké, ha  $x$  jobb oldali gyermek; és legyen  $x$  **távoli unokaöccse**  $x$  másik unokaöccse.

Két aleset létezik:

- (i) A duplán fekete csomópont távoli unokaöccse fekete, tehát a közeli unokaöccse piros.
- (ii) A távoli unokaöcs piros, tehát a közeli unokaöcs bármilyen színű lehet.

Ahogy a következő ábrán látható, az (i) alesetet egy forgatással és egy színcserével a (ii) alestre transzformáljuk, a (ii) alesetet pedig egy újabb forgatással és színcserével oldjuk meg. A két sor szimmetrikus, attól függően, hogy a duplán fekete csomópont bal vagy jobb oldali gyermek-e.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törles

Többágú fák

B-fa

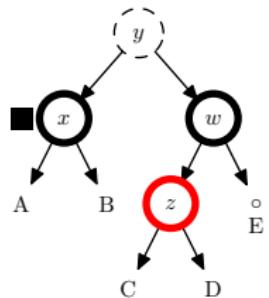
Keresés

Bővítés

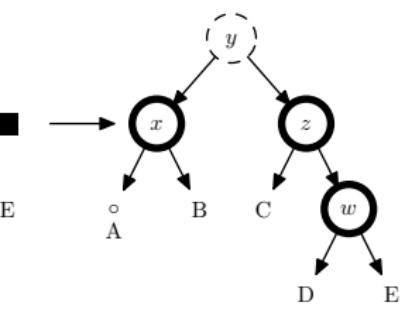
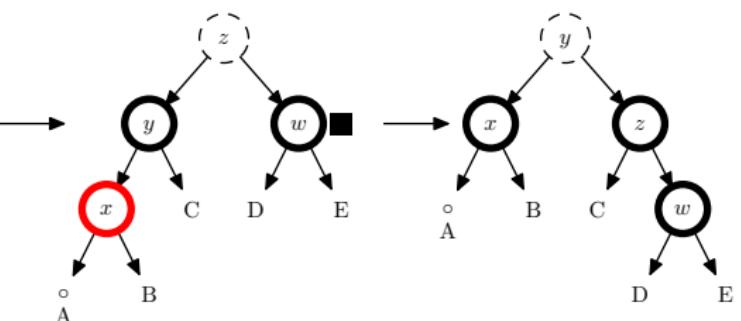
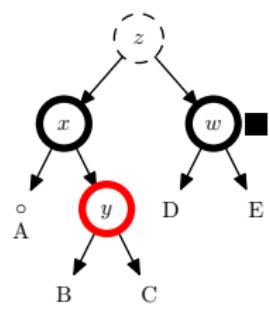
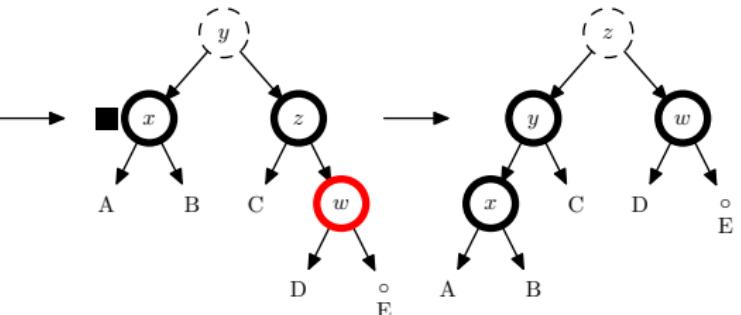
Törles

# Törlés piros-fekete fából (4. eset)

(i) aleset



(ii) aleset



1

<sup>1</sup>Ebben az esetben előállítunk egy extra fekete csomópontot, a tokenet eldobjuk, és készen vagyunk. Ahogy az ábrán látható, a token alatti levelekhez vezető utakon a fekete csomópontok száma eggyel nő, míg a többi útvonalon változatlan marad, és a többi piros-fekete tulajdonság sem sérül.



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Többágú fák

A többágú fák kezelésére nincsenek általános elvek, implementációjuk elsősorban alkalmazásfüggő. A rendezett bináris fáknál bemutatott bejárási algoritmusok közül a **preorder** és a **postorder bejárás** azonban egyszerűen kiterjeszthető a rendezett többágú fák esetére is.

**Reprezentációjuk** az alábbi módokon történhet:

- ▶ A fát **bináris fává** alakítjuk.
- ▶ Az adatelemekben a rákövetkezők számára felső korlátot szabunk. Szétszort reprezentáció esetén ilyenkor az adatrész mellett egy **mutatótömb** jelenik meg.
- ▶ Ha nincs felső korlátja a rákövetkezők számának, akkor a rákövetkezőket címző mutatókat felfűzhetjük egy **egyirányban láncolt listába**.



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

**Többágú fák****B-fa**
 Keresés  
 Bővítés  
 Törlés

# Lapok

A többágú fák speciális esete, ha **lemezen** jelennek meg. A mutatók ilyenkor lemezcímeket tartalmaznak. A fában egyszerre nem egy elemet mozgatunk a memória és a lemez között (hiszen az lassú lenne), hanem egy **elemcsoportot**. A fát felosztjuk **lapakra**, amelyeken a fa több eleme is lehet, és egyszerre egy lapot mozgatunk. A legjobb, ha a lap mérete megegyezik az operációs rendszer által használt blokkmérettel. A fa elemeit két lépésben tudjuk elérni: először a lapot érjük el, azon belül pedig az elemet.

A **B-fa** az alábbi tulajdonságokkal rendelkezik:

- ▶ többágú fa
- ▶ keresőfa
- ▶ kiegyensúlyozott fa
- ▶ az elemek lapokon helyezkednek el



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## B-fa

### A B-fa definíciója és tulajdonságai (1)

A B-fa olyan keresőfa, amely **lapokból** épül fel. A lapokon **mutatók** és **adatelemek** helyezkednek el. Egy-egy lapon az adatelemek maximális számát a B-fa **rendje** határozza meg. Ha a B-fa rendje ***n***, akkor

- ▶ a gyökér lap kivételével minden lapon **legalább *n*** darab adatelemnek kell szerepelnie, és
- ▶ minden lapon **legfeljebb *2n*** darab adatelem helyezkedhet el.

Egy-egy lapon az adatelemek **kulcsaik szerint** növekvőleg rendezettek.

$p_0$	$a_1$	$p_1$	$a_2$	$p_2$	$\dots$	$p_{m-1}$	$a_m$	$p_m$
$k_1$	$<$	$k_2$	$<$			$<$		$k_m$



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

## B-fa

Keresés

Bővítés

Törlés

## A B-fa definíciója és tulajdonságai (2)

Tegyük fel, hogy a B-fa egy lapján az adatelemek aktuális száma éppen  $m$  ( $\leq 2n$ ). Ekkor a B-fa egy lapja vagy levéllap, vagy pontosan  $m + 1$  darab rákövetkezője van. A levéllapon mindegyik mutató értéke NIL.

## Egy nem levéllap

- ▶  $p_0$  mutatója azt a részfát címzi (arra a részfára mutat), amelyben minden kulcsérték kisebb ezen lap  $k_1$  értékű kulcsánál.
- ▶  $p_i$  mutatója ( $0 < i < m$ ) azt a részfát címzi, amelyben minden kulcsérték nagyobb ezen lap  $k_{i-1}$  értékű kulcsánál és kisebb a  $k_i$  értékű kulcsánál.
- ▶  $p_m$  mutatója azt a részfát címzi, amelyben minden kulcsérték nagyobb ezen lap  $k_m$  értékű kulcsánál.

A B-fában a levéllapok azonos szinten helyezkednek el.

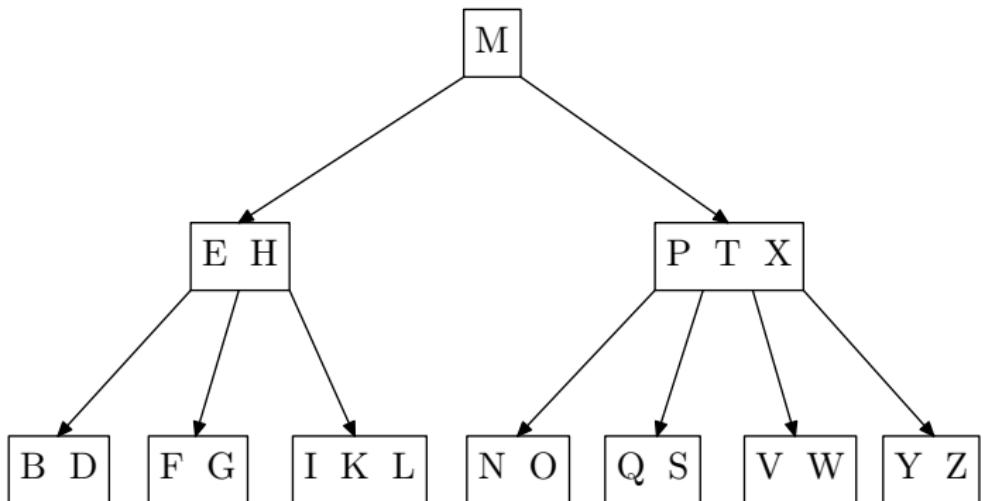
## Példa B-fára

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Egy másodrendű B-fa karakter típusú kulcsokkal:



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Keresés B-fában

A B-fában a  $K$  kulcsot keressük.

- ① Ha a B-fa **üres**, az algoritmus véget ér, a keresett elem **nincs** a B-fában.
- ② A B-fa gyökérlapján **lineáris keresést** hajtunk végre. Ha a lineáris keresés **sikeres, megtaláltuk** a keresett elemet, az algoritmus véget ér.
- ③ Ha a lineáris keresés **sikertelen**, akkor
  - vagy a gyökérlap első olyan eleménél ( $a_i$ ), melynek kulcsa ( $k_i$ ) nagyobb, mint  $K$ ,
  - vagy a gyökérlap **utolsó eleme után**

áll meg a lineáris keresés. Előbbi esetben a  $p_{i-1}$  mutató által mutatott részfában, utóbbi esetben a  $p_m$  mutató által mutatott részfában folytatjuk a keresést (rekurzívan).

Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# B-fa bővítése

## B-fa bővítése

A B-fát minden levéllapon bővítjük, ha a bővítendő elem még nem szerepel a B-fában.

- ① Megkeressük azt a **levéllapot**, ahová a bővítendő adatelemet el fogjuk helyezni.
- ② Ha a levéllapon  **$2n$ -nél kevesebb** elem szerepel, akkor a rendezettség figyelembevételével **elhelyezzük** a bővítendő elemet a lapon.
- ③ Ha a levéllapon a bővítést megelőzően **már  $2n$  elem** szerepelne (**tele van**), akkor a – bővített elemmel együtt –  **$2n + 1$**  elem közül kiválasztjuk a rendezettség szerinti **középsőt**, amit kiemelünk ezen elemek közül, s beszűrunk a **szülőlapra**. A megmaradt elemeket két lapra osztjuk szét: az **első  $n$**  darab a kiemelt középső elem bal oldali mutatója által mutatott lapra, az **utolsó  $n$**  darab elem pedig a kiemelt középső elem jobb oldali mutatója által mutatott lapra kerül.

# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

C

# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

C N

# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

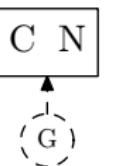
Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

C	G	N
---	---	---

# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

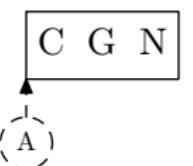
Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

A	C	G	N
---	---	---	---



## Példa B-fa bővítésére



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

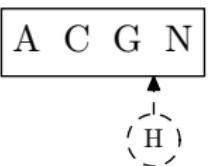
Többágú fák

B-fa

Keresés

Bővítés

Törlés





## Példa B-fa bővítésére



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

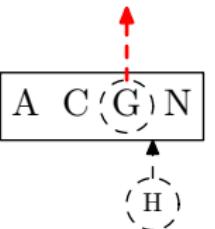
Piros-fekete fa

Okasaki-féle beszűrás  
CLRS-féle beszűrás  
Törlés

Többágú fák

B-fa

Keresés  
Bővítés  
Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

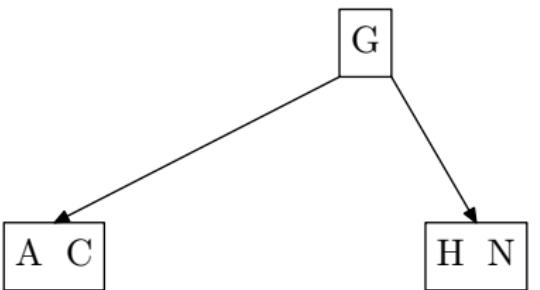
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

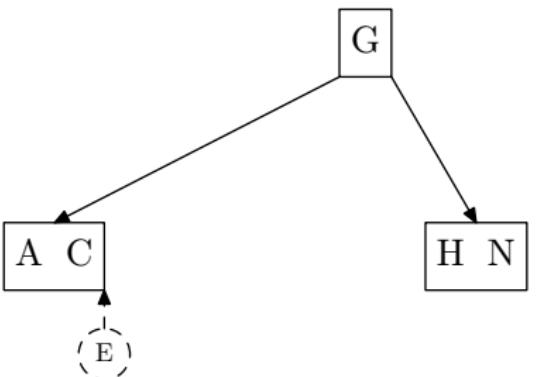
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

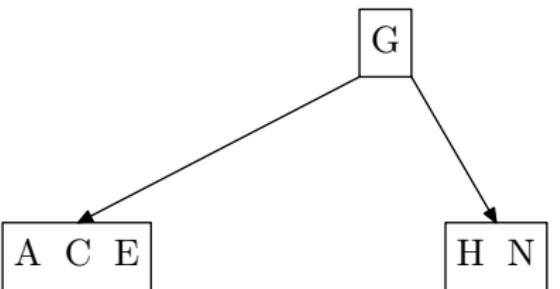
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

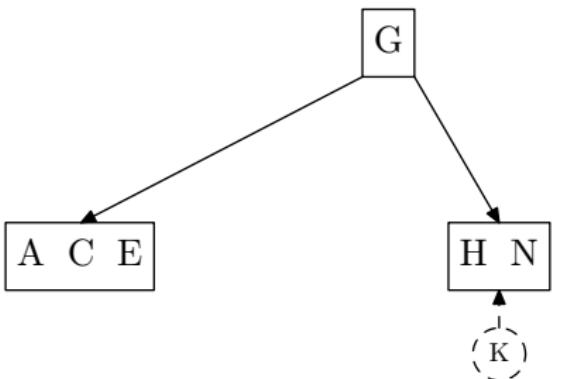
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

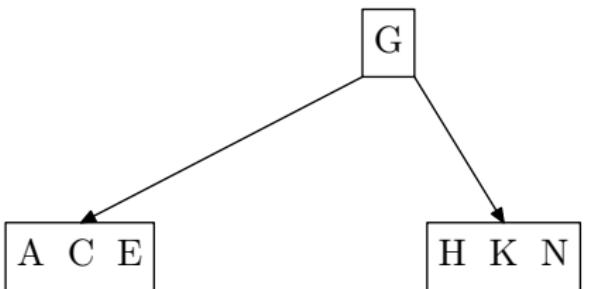
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

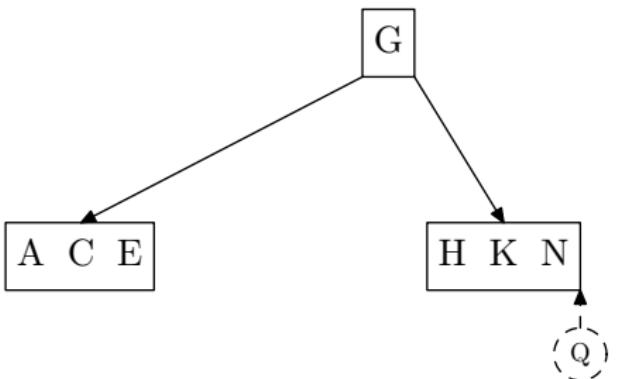
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

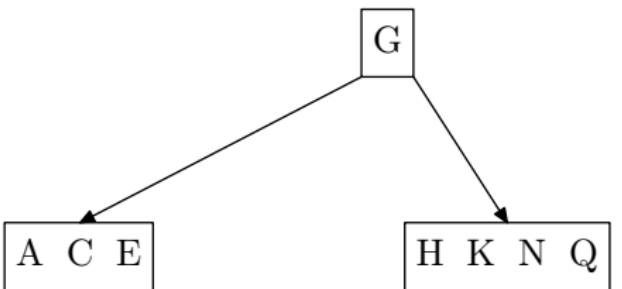
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

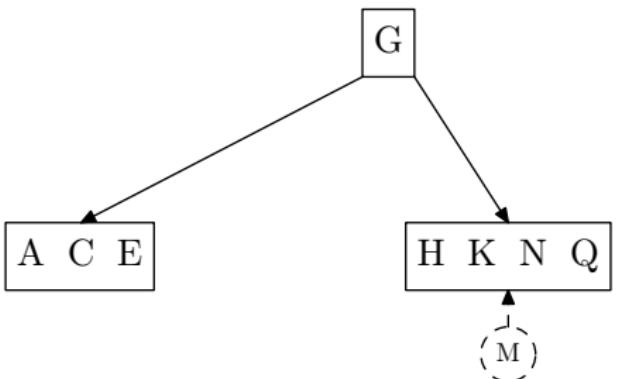
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

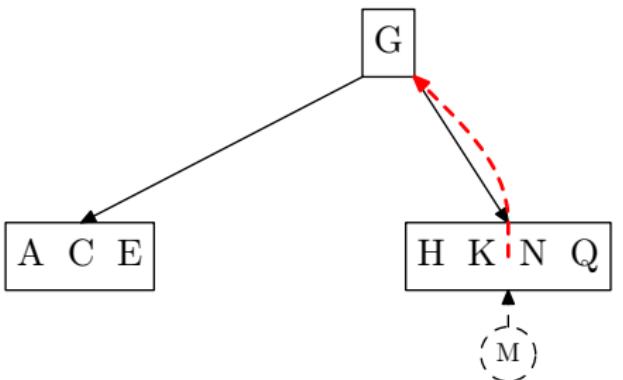
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

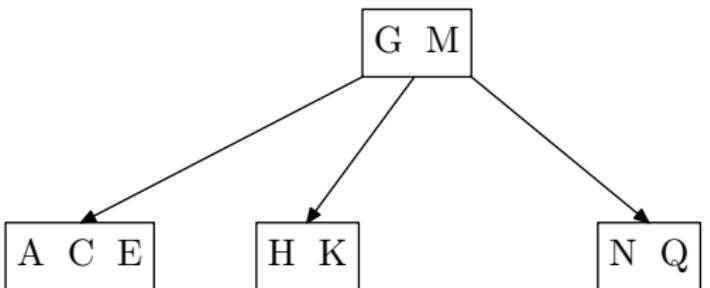
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

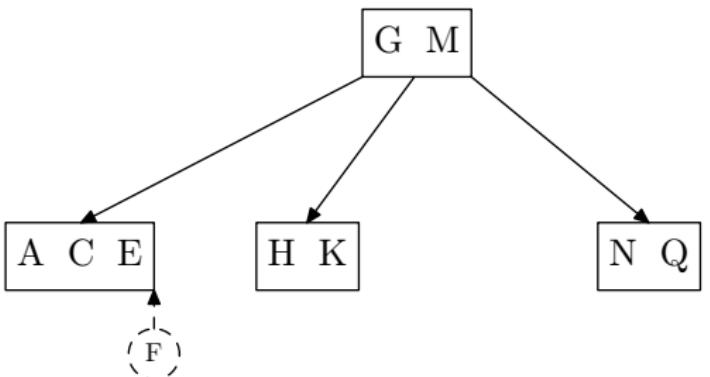
Többágú fák

B-fa

Keresés

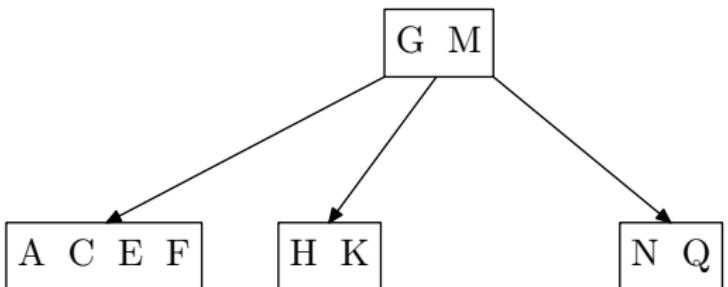
Bővítés

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U



Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

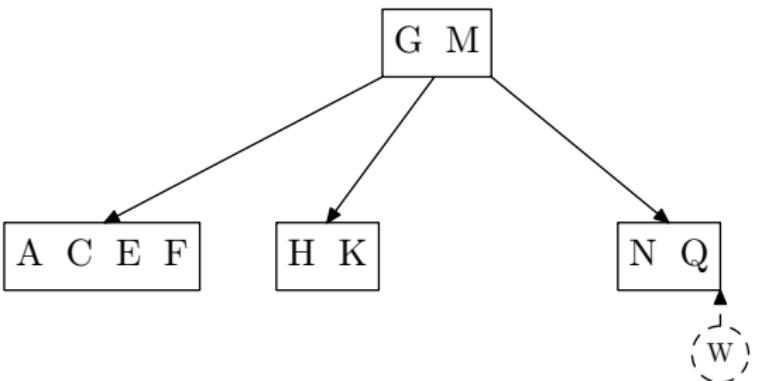
Többágú fák

B-fa

Keresés

**Bővítés**

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

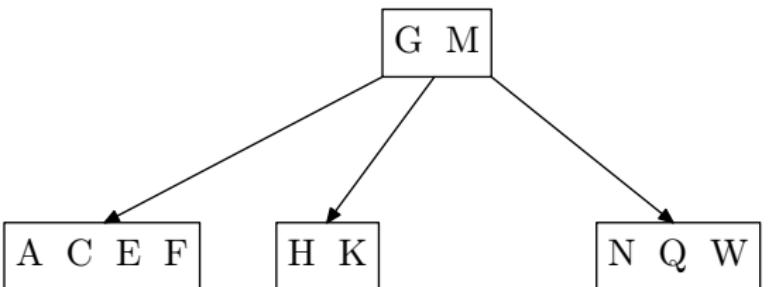
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

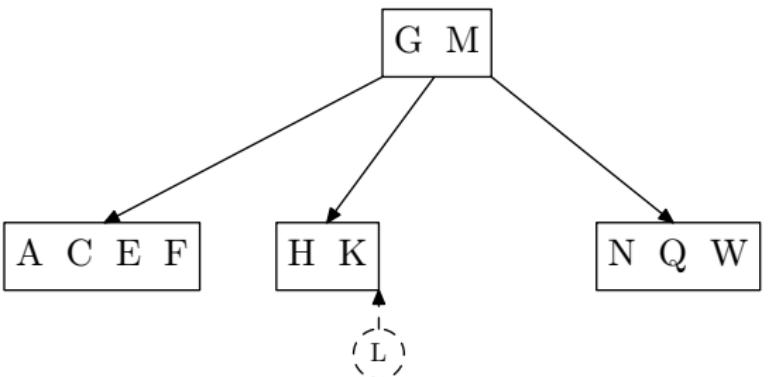
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

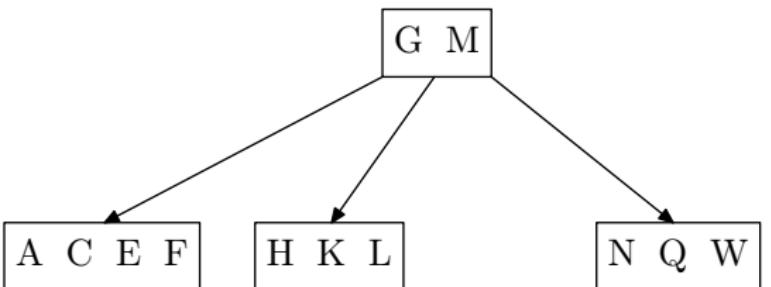
Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

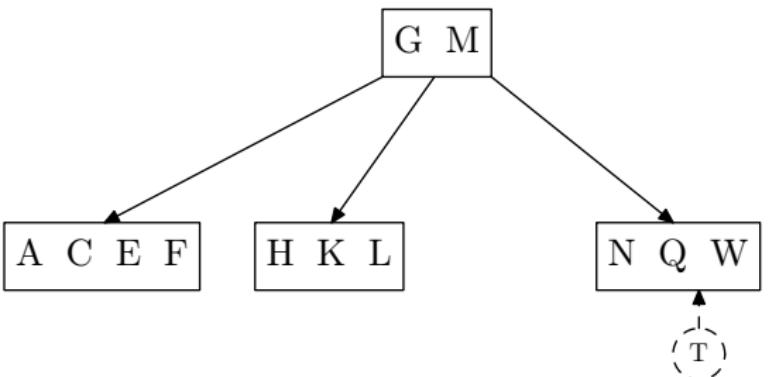
Többágú fák

B-fa

Keresés

Bővítés

Törlés





## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

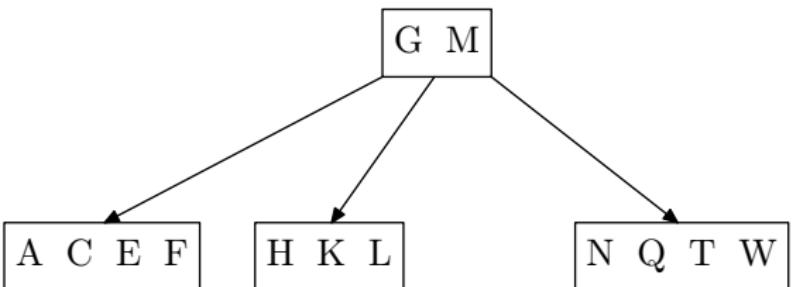
Többágú fák

B-fa

Keresés

**Bővítés**

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

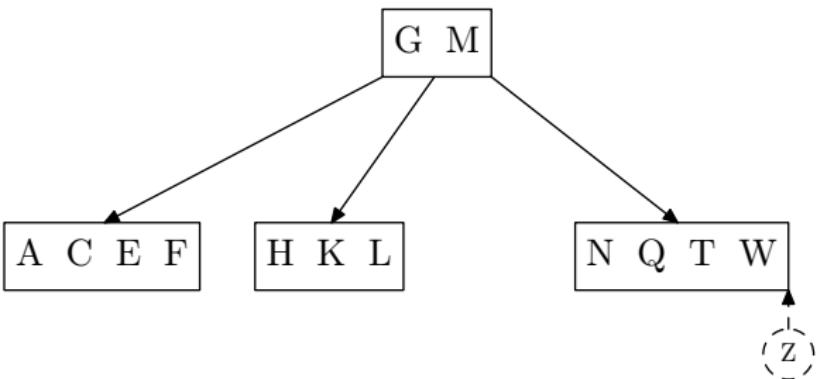
Többágú fák

B-fa

Keresés

Bővítés

Törlés



# Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

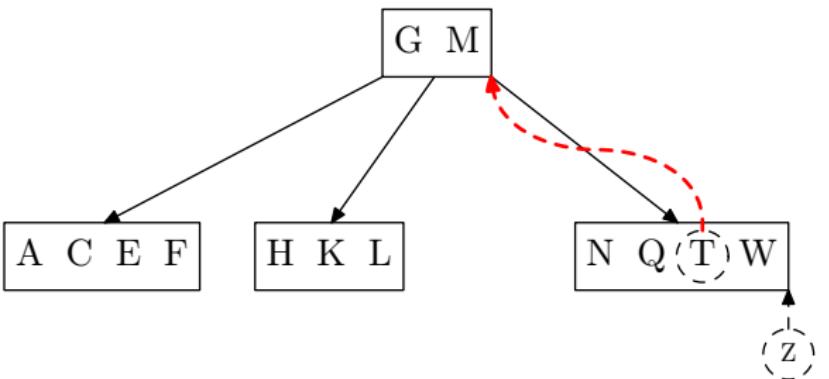
Többágú fák

B-fa

Keresés

Bővítés

Törlés





## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

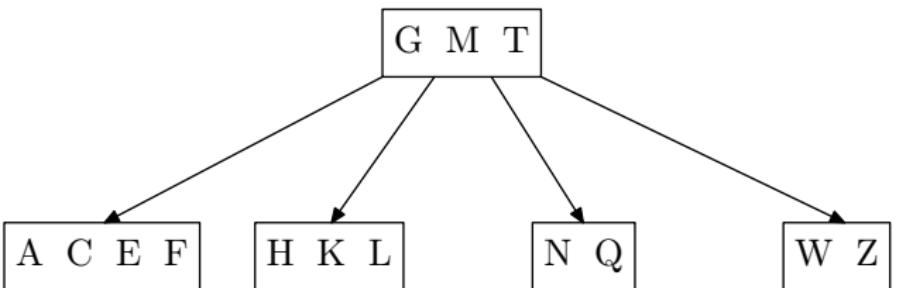
Többágú fák

B-fa

Keresés

**Bővítés**

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

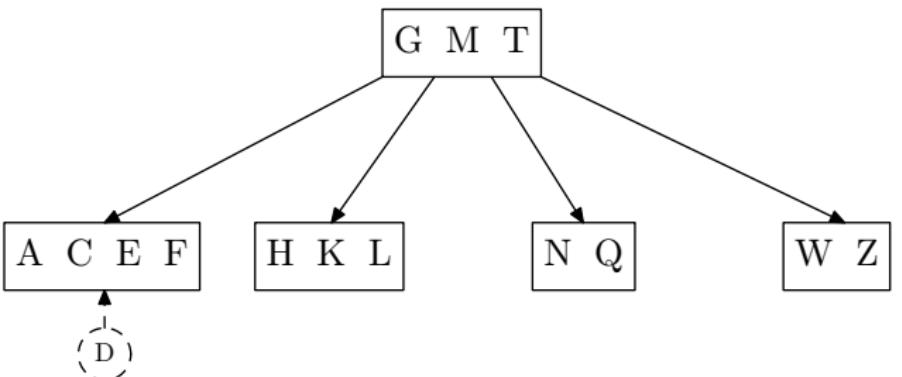
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

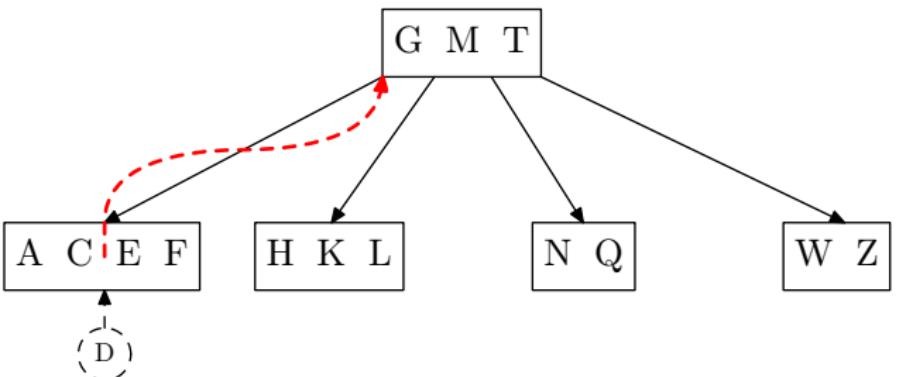
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

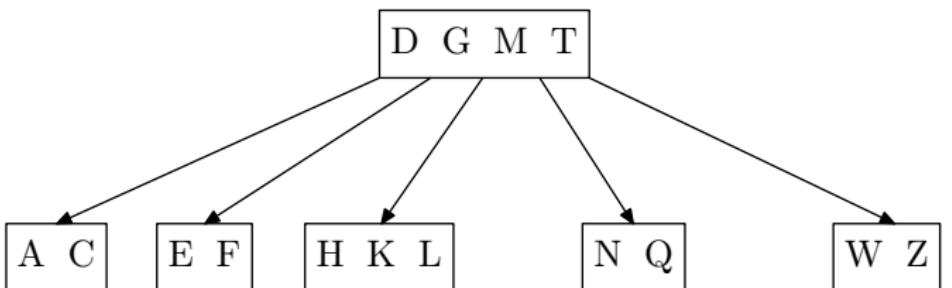
Többágú fák

B-fa

Keresés

Bővítés

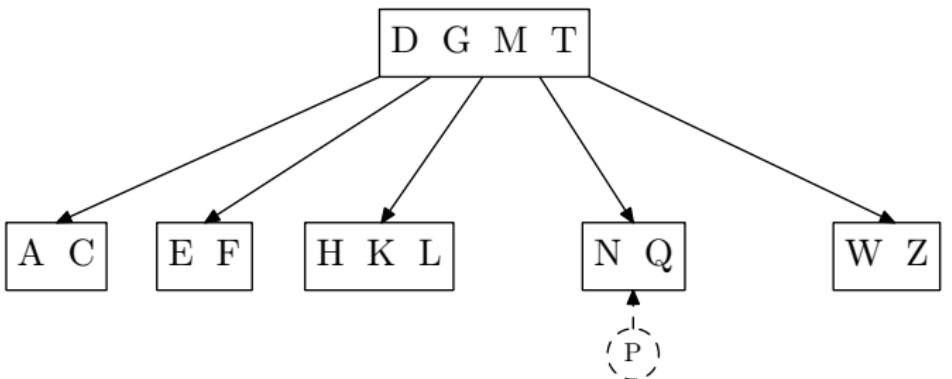
Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

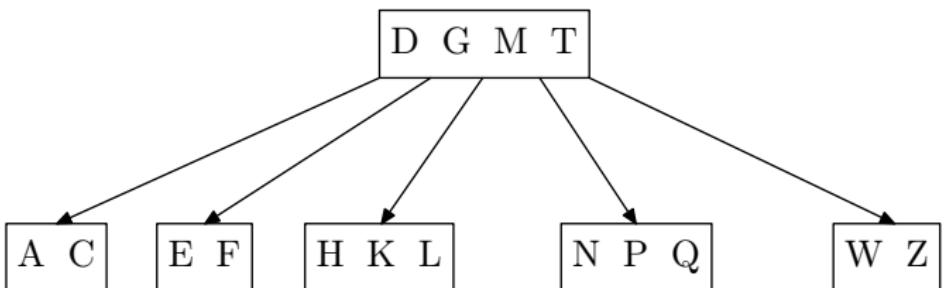
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

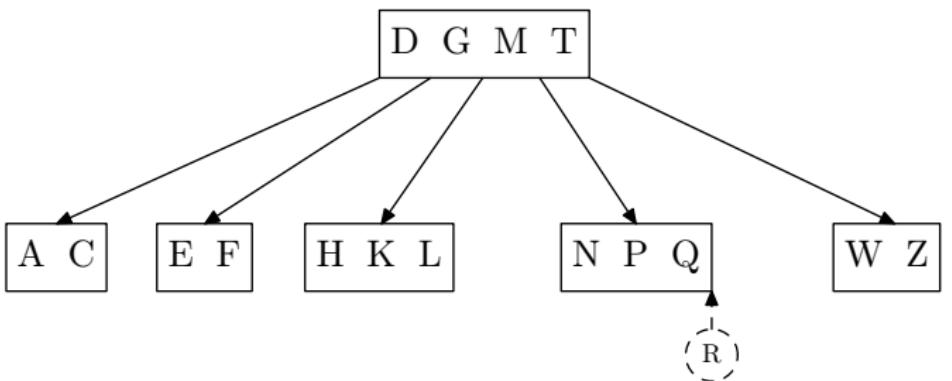
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

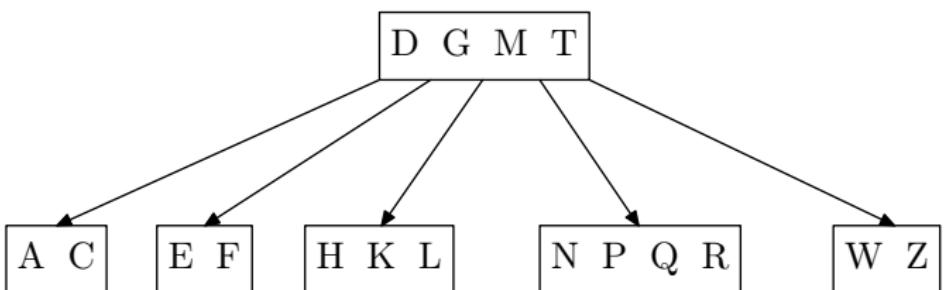
Többágú fák

B-fa

Keresés

Bővítés

Törlés





Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

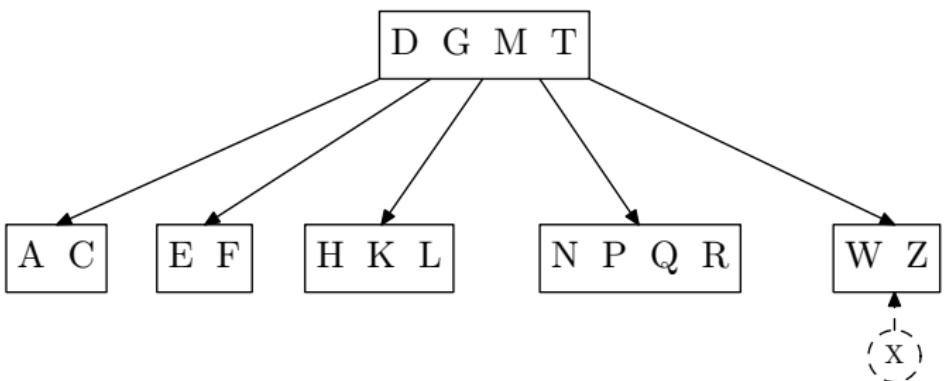
Keresés

Bővítés

Törlés

## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U





Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

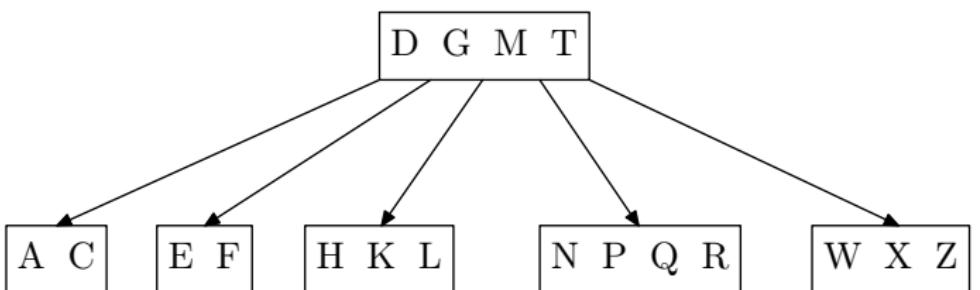
B-fa

Keresés

Bővítés

Törlés

## Példa B-fa bővítésére



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

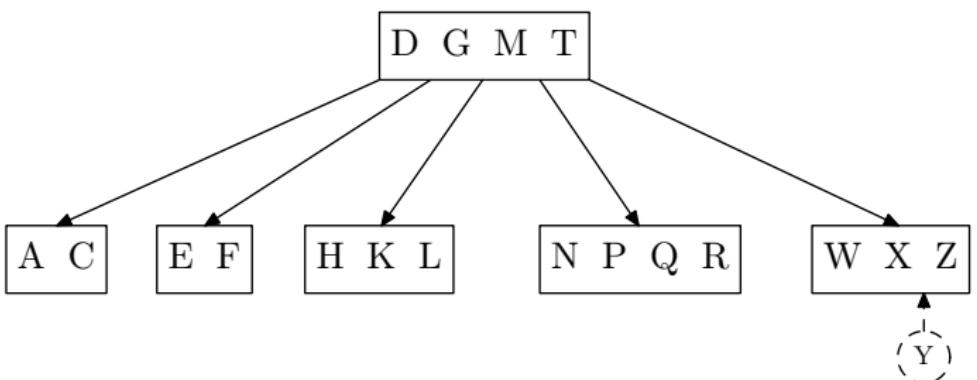
Többágú fák

B-fa

Keresés

Bővítés

Törlés





## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

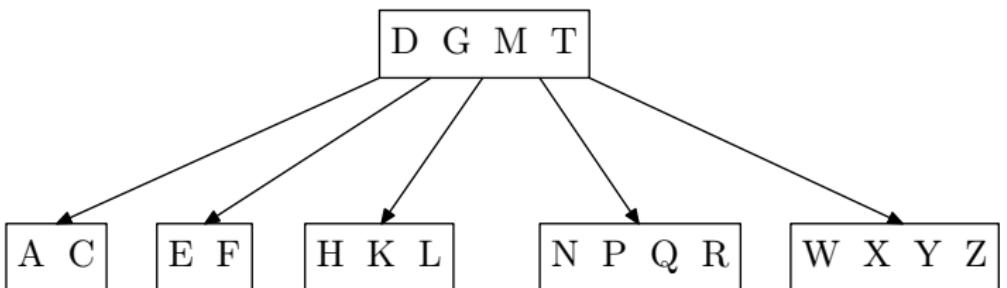
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U

Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

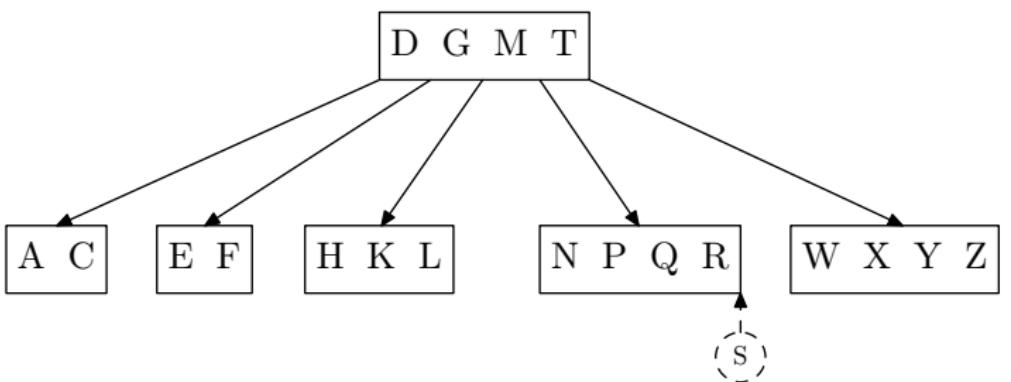
Többágú fák

B-fa

Keresés

Bővítés

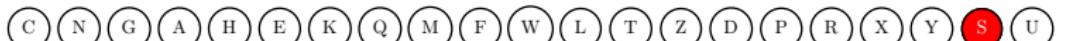
Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

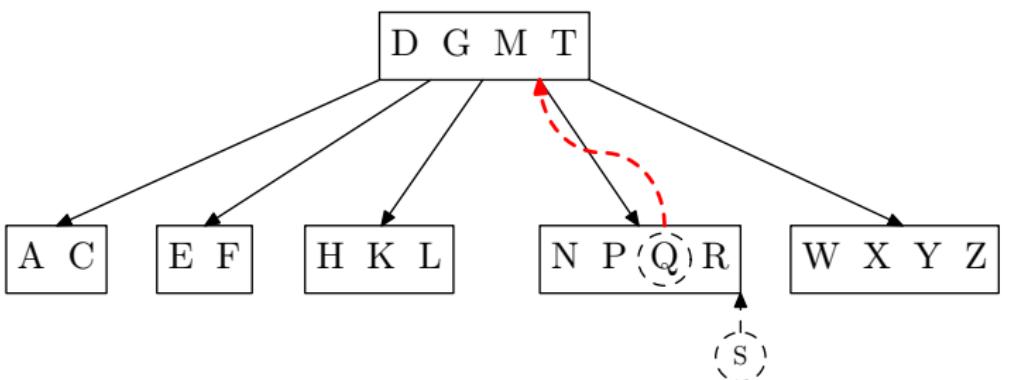
Többágú fák

B-fa

Keresés

Bővítés

Törlés



## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

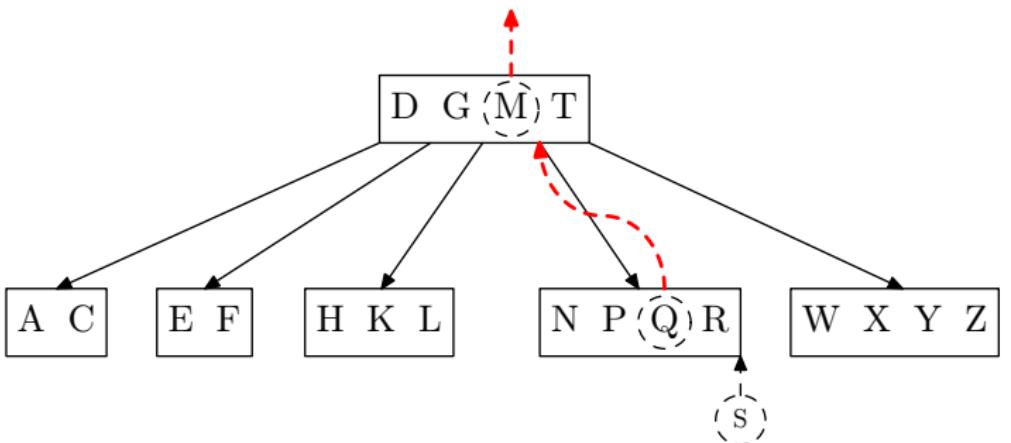
Többágú fák

B-fa

Keresés

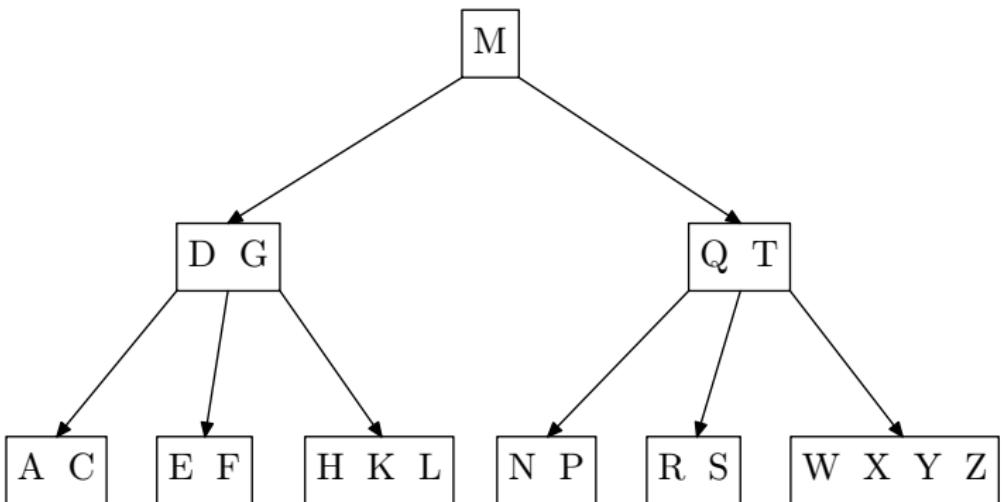
Bővítés

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

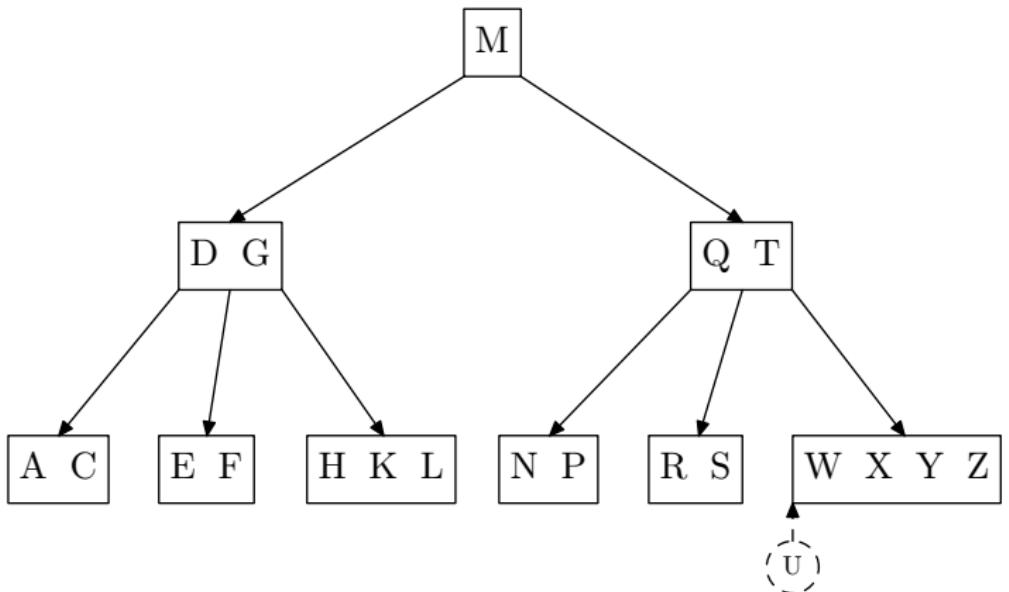
## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

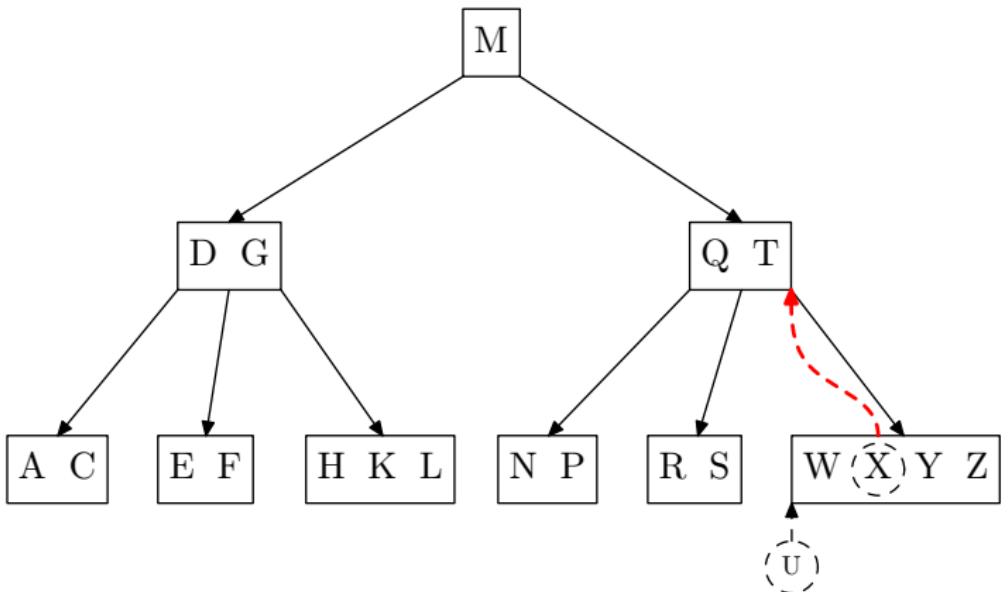
## Példa B-fa bővítésére

Bináris keresőfák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



C N G A H E K Q M F W L T Z D P R X Y S U



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

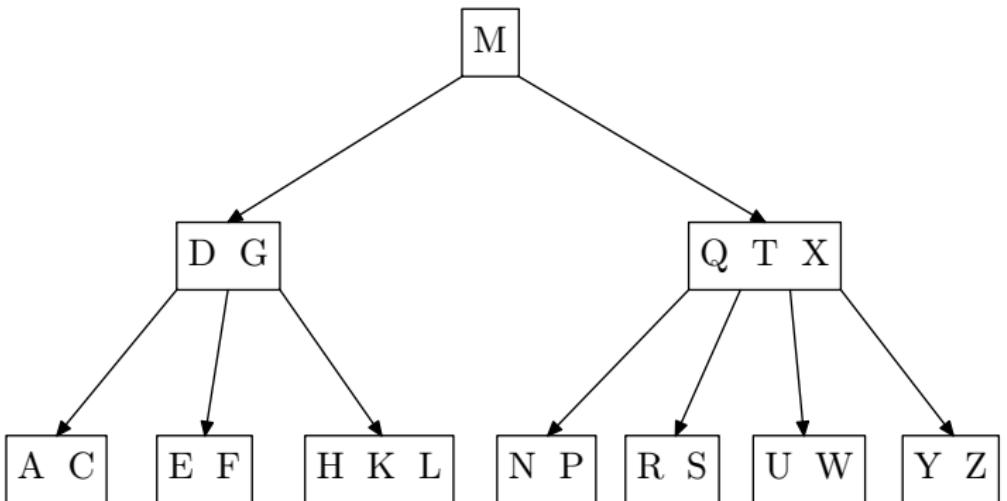
Bővítés

Törlés



## Példa B-fa bővítésére

C N G A H E K Q M F W L T Z D P R X Y S U



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

**Bővítés**

Törlés



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

## Törlés B-fából

Ha a törlendő elem nem levéllapon van, akkor felülírjuk azzal a levéllapon lévő elemmel, amelyik a törlendő elem inorder módon megelőzője (vagy rákövetkezője). Fizikailag tehát minden esetben levéllapról törlünk.

A törlés után előfordulhat, hogy a levéllapon  $n - 1$  elem marad. Ekkor két eset lehetséges:

- ① Ha valamelyik szomszédos testvérlap legalább  $n + 1$  elemet tartalmaz, akkor a közös szülőlapjukon keresztül átveszünk tőle egy elemet. Ez azt jelenti, hogy a testvérlap legszélső eleme a szülőlapra kerül a megfelelő helyre, az eredetileg ott lévő elem pedig lekerül az  $n - 1$  elemet tartalmazó gyermeklapra. Ezzel egyidejűleg a testvérlapon lévő legszélső mutató (és így az onnan kiinduló részfa) is átkerül a „csonka” lapra. Ez a művelet tehát hasonló az AVL-fa forgatás műveletéhez.
- ② Ha minden szomszédos testvérlap  $n$  elemet tartalmaz (vagy csak egy szomszédos testvérlap van, és az  $n$  elemet tartalmaz), akkor valamelyikükkel **lapösszevonást** kell végrehajtani. Ez azt jelenti, hogy ebből a két lapból, valamint a szülőlapon a két lap közötti elemből egy  $2n$  elemet tartalmazó lapot képzünk. Ezáltal a szülőlapon egyelőre csökken az elemek száma, így előfordulhat, hogy ott is  $n - 1$  elem marad. A fenti lépésekkel tehát iterálni kell. Legrosszabb esetben a lapösszevonás egészen a gyökérlapig felgyűrűzhet, ilyenkor egyelőre csökken a fa magassága.

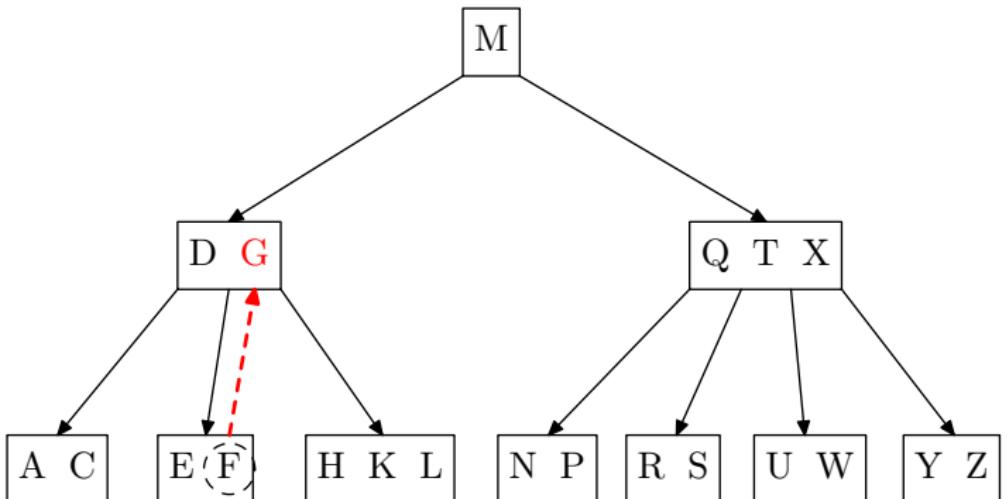
# Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

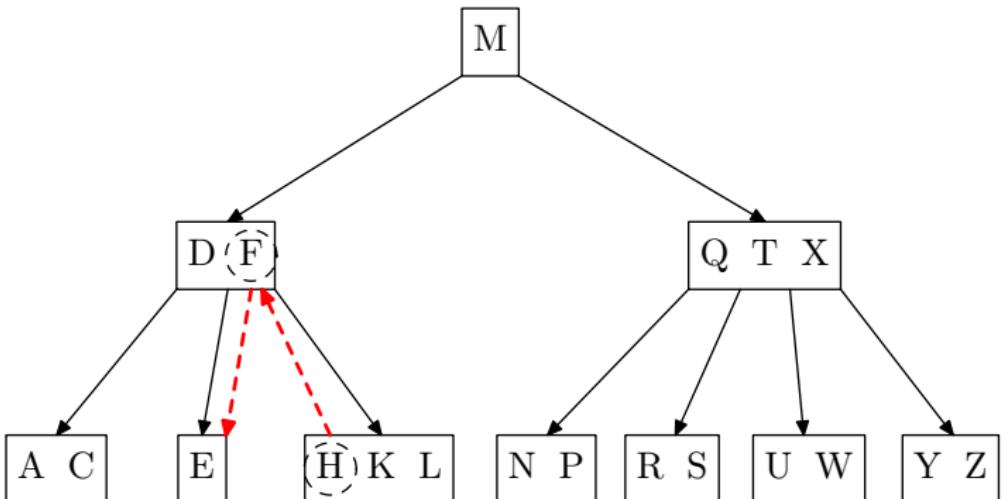
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

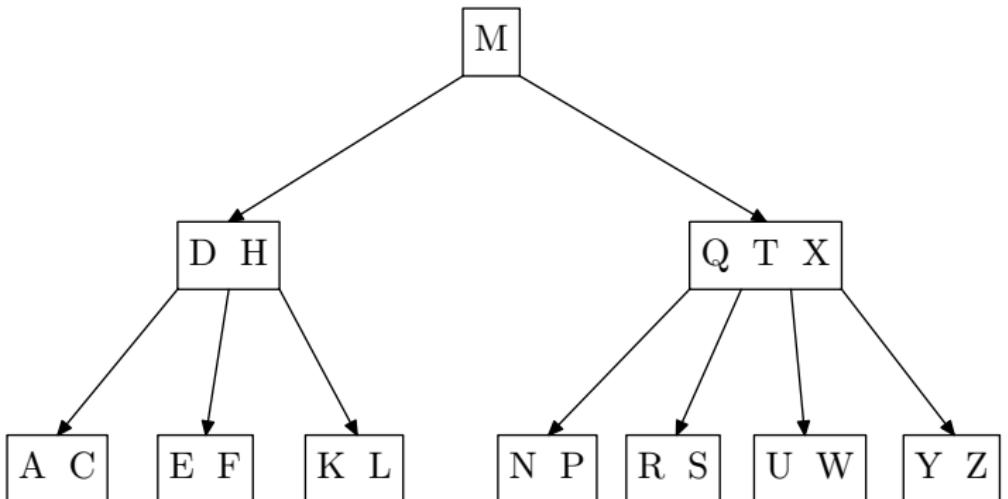
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

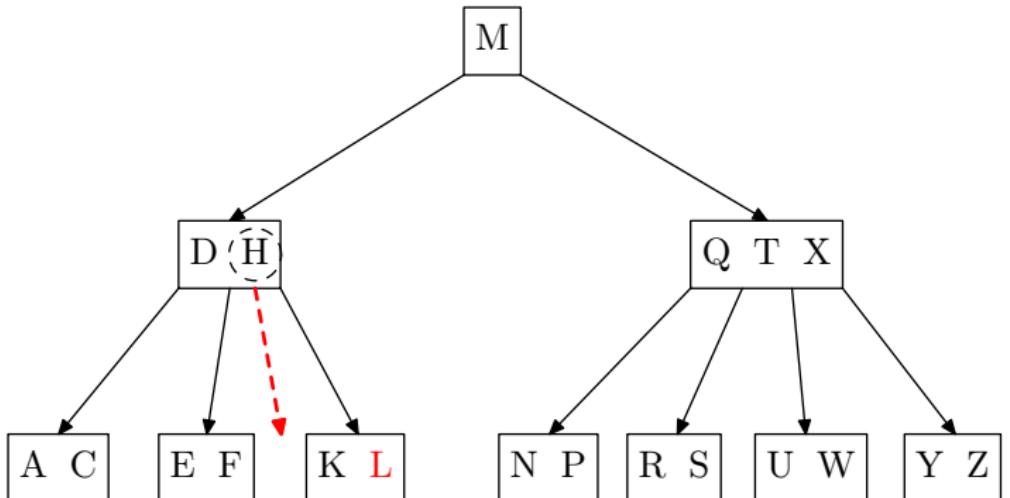
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(G) (L) (Q) (X)



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

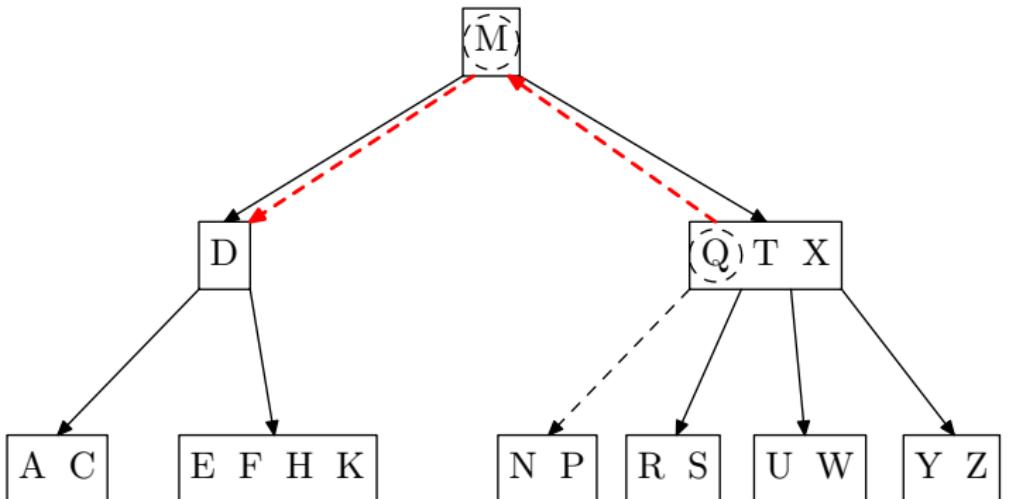
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(G) (L) (Q) (X)



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

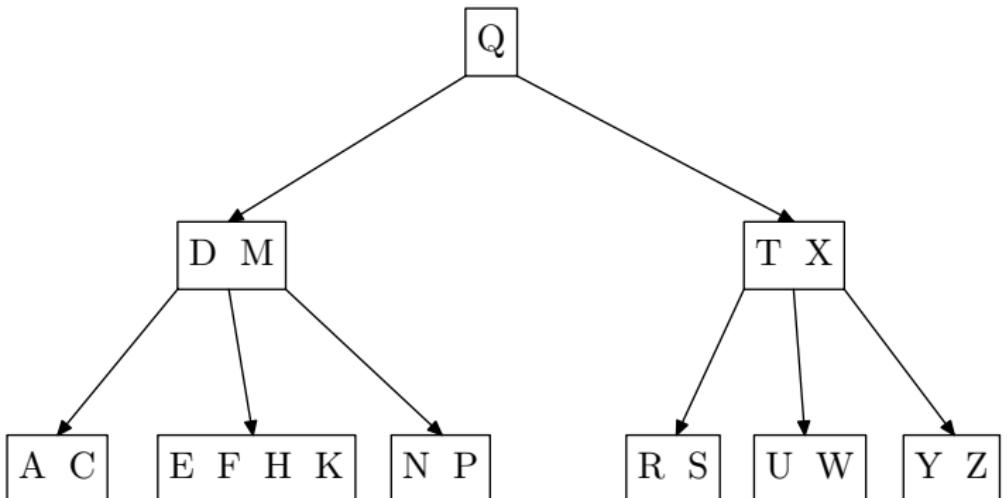
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(G) (L) (Q) (X)



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

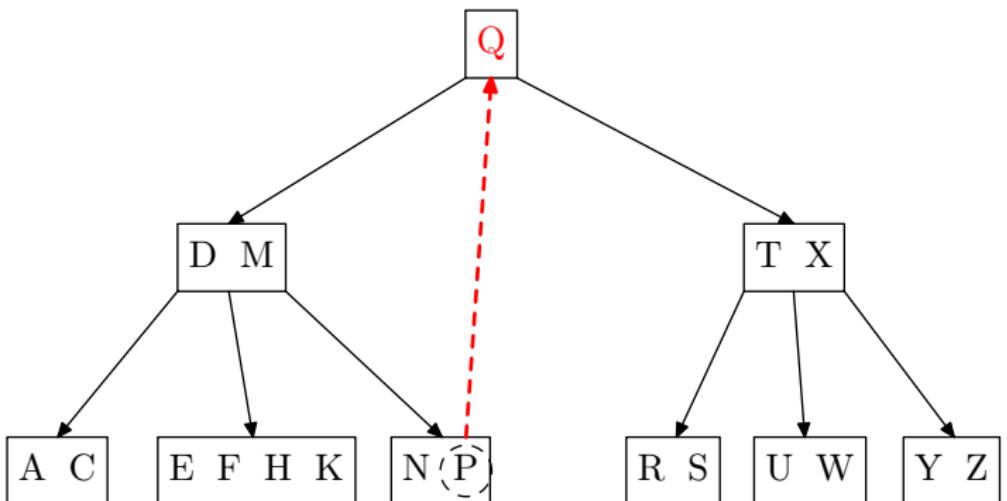
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

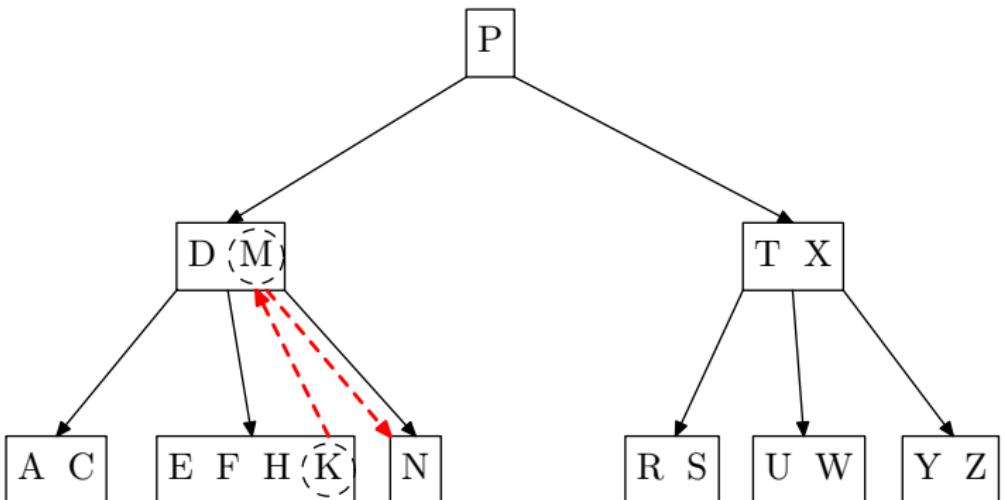
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(G) (L) (Q) (X)



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

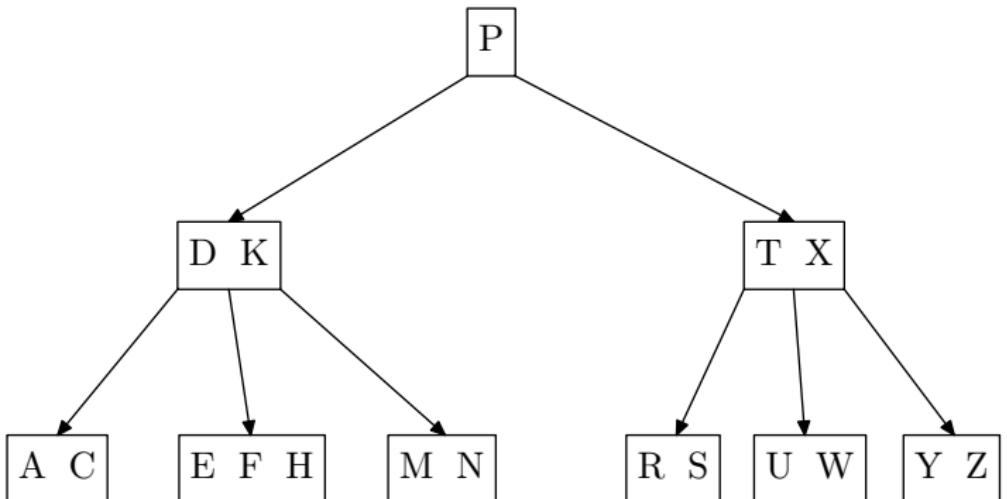
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



(G) (L) (Q) (X)



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

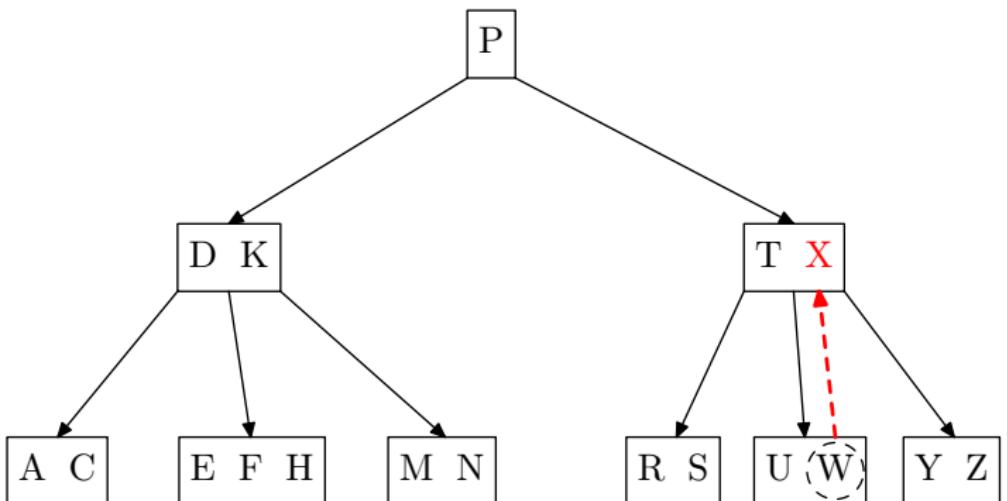
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegynsúlyozottság

Bináris keresőfa

Kiegynsúlyozott fa

Kiegynsúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

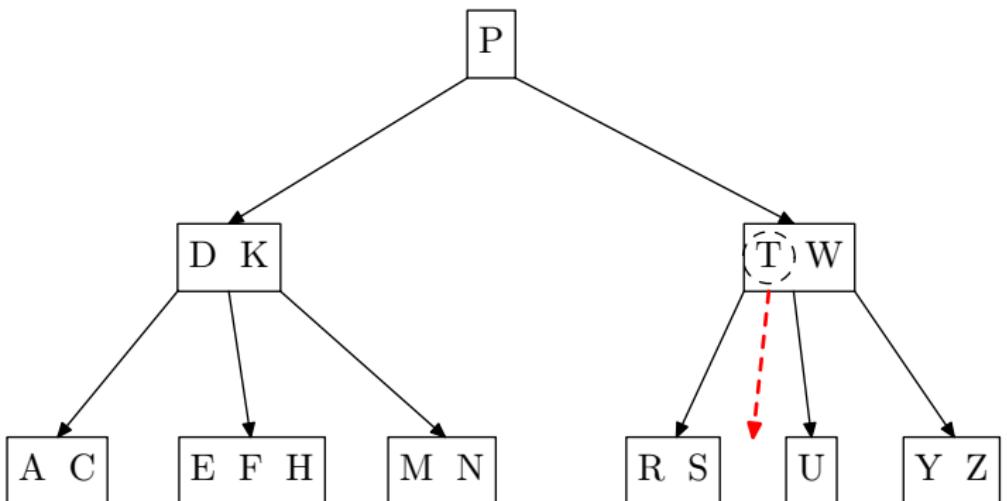
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

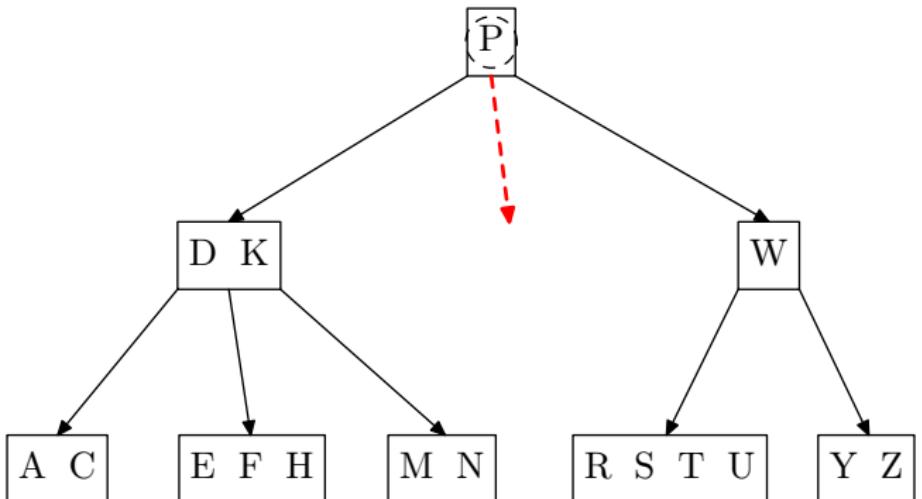
## Példa B-fából való törlésre

Bináris keresőfák

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



G L Q X



Kiegysúlyozottság

Bináris keresőfa

Kiegysúlyozott fa

Kiegysúlyozott  
keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

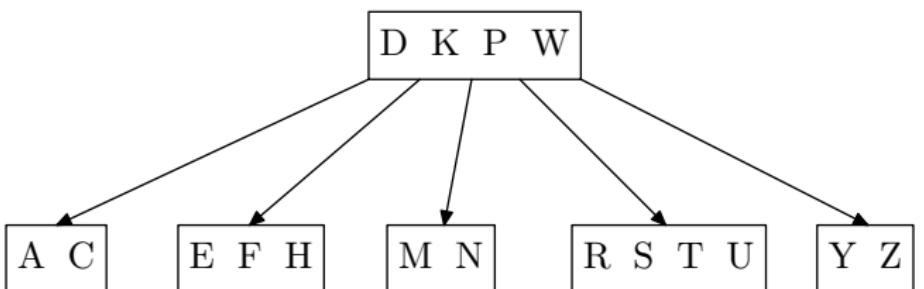
Bővítés

Törlés



## Példa B-fából való törlésre

G L Q X



Kiegyensúlyozottság

Bináris keresőfa

Kiegyensúlyozott fa

Kiegyensúlyozott keresőfa

Piros-fekete fa

Okasaki-féle beszűrás

CLRS-féle beszűrás

Törlés

Többágú fák

B-fa

Keresés

Bővítés

Törlés

# 7. előadás

## Rendező algoritmusok

Gyorsrendezés, rendezés lineáris lépésszámmal

*Adatszerkezetek és algoritmusok előadás*

2018. március 6.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

# Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Rendező algoritmusok

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



## A gyorsrendezés leírása

A gyorsrendezés, az összefésülő rendezéshez hasonlóan az oszd-meg-és-uralkodj elven alapszik.

**Felosztás:** Az  $A[p \dots r]$  tömböt két (esetleg üres)

$A[p \dots q-1]$  és  $A[q+1 \dots r]$  résztömbre osztjuk úgy, hogy az  $A[p \dots q-1]$  minden eleme kisebb vagy egyenlő  $A[q]$ -nál, ez utóbbi elem viszont kisebb vagy egyenlő  $A[q+1 \dots r]$  minden eleménél. A  $q$  index kiszámítása része ennek a felosztó eljárásnak.

**Uralkodás:** Az  $A[p \dots q-1]$  és  $A[q+1 \dots r]$  résztömbököt a gyorsrendezés rekurzív hívásával rendezzük.

**Összevonás:** Mivel a két résztömböt helyben rendeztük, nincs szükség egyesítésre: az egész  $A[p \dots r]$  tömb rendezett.

### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Édényrendezés

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



## A gyorsrendezés egy algoritmusa

```

procedure GYORS_RENDEZ(A, p, r)
1: if p < r then
2:   q  $\leftarrow$  FELOSZT(A, p, r)
3:   GYORS_RENDEZ(A, p, q - 1)
4:   GYORS_RENDEZ(A, q + 1, r)
5: end if
end procedure
```

```
function FELOSZT(A, p, r)
```

```

1: x  $\leftarrow$  A[r]
2: i  $\leftarrow$  p - 1
3: for j  $\leftarrow$  p to r - 1 do
4:   if A[j]  $\leq$  x then
5:     i  $\leftarrow$  i + 1
6:     A[i] és A[j] felcserélése
7:   end if
8: end for
9: A[i + 1] és A[r] felcserélése
10: return i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a  $k$  tömbindexre fennáll:

1. Ha  $p \leq k \leq i$ , akkor  $A[k] \leq x$ .
2. Ha  $i < k < j$ , akkor  $A[k] > x$ .
3. Ha  $k = r$ , akkor  $A[k] = x$ .
4. Ha  $j \leq k < r$ , akkor  $A[k] ???$

### Gyorsrendezés

#### A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlentíett változata

#### Összehasonlítható rendezések

A döntésífa-modell

Összehasonlítható rendezések hatékonysága

#### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



## A gyorsrendezés egy algoritmusá

```

procedure GYORS_RENDEZ(A, p, r)
1: if p < r then
2:   q ← FELOSZT(A, p, r)
3:   GYORS_RENDEZ(A, p, q - 1)
4:   GYORS_RENDEZ(A, q + 1, r)
5: end if
end procedure
function FELOSZT(A, p, r)
1: x ← A[r]
2: i ← p - 1
3: for j ← p to r - 1 do
4:   if A[j] ≤ x then
5:     i ← i + 1
6:     A[i] és A[j] felcserélése
7:   end if
8: end for
9: A[i + 1] és A[r] felcserélése
10: return i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a  $k$  tömbindexre fennáll:

1. Ha  $p \leq k \leq i$ , akkor  $A[k] \leq x$ .
2. Ha  $i < k < j$ , akkor  $A[k] > x$ .
3. Ha  $k = r$ , akkor  $A[k] = x$ .
4. Ha  $j \leq k < r$ , akkor  $A[k] ???$

### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

Kósá Márk

Pánovics János

Szathmáry László

Halász Gábor



## A gyorsrendezés egy algoritmusá

```

procedure GYORS_RENDEZ(A, p, r)
1: if p < r then
2:   q ← FELOSZT(A, p, r)
3:   GYORS_RENDEZ(A, p, q - 1)
4:   GYORS_RENDEZ(A, q + 1, r)
5: end if
end procedure
function FELOSZT(A, p, r)
1: x ← A[r]
2: i ← p - 1
3: for j ← p to r - 1 do
4:   if A[j] ≤ x then
5:     i ← i + 1
6:     A[i] és A[j] felcserélése
7:   end if
8: end for
9: A[i + 1] és A[r] felcserélése
10: return i + 1
end function
```

A 3–6. sorokban levő ciklus minden iterációjának kezdetén a  $k$  tömbindexre fennáll:

1. Ha  $p \leq k \leq i$ , akkor  $A[k] \leq x$ .
2. Ha  $i < k < j$ , akkor  $A[k] > x$ .
3. Ha  $k = r$ , akkor  $A[k] = x$ .
4. Ha  $j \leq k < r$ , akkor  $A[k] ???$

### Gyorsrendezés

#### A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

#### Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

#### Rendezés lineáris időben

Leszámláló rendezés

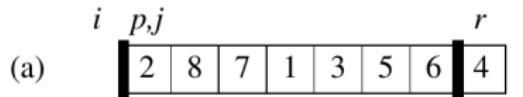
Számjegyes rendezés

Edényrendezés

# A FELOSZT algoritmus működése

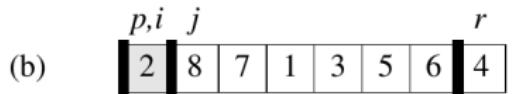
Rendező algoritmusok

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



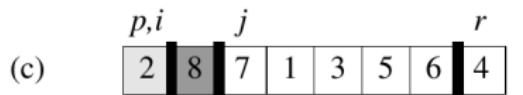
(e)

$p$	$i$	$j$	$r$
2	1	7	8 3 5 6 4



(f)

$p$	$i$	$j$	$r$
2	1	3	8 7 5 6 4



(g)

$p$	$i$	$j$	$r$
2	1	3	8 7 5 6 4

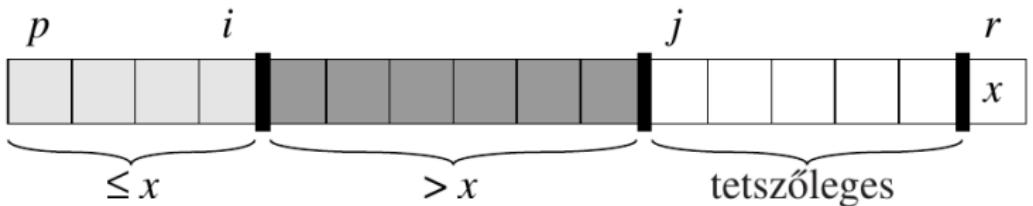


(h)

$p$	$i$	$j$	$r$
2	1	3	8 7 5 6 4

(i)

$p$	$i$	$j$	$r$
2	1	3	4 7 5 6 8



## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlentített változata

## Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

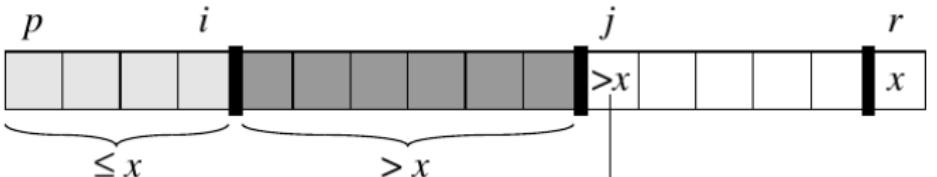
Számjegyes rendezés

Edényrendezés

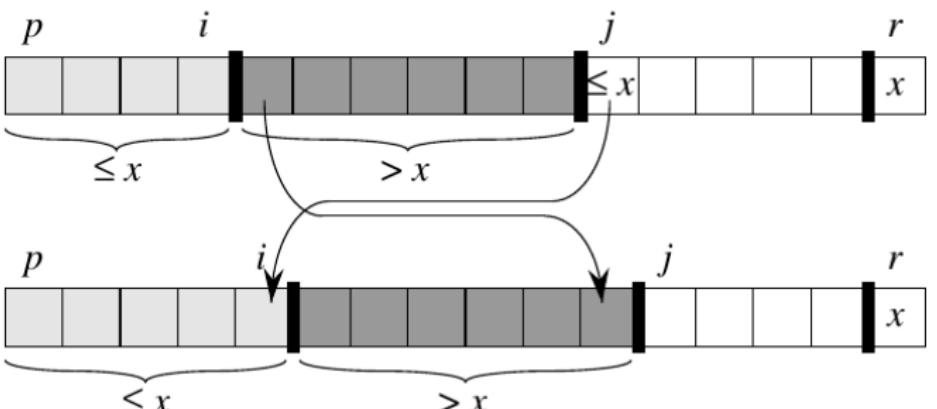


# A FELOSZT algoritmus működése

(a)



(b)



## Gyorsrendezés

[A gyorsrendezés leírása](#)

A gyorsrendezés hatékonysága

A gyorsrendezés véletlennel változata

## Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## Elemzés

**Teljesül:** A ciklus első iterációja előtt  $i = p - 1$  és  $j = p$ .

Mivel egyetlen elem sincs  $p$  és  $i$  között, sem pedig  $i + 1$  és  $j - 1$  között, a ciklusinvariáns első két feltétele triviálisan igaz. Az 1. sor értékadása miatt a harmadik feltétel is igaz.

**Megmarad:** Amikor  $A[j] > x$ , akkor csupán a  $j$  értékét kell növelni. Amikor  $j$  értéke megnő, a második feltétel  $A[j-1]$ -re igaz, míg a többi elem változatlan marad.

Amikor  $A[j] \leq x$ , akkor  $i$  megnő,  $A[i]$  és  $A[j]$  felcserélődik, majd  $j$  megnő. A csere miatt most  $A[i] \leq x$ , és az 1. feltétel teljesül. Mivel a ciklusinvariáns miatt az az elem, amely az  $A[j - 1]$ -be került nagyobb, mint  $x$ :  $A[j - 1] > x$ .

**Befejeződik:** Befejezéskor  $j = r$ , ezért a tömb minden eleme az invariáns által leírt valamelyik halmazban van. A tömb elemeit három halmazba tettük: az elsőben  $x$ -nél kisebbek vagy vele egyenlők vannak, a másodikban  $x$ -nél nagyobbak, és a harmadikban csak az  $x$ .



### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

**Gyorsrendezés**

A gyorsrendezés leírása

**A gyorsrendezés hatékonysága**

A gyorsrendezés véletlenített változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonysága

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# A gyorsrendezés hatékonysága

## Legrosszabb felosztás

A gyorsrendezés legrosszabb esete az, amikor a felosztó eljárás az eredeti tömböt egy  $n - 1$  és egy  $0$  elemű tömbre osztja.

A felosztási idő  $\Theta(n)$ . A rekurzív hívás egy  $0$  nagyságú tömbre nem csinál egyebet, éppen csak visszatér, ezért  $T(0) = \Theta(1)$ , és a gyorsrendezés futási idejének rekurzív képlete:

$$\begin{aligned} T(n) &= T(n - 1) + T(0) + \Theta(n) \\ &= T(n - 1) + \Theta(n). \end{aligned}$$

Intuitív módon, egy számtani sorhoz jutunk, aminek az összege alapján azt várhatjuk:  $T(n) = \Theta(n^2)$ .

Ezt pl. a helyettesítő módszerrel könnyű ellenőrizni.



## Gyorsrendezés

A gyorsrendezés leírása

## A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# A gyorsrendezés hatékonysága

## Legjobb felosztás

A legjobb felosztásnál a FELOSZT eljárás két,  $n/2$  eleműnél nem nagyobb tömböt hoz létre, mivel az egyik  $\lfloor n/2 \rfloor$ , a másik pedig  $\lceil n/2 \rceil - 1$  elemű.

A futási idő rekurzív képlete

$$T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n),$$

amelynek megoldása a mester téTEL második esete alapján

$$T(n) = \Theta(n \lg n),$$

így ez a legjobb felosztás aszimptotikusan gyorsabb algoritmust eredményez.



## Theorem (Mester téTEL)

Legyenek  $a \geq 1$ ,  $b > 1$  állandók,  $f(n)$  egy függvény,  $T(n)$  pedig a nemnegatív egészeken a

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

rekurzív egyenlettel definiált függvény, ahol  $n/b$  jelentheti akár az  $\lfloor n/b \rfloor$  egészrészét, akár az  $\lceil n/b \rceil$  egészrészét. Ekkor:

- ① Ha  $f(n) = O(n^{\log_b a - \epsilon})$ , egy  $\epsilon > 0$  állandóval, akkor

$$\longrightarrow T(n) = \Theta(n^{\log_b a})$$

- ② Ha  $f(n) = \Theta(n^{\log_b a})$ , akkor

$$\longrightarrow T(n) = \Theta(n^{\log_b a} \lg n) (= \Theta(f(n) \lg n))$$

- ③ Ha  $f(n) = \Omega(n^{\log_b a + \epsilon})$  és  $a \cdot f(n/b) \leq c \cdot f(n)$  valamely  $\epsilon > 0$  és  $c < 1$  állandóra és elég nagy  $n$ -re, akkor

$$\longrightarrow T(n) = \Theta(f(n))$$

### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



## Gyorsrendezés

A gyorsrendezés leírása

## A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# A gyorsrendezés hatékonysága

## Kiegyensúlyozott felosztás

Tételezzük fel, hogy a felosztó eljárás minden 9 az 1-hez felosztást ad, amely kiegyensúlyozatlan felosztásnak tűnik.

Ekkor a gyorsrendezés futási idejének rekurzív képlete

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn,$$

ahol expliciten kiírtuk a  $\Theta(n)$ -ben rejlő **c** állandót. A rekurziós fa pedig:



# A gyorsrendezés hatékonysága

## Kiegyensúlyozott felosztás

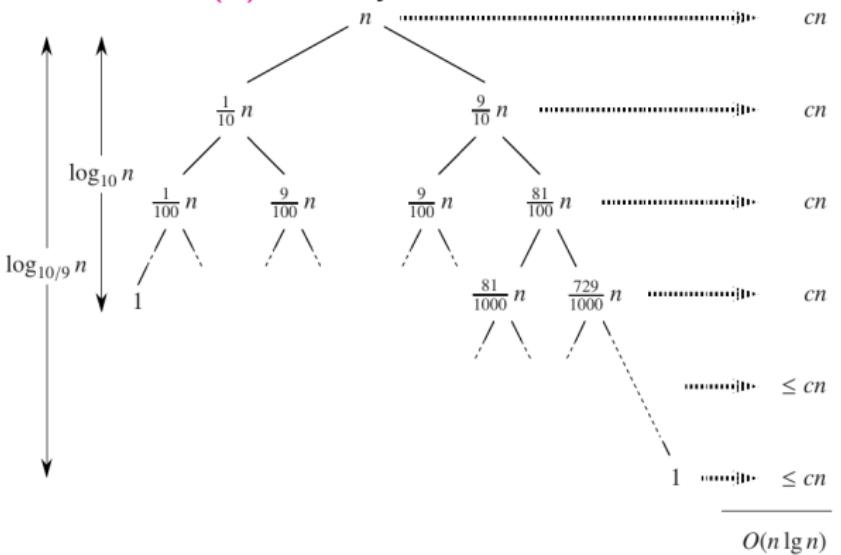
Tételezzük fel, hogy a felosztó eljárás minden 9 az 1-hez felosztást ad, amely kiegyensúlyozatlan felosztásnak tűnik.

Ekkor a gyorsrendezés futási idejének rekurzív képlete

$$T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn,$$

ahol expliciten kiírtuk a  $\Theta(n)$ -ben rejlő  $c$  állandót. A rekurziós fa pedig:

Minden állandó arányú felosztáskor a rekurziós fa mélysége  $\Theta(\lg n)$ , és minden szinten a költség  $O(n)$ .



A futási idő tehát

$O(n \lg n)$ .

## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



### Gyorsrendezés

A gyorsrendezés leírása

### A gyorsrendezés hatékonysága

A gyorsrendezés véletlennel történő változata

### Összehasonlító rendezések

A döntésífa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# A gyorsrendezés hatékonysága

## Az átlagos viselkedés megsejtése

- ▶ Ahhoz, hogy a gyorsrendezés átlagos viselkedését pontosan meghatározhassuk, egy feltevést kell megfogalmaznunk a bemenő tömbök gyakoriságáról.
- ▶ A gyorsrendezés viselkedése nem a bemenő elemektől, hanem azoknak az egymáshoz viszonyított helyétől függ.
- ▶ A legnyilvánvalóbb feltételezés az, hogy a bemeneti elemek minden permutációja ugyanolyan eséllyel fordulhat elő.
- ▶ Amikor a gyorsrendezés véletlen bemenetekre fut, nem valószínű, hogy a felosztás minden szinten egyformán történik.
  - Várható, hogy bizonyos felosztások kiegyensúlyozottak lesznek, mások pedig nem.
  - Például, bizonyítható, hogy a FELOSZT eljárás
    - 80% körül 9:1 aránynál kiegyensúlyozottabb, míg
    - 20% körül legfeljebb ennyire kiegyensúlyozott felosztást ad.
  - Általában a FELOSZT eljárás a „jó” és „rossz” felosztások keverékét adja.
  - Egy rossz és utána egy jó felosztás három résztömböt hoz létre  $\Rightarrow T(n) \leq T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + 2cn = \Theta(n \lg n)$ .

# A gyorsrendezés hatékonysága

Rendező algoritmusok

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Gyorsrendezés

A gyorsrendezés leírása

## A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésifa-modell

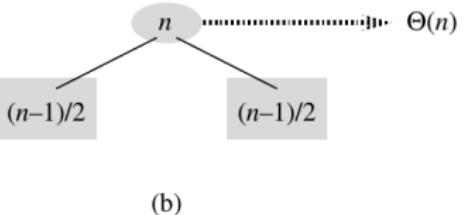
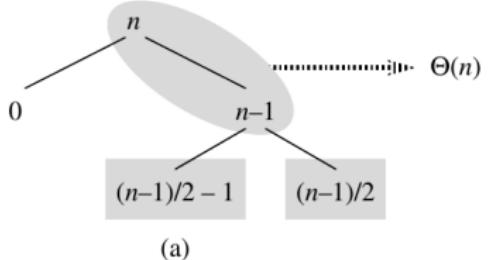
Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésífa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## A gyorsrendezés véletlenített változata

A „véletlenítés” egyik módszere lehetne, ha permutálnánk a bemenetet.

Egy másik változat, a **véletlen mintavétel** nevezetű, egyszerűbb elemzést biztosít. (És gyorsabban kivitelezhető.)

**Őrszemnek** nem mindig az  $A[r]$  elemet tekintjük, hanem helyette az  $A[p \dots r]$  résztömb egy véletlenszerűen kiválasztott elemét. Ezt úgy oldjuk meg egyszerűen, hogy az  $A[r]$  elemet felcseréljük az  $A[p \dots r]$  résztömb egy véletlenszerűen választott elemével. Ez a módosítás biztosítja, hogy az  $x = A[r]$  őrszem ugyanolyan valószínűsséggel lehet az  $A[p \dots r]$  résztömb bármelyik eleme. Mivel az őrszemet véletlenszerűen választjuk ki, az átlagos viselkedésben a felosztás várhatóan jól kiegyensúlyozott lesz.

```
function VÉLETLEN_FELOSZT(A, p, r)
```

- 1:  $i \leftarrow VÉLETLEN(p, r)$
- 2:  $A[r]$  és  $A[i]$  cseréje
- 3: return FELOSZT(A,p,r)

```
end function
```

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésífa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## A gyorsrendezés véletlenített változata

A „véletlenítés” egyik módszere lehetne, ha permutálnánk a bemenetet.

Egy másik változat, a **véletlen mintavétel** nevezetű, egyszerűbb elemzést biztosít. (És gyorsabban kivitelezhető.)

**Őrszemnek** nem mindig az  $A[r]$  elemet tekintjük, hanem helyette az  $A[p \dots r]$  résztömb egy véletlenszerűen kiválasztott elemét. Ezt úgy oldjuk meg egyszerűen, hogy az  $A[r]$  elemet felcseréljük az  $A[p \dots r]$  résztömb egy véletlenszerűen választott elemével. Ez a módosítás biztosítja, hogy az  $x = A[r]$  őrszem ugyanolyan valószínűsséggel lehet az  $A[p \dots r]$  résztömb bármelyik eleme. Mivel az őrszemet véletlenszerűen választjuk ki, az átlagos viselkedésben a felosztás várhatóan jól kiegyensúlyozott lesz.

**function** VÉLETLEN\_FELOSZT(A, p, r)

- 1: **i**  $\leftarrow$  VÉLETLEN(p, r)
- 2:  $A[r]$  és  $A[i]$  cseréje
- 3: **return** FELOSZT(A,p,r)

**end function**

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

**Összehasonlító rendezések**

A döntésífa-modell

Összehasonlító rendezések hatékonysága

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# A „várható” futási idő

## Lemma (Tankönyvben: 7.1. lemma)

Legyen  $X$  a GYORSRENDEZÉS egész futási ideje alatt a FELOSZT 4. sorában végrehajtott összehasonlítások száma, ha  $n$  elemű tömböt rendezünk.

Ekkor a GYORSRENDEZÉS futási ideje  $O(n + X)$ .

## Bizonyítás.

A FELOSZT eljárást legfeljebb  $n$ -szer hívjuk meg, ennek futási ideje egy állandó, amelyhez hozzáadódik a **for** ciklus végrehajtásából adódó idő. A **for** minden iterációjakor végrehajtjuk a 4. sort is. □

Célunk, hogy kiszámítsuk  $X$ -et, az összehasonlítások számát a FELOSZT összes hívásában. Nem fogjuk megszámolni, hogy a FELOSZT egy-egy híváskor hány összehasonlítást végez, hanem inkább egy felső korlátot adunk az összehasonlítások összértékére.



## A „várható” futási idő

### Lemma (Tankönyvben: 7.1. lemma)

Legyen  $X$  a GYORSRENDEZÉS egész futási ideje alatt a FELOSZT 4. sorában végrehajtott összehasonlítások száma, ha  $n$  elemű tömböt rendezünk.

Ekkor a GYORSRENDEZÉS futási ideje  $O(n + X)$ .

### Bizonyítás.

A FELOSZT eljárást legfeljebb  $n$ -szer hívjuk meg, ennek futási ideje egy állandó, amelyhez hozzáadódik a **for** ciklus végrehajtásából adódó idő. A **for** minden iterációjakor végrehajtjuk a 4. sort is. □

Célunk, hogy kiszámítsuk  $X$ -et, az összehasonlítások számát a FELOSZT összes hívásában. Nem fogjuk megszámolni, hogy a FELOSZT egy-egy híváskor hány összehasonlítást végez, hanem inkább egy felső korlátot adunk az összehasonlítások összértékére.

### Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Édényrendezés



## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlennelített változata

## Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## A „várható” futási idő

Az elemzés megkönnyítéséért nevezzük át az A tömb elemeit, legyenek ezek  $z_1, z_2, \dots, z_n$ , ahol  $z_i$  az  $i$ -edik legkisebb elem. Ugyancsak értelmezzük a  $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$  halmazt, amely a  $z_i$  és  $z_j$  közötti elemeket tartalmazza, ezekkel bezárólag. (És tegyük fel, hogy minden elem eltérő értékű.)

Legyen

$$X_{ij} = I \{ z_i \text{ összehasonlítása } z_j\text{-vel} \},$$

$$\left( = \begin{cases} 1 & \text{ha } z_i \text{ és } z_j \text{ össze lesznek hasonlítva valamikor,} \\ 0 & \text{egyébként (I=indikátor)} \end{cases} \right)$$

Mikor hasonlítja össze az algoritmus a  $z_i$  és  $z_j$  elemeket? Hogy megválaszolhassuk a kérdést, vegyük észre, hogy bármely két számpárt legfeljebb egyszer hasonlítjuk össze. Miért?

Az elemeket csak az órszemmel hasonlítjuk össze, és a FELOSZT egy hívása után az abban használt órszemet többé már nem használjuk összehasonlításra.



## Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésífa-modell  
 Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Edényrendezés

## A „várható” futási idő

Az elemzés megkönnyítéséért nevezzük át az A tömb elemeit, legyenek ezek  $z_1, z_2, \dots, z_n$ , ahol  $z_i$  az  $i$ -edik legkisebb elem. Ugyancsak értelmezzük a  $Z_{ij} = \{z_i, z_{i+1}, \dots, z_j\}$  halmazt, amely a  $z_i$  és  $z_j$  közötti elemeket tartalmazza, ezekkel bezárólag. (És tegyük fel, hogy minden elem eltérő értékű.)

Legyen

$$X_{ij} = I \{ z_i \text{ összehasonlítása } z_j\text{-vel} \},$$

$$\left( = \begin{cases} 1 & \text{ha } z_i \text{ és } z_j \text{ össze lesznek hasonlítva valamikor,} \\ 0 & \text{egyébként (I=indikátor)} \end{cases} \right)$$

Mikor hasonlítja össze az algoritmus a  $z_i$  és  $z_j$  elemeket? Hogy megválaszolhassuk a kérdést, vegyük észre, hogy bármely két számpárt legfeljebb egyszer hasonlítjuk össze. Miért?

Az elemeket csak az űrszemmel hasonlítjuk össze, és a FELOSZT egy hívása után az abban használt űrszemet többé már nem használjuk összehasonlításra.



## A „várható” futási idő

Innen az összehasonlítások száma:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

Ennek várható értéke:

$$\begin{aligned} E[X] &= E \left[ \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij} \right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n Pr \{ z_i \text{ összehasonlítása } z_j\text{-vel} \} \end{aligned}$$

Ha az  $x$  őrszemet úgy választjuk meg, hogy  $z_i < x < z_j$ , akkor  $z_i$  és  $z_j$  sem most, sem később nem lesznek összehasonlítva.

Ellenben, ha  $x = z_i$  vagy  $x = z_j$ , akkor most lesznek összehasonlítva.

### Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonyiséga

A gyorsrendezés véletlennitett változata

### Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonyiséga

### Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Edényrendezés



## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésífa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## A „várható” futási idő

Ennek alapján:

$$\begin{aligned}
 \Pr \{ Z_i \text{ összehasonlítása } Z_j\text{-vel} \} &= \Pr \{ Z_i \text{ vagy } Z_j \text{ az első őrszem } Z_{ij}\text{-ből} \} \\
 &= \Pr \{ Z_i \text{ az első őrszem } Z_{ij}\text{-ből} \} \\
 &\quad + \Pr \{ Z_j \text{ az első őrszem } Z_{ij}\text{-ből} \} \\
 &= \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}.
 \end{aligned}$$

Innen:

$$\begin{aligned}
 E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\
 &= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k} \\
 &= \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)
 \end{aligned}$$

$$\left( \sum_{k=1}^n \frac{1}{k} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i + j} \leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i - 1} \frac{1}{2^i} = \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \leq \lg n + 1 \right)$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Gyorsrendezés

- A gyorsrendezés leírása
- A gyorsrendezés hatékonysága
- A gyorsrendezés véletlennelített változata

## Összehasonlító rendezések

- A döntésifa-modell
- Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

- Leszámláló rendezés
- Számjegyes rendezés
- Edényrendezés

## Összehasonlító rendezések

Az eddig tárgyalt rendező algoritmusoknak van egy érdekes, közös tulajdonságuk:

- ▶ a rendezéshez csak a bemeneti elemek összehasonlítását használják.  
(Az  $a_i$  és  $a_j$  elemek esetén az  $a_i < a_j$ ,  $a_i \leq a_j$ ,  $a_i = a_j$ ,  $a_i \geq a_j$  vagy  $a_i > a_j$  egyikét végezzük el, hogy megtudjuk a két elem egymáshoz viszonyított sorrendjét.)

Éppen ezért ezeket az algoritmusokat **összehasonlító rendezéseknek** nevezzük.

A következőkben megmutatjuk, hogy bármely összehasonlító algoritmusnak a legrosszabb esetben  $\Omega(n \lg n)$  összehasonlításra van szüksége  $n$  elem rendezéséhez.

- ▶ Következésképpen az összefésüléses rendezés és a kupacrendezés aszimptotikusan optimális, és
- ▶ minden összehasonlító rendezés legfeljebb egy állandó szorzóval lehet gyorsabb.

(Továbbiakban feltételezzük, hogy minden elem eltérő értékű:  $a_i \neq a_j$ , így csak  $a_i \leq a_j$ -t fogunk használnunk.)



## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlennelített változata

## Összehasonlító rendezések

## A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

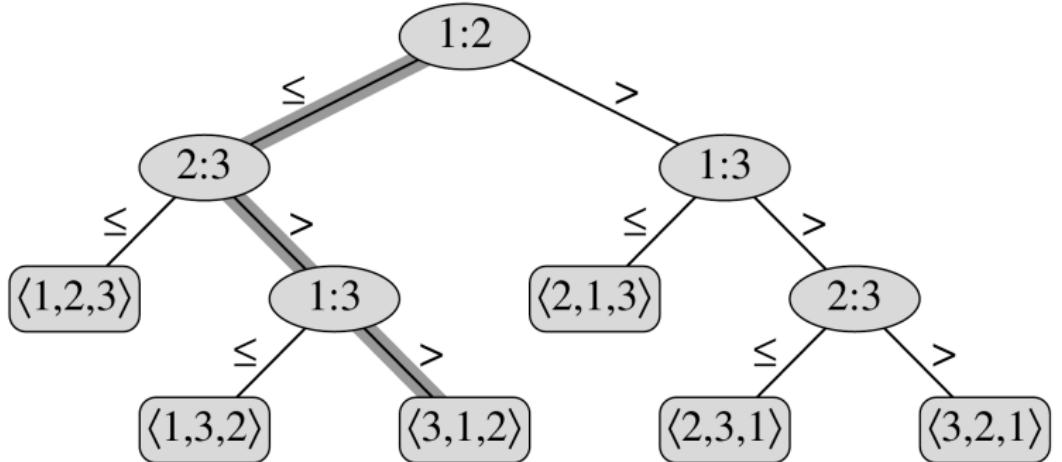
Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

## A döntésifa-modell

Az összehasonlító rendezéseket tekinthetjük **döntési fáknak**:



A döntési fában:

- ▶ minden belső csúcsot egy  $i:j$  számpárral jelölünk, ahol  $1 \leq i, j \leq n$ ,  $n$  pedig a bemenet elemeinek a száma.
- ▶ minden levél egy  $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$  permutációval jelölhető.

A rendezési algoritmus végrehajtása megfelel a döntési fában a gyökértől valamely levélig vezető út bejárásának.



## Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlennelített változata

## Összehasonlító rendezések

## A döntésifa-modell

Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

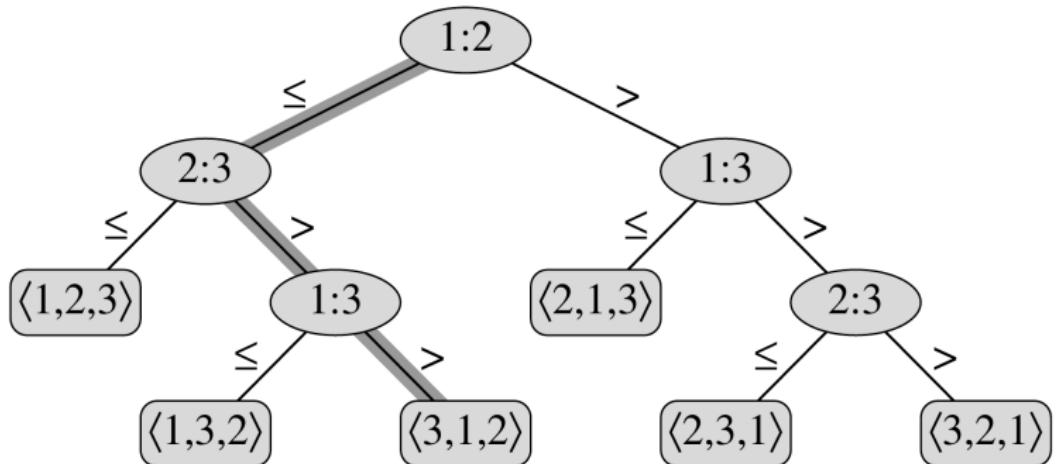
Edényrendezés

## A döntésifa-modell

Mivel minden helyes rendezési algoritmusnak elő kell tudni állítania a bemeneti elemek összes permutációját, ezért a helyes rendezés szükséges feltétele, hogy

- ▶ az  $n$  elem összes  $n!$  permutációjának meg kell jelennie a döntési fa levelei között,
- ▶ és ezen levelek mindegyikének elérhetőnek kell lennie a gyökérből egy összehasonlító rendezéshez tartozó úttal.

Vagyis csak olyan döntési fákat tekintünk, ahol minden permutáció megjelenik egy elérhető levélként.



# Összehasonlító rendezések

## Theorem (Tankönyv: 8.1. téTEL.)

Bármely összehasonlító rendezőalgoritmus a legrosszabb esetben  $\Omega(n \lg n)$  összehasonlítást végez.

### Bizonyítás.

Vegyük egy  $n$  elemet rendező döntési fát, amelynek magassága  $h$ , elérhető leveleinek száma pedig  $l$ . Mivel a bemenet összes  $n!$  permutációja meg kell jelenjen levélként, ezért  $n! \leq l$ . Mivel egy  $h$  mélységű bináris fa leveleinek a száma nem lehet nagyobb, mint  $2^h$ , ezért

$$n! \leq l \leq 2^h,$$

ahonnan minden oldal logaritmusát véve

$$\begin{aligned} h &\geq \lg(n!) && \text{(a logaritmusfüggvény monoton növekvő)} \\ &= \Omega(n \lg n) && \text{(Stirling-formula alapján).} \end{aligned}$$

Kós Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



### Gyorsrendezés

- A gyorsrendezés leírása
- A gyorsrendezés hatékonyiséga
- A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

- A döntésifa-modell
- Összehasonlító rendezések hatékonyiséga

### Rendezés lineáris időben

- Leszámláló rendezés
- Számjegyes rendezés
- Edényrendezés



Kós Márk

Pánovics János

Szathmáry László

Halász Gábor

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonysága

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# Rendezés lineáris időben

A következőkben tárgyalandó algoritmusok nem az összehasonlítást használják a rendezéshez,

- ▶ így az  $\Omega(n \lg n)$  alsó korlát rájuk nem vonatkozik.



## Leszámláló rendezés

A leszámláló rendezésnél **feltételezzük**, hogy az  $n$  bemeneti elem mindegyike  $0$  és  $k$  közötti egész szám, ahol  $k$  egy ismert egész.

- ▶ Ha  $k = O(n)$ , a rendezés futási ideje  $\Theta(n)$ .

A leszámláló rendezés alapötlete az, hogy minden egyes  $x$  bemeneti elemre meghatározza azoknak az elemeknek a számát, amelyek kisebbek, mint az  $x$ .

Ezzel az információval az  $x$  elemet közvetlenül a saját pozíójába tudjuk helyezni a kimeneti tömbben. Ha például  $17$  olyan elem van, amelyik kisebb, mint az  $x$ , akkor az  $x$  elem a  $18$ . helyen lesz a kimeneti tömbben.

Ez a séma valamelyest megváltozik abban az esetben, amikor néhány elem egyenlő, hiszen nem akarjuk minden ugyanabba a pozícióba helyezni.

### Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonyiséga  
 A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonyiséga

### Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Edényrendezés



## Leszámláló rendezés

-- A[1..n], B[1..n], C[0..k]

**procedure** LESZÁMLÁLÓ\_RENDEZÉS(A, B, k)

```

1: for i  $\leftarrow$  0 to k do
2:   C[i]  $\leftarrow$  0
3: end for
4: for j  $\leftarrow$  1 to méret(A) do
5:   C[A[j]]  $\leftarrow$  C[A[j]]+1
6: end for
7: for i  $\leftarrow$  1 to k do
8:   C[i]  $\leftarrow$  C[i]+C[i-1]
9: end for
10: for j  $\leftarrow$  méret(A) downto 1 do
11:   B[C[A[j]]]  $\leftarrow$  A[j]
12:   C[A[j]]  $\leftarrow$  C[A[j]]-1
13: end for
end procedure
```

Könnyű belátni,  
hogy:

$$T(n) = \Theta(n+k)$$

### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



## Leszámláló rendezés

-- A[1..n], B[1..n], C[0..k]

**procedure** LESZÁMLÁLÓ\_RENDEZÉS(A, B, k)

```

1: for i  $\leftarrow$  0 to k do
2:   C[i]  $\leftarrow$  0
3: end for
4: for j  $\leftarrow$  1 to méret(A) do
5:   C[A[j]]  $\leftarrow$  C[A[j]]+1
6: end for
7: for i  $\leftarrow$  1 to k do
8:   C[i]  $\leftarrow$  C[i]+C[i-1]
9: end for
10: for j  $\leftarrow$  méret(A) downto 1 do
11:   B[C[A[j]]]  $\leftarrow$  A[j]
12:   C[A[j]]  $\leftarrow$  C[A[j]]-1
13: end for
end procedure
```

Könnyű belátni,  
hogy:

$$T(n) = \Theta(n+k)$$

### Gyorsrendezés

A gyorsrendezés leírása

A gyorsrendezés hatékonysága

A gyorsrendezés véletlenített változata

### Összehasonlító rendezések

A döntésifa-modell

Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés



### Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonysága  
 A gyorsrendezés véletlennel történő változata

### Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonysága

### Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés  
 Edényrendezés

## Leszámláló rendezés

A leszámláló rendezés egy fontos tulajdonsága az, hogy **stabil**: az azonos értékű elemek ugyanabban a sorrendben jelennek meg a kimeneti tömbben, mint ahogyan a bemeneti tömbben szerepeltek.

Azaz két azonos értékű szám között a kapcsolat megmarad azon szabály szerint, hogy amelyikük először jelenik meg a bemeneti tömbben, az jelenik meg először a kimeneti tömbben is.

- ▶ Általában a stabilitás csak akkor fontos tulajdonság, amikor a rendezendő elemek mellett kísérő adatok is vannak.
- ▶ A leszámláló rendezés stabilitása egy másik szempontból is fontos:
  - a leszámláló rendezést gyakran felhasználjuk a számjegyes rendezés eljárásaként.
  - Ahogyan ez a következő alfejezetben kiderül, a leszámláló rendezés stabilitása életbevágó a számjegyes rendezés helyes működéséhez.

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonyiséga

A gyorsrendezés véletlennitett változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonyiséga

**Rendezés lineáris időben**

Leszámláló rendezés

**Számjegyes rendezés**

Edényrendezés

# Számjegyes rendezés

A **számjegyes rendezés** – másnéven: **radix rendezés** – kódja értelemszerű.

A következő eljárás feltételezi, hogy az  $n$  elemű  $A$  tömb minden egyes eleme  $d$  jegyű, ahol az első számjegy a legalacsonyabb helyértékű számjegy és a  $d$ -edik számjegy a legmagasabb helyértékű számjegy.

**procedure** SZÁMJEGYES\_RENDEZÉS( $A$ ,  $d$ )

1: **for**  $i \leftarrow 1$  **to**  $d$  **do**

2:   stabil algoritmussal rendezzük az  $A$  tömböt az  $i$ -edik számjegy szerint.

3: **end for**

**end procedure**



# Számjegyes rendezés

## Lemma (Tankönyv: 8.3. lemma.)

Legyen adott  $n$  darab  $d$  jegyből álló szám, ahol a számjegyek legfeljebb  $k$  értéket vehetnek fel. Ekkor a SzÁMJEYES-RENDEZÉS  $\Theta(d \cdot (n + k))$  időben rendez helyesen ezeket a számokat.

## Bizonyítás.

A számjegyes rendezés helyessége a rendezendő oszlopon végzett indukcióból következik. Ha minden számjegy  $0$  és  $k - 1$  között van (azaz  $k$  lehetséges értéket vehet fel), és  $k$  nem túl nagy, a leszámláló rendezés a kézenfekvő választás a használandó stabil algoritmusra. Így  $n$  darab  $d$  számjegyű szám esetén egy lépés  $\Theta(n + k)$  időt igényel. Mivel  $d$  lépésünk van, a számjegyes rendezés teljes időigénye  $\Theta(d \cdot (n + k))$ .  $\square$

Ha  $d$  állandó és  $k = O(n)$ , a számjegyes rendezés lineáris idejű. Általánosabb esetben, a kulcsokat bizonyos rugalmassággal bonthatjuk számjegyekre.

## Gyorsrendezés

- [A gyorsrendezés leírása](#)
- [A gyorsrendezés hatékonysága](#)
- [A gyorsrendezés véletlenített változata](#)

## Összehasonlító rendezések

- [A döntésifa-modell](#)
- [Összehasonlító rendezések hatékonysága](#)

## Rendezés lineáris időben

- [Leszámláló rendezés](#)
- [Számjegyes rendezés](#)

[Edényrendezés](#)

## Lemma (Tankönyvben: 8.4. lemma.)

Legyen adott  $n$  darab  $b$  bites szám és egy tetszőleges  $1 \leq r \leq b$  pozitív egész. Ekkor a SzÁMJEGYES-RENDEZÉS  $\Theta((b/r) \cdot (n + 2^r))$  időben rendez helyesen ezeket a számokat.

## Bizonyítás.

Egy  $r \leq b$  értékre minden kulcs  $d = \lceil b/r \rceil$  darab egyenként  $r$  bites számjegyként tekinthető. minden számjegy egy 0 és  $2^r - 1$  közötti egész, ezért alkalmazhatjuk a leszámláló rendezést a  $k = 2^r - 1$  paraméterrel.

A leszámláló rendezés minden lépése  $\Theta(n + k) = \Theta(n + 2^r)$  ideig tart, így a  $d$  lépés megtételéhez szükséges teljes idő  $\Theta(d \cdot (n + 2^r)) = \Theta((b/r) \cdot (n + 2^r))$ .  $\square$

(Egy 32 bites szót tekinthetünk például 4 darab 8 bites számjegynek, azaz  $b = 32$ ,  $r = 8$ ,  $k = 2^r - 1 = 255$  és  $d = b/r = 4$  és így  $(b/r) \cdot (n + 2^r) = 4 \cdot (n + 256) = 4n + 1024$ .)

Kós Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Gyorsrendezés

A gyorsrendezés leírása  
A gyorsrendezés hatékonysága  
A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésifa-modell  
Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonyiséga

A gyorsrendezés véletlennitett változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonyiséga

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

Edényrendezés

# Edényrendezés

Az **edényrendezés** – szokták **vödör rendezésnek** is hívni – várható futási ideje lineáris, amennyiben a bemenet egyenletes eloszlásból származik.

Éppúgy, mint a leszámláló rendezés, az edényrendezés is azért lehet gyors, mert feltesz valamit a bemenetről.

- ▶ Míg a leszámláló rendezés azt feltételezi, hogy a bemenet olyan egészekből áll, amelyek egy kis intervallumba tartoznak,
- ▶ addig az edényrendezés azt, hogy a bemenetet egy olyan véletlen folyamat generálja, amelyik egyenletesen osztja el az elemeket a  $[0, 1)$  intervallumon.

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonyiséga

A gyorsrendezés véletlennitett változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonyiséga

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

**Edényrendezés**

## Edényrendezés

Az edényrendezés alábbi kódja feltételezi, hogy a bemenet egy  $n$  elemű  $A$  tömb, és a tömb minden egyes  $A[i]$  elemére teljesül, hogy  $0 \leq A[i] < 1$ . A kódban szükség van továbbá egy, a láncolt listák (edények – vödrök)  $B[0 \dots n-1]$  segédtömbjére, és feltezzük, hogy az ilyen listák kezeléséhez szükséges eljárások is rendelkezésünkre állnak.

**procedure** EDÉNY\_RENDEZÉS( $A$ )

```

1:  $n \leftarrow$  méret( $A$ )
2: for  $i \leftarrow 1$  to  $n$  do
3:   szúrjuk be az  $A[i]$  elemet a  $B[\lfloor nA[i] \rfloor]$  listába.
4: end for
5: for  $i \leftarrow 0$  to  $n-1$  do
6:   rendezzük a  $B[i]$  listát beszúrásos rendezéssel.
7: end for
8: sorban összefűzzük a  $B[0], B[1], \dots, B[n - 1]$  listákat.
end procedure
```



## Gyorsrendezés

A gyorsrendezés leírása  
 A gyorsrendezés hatékonysága  
 A gyorsrendezés véletlennitett változata

## Összehasonlító rendezések

A döntésifa-modell  
 Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés  
 Számjegyes rendezés

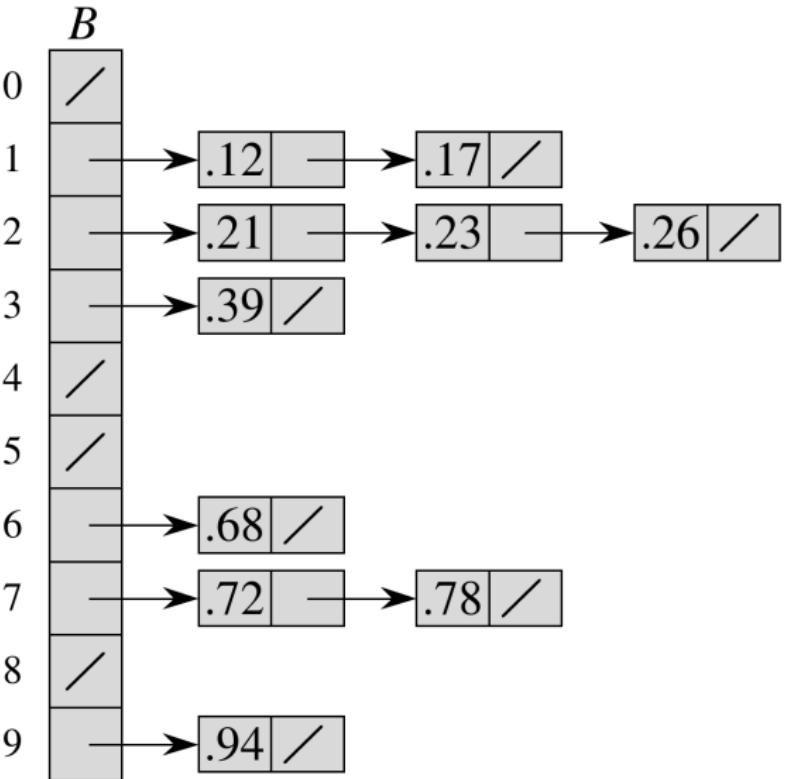
## Edényrendezés

## Edényrendezés

## Egy példa

	A
1	.78
2	.17
3	.39
4	.26
5	.72
6	.94
7	.21
8	.12
9	.23
10	.68

(a)



(b)

**Gyorsrendezés**

A gyorsrendezés leírása

A gyorsrendezés hatékonyiséga

A gyorsrendezés véletlennel tett változata

**Összehasonlító rendezések**

A döntésifa-modell

Összehasonlító rendezések hatékonyiséga

**Rendezés lineáris időben**

Leszámláló rendezés

Számjegyes rendezés

**Edényrendezés**

# Edényrendezés

## Helyesség

Ahhoz, hogy lássuk az algoritmus helyes működését, tekintsünk két elemet,  $A[i]$ -t és  $A[j]$ -t.

- ▶ Tegyük fel az általánosság megszorítása nélkül, hogy  $A[i] \leq A[j]$ .
- ▶ Mivel ilyenkor  $\lfloor nA[i] \rfloor \leq \lfloor nA[j] \rfloor$  is teljesül, az  $A[i]$  elem
  - vagy ugyanabba az edénybe kerül, mint az  $A[j]$ ,
  - vagy egy kisebb indexű edénybe.
- ▶ Ha  $A[i]$  és  $A[j]$  ugyanabba az edénybe került, akkor a 4–5. sorban található **for** ciklus a helyes sorrendbe állítja őket.
- ▶ Ha  $A[i]$  és  $A[j]$  különböző edényekbe kerültek, akkor a 6. sor állítja a helyes sorrendbe őket.

Tehát az edényrendezés helyesen működik.



# Edényrendezés

## Hatókonyság

A futási időt vizsgálva észrevehetjük, hogy az 5. sort kivéve az összes sor időigénye legrosszabb esetben  $O(n)$ . Marad még az 5. sorban levő  $n_i$  elem beszúrásos rendezése időigényének az elemzése

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)$$

Ennek várható értéke:

$$\begin{aligned} E[T(n)] &= E\left[\Theta(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= \Theta(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \quad (\text{a várható érték linearitása miatt}) \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(E[n_i^2]) \quad (\text{nem bizonyított téTEL alapján}) \\ &= \Theta(n) + \sum_{i=0}^{n-1} O(2 - \frac{1}{n}) \quad (\text{lásd a tankönyv 8.4 fejezetben}) \end{aligned}$$

## Gyorsrendezés

- A gyorsrendezés leírása
- A gyorsrendezés hatékonysága
- A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

- A döntésífa-modell
- Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

- Leszámláló rendezés
- Számjegyes rendezés

## Edényrendezés

**Gyorsrendezés**

A gyorsrendezés leírása  
 A gyorsrendezés hatékonysága  
 A gyorsrendezés véletlennitett változata

**Összehasonlító rendezések**

A döntésifa-modell  
 Összehasonlító rendezések hatékonysága

**Rendezés lineáris időben**

Leszámláló rendezés  
 Számjegyes rendezés

**Edényrendezés**

# Edényrendezés

## Hatókonyáság

Még ha a bemenet nem is egyenletes eloszlásból származik, az edényrendezés futhat lineáris időben.

Amíg a bemenet rendelkezik azzal a tulajdonsággal, hogy

- ▶ az edényméretek négyzeteinek összege lineáris a teljes elemszámban,

addig az edényrendezés futási ideje lineáris lesz.

# Most akkor melyik rendezést használjuk?

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Gyorsrendezés

A gyorsrendezés leírása  
A gyorsrendezés hatékonysága  
A gyorsrendezés véletlenített változata

## Összehasonlító rendezések

A döntésifa-modell  
Összehasonlító rendezések hatékonysága

## Rendezés lineáris időben

Leszámláló rendezés  
Számjegyes rendezés

## Edényrendezés



# 8. előadás

## Sztring

Sztringkereső algoritmusok.

*Adatszerkezetek és algoritmusok előadás*

2018. április 17.

A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt-  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar



A sztring  
 adatszerkezet

A meztílabas (brute  
 force) algoritmus

A Knuth–Morris–Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus

A kereső algoritmus

A Shift-And  
 (Dömölkki-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

# Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
 Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
 1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
 3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
 Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
 Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>



## A sztring adatszerkezet

A meztáblas (brute force) algoritmus

A Knuth–Morris–Pratt-algoritmus

A előfeldolgozó algoritmus

A kereső algoritmus

A Shift-And (Dömölk-féle) algoritmus

A Shift-Or algoritmus

Rabin-Karp mintaillesztés

Reguláris kifejezések

# A sztring adatszerkezet

A sztring olyan szekvenciális lista, amelynek az elemei egy ábécé szimbólumai. Ezeket a szimbólumokat **karaktereknek** nevezük.

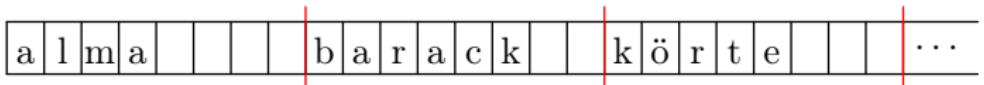
## Sztringgel végezhető műveletek

- ▶ **Létrehozás:** explicit módon felsoroljuk a sztring összes karakterét.
- ▶ **Bővítés:** bárhol bővíthető. Bővítéskor két részsztringet képzünk, majd koncatenáljuk azokat a beszúrandó sztringgel.
- ▶ **Törlés:** megvalósítható a fizikai törlés, melynek során két részsztringet képzünk (melyekben már nem szerepel a törlendő részsztring), majd koncatenáljuk azokat.
- ▶ **Csere:** cserélhetünk egy karaktert, de részsztring is cserélhető másik részsztringre. Két részsztringet képzünk, majd koncatenáljuk azokat az új értéket képviselő sztringgel (törles+bővítés).
- ▶ **Rendezés:** nem értelmezett.
- ▶ **Keresés:** értelmezhető, kereshetünk egy karaktert vagy egy részsztringet.
- ▶ **Elérés:** soros vagy közvetlen.
- ▶ **Bejárás:** értelmezhető.



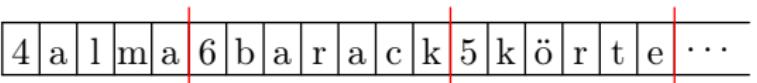
# A sztring adatszerkezet folytonos reprezentációi

Fix hosszon:

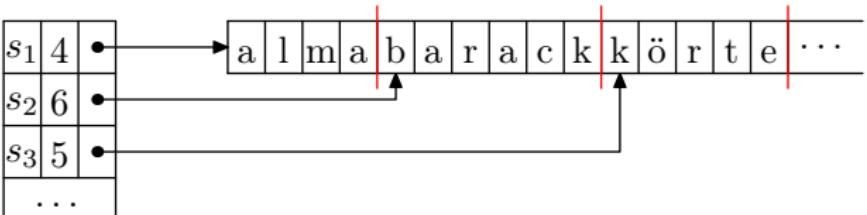


Változó hosszon:

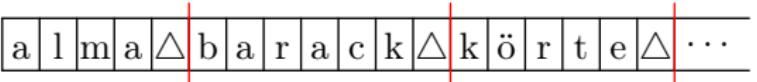
- hossz a sztring előtt:



- információs táblázattal:



- végjellel:



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A sztring adatszerkezet szétszort reprezentációja

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## A listaelemek tartalmazhatnak

- ▶ egy karaktert  
(rossz helykihasználás)
- ▶ vagy egy részsztringet.  
Ebben esetben a részsztringek eltérő hosszúságúak lehetnek, és valamelyik folytonos reprezentációval ábrázoljuk őket.

## A sztring adatszerkezet

A meztílabas (brute force) algoritmus

A Knuth–Morris–Pratt-algoritmus

A előfeldolgozó algoritmus

A kereső algoritmus

A Shift-And (Dömököi-féle) algoritmus

A Shift-Or algoritmus

Rabin-Karp mintaillesztés

Reguláris kifejezések

# Sztringkereső (mintaillesztő) algoritmusok

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Egy sztringben keresünk egy másik sztringet. Azt a sztringet, amelyikben keresünk, **alapsztringnek**, azt a sztringet pedig, amit keresünk, **mintasztringnek** nevezzük. A pszeudokódokban az alapsztringet  $A$ -val, a mintasztringet  $P$ -vel fogjuk jelölni.

Néhány sztringkereső algoritmus:

- ▶ mezítlábas (brute force) algoritmus
- ▶ Knuth–Morris–Pratt-algoritmus
- ▶ Boyer–Moore-algoritmus
- ▶ Rabin–Karp-algoritmus
- ▶ Shift-And (Dömölki Bálint-féle) és Shift-Or algoritmus

A sztring  
adatszerkezet

A meztlábas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



A sztring  
 adatszerkezet

A mezítlábas (brute force)  
 algoritmus

A Knuth–Morris–Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus  
 A kereső algoritmus

A Shift-And  
 (Dömölkki-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

## A mezítlábas (brute force) algoritmus

**function** MEZÍTLÁBAS\_KERES(A, P)

```

1: n ← HOSSZ(A)
2: m ← HOSSZ(P)
3: for i ← 1 to n-m+1 do
4:     j ← 0
5:     while j < m és A[i + j] = P[1 + j] do
6:         j ← j + 1
7:     end while
8:     if j = m then
9:         return i
10:    end if
11: end for
12: return 0
end function
```



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus  
A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A Knuth–Morris–Pratt-algoritmus

## Prefix, szuffix

Legyen  $\Sigma$  egy ábécé és  $x = x_1 \dots x_k$  ( $k \in \mathbb{N}$ ) egy  $k$  hosszúságú sztring  $\Sigma$  felett ( $x \in \Sigma^*$ )!

Az  $x$ -nek az  $u \in \Sigma^*$  részsztring egy **prefixe**, ha

$$u = x_1 \dots x_b, \text{ ahol } 0 \leq b \leq k,$$

azaz ha  $x$   $u$ -val kezdődik. Az  $x$ -nek az  $u$  részsztring egy **szuffixe**, ha

$$u = x_{k-b+1} \dots x_k, \text{ ahol } 0 \leq b \leq k,$$

azaz ha  $x$   $u$ -val végződik.

## Valódi prefix, valódi szuffix

Az  $x$  egy  $u$  prefixét vagy  $x$  egy  $u$  szuffixét **valódi** prefixnek vagy szuffixnek nevezzük, ha  $u \neq x$ , azaz ha  $b < k$ .

**Ha  $k = 0$ , akkor  $x = \varepsilon$ , ill. ha  $b = 0$ , akkor  $u = \varepsilon$  ( $\varepsilon \equiv$  üres sztring).**



## Border

Legyen  $\Sigma$  egy ábécé és  $x = x_1 \dots x_k$  ( $k \in \mathbb{N}$ ) egy  $k$  hosszúságú sztring  $\Sigma$  felett!

Az  $x$ -nek az  $r$  részsztring egy **bordere**, ha

$$r = x_1 \dots x_b \text{ és } r = x_{k-b+1} \dots x_k, \text{ ahol } 0 \leq b < k.$$

Az  $x$  bordere egy olyan részsztring, amely valódi prefixe és valódi szuffixe is  $x$ -nek. Ekkor a részsztring  $b$  hosszát a border **hosszának** nevezük. Ha  $b = 0$ , akkor  $r = \varepsilon$  (üres sztring).

A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkai-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## Példa

Legyen  $x = abacab$ .

- Az  $x$  valódi prefixei:

$\varepsilon, a, ab, aba, abac, abaca$ .

- Az  $x$  valódi szuffixei:

$\varepsilon, b, ab, cab, acab, bacab$ .

- Az  $x$  borderei:

$\varepsilon, ab$ .

Az  $\varepsilon$  border hossza 0, az  $ab$  border hossza 2.

## Megjegyzés

- Az  $\varepsilon$  üres sztring minden  $x \in \Sigma^+$  sztringnek bordere.
- Az  $\varepsilon$  üres sztringnek nincs bordere.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



## Példa

1	2	3	4	5	6	7	8	9	...
a	b	c	a	b	c	a	b	d	
a	b	c	a	b	<b>d</b>				
			a	b	c	a	b	d	

Az  $1, \dots, 5$  pozíciókon lévő karakterek megegyeznek. A 6. pozíción a  $c$  és  $d$  karakterek eltérnek. A minta 3 pozícióval tovább léptethető, és az összehasonlítások a 6. pozíciótól folytathatók.

A léptetés mértékét a  $p$  egyező prefixének a legszélesebb bordere határozza meg. Ebben a példában az egyező prefix  $abca$ , a hossza  $j = 5$ . Az ő legszélesebb bordere  $ab$ , amely  $b = 2$  hosszúságú. A léptetés mértéke  $j - b = 5 - 2 = 3$ .

A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkiféle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

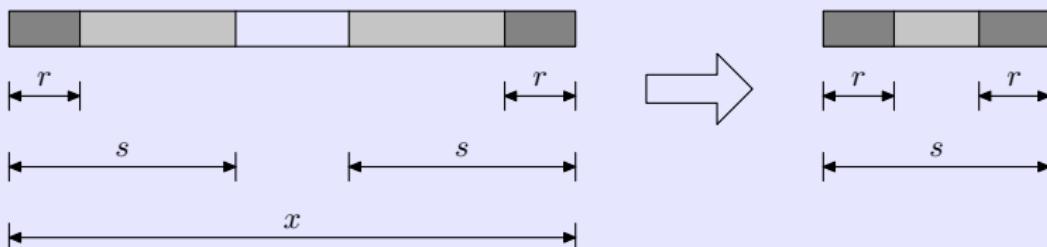
Reguláris kifejezések

Az előfeldolgozási szakaszban a minta minden egyes prefixéhez meg kell határozni a legszélesebb border hosszát. Később a keresési szakaszban a léptetés mértéke az egyező prefixeknek megfelelően számítható.

## Theorem

Legyen  $r$  és  $s$  egy  $x$  sztring bordere, ahol  $|r| < |s|$ . Ekkor  $r$  egy bordere  $s$ -nek.

## Bizonyítás.



Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

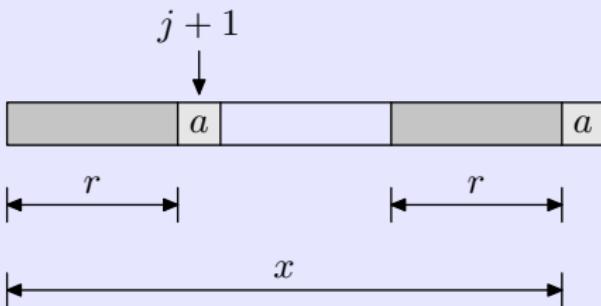
## Megjegyzés

Ha  $s$  a legszélesebb bordere  $x$ -nek, akkor  $x$  következő legszélesebb  $r$  borderét megkapjuk  $s$  legszélesebb bordereként, és így tovább...

## Definition

Legyen  $x \in \Sigma^*$  a  $P$  minta sztring egy valódi prefixe és  $a \in \Sigma$  egy karakter.

Az  $x$  egy  $r$  bordere **bővíthető**  $a$ -val, ha  $ra$  egy bordere  $xa$ -nak.



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

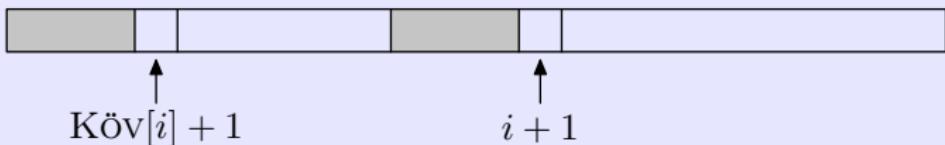
Rabin-Karp  
mintaillesztés

Reguláris kifejezések



## A Köv tömb

Az előfeldolgozási szakaszban egy  $m + 1$  elemű Köv tömböt számítunk ki. A tömb  $\text{Köv}[i]$  eleme a mintasztring  $i$  hosszúságú prefixéhez tartozó legszélesebb border hossza ( $i = 0, \dots, m$ ). Mivel az  $i = 0$  hosszúságú  $\varepsilon$  sztringnek nincsen bordere, ezért  $\text{Köv}[0] = -1$ .



Feltéve, hogy a  $\text{Köv}[0], \dots, \text{Köv}[i]$  értékeket már ismerjük, a  $\text{Köv}[i + 1]$  értékét kiszámíthatjuk, ha ellenőrizzük, hogy a  $p_1 \dots p_i$  prefix egy bordere bővíthető-e a  $p_{i+1}$  karakterrel.

Ez abban az esetben tehető meg, ha  $p_{\text{Köv}[i]+1} = p_{i+1}$ . A bordereket a  $\text{Köv}[i]$ ,  $\text{Köv}[\text{Köv}[i]]$ , ... értékek csökkenő sorrendjében kell megvizsgálni.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# Az előfeldolgozó algoritmus

**function** Köv\_FELTÖLT(P)

```

1: m ← HOSSZ(P)
2: i ← 0
3: j ← KÖV[0] ← − 1
4: while i < m do
5:   if j = − 1 vagy P[i + 1] = P[j + 1] then
6:     i ← i + 1
7:     j ← j + 1
8:     KÖV[i] ← j
9:   else
10:    j ← KÖV[j]
11:   end if
12: end while
13: return KÖV
end function
```



## Példa

A  $p = ababaa$  minta esetén a borderek szélességei a következő értékeket veszik fel a Köv tömbben:

$i :$	0	1	2	3	4	5	6
$p[i] :$	$a$	$b$	$a$	$b$	$a$	$a$	
$\text{Köv}[i] :$	-1	0	0	1	2	3	1

Láthatjuk például, hogy  $\text{Köv}[5] = 3$ , mivel az 5 hosszúságú  $ababa$  prefixnek van egy 3 hosszúságú bordere.

A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

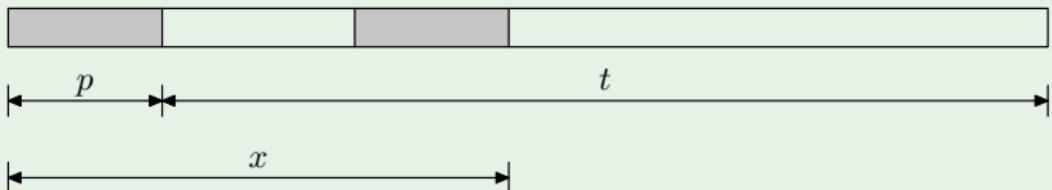
Rabin-Karp  
mintaillesztés

Reguláris kifejezések



## Megjegyzés

Elméletileg semmi akadálya annak, hogy az előző előfeldolgozó algoritmust a  $pt$  sztringre alkalmazzuk a  $p$  helyett. Ha a bordereket csak a  $p$  minta  $m$  szélességéig számoljuk ki, akkor a  $pt$  valamely  $x$  prefixének egy  $m$  szélességű bordere megfelel a minta egy előfordulásának  $t$ -ben (feltéve, hogy a border nem önátfedő).



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## A kereső algoritmus

**function** KMP\_KERESÉS(A, P)

1: n  $\leftarrow$  HOSSZ(A)

2: m  $\leftarrow$  HOSSZ(P)

3: KÖV  $\leftarrow$  Köv\_FELTÖLT(P)

4: i  $\leftarrow$  j  $\leftarrow$  0

5: **while** i < n és j < m **do**

6:     **if** j = -1 vagy A[i + 1] = P[j + 1] **then**

7:         i  $\leftarrow$  i + 1

8:         j  $\leftarrow$  j + 1

9:     **else**

10:         j  $\leftarrow$  KÖV[j]

11:     **end if**

12: **end while**

13: **if** j = m **then**

14:     **return** i - m + 1

15: **else**

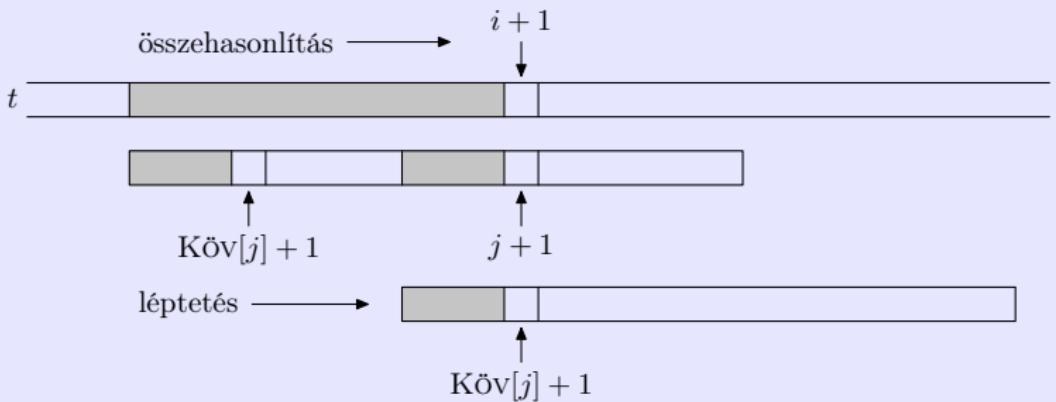
16:     **return** 0

17: **end if**

**end function**



## A minta léptetése



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A Shift-And (Dömölki-féle) algoritmus

## Az ötlet

Legyen a  $p$  mintasztring hossza  $m!$  Vegyünk egy  $m$  elemű  $D$  vektort, amelynek  $j$ -edik eleme akkor és csak akkor 1, ha  $p_1 \dots p_j$  szuffixe  $a_1 \dots a_i$ -nek, egyébként 0! ( $i$ -vel az alapsztring aktuálisan vizsgált karakterének az indexét jelöljük.)

## Megjegyzés

Ha  $p$  mérete kisebb, mint egy gépi szó hossza, akkor ez a vektor a számítógép egy regiszterében is tárolható, így gyorsítható a majdani keresés.

Amikor az alapsztring következő,  $(i + 1)$ -edik karakterét olvassuk, meg kell határoznunk egy új  $D'$  vektort.

Ehhez a következő megfigyelést használjuk fel: a  $D'$  vektor  $(j + 1)$ -edik elemének értéke akkor és csak akkor lesz 1, ha

- ① a  $j$ -edik eleme 1 volt  $D$ -nek, azaz  $p_1 \dots p_j$  szuffixe volt  $a_1 \dots a_i$ -nek, és
- ②  $a_{i+1}$  megegyezik  $p_{j+1}$ -gyel.



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus  
A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A Shift-And (Dömölki-féle) algoritmus

## A megvalósítás

Az algoritmus az előfeldolgozás során felépít egy  $B$  bitmátrixot, amelynek az oszlopait a  $p$  mintasztring karaktereivel (illetve ezen karakterek indexeivel), a sorait pedig az ábécé (egymástól különböző) karaktereivel címkézi. Egy  $c$  karakterhez tartozó sorban

$$B_{c,j} = \begin{cases} 1, & \text{ha } p_j = c, \\ 0 & \text{egyébként.} \end{cases}$$

A kereséshez az algoritmus három  $m$  elemű segédvektort ( $D$ ,  $U$  és  $V$ ) használ, melyeket a következőképpen definiálunk:

$$D_j = 0 \quad 1 \leq j \leq m \text{ esetén,}$$

$$U_j = \begin{cases} 1, & \text{ha } j = 1, \\ 0 & \text{egyébként,} \end{cases}$$

$$V_j = \begin{cases} 1, & \text{ha } j = m, \\ 0 & \text{egyébként.} \end{cases}$$



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A Shift-And (Dömölki-féle) algoritmus

## A SHIFT művelet

Legyen  $X = (x_1, x_2, \dots, x_{m-1}, x_m)$ !

Jelölje  $\text{SHIFT}(X)$  azt a vektort, melyre

$$\text{SHIFT}(X) = \text{SHIFT}(x_1, x_2, \dots, x_{m-1}, x_m) = (0, x_1, x_2, \dots, x_{m-1})$$

Az a alapsztringet karakterenként vizsgáljuk végig, és minden  $a_i$  karakter érintésekor frissítjük a  $D$  vektort a következő formula felhasználásával:

$$D = (\text{SHIFT}(D) \vee U) \wedge B_{a_i}.$$

Ha a keresés során az  $i$ -edik karakter feldolgozása után teljesül a

$$D \wedge V \neq 0^m = (\underbrace{0, 0, \dots, 0}_m)$$

feltétel, akkor megtaláltuk a  $p$  mintasztring egy előfordulását az  $a$  alapsztringben. A minta első karaktere ekkor az alapsztring  $(i - m + 1)$ -edik karakterére illeszkedik.

Feltéve, hogy a  $D$  kiszámításához szükséges műveletek konstans időben elvégezhetők, az algoritmus futási ideje:  $\Theta(n)$ .



A sztring  
 adatszerkezet

A meztílabas (brute  
 force) algoritmus

A Knuth–Morris–Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus

A kereső algoritmus

A Shift-And  
 (Dömököi-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

## A kereső algoritmus

**function** SHIFT\_AND(A, P)

```

1: n ← HOSSZ(A)
2: m ← HOSSZ(P)
3: for all c ∈ Σ do
4:   for j ← 1 to m do
5:     if P[j] = c then
6:       Bc,j ← 1
7:     else
8:       Bc,j ← 0
9:     end if
10:   end for
11: end for

12: for j ← 1 to m do
13:   Dj ← Uj ← Vj ← 0
14: end for
15: U1 ← Vm ← 1
16: for i ← 1 to n do
17:   D ←
        (SHIFT(D) ∨ U) ∧ BA[i]
18:   if D ∧ V ≠ 0m then
19:     return i-m+1
20:   end if
21: end for
22: return 0
end function
```



A sztring  
 adatszerkezet

A meztábas (brute  
 force) algoritmus

A Knuth–Morris–Pratt  
 algoritmus

Az előfeldolgozó  
 algoritmus  
 A kereső algoritmus

A Shift-And  
 (Dömölki-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

## A Shift-And (Dömölki-féle) algoritmus

Legyen  $a = atacgatata$  és  $p = atat!$

A Shift-And algoritmus által használt  $B$  bitmátrix a következő lesz:

	$a$	$t$	$a$	$t$
$a$	1	0	1	0
$t$	0	1	0	1
*	0	0	0	0

A mátrixban \* jelzi az ábécé  $a$ -tól és  $t$ -től különböző összes többi karakterét.



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintailleszték

Reguláris kifejezések

<i>i</i>	<i>ai</i>	<i>D</i>	$S_H(D)$	$S_H(D) \vee U$	<i>Bai</i>	<i>D'</i>
1	<i>a</i>	(0, 0, 0, 0)	(0, 0, 0, 0)	(1, 0, 0, 0)	(1, 0, 1, 0)	(1, 0, 0, 0)
2	<i>t</i>	(1, 0, 0, 0)	(0, 1, 0, 0)	(1, 1, 0, 0)	(0, 1, 0, 1)	(0, 1, 0, 0)
3	<i>a</i>	(0, 1, 0, 0)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)
4	<i>c</i>	(1, 0, 1, 0)	(0, 1, 0, 1)	(1, 1, 0, 1)	(0, 0, 0, 0)	(0, 0, 0, 0)
5	<i>g</i>	(0, 0, 0, 0)	(0, 0, 0, 0)	(1, 0, 0, 0)	(0, 0, 0, 0)	(0, 0, 0, 0)
6	<i>a</i>	(0, 0, 0, 0)	(0, 0, 0, 0)	(1, 0, 0, 0)	(1, 0, 1, 0)	(1, 0, 0, 0)
7	<i>t</i>	(1, 0, 0, 0)	(0, 1, 0, 0)	(1, 1, 0, 0)	(0, 1, 0, 1)	(0, 1, 0, 0)
8	<i>a</i>	(0, 1, 0, 0)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)
9	<i>t</i>	(1, 0, 1, 0)	(0, 1, 0, 1)	(1, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
10	<i>a</i>	(0, 1, 0, 1)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)
11	<i>t</i>	(1, 0, 1, 0)	(0, 1, 0, 1)	(1, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
12	<i>a</i>	(0, 1, 0, 1)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)

A táblázatból látható, hogy az algoritmus kétszer, a 9. és a 11. karakter feldolgozása után állapíthatja meg, hogy  $D' \wedge V \neq 0^m$ . A mintasztring így a  $9 - 4 + 1 = 6$ . és  $11 - 4 + 1 = 8$ . pozíciótól kezdve fordul elő az alapsztringben.



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## A Shift-And (Dömölki-féle) algoritmus

Az algoritmus hatalmas előnye, hogy párhuzamosan több mintát is kereshetünk a segítségével. Hogyan?

- ▶ A  $B$  bitmátrixnak annyi oszlopa lesz, ahány karakterből állnak a mintasztringek összesen. Kitöltése ugyanúgy történik, mint eddig.
- ▶ Az  $U$  bitvektor is annyi elemű lesz, ahány karakterből állnak a mintasztringek összesen. A vektorban az egyes minták első karakterének megfelelő pozíciókba 1-et írunk, a többibe 0-t.
- ▶ Az  $V$  bitvektor is annyi elemű lesz, ahány karakterből állnak a mintasztringek összesen. A vektorban az egyes minták utolsó karakterének megfelelő pozíciókba 1-et írunk, a többibe 0-t.
- ▶ Maga az algoritmus alapvetően nem változik. Ha a  $D \wedge V \neq 0^m$  feltétel teljesül, akkor az(oka)t a mintá(ka)t találtuk meg, amely(ek)nek az utolsó karakteréhez tartozó bitpozícióban 1 szerepel a  $D \wedge V$  vektorban.



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## A Shift-And (Dömölki-féle) algoritmus

Az algoritmus hatalmas előnye, hogy párhuzamosan több mintát is kereshetünk a segítségével. Hogyan?

- ▶ A  $B$  bitmátrixnak annyi oszlopa lesz, ahány karakterből állnak a mintasztringek összesen. Kitöltése ugyanúgy történik, mint eddig.
- ▶ Az  $U$  bitvektor is annyi elemű lesz, ahány karakterből állnak a mintasztringek összesen. A vektorban az egyes minták első karakterének megfelelő pozíciókba 1-et írunk, a többibe 0-t.
- ▶ Az  $V$  bitvektor is annyi elemű lesz, ahány karakterből állnak a mintasztringek összesen. A vektorban az egyes minták utolsó karakterének megfelelő pozíciókba 1-et írunk, a többibe 0-t.
- ▶ Maga az algoritmus alapvetően nem változik. Ha a  $D \wedge V \neq 0^m$  feltétel teljesül, akkor az(oka)t a mintá(ka)t találtuk meg, amely(ek)nek az utolsó karakteréhez tartozó bitpozícióban 1 szerepel a  $D \wedge V$  vektorban.



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## A Shift-And (Dömölki-féle) algoritmus

Legyen  $a = atacgatata, p_1 = atat, p_2 = gat$  és  $p_3 = tata!$

Az algoritmus által használt  $B$  bitmátrix a következő lesz:

	<i>a</i>	<i>t</i>	<i>a</i>	<i>t</i>	<i>g</i>	<i>a</i>	<i>t</i>	<i>t</i>	<i>a</i>	<i>t</i>	<i>a</i>
<i>a</i>	1	0	1	0	0	1	0	0	1	0	1
<i>g</i>	0	0	0	0	1	0	0	0	0	0	0
<i>t</i>	0	1	0	1	0	0	1	1	0	1	0
*	0	0	0	0	0	0	0	0	0	0	0

A mátrixban \* jelzi az ábécé *a*-tól, *g*-től és *t*-től különböző összes többi karakterét.

Az  $U$ ,  $V$  és  $D$  vektorok pedig így fognak kinézni:

$$U = (1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0)$$

$$V = (0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1)$$

$$D = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$



A sztring  
 adatszerkezet

A meztílabas (brute  
 force) algoritmus

A Knuth–Morris–Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus  
 A kereső algoritmus

A Shift-And  
 (Dömölki-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

# A Shift-Or algoritmus

## Az ötlet

Ha a Shift-And algoritmus által használt minden bitet negálunk, és felcseréljük egymással az algoritmusban a konjunkció ( $\wedge$ ) és a diszjunkció ( $\vee$ ) műveleteket, akkor ugyanazt az algoritmust kapjuk. Mivel azonban a SHIFT művelet mindenkor egy 0 bitet léptet be balról a  $D$  vektorban, az eltolás után elhagyható az  $U$  vektorral végzendő konjunkciós művelet (mivel az úgyis csak az első bitet nullázná ki), és ezért maga az  $U$  vektor is.

## Megjegyzés

Ha párhuzamosan több mintát szeretnénk keresni, akkor nem hagyható el sem az  $U$  vektor, sem a vele végzendő konjunkciós művelet. Ebben az esetben a Shift-Or algoritmus egyenértékű a Shift-And algoritmussal.



A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus  
A kereső algoritmus

A Shift-And  
(Dömölli-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# A Shift-Or algoritmus

## A megvalósítás

Az előfeldolgozás során felépített  $B$  bitmátrix egy  $c$  karakterhez tartozó sorában

$$B_{c,j} = \begin{cases} 0, & \text{ha } p_j = c, \\ 1 & \text{egyébként.} \end{cases}$$

A kereséshez az algoritmus most csak két  $m$  elemű segédvektort ( $D$  és  $V$ ) használ, melyeket a következőképpen definiálunk:

$$D_j = 1 \quad 1 \leq j \leq m \text{ esetén,}$$

$$V_j = \begin{cases} 0, & \text{ha } j = m, \\ 1 & \text{egyébként.} \end{cases}$$



A sztring  
 adatszerkezet

A meztáblas (brute  
 force) algoritmus

A Knuth-Morris-Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus  
 A kereső algoritmus

A Shift-And  
 (Dömölkai-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

# A Shift-Or algoritmus

## A SHIFT művelet

Legyen  $x = (x_1, x_2, \dots, x_{m-1}, x_m)$ ! Jelölje  $\text{SHIFT}(x)$  azt a vektort, melyre

$$\text{SHIFT}(x) = \text{SHIFT}(x_1, x_2, \dots, x_{m-1}, x_m) = (0, x_1, x_2, \dots, x_{m-1})!$$

Az a alapsztringet karakterenként vizsgáljuk végig, és minden  $a_i$  karakter érintésekor frissítjük a  $D$  vektort a következő formula felhasználásával:

$$D = \text{SHIFT}(D) \vee B_{a_i}.$$

Ha a keresés során az  $i$ -edik karakter feldolgozása után teljesül a

$$D \vee V \neq 1^m = (\underbrace{1, 1, \dots, 1}_m)$$

feltétel, akkor megtaláltuk a  $p$  mintasztring egy előfordulását az  $a$  alapsztringben. A minta első karaktere ekkor az alapsztring  $(i - m + 1)$ -edik karakterére illeszkedik.



A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth-Morris-Pratt  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölk-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

## A kereső algoritmus

**function** SHIFT\_Or(A, P)

```

1: n ← HOSSZ(A)
2: m ← HOSSZ(P)
3: for all c ∈ Σ do
4:   for j ← 1 to m do
5:     if P[j] = c then
6:       Bc,j ← 0
7:     else
8:       Bc,j ← 1
9:     end if
10:   end for
11: end for

12: for j ← 1 to m do
13:   Dj ← Vj ← 1
14: end for
15: Vm ← 0
16: for i ← 1 to n do
17:   D ← SHIFT(D) ∨ BA[i]
18:   if D ∨ V ≠ 1m then
19:     return i-m+1
20:   end if
21: end for
22: return 0
end function
```



A sztring  
 adatszerkezet

A meztáblas (brute  
 force) algoritmus

A Knuth–Morris–Pratt  
 algoritmus

A előfeldolgozó  
 algoritmus

A kereső algoritmus

A Shift-And  
 (Dömölk-féle)  
 algoritmus

A Shift-Or algoritmus

Rabin-Karp  
 mintaillesztés

Reguláris kifejezések

## A Shift-Or algoritmus

Legyen  $a = atacgatatata$  és  $p = atat!$

A Shift-Or algoritmus által használt  $B$  bitmátrix a következő lesz:

	$a$	$t$	$a$	$t$
$a$	0	1	0	1
$t$	1	0	1	0
*	1	1	1	1

A mátrixban \* jelzi az ábécé  $a$ -tól és  $t$ -től különböző összes többi karakterét.


 A sztring  
adatszerkezet

 A meztáblas (brute  
force) algoritmus

 A Knuth–Morris–Pratt  
algoritmus

 Az előfeldolgozó  
algoritmus

A kereső algoritmus

 A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

 Rabin-Karp  
mintaillesztés

Reguláris kifejezések

<i>i</i>	<i>ai</i>	<i>D</i>	$\text{SH}(D)$	<i>Bai</i>	<i>D'</i>
1	<i>a</i>	(1, 1, 1, 1)	(0, 1, 1, 1)	(0, 1, 0, 1)	(0, 1, 1, 1)
2	<i>t</i>	(0, 1, 1, 1)	(0, 0, 1, 1)	(1, 0, 1, 0)	(1, 0, 1, 1)
3	<i>a</i>	(1, 0, 1, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
4	<i>c</i>	(0, 1, 0, 1)	(0, 0, 1, 0)	(1, 1, 1, 1)	(1, 1, 1, 1)
5	<i>g</i>	(1, 1, 1, 1)	(0, 1, 1, 1)	(1, 1, 1, 1)	(1, 1, 1, 1)
6	<i>a</i>	(1, 1, 1, 1)	(0, 1, 1, 1)	(0, 1, 0, 1)	(0, 1, 1, 1)
7	<i>t</i>	(0, 1, 1, 1)	(0, 0, 1, 1)	(1, 0, 1, 0)	(1, 0, 1, 1)
8	<i>a</i>	(1, 0, 1, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
9	<i>t</i>	(0, 1, 0, 1)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)
10	<i>a</i>	(1, 0, 1, 0)	(0, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)
11	<i>t</i>	(0, 1, 0, 1)	(0, 0, 1, 0)	(1, 0, 1, 0)	(1, 0, 1, 0)
12	<i>a</i>	(1, 0, 1, 0)	(0, 1, 0, 1)	(0, 1, 0, 1)	(0, 1, 0, 1)

A táblázatból látható, hogy az algoritmus kétszer, a 9. és a 11. karakter feldolgozása után állapíthatja meg, hogy  $D' \vee V \neq 1^m$ . A mintasztring így a  $9 - 4 + 1 = 6$ . és  $11 - 4 + 1 = 8$ . pozíciótól kezdve fordul elő az alapsztringben.



## Rabin-Karp mintaillesztés

```
function RABIN_KARP_ILLESZTŐ(A, P, d, q)
```

```

1: n ← HOSSZ(A)
2: m ← HOSSZ(P)
3: h ←  $d^{m-1} \bmod q$ 
4: p ← t ← 0
5: for i ← 1 to m do
6:   p ← ( $d \cdot p + P[i]$ ) mod q
7:   a ← ( $d \cdot a + A[i]$ ) mod q
8: end for
9: for j ← 0 to n - m do
10:   if p = a és P = A[j+1..j+m] then
11:     return j+1
12:   end if
13:   a ← ( $d \cdot (a - A[j+1] \cdot h) + A[j + m + 1]$ ) mod q
14: end for
end function
```

Jel alkalmazható – pl. két dimenziós mintákra (képek) is.

A sztring  
adatszerkezet

A meztlábas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



## Rabin-Karp mintaillesztés

```
function RABIN_KARP_ILLESZTŐ(A, P, d, q)
```

```

1: n ← HOSSZ(A)
2: m ← HOSSZ(P)
3: h ←  $d^{m-1} \bmod q$ 
4: p ← t ← 0
5: for i ← 1 to m do
6:   p ← ( $d \cdot p + P[i]$ ) mod q
7:   a ← ( $d \cdot a + A[i]$ ) mod q
8: end for
9: for j ← 0 to n - m do
10:   if p = a és P = A[j+1..j+m] then
11:     return j+1
12:   end if
13:   a ← ( $d \cdot (a - A[j+1] \cdot h) + A[j + m + 1]$ ) mod q
14: end for
end function
```

Jól általánosítható – pl. két dimenziós mintákra (képek) is.

A sztring  
adatszerkezet

A meztílabas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

A előfeldolgozó  
algoritmus  
A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések



„ab\*c?d.e+f”

- \*: a megelőző 'karakter' tetszőleges számú előfordulása
- ?\*: a megelőző 'karakter' nulla vagy egy előfordulása
- +\*: a megelőző 'karakter' egy vagy több előfordulása
- .: tetszőleges karakter

Az „ab\*c?d.e+f” minta illeszkedik az alábbi alapsztringekre:

„fgabbbdreefad”, „rtacdhefr”, „sabbbcdeeffh”, „addef”,

és nem illeszkedik az alábbiakra:

„abbckld”, „acdb”, „cvfdcd”

A sztring  
adatszerkezet

A meztáblas (brute  
force) algoritmus

A Knuth–Morris–Pratt–  
algoritmus

Az előfeldolgozó  
algoritmus

A kereső algoritmus

A Shift-And  
(Dömölkki-féle)  
algoritmus

A Shift-Or algoritmus

Rabin-Karp  
mintaillesztés

Reguláris kifejezések

# 9. előadás

## A táblázat

A táblázatról általában, soros, önátrendező, rendezett és kulcstranszformációs táblázat

*Adatszerkezetek és algoritmusok előadás*

2018. április 10.



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing

Kulcstranszformációs  
módszerek

Szinonimák

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek  
Szinonimák

## Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing

Kulcstranszformációs  
módszerek

Színönmák

## Asszociatív adatszerkezetek

Adatelemei tartalmuk alapján, valamilyen kulcs segítségével címezhetők (pl.: tömb, mátrix, táblázat).

Legegyszerűbb eset: kulcs = index (pl. tömb).

Megkülönböztetünk:

- ① közvetlen (direkt) szervezésű struktúrák
- ② véletlen (indirekt) szervezésű struktúrák

Feladat: a kulcs alapján feleltessünk meg valamilyen memóriaterületet az adatalemnek.

Közvetlen szervezésű struktúrák esetén a kulcs és az adatalem között egyértelmű megfeleltetés állítható fel.



## A táblázat

A táblázat **dinamikus**, **homogén** és **asszociatív** adatszerkezet.

KULCS	ÉRTÉK

← a táblázat  
egy eleme

Elemei **összetettek**: **kulcs** és **érték** komponensből állnak.

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek  
Szinonimák

## A táblázat

A kulcsnak a táblázatban előforduló értékeit **kulcsértékeknek** nevezzük.

A táblázat elemeit a kulcsuk értékei alapján különböztetjük meg egymástól. A táblázatban nem szerepelhet két azonos kulcsérték. (Ezért nevezhetjük őket kulcsértékeknek.)



## Asszociatív adatszerkezetek

#### A táblázatról általában

## Soros táblázat

## Önájtrendező táblázat

## Rendezett táblázat

## Kulcstranszformációs táblázat

## Hash függvény, hashing Kulcstranszformációs módszerek Szinonimák



Asszociatív  
 adatszerkezetek

A táblázatról általában

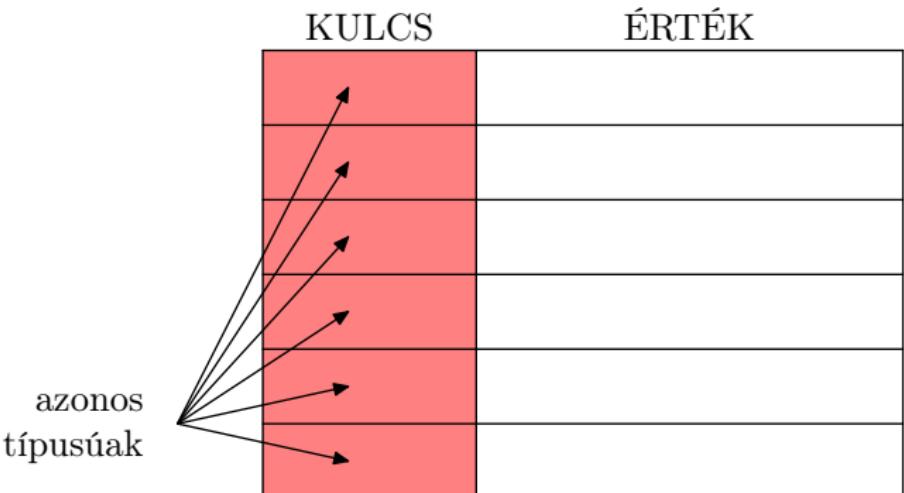
Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

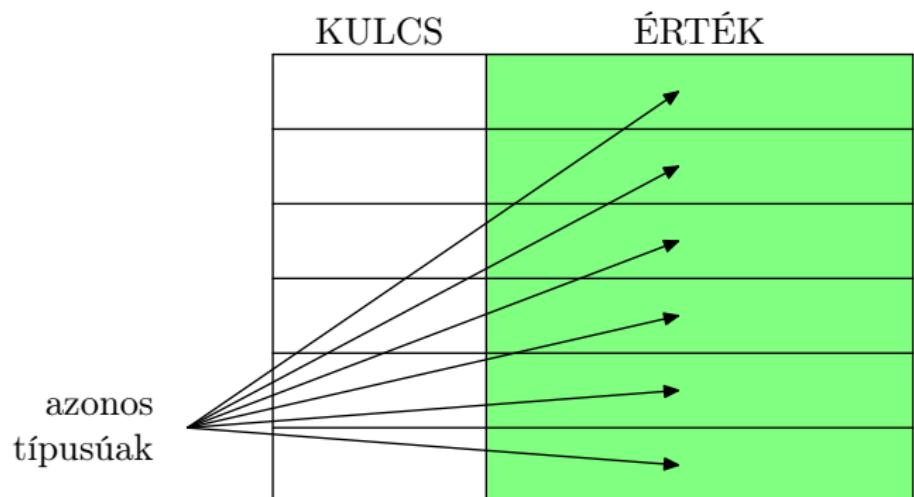
Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák



A **KULCS** és **ÉRTÉK** komponensek típusának sem feltétlenül kell megegyeznie egymással. Az adatelemekben a kulcsértékek típusát a KULCS típusa, az értékek típusát az ÉRTÉK típusa határozza meg.

## A táblázat

A **táblázat** az egydimenziós tömb **általánosításának** tekinthető: az adatelemeket a kulcsuk értéke alapján azonosítjuk, ezeknek az értékeknek azonban nem feltétlenül kell egész számoknak lenniük, lehet más típusuk is.



A **KULCS** és **ÉRTÉK** komponensek típusának sem feltétlenül kell megegyeznie egymással. Az adatelemekben a kulcsértékek típusát a KULCS típusa, az értékek típusát az ÉRTÉK típusa határozza meg.



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing

Kulcstranszformációs  
módszerek

Szinonimák



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

## Soros táblázat

Az adataelemek táblázatbeli sorrendje megegyezik az elemek táblázatba kerülésének időbeli (érkezési) sorrendjével.

### Soros táblázattal végezhető műveletek

- ▶ **Létrehozás:** a KULCS és az ÉRTÉK típusának meghatározása, és az elemek elhelyezése érkezési sorrendben.
- ▶ **Bővítés:** a táblázat végén<article>, az új elemet az eddig utolsó elem mögé helyezzük el.
- ▶ **Törlés:** mind folytonos, mind szétszórt reprezentáció mellett megvalósítható a fizikai törlés.
- ▶ **Csere:** az érték rész bármikor, kulcsérték csak akkor cserélhető, ha az új érték még nem szerepel.
- ▶ **Rendezés:** nem értelmezett.
- ▶ **Keresés:** teljes keresés a kulcs értéke alapján.
- ▶ **Elérés:** soros.
- ▶ **Bejárás:** általában az elejtől a végéig.
- ▶ A **feldolgozás** alapja a kulcs és a teljes keresés.



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek  
Szinonimák

## Soros táblázat

Folytonosan és szétszórtan (pl. egyirányban láncolt listával) egyaránt ábrázolható.

Előbbi esetben a **fizikai törlést** úgy valósíthatjuk meg, hogy a törlendő adatelemet a táblázat utolsó elemével felülírjuk, és csökkentjük a táblázat aktuális elemszámát.

Utóbbi esetben a törlés – szokás szerint – mutatóértékek cseréjével oldható meg.

### Mikor érdemes használni?

Ha az elemek feldolgozási gyakorisága nagyjából azonos, és nem lényeges az elemek feldolgozásának a sorrendje.



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

## Önárendező táblázat

### Önárendező táblázattal végezhető műveletek

A műveletek legtöbbje megegyezik a soros táblázatéval.  
 Kivételt képez a **feldolgozás**.

- ▶ Egy elem **feldolgozása** után a feldolgozott elemet a táblázat elejére helyezzük, az addigi első elem elő. Ennek következtében mindenkor a legutoljára feldolgozott elem lesz a táblázat elején.

Nagyszámú adatelem feldolgozása után az elemek sorrendje a táblázatban jól fogja közelíteni a feldolgozási gyakoriságot: a táblázat elején lesznek a gyakrabban feldolgozott elemek.

Az önárendező táblázat legjobban szétszórtan, egyirányban láncolt listával reprezentálható. Ekkor a feldolgozás művelete minden össze három mutató értékének a cseréjét jelenti.

### Mikor érdemes használni?

Ha az elemek feldolgozási gyakorisága (nagyon) eltérő.



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek  
Szinonimák

## Rendezett táblázat

Elemei a kulcsértékek alapján rendezettek, az adatelemek sorrendjét a kulcsértékek (általában) növekvő sorrendje definiálja.

### Rendezett táblázattal végezhető műveletek

A műveletek legtöbbje megegyezik a soros táblázatéval.  
Kivételt képeznek a következő műveletek:

- ▶ **Létrehozás** és **bővítés**: az érkező elemeket rendezetten helyezzük el a táblázatban, rendezett sorozatba történő beszúrással.
- ▶ **Csere**: az adatelem értékrésze bármikor cserélhető.
- ▶ **Keresés**: lineáris vagy bináris (reprezentációfüggő).

Folytonos reprezentációnál gyorsabb a keresés, de nehézkes a bővítés és a törlés.

Szétszórt reprezentációnál ennek az ellenkezője igaz.

### Mikor érdemes használni?

Ha az elemek feldolgozási gyakorisága nagyjából azonos, és fontos a feldolgozásuk sorrendje és gyorsasága.



# Kulcstranszformációs táblázat

Véletlen szervezésű struktúrák.

## Cél

A közvetlen elérés biztosítása a kulcsértékek alapján.

## Hash függvény

A kulcstranszformációs táblázatban egy  $K$  értékű kulccsal rendelkező elem helyét (címét) egy  $h$  függvény  $h(K)$  értéke határozza meg. Ezt a  $h$  függvényt hívjuk **hasító** vagy **hash** függvénynek.

## Hashing

Azt az eljárást, melynek során egy adatelem  $K$  értékű kulcsához meghatározzuk a  $h(K)$  értéket (az adatelem táblázatbeli helyét), **hasításnak**, **hashingnek**, **randomizálásnak** vagy **kulcstranszformációnak** nevezük.

Ennek a táblázatfajtának az ábrázolása **folytonos** – vagy legalább a folytonos ábrázoláson „alapul”.

Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing

Kulcstranszformációs  
 módszerek

Szinonimák



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

## Kulcstranszformációs táblázat

**Kölcsönösen egyértelmű** hash függvény használható, ha

- ▶ a gyakorlatban előforduló kulcsértékek száma közel azonos az elvileg lehetséges kulcsértékek számával;
- vagy
- ▶ a gyakorlatban előforduló kulcsértékek egyenletesen oszlanak el az elvileg lehetséges kulcsértékek között.

Ha a gyakorlatban előforduló kulcsértékek száma és az elvi lehetőségek száma között nagy az eltérés, és a gyakorlatban előforduló kulcsértékeknek nem egyenletes az eloszlása, akkor csak **egyértelmű** hash függvények használatára van lehetőség.

Egy **egyértelmű** hash függvénytől a következőket várjuk el:

- ▶ a gyakorlatban előforduló kulcsokat képezze le a rendelkezésre álló címtartományba;
- ▶ a rendelkezésre álló címtartományon belül tegye **egyenletessé** az elemek eloszlását.



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing

Kulcstranszformációs  
 módszerek

Szinonimák

# Kulcstranszformációs módszerek

## Kulcstranszformációs módszer

A kulcstranszformációs módszer egy algoritmus, amely azt írja le, hogy a hash függvény hogyan képezi le a kulcsértéket a tábeli címre.

A gyakorlatban a kulcsok típusa alapján megkülönböztetünk

- ▶ **szöveges** és
- ▶ **numerikus** kulcsokat.

**Szöveges** típusú kulcsok esetén a szöveget alkotó karakterek belső kódjainak valamelyen numerikus függvényét tekintjük, amellyel a kulcstranszformációt a numerikus típusú kulcsok esetére vezethetjük vissza.

**Numerikus** típusú kulcsok esetén – többek között – az alábbi módszerek használhatók:

- ▶ prímszámmal való osztás
- ▶ szorzás
- ▶ helyérték-kiválasztás
- ▶ bázistranszformáció



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Szinonimák, szinonimakezelő módszerek

Univerzális hash függvény nem létezik.

## Szinonimák

A csak egyértelmű hash függvényeknél előfordulhat, hogy a függvény a **különböző** kulcsértékekkel rendelkező adatelemekhez **ugyanazt** a tárcímet rendeli. Az ilyen adatelemeket **szinonimáknak**, magát a jelenséget pedig **túlcsordulásnak** nevezzük.

(Egyéb elnevezések: ütközés, ütközéskezelés.)

A szinonimák felbukkanását kezelní kell, mert nem helyezhetünk el két vagy több adatelemt ugyanazon a tárhelyen.

Néhány szinonimakezelő módszer:

- ▶ független túlcsordulási lista alkalmazása
- ▶ nyílt címzés módszere
- ▶ nyílt címzés (belso) láncolással



# Független túlcsordulási lista

21  
sárga

1  
piros

12  
barna

5  
barna

23  
kék

9  
barna

44  
zöld

31  
sárga

$$h(k) = k \bmod 11 + 1$$

1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

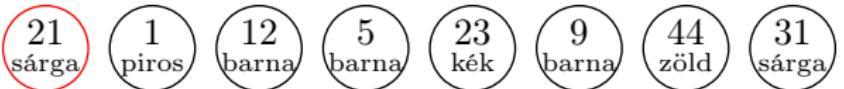
Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$

1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11	21	sárga	NIL

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák

# Független túlcsordulási lista

21  
sárga

1  
piros

12  
barna

5  
barna

23  
kék

9  
barna

44  
zöld

31  
sárga

$$h(k) = k \bmod 11 + 1$$

1			
2	1	piros	NIL
3			
4			
5			
6			
7			
8			
9			
10			
11	21	sárga	NIL



[Asszociatív  
adatszerkezetek](#)

[A táblázatról általában](#)

[Soros táblázat](#)

[Önárendező táblázat](#)

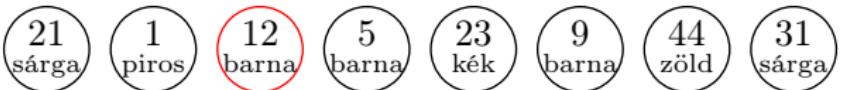
[Rendezett táblázat](#)

[Kulcstranszformációs  
táblázat](#)

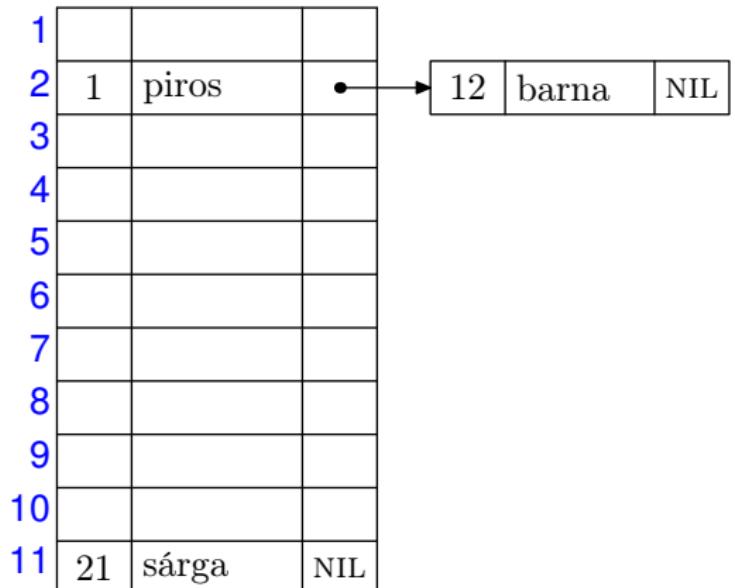
Hash függvény, hashing  
Kulcstranszformációs  
módszerek

[Szinonimák](#)

# Független túlcsordulási lista

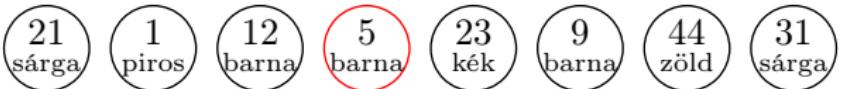


$$h(k) = k \bmod 11 + 1$$





# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$

1			
2	1	piros	• →
3			
4			
5			
6	5	barna	NIL
7			
8			
9			
10			
11	21	sárga	NIL

Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

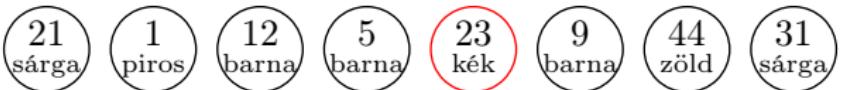
Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

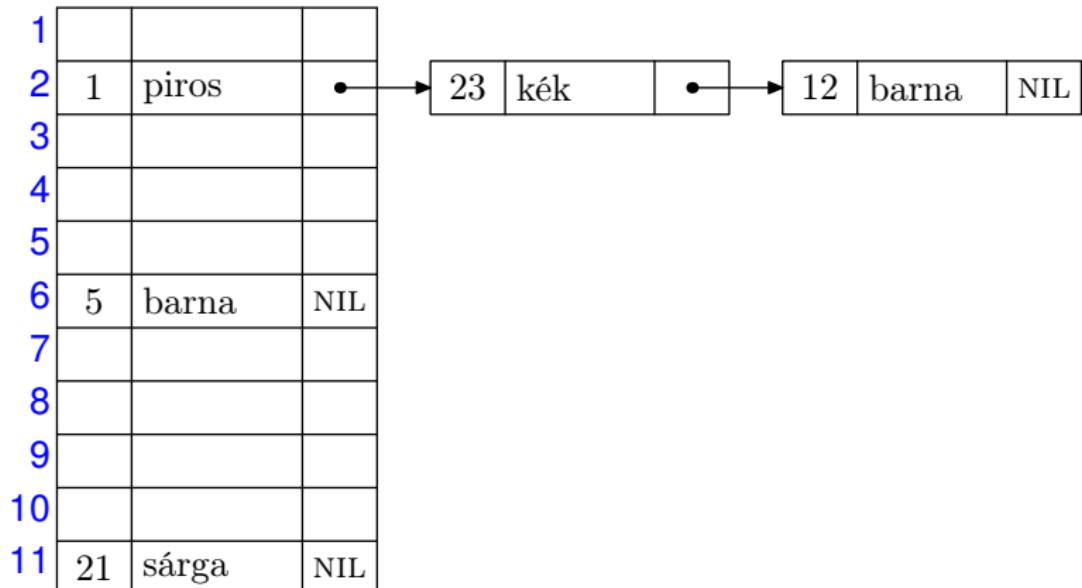
Szinonimák



# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák



# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$

1			
2	1 piros	•	→ [23   kék   •] → [12   barna   NIL]
3			
4			
5			
6	5 barna	NIL	
7			
8			
9			
10	9 barna	NIL	
11	21 sárga	NIL	

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák



# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$

1	44	zöld	NIL
2	1	piros	• →
3			
4			
5			
6	5	barna	NIL
7			
8			
9			
10	9	barna	NIL
11	21	sárga	NIL



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önátrendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák



# Független túlcsordulási lista



$$h(k) = k \bmod 11 + 1$$

1	44	zöld	NIL
2	1	piros	• → [23   kék   NIL]
3			
4			
5			
6	5	barna	NIL
7			
8			
9			
10	9	barna	• → [31   sárga   NIL]
11	21	sárga	NIL

Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

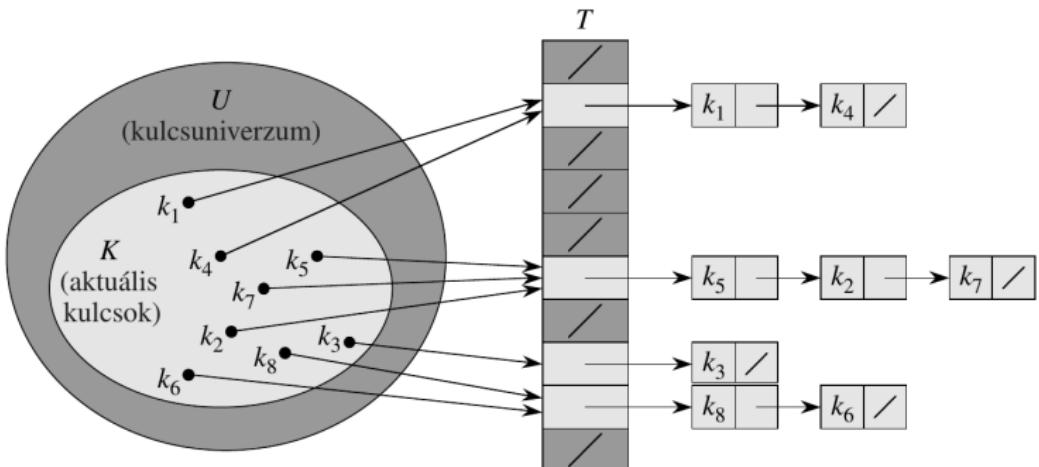
Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Független túlcordulási lista





Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

# Független túlcsordulási lista

## Hatókonyság

Legyen  $T$  egy  $m$  rést tartalmazó hasító táblázat, melyben  $n$  elem van. Definiáljuk  $T$ -ben az  $\alpha$  kitöltési tényezőt, mint az  $n/m$  hányadost, ami nem más, mint az egy láncba fűzött elemek átlagos száma.

Feltételezzük, hogy minden elem egyforma valószínűsséggel képződik le bármely résre, függetlenül attól, hogy a többiek hová kerültek. Ezt **egyszerű egyenletes hasítási feltételnek** nevezzük.

Ha a  $T[j]$  lista hosszát  $n_j$ -vel ( $j = 1, 2, \dots, m$ ), jelöljük, akkor

$$n = n_1 + n_2 + \dots + n_m$$

és  $n_j$  várható értéke  $E[n_j] = n/m = \alpha$ .

Feltesszük, hogy a  $h(k)$  hasított érték  $O(1)$  idő alatt számítható ki, s így egy  $k$  kulcsú elem keresésének ideje lineárisan függ a  $T[h(k)]$  lista  $n_{h(k)}$  hosszától.



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

# Független túlcsordulási lista

## Hatókonyság

### Theorem (Tankönyv: 11.1. téTEL.)

*Ha egy hasító táblázatban az ütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikertelen keresés átlagos ideje  $\Theta(1 + \alpha)$ .*

### Bizonyítás.

Az egyszerű egyenletesség feltételezése miatt bármely  $k$  kulcs, amely még nincs a táblázatban, egyforma valószínűsséggel képződik le az  $m$  rés bármelyikére. Ezért egy  $k$  kulcs sikertelen keresésének átlagos ideje megegyezik annak átlagos idejével, hogy a  $T[h(k)]$  listát végigkeressük. Ennek a listának az átlagos hossza  $E[n_{h(k)}] = \alpha$ . Ezért a sikertelen keresés során megvizsgált elemek várható száma  $\alpha$ , s így az összes szükséges idő (beszámítva a  $h(k)$  kisszámítási idejét is) valóban  $\Theta(1 + \alpha)$ .  $\square$

### Theorem (Tankönyv: 11.2. téTEL.)

*Ha egy hasító táblázatban a kulcsütöközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikeres keresés átlagos ideje  $\Theta(1 + \alpha)$ .*



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

# Független túlcsordulási lista

## Hatókonyság

### Theorem (Tankönyv: 11.1. téTEL.)

*Ha egy hasító táblázatban az ütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikertelen keresés átlagos ideje  $\Theta(1 + \alpha)$ .*

### Bizonyítás.

Az egyszerű egyenletesség feltételezése miatt bármely  $k$  kulcs, amely még nincs a táblázatban, egyforma valószínűsséggel képződik le az  $m$  rés bármelyikére. Ezért egy  $k$  kulcs sikertelen keresésének átlagos ideje megegyezik annak átlagos idejével, hogy a  $T[h(k)]$  listát végigkeressük. Ennek a listának az átlagos hossza  $E[n_{h(k)}] = \alpha$ . Ezért a sikertelen keresés során megvizsgált elemek várható száma  $\alpha$ , s így az összes szükséges idő (beszámítva a  $h(k)$  kiszámítási idejét is) valóban  $\Theta(1 + \alpha)$ .  $\square$

### Theorem (Tankönyv: 11.2. téTEL.)

*Ha egy hasító táblázatban a kulcsütöközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikeres keresés átlagos ideje  $\Theta(1 + \alpha)$ .*

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

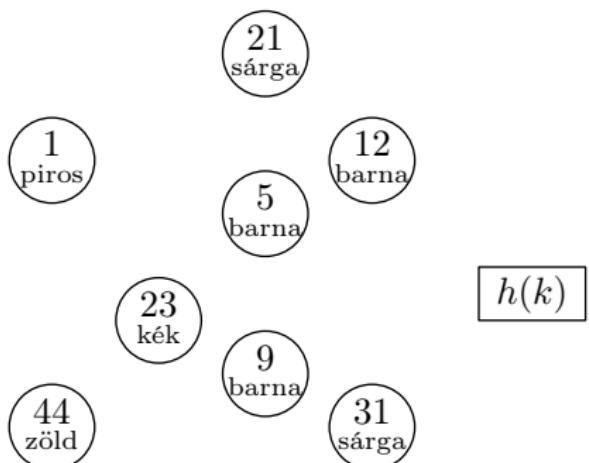
Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

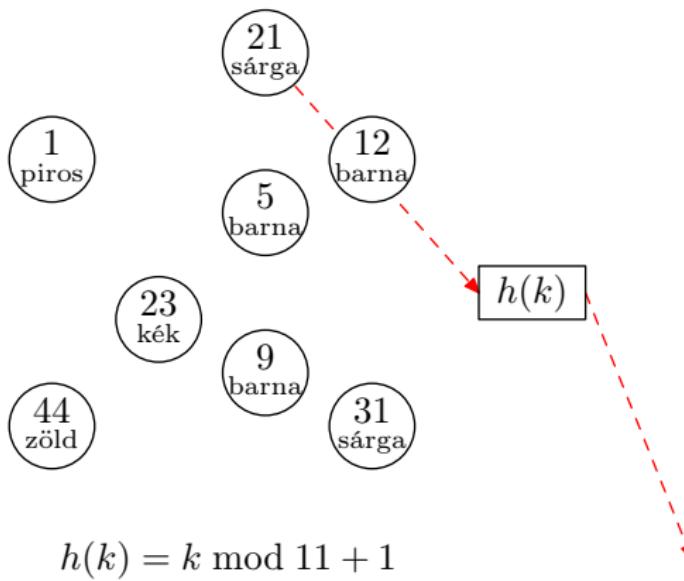
Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	21 sárga


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

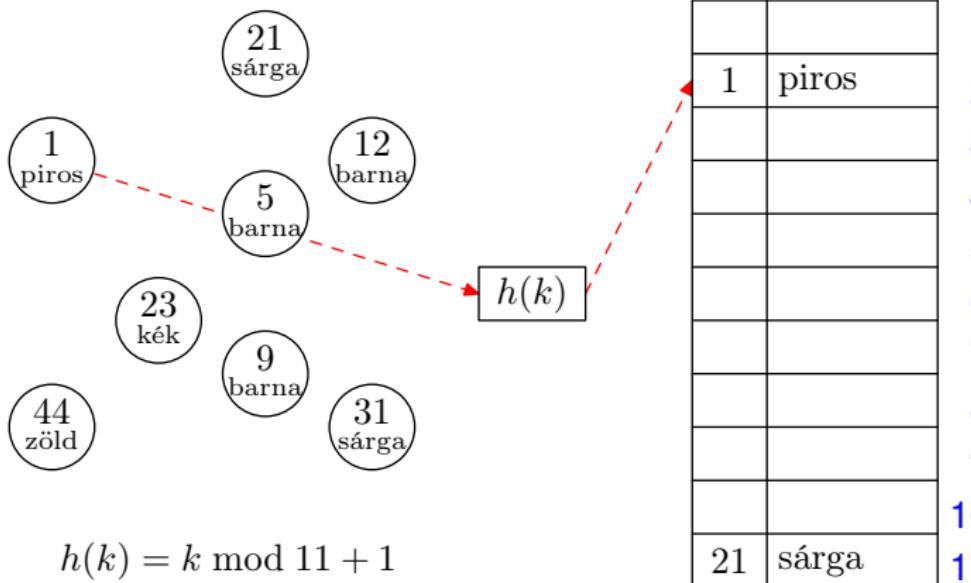
 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:





Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módserek

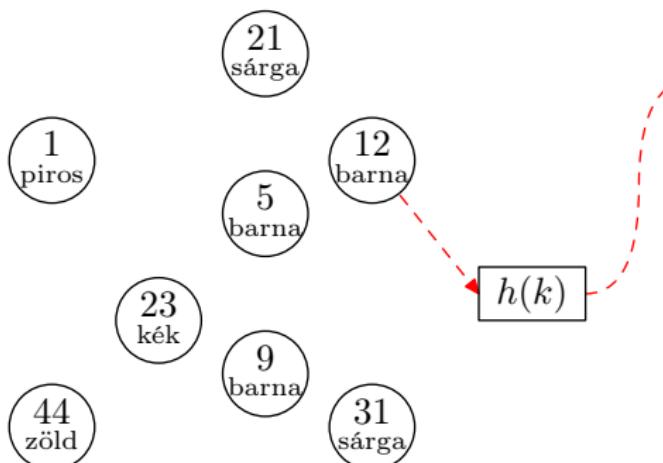
Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

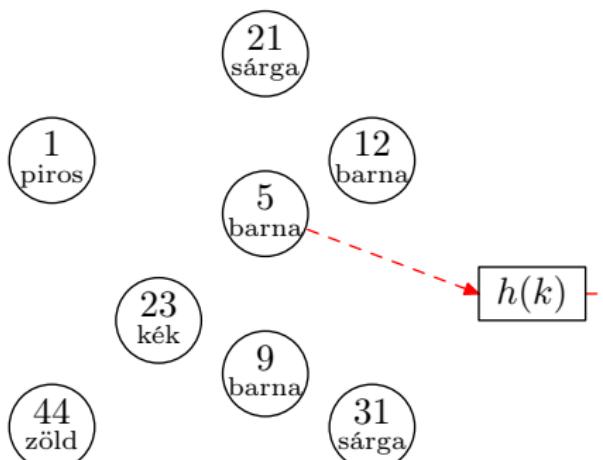
 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros
2	barna
3	
4	
5	barna
6	
7	
8	
9	
10	
11	sárga


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

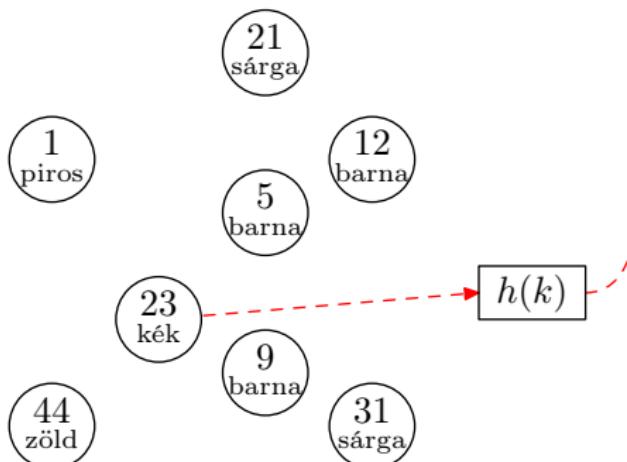
Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros
2	barna
3	kék
4	
5	barna
6	
7	
8	
9	
10	
11	sárga


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

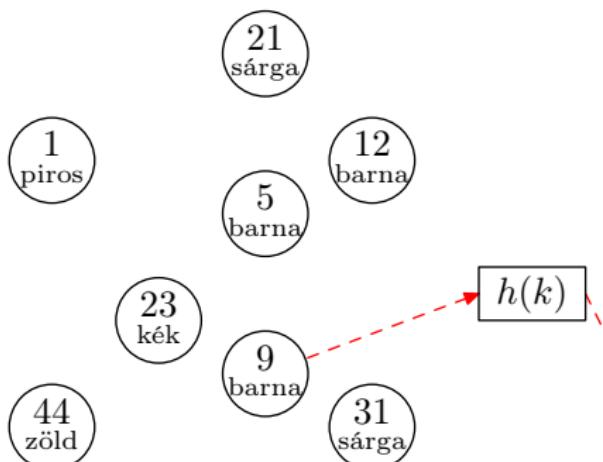
 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros
2	barna
3	kék
4	
5	barna
6	
7	
8	
9	barna
10	sárga
11	


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

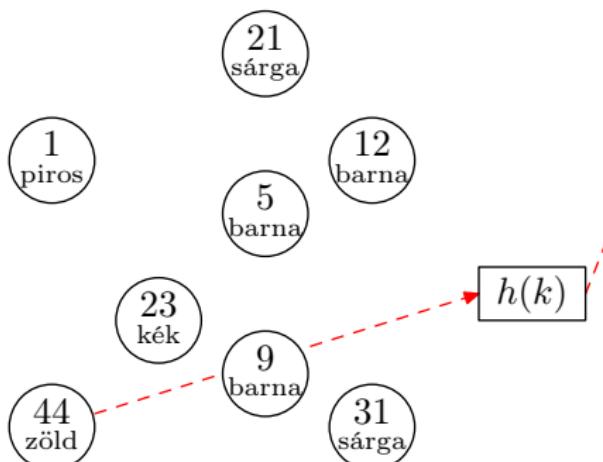
 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek  
 Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

44	zöld
1	piros
12	barna
23	kék
5	barna
9	barna
31	sárga
44	zöld
9	barna
21	sárga


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

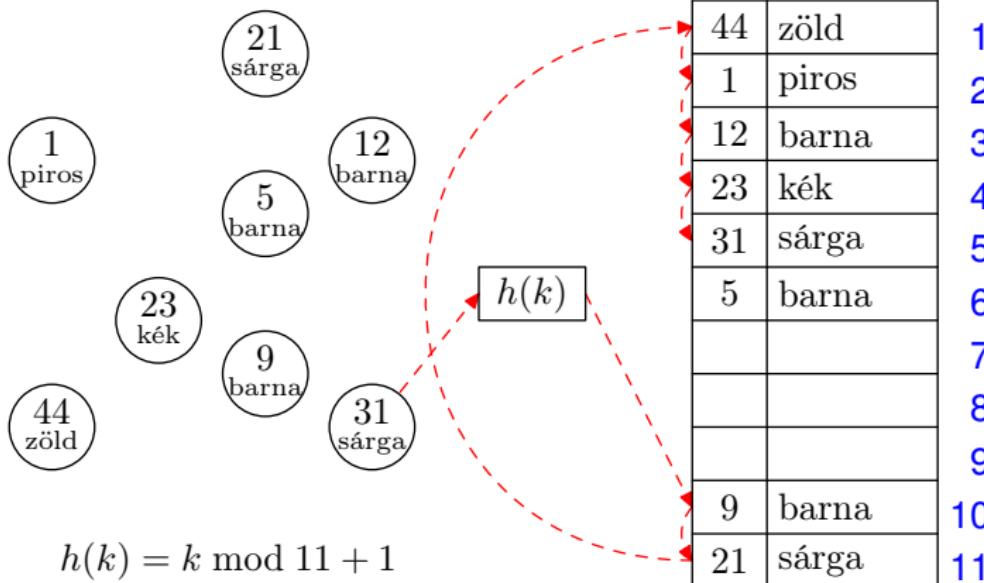
Szinonimák

# Nyílt címzés módszere

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:





Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

## Nyílt címzés módszere

- ▶ Ha adott egy  $h' : U \rightarrow \{1, 2, \dots, m\}$  közönséges hasító függvény, akkor a **lineáris kipróbálás** módszere az alábbi hasító függvényt használja ( $i=1,2,\dots,m$ ):

$$h(k, i) = (h'(k) + i - 1) \mod m + 1.$$

- ▶ Négyzetes kipróbálás módszere:

$$h(k, i) = (h'(k) + c_1(i-1) + c_2(i-1)^2) \mod m + 1.$$

- ▶ Dupla hasítás módszere:

$$h(k, i) = (h_1(k) + (i-1)h_2(k)) \mod m + 1.$$

Itt  $h_2(k)$ -nak relatív prímnek kell lennie  $m$ -hez képest.  
 (Pl.  $m = 2^k$  és  $h_2(k) = h_2(k) * 2 + 1.$ )

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák

## Nyílt címzés módszere

- ▶ Ha adott egy  $h' : U \rightarrow \{1, 2, \dots, m\}$  közönséges hasító függvény, akkor a **lineáris kipróbálás** módszere az alábbi hasító függvényt használja ( $i=1,2,\dots,m$ ):

$$h(k, i) = (h'(k) + i - 1) \mod m + 1.$$

- ▶ Négyzetes kipróbálás módszere:

$$h(k, i) = (h'(k) + c_1(i-1) + c_2(i-1)^2) \mod m + 1.$$

- ▶ Dupla hasítás módszere:

$$h(k, i) = (h_1(k) + (i-1)h_2(k)) \mod m + 1.$$

Itt  $h_2(k)$ -nak relatív prímnek kell lennie  $m$ -hez képest.  
(Pl.  $m = 2^k$  és  $h_2(k) = h_2(k) * 2 + 1.$ )



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

## Nyílt címzés módszere

- Ha adott egy  $h' : U \rightarrow \{1, 2, \dots, m\}$  közönséges hasító függvény, akkor a **lineáris kipróbálás** módszere az alábbi hasító függvényt használja ( $i=1,2,\dots,m$ ):

$$h(k, i) = (h'(k) + i - 1) \mod m + 1.$$

- Négyzetes kipróbálás módszere:

$$h(k, i) = (h'(k) + c_1(i-1) + c_2(i-1)^2) \mod m + 1.$$

- Dupla hasítás módszere:

$$h(k, i) = (h_1(k) + (i-1)h_2(k)) \mod m + 1.$$

Itt  $h_2(k)$ -nak relatív prímnek kell lennie  $m$ -hez képest.  
 (Pl.  $m = 2^k$  és  $h_2(k) = h'_2(k) * 2 + 1$ .)



Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
 táblázat

Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

Szinonimák

## Nyílt címzés módszere

### Theorem (Tankönyv: 11.6. téTEL.)

Ha adott egy nyílt címzéses hasító táblázat az  $\alpha = n/m < 1$  kitöltöttségi aránnal, akkor a sikertelen keresés várható próbaszáma az egyenletes hasítási feltétel teljesülése esetén legfeljebb  $1/(1 - \alpha)$ .

### Corollary (Tankönyv: 11.7. következmény.)

Egy  $\alpha$  kitöltési tényezőjű nyílt címzéses hasító táblázatba való beszúrás várható próbaszáma az egyenletes hasítási feltétel teljesülése esetén legfeljebb  $1/(1 - \alpha)$ .

### Theorem (Tankönyv: 11.8. téTEL.)

Legyen adott egy nyílt címzéses hasító táblázat az  $\alpha < 1$  kitöltöttségi aránnal. Tegyük fel az egyenletes hasítási feltételt és azt, hogy a táblázat minden elemét egyforma valószínűséggel keressük. Ekkor a sikeres keresés várható próbaszáma

$$\frac{1}{\alpha} \ln \frac{1}{1 - \alpha}$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

Szinonimák

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

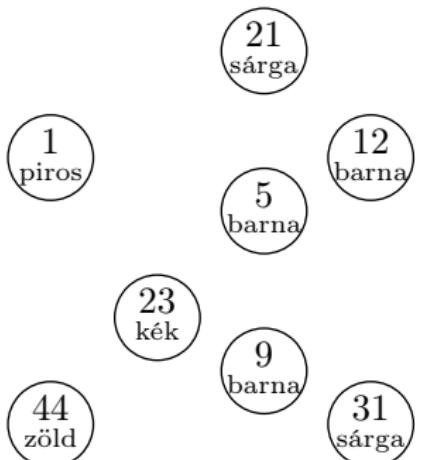

$$h(k)$$

## Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

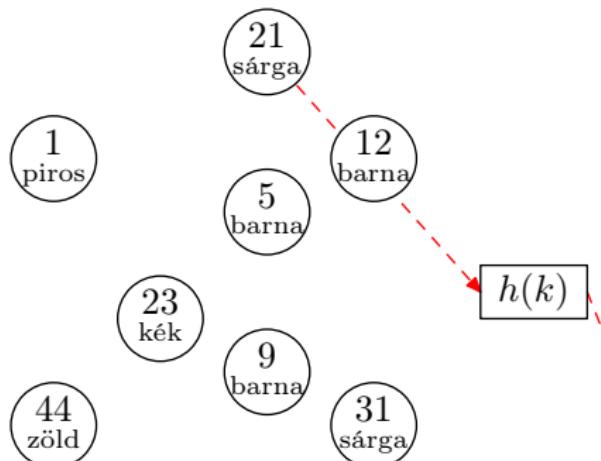
Szinonimák

# Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

21	sárga	0

1

2

3

4

5

6

7

8

9

10

11


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

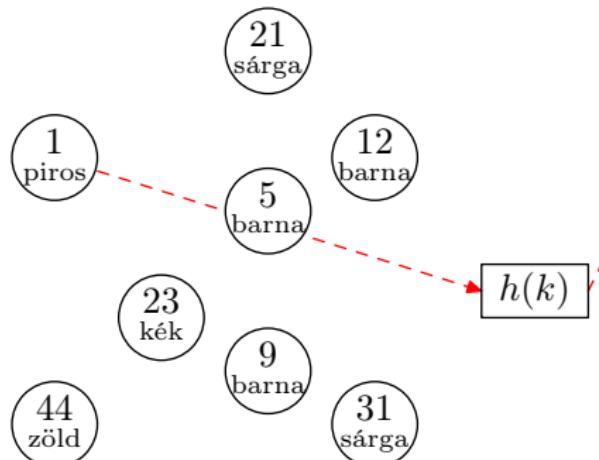
Szinonimák

# Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros	0
2		
3		
4		
5		
6		
7		
8		
9		
10		
11	sárga	0


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

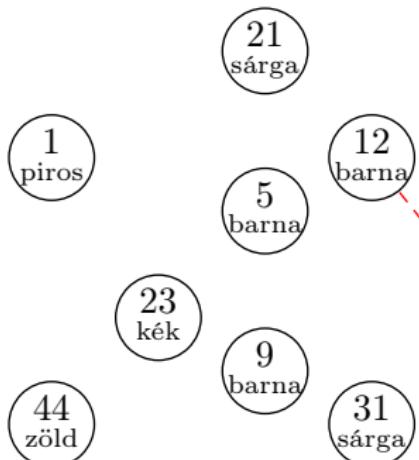
Szinonimák

# Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros	3
12	barna	0
21	sárga	0

 1  
 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 10  
 11


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

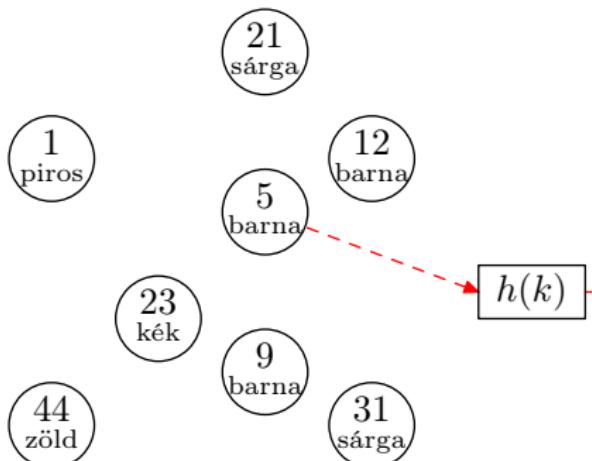
Szinonimák

# Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros	3
12	barna	0
5	barna	0
21	sárga	0

 1  
 2  
 3  
 4  
 5  
 6  
 7  
 8  
 9  
 10  
 11

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

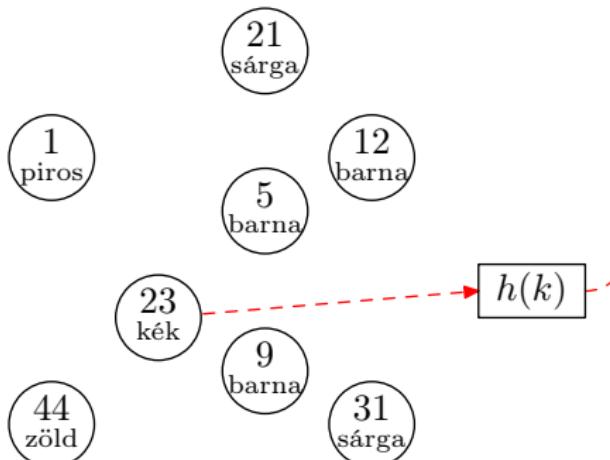
Szinonimák

## Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1	piros	3
12	barna	4
23	kék	0
5	barna	0
21	sárga	0

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

Hash függvény, hashing

 Kulcstranszformációs  
 módszerek

Szinonimák

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

1	piros	3
12	barna	4
23	kék	0
5	barna	0
9	barna	0
31	sárga	0

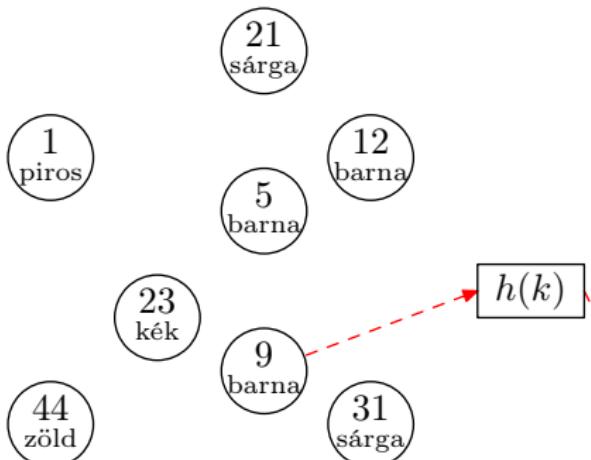
$$h(k)$$

## Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$


 Asszociatív  
 adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

 Kulcstranszformációs  
 táblázat

 Hash függvény, hashing  
 Kulcstranszformációs  
 módszerek

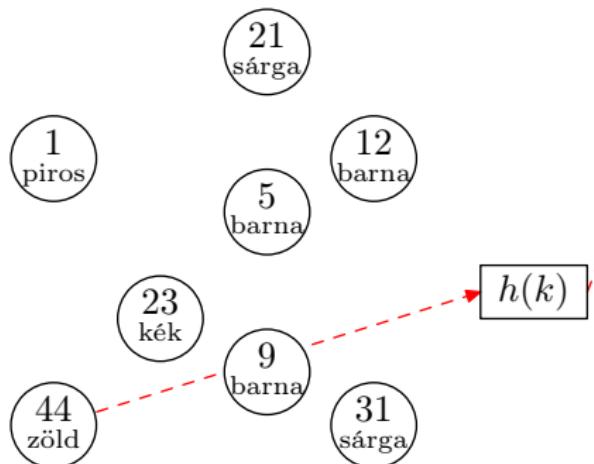
Szinonimák

# Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

44	zöld	0
1	piros	3
12	barna	4
23	kék	0
5	barna	0
9	barna	0
21	sárga	0

1

2

3

4

5

6

7

8

9

10

11

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Asszociatív  
adatszerkezetek

A táblázatról általában

Soros táblázat

Önárendező táblázat

Rendezett táblázat

Kulcstranszformációs  
táblázat

Hash függvény, hashing  
Kulcstranszformációs  
módszerek

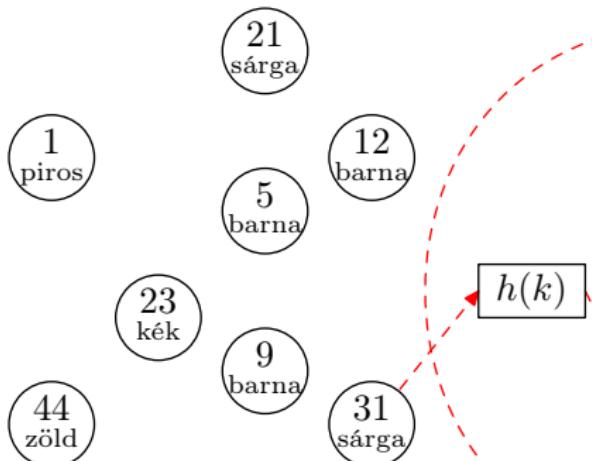
Szinonimák

## Nyílt címzés (belsı) láncolással

Egy tárhely státusza lehet:

- ▶ szabad (üres vagy logikailag törölt)
- ▶ foglalt

A táblázat bővítése:



$$h(k) = k \bmod 11 + 1$$

1  
piros

21  
sárga

12  
barna

5  
barna

23  
kék

9  
barna

31  
sárga

44  
zöld

$h(k)$

44	zöld	0
1	piros	3
12	barna	4
23	kék	0
31	sárga	0
5	barna	0
9	barna	5
21	sárga	0



[Asszociatív adatszerkezetek](#)

[A táblázatról általában](#)

[Soros táblázat](#)

[Önárendező táblázat](#)

[Rendezett táblázat](#)

[Kulcstranszformációs táblázat](#)

Hash függvény, hashing

Kulcstranszformációs módszerek

Szinonimák

## Kulcstranszformációs táblázattal végezhető műveletek

- ▶ **Létrehozás**kor a gyakorlatban előforduló kulcsértékek darabszámát megbecsüljük, kiválasztunk egy hash függvényt és (ha szükséges) egy szinonimakezelési módszert, majd lefoglaljuk a tárhelyeket a becslésnek megfelelően, és mindenket üresre állítjuk.
- ▶ **Bővítés**: az adatelem kulcsértéke alapján, a hash függvény segítségével.
- ▶ **Törlés**: szinte kizárolag logikai; fizikai csak a túlcordulási listából.
- ▶ **Csere**: kulcs alapján a hozzá tartozó értéket lehet cserálni.
- ▶ **Rendezés**: nincs.
- ▶ **Elérés**: kvázi közvetlen, a hash függvény a közvetlen elérést szolgálja, a szinonimákat keresni kell.
- ▶ **Keresés**: általában nincs, csak szinonimákat kereshetünk, az pedig a szinonimakezelési módszertől függ.
- ▶ **Bejárás**: nincs, mivel csak közvetlen elérés van.
- ▶ A **feldolgozás** alapja a hash függvény, illetve a közvetlen elérés.

# 10. előadás

## Gráfok

Gráfok, bejárások, legrövidebb út keresése.

Adatszerkezetek és algoritmusok előadás

2018. április 17.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hálós adatszerkezet  
(gráf)

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

# Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>



Hálós adatszerkezet  
(gráf)



## Hálós adatszerkezetek

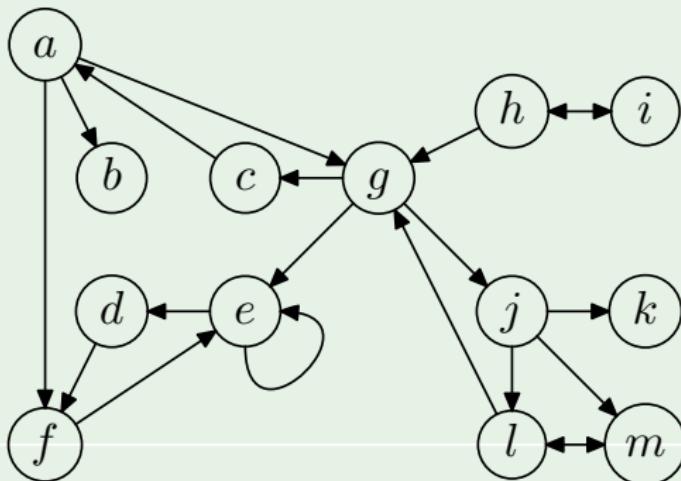
Minden adatelemnek tetszőleges számú megelőzője és tetszőleges számú rákövetkezője lehet.

Egy elem lehet egy másiknak (beleértve saját magát is) a

- ▶ megelőzője,
- ▶ mindkettő vagy
- ▶ rákövetkezője,
- ▶ egyik sem.

Ilyen adatszerkezet a **gráf**, amely a matematikában ismert (véges méretű, irányított) gráf struktúrának felel meg.

### Példa





## Gráfelméleti fogalmak

A **gráf** hálós adatszerkezet. A gráf **csúcsok** és **élek** halmaza. Egy él két csúcs közötti kapcsolat. Egy gráfot az határoz meg, hogy mely csúcsai vannak élekkel összekötve.

**Irányított gráf** esetében az éleknek irányuk van.

**Irányítatlan gráf** esetében az élekhez nincs irány rendelve, vagyis nem teszünk különbséget az „A-ból B-be” illetve a „B-ből A-ba” menő élek között.

**Út**nak nevezzük az élek olyan sorozatát, amelyben nem ismétlünk sem éleket, sem csúcsokat.

**Körnek** nevezzük azt az utat, amelynek kezdő- és végpontja azonos.

**Összefüggő** egy gráf, ha (élei esetleges irányításáról megfeledkezve) bármely két csúcs között van út.

**Egyszerű** gráfban bármely két csúcs között legfeljebb egy él lehet (kizártuk a többszörös éleket) és nem engedünk meg olyan éleket, amelyek kezdő- és végpontja azonos.

**Súlyozott gráf** esetén van egy w súlyfüggvény is.



# Gráffal végezhető műveletek

A gráf dinamikus és homogén adatszerkezet.

## Gráffal végezhető műveletek

- ▶ **Létrehozás:** üres gráfot hozunk létre.
- ▶ **Bővítés:** az új elem értéke mellett meg kell adni a megelőzőinek és a rakkövetkezőinek a listáját is.
  - Érinti az új elem megelőzőit is.
  - Speciális esetben új elem hozzáadása nélkül, új élekkel is bővíthetjük a gráfot.
- ▶ **Törlés:** fizikai. Érinti azokat az elemeket is, amelyekkel a törölt elem szomszédsági viszonyban állt.
- ▶ **Csere:** megoldható.
- ▶ **Rendezés:** nem értelmezett.
- ▶ **Keresés, elérés, feldolgozás:** a bejárás alapján.
- ▶ **Bejárás:** szélességi vagy mélységi.



## A gráf bejárása

Kiválasztunk egy tetszőleges elemet ( $S$ ), és ebből kiindulva térképezzük fel a gráf  $S$ -ből elérhető elemeit. Felépítünk egy  $S$  gyökerű feszítőfát (**szélességi** vagy **mélységi fa**). Ha  $S$ -ből nem érhető el a gráf összes eleme, akkor a maradék elemekből újra kiválasztunk egyet, és újabb feszítőfát építünk fel. Hogy minél kevesebb feszítőfa jöjjön létre, célszerű  $S$ -nek azt az elemet választani, amelyik a legtöbb rákövetkezővel rendelkezik.

A gráf elemeit háromféle színnel látjuk el:

- ▶ fekete (1. kategória): már bejárt elemek
- ▶ szürke (2. kategória): amiket látunk
- ▶ fehér (3. kategória): amiket *nem* látunk

A két algoritmus abban különbözik egymástól, hogy **szélességi bejárás** esetén a legkorábban, **mélységi bejárás** esetén pedig a legkésőbb elért szürke elemből kiindulva folytatjuk a feszítőfa építését.

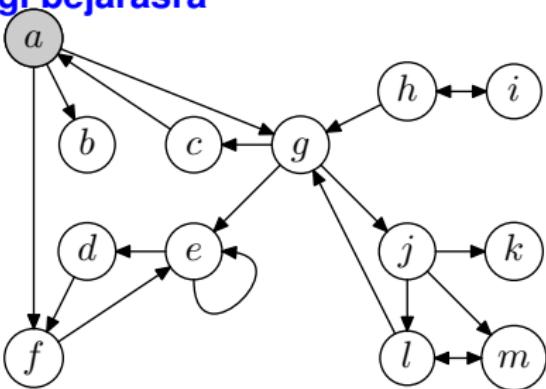


# Szélességi bejárás

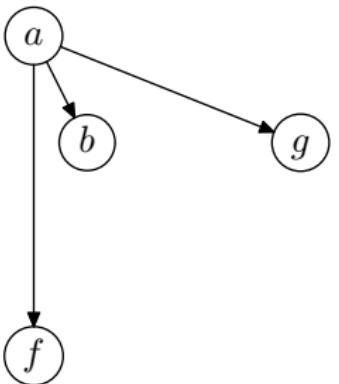
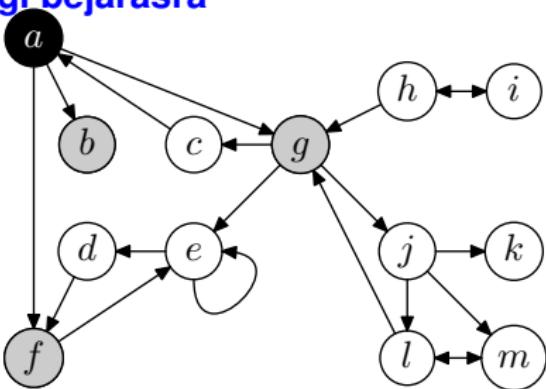
## Szélességi bejárás

A bejárás során az adatelemeket fehér, szürke és fekete színűre fogjuk színezni, valamint egy **sort** fogunk használni a szürke színű adatelemek tárolására. A bejárás során a feldolgozott elemekből feszítőfá(ka)t építünk fel.

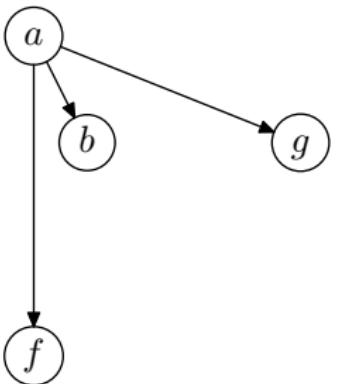
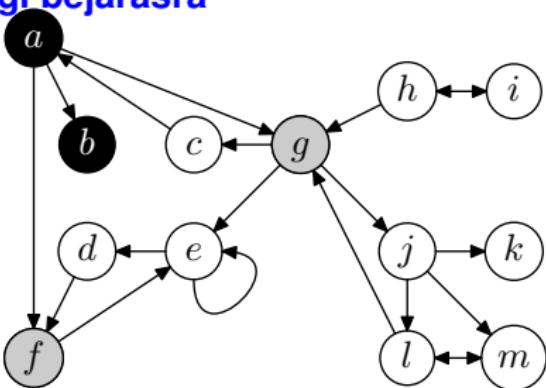
- 1 Színezzük a gráf adatelemeit fehér színűre, és hozzuk létre az üres sort.
- 2 Ha minden adatelem fekete színű, a bejárás véget ér.
- 3 Ha a sor üres, válasszunk tetszőlegesen egy fehér színű elemet, színezzük szürkére, és helyezzük el a sorban. Ennek az elemnek a feldolgozásakor új feszítőfát fogunk elkezdeni építeni.
- 4 Ha a sorban van elem, vegyük ki a sor első elemét, fehér színű gyermeket szürkére festve helyezzük el a sorban, majd az adatelemet fessük feketére, és helyezzük el a megfelelő feszítőfában.
- 5 Folytassuk az algoritmust a 2. lépéssel.



Sor: *a*

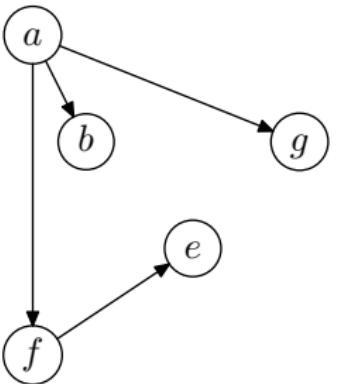
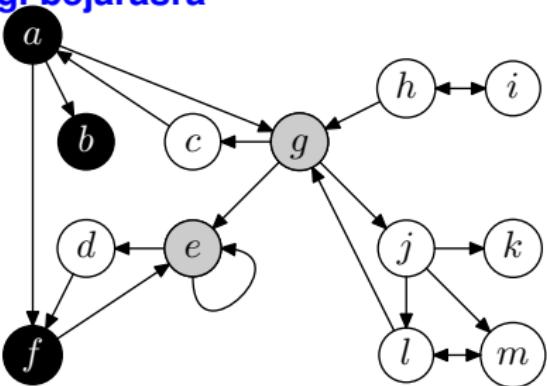


Sor:  $b, f, g$



Sor:  $f, g$

## Példa szélességi bejárásra



Sor:  $g, e$

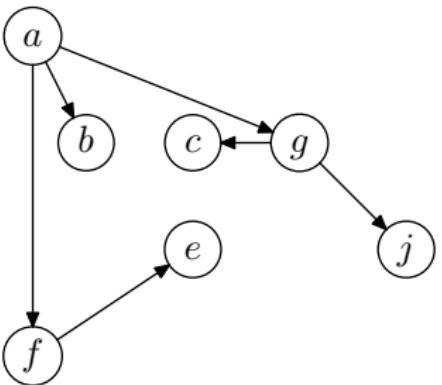
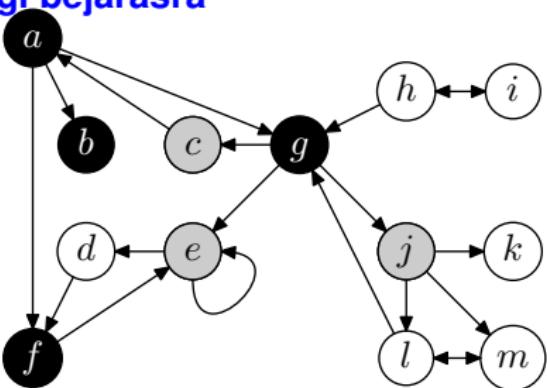
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



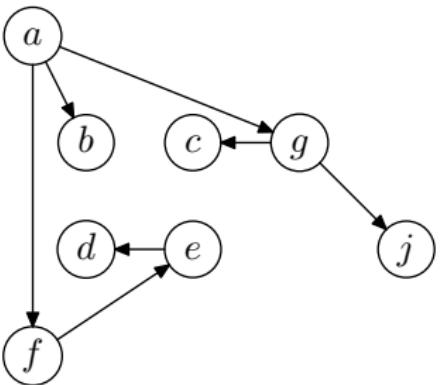
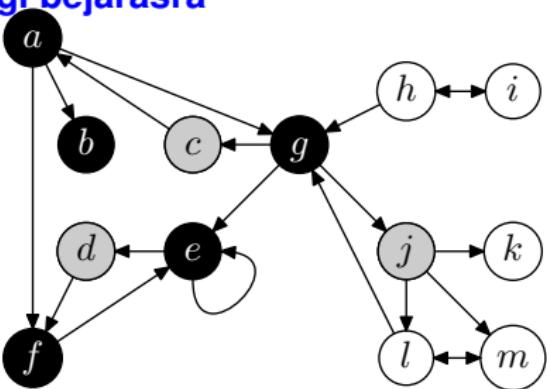
Hálós adatszerkezet  
(gráf)



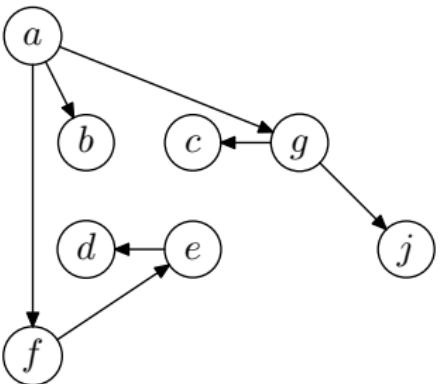
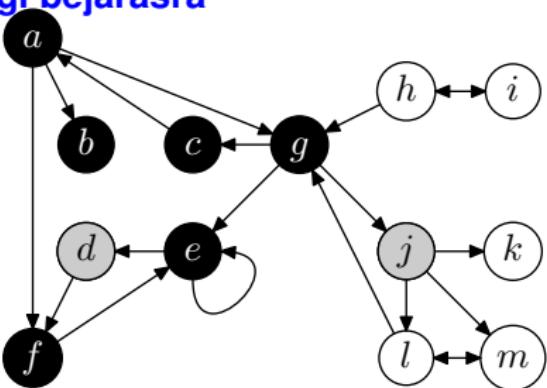
Hálós adatszerkezet  
(gráf)



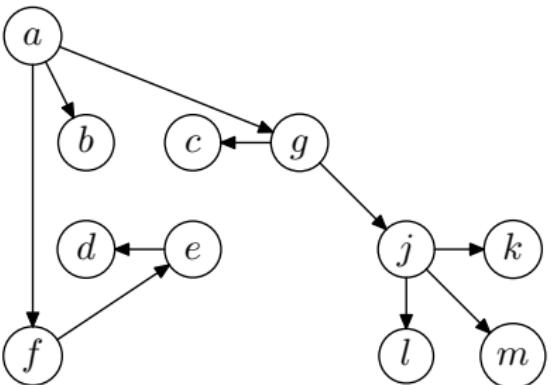
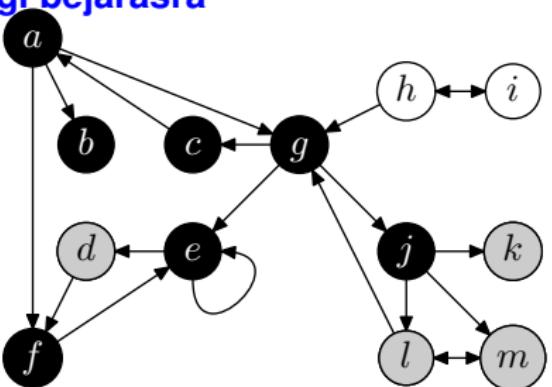
Sor:  $e, c, j$



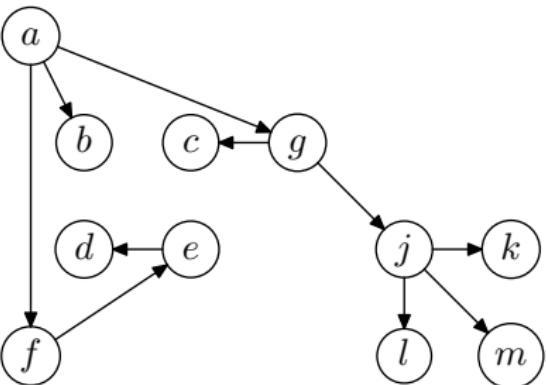
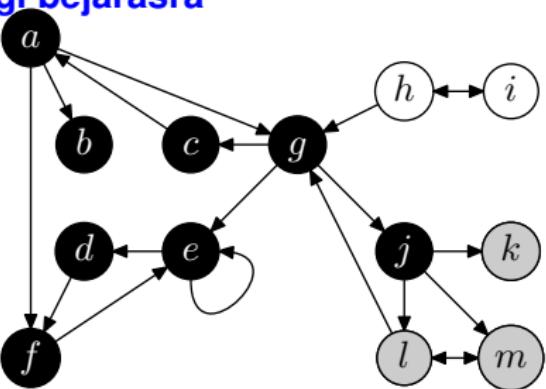
Sor:  $c, j, d$



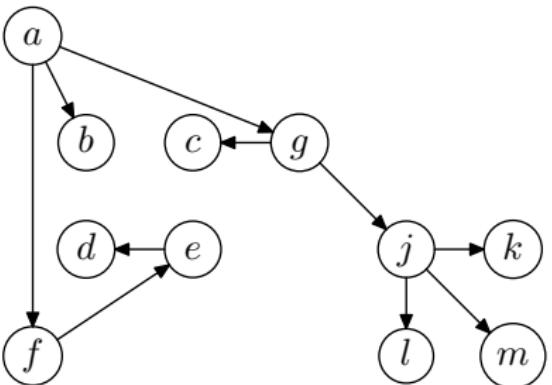
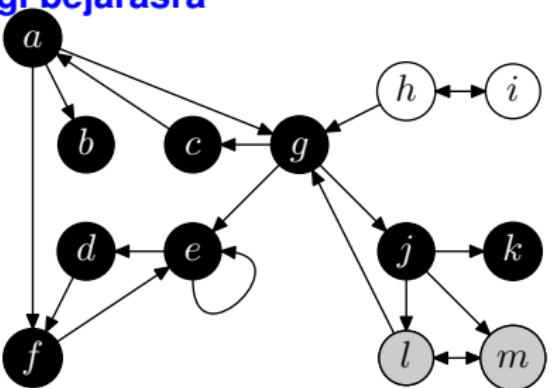
Sor:  $j, d$



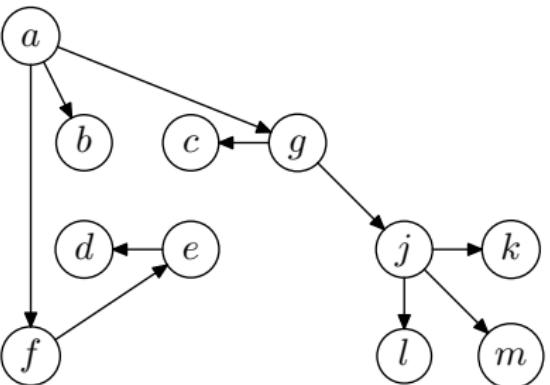
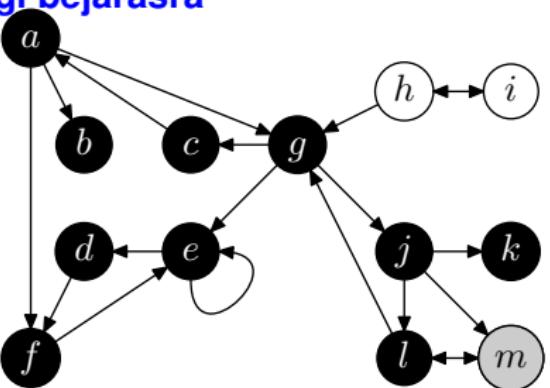
Sor:  $d, k, l, m$



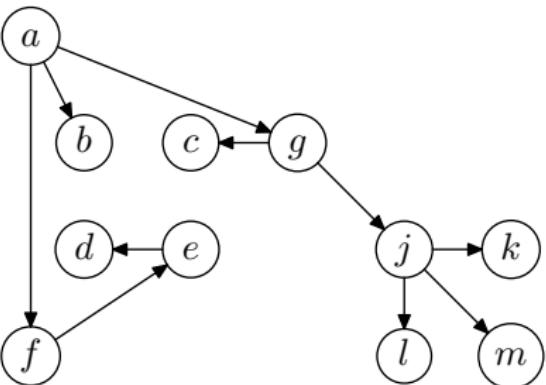
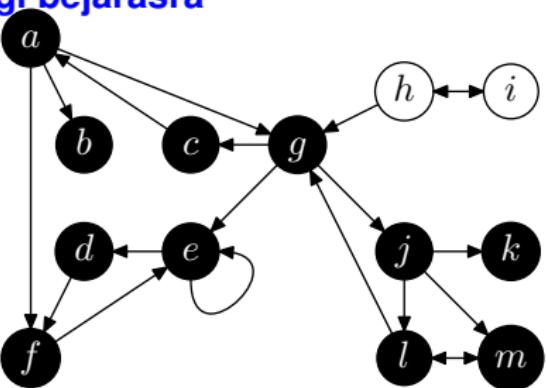
Sor:  $k, l, m$



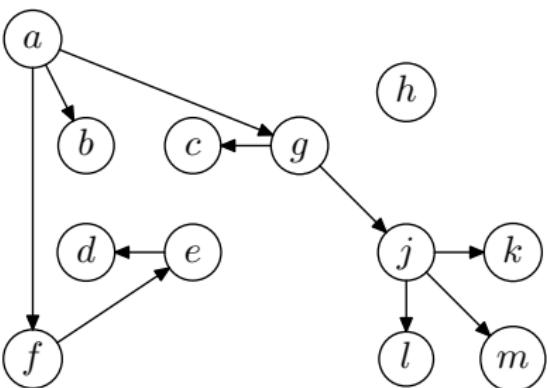
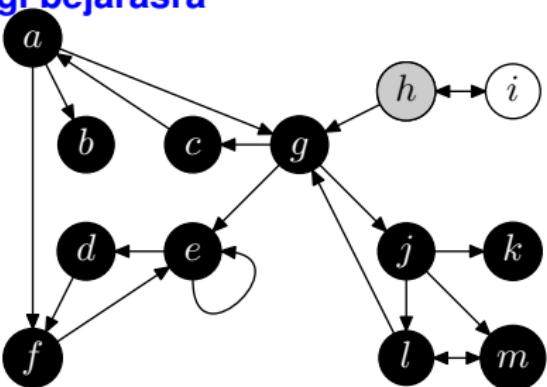
Sor:  $l, m$



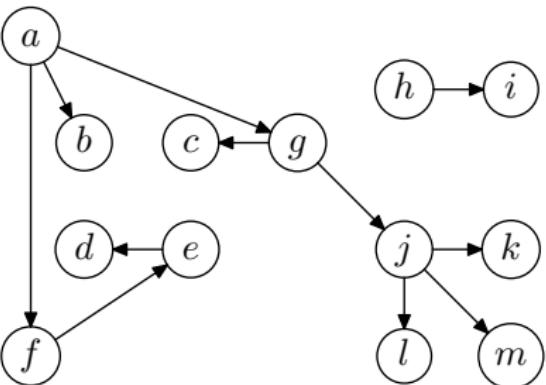
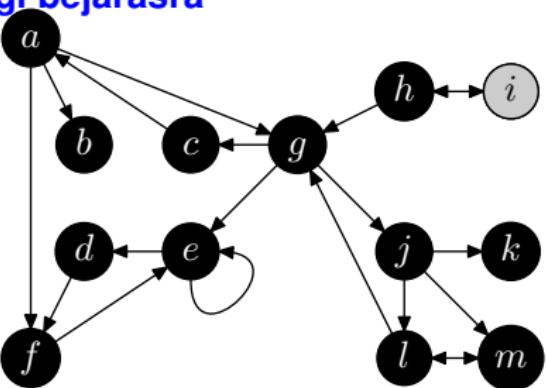
Sor: m



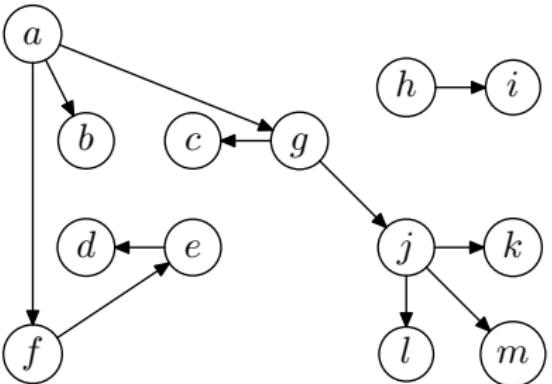
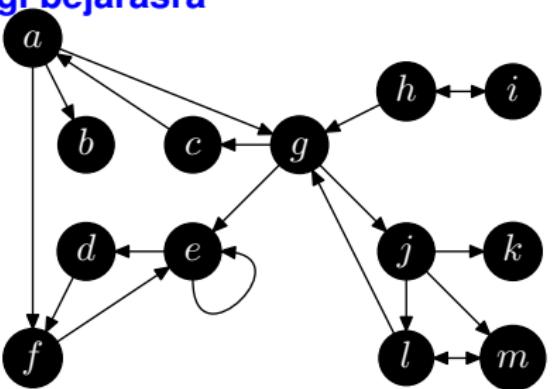
Sor: -



Sor: h



Sor: *i*



Sor: -



## Szélességi Keresés

**procedure** SZÉLESSÉGI\_KERESÉS( $G, s, D, \pi$ )

```

1: for  $i \leftarrow 1$  to méret( $G$ ) do
2:    $SZÍN[i] \leftarrow$  fehér
3:    $D[i] \leftarrow \infty$ 
4:    $\pi[i] \leftarrow 0$ 
5: end for
6:  $SZÍN[s] \leftarrow$  szürke
7:  $D[s] \leftarrow 0$ 
8:  $Q \leftarrow$  új_üres_SOR
9: PUT( $Q, s$ )
10: while NEM_ÜRES( $Q$ ) do
11:    $u \leftarrow GET(Q)$ 
12:   for  $i \leftarrow 1$  to
        méret( $G[u].KÖV$ ) do
13:      $v \leftarrow G[u].KÖV[i]$ 
14:     if  $SZÍN[v] =$  fehér
        then
15:        $SZÍN[v] \leftarrow$  szürke
16:        $D[v] \leftarrow D[u]+1$ 
17:        $\pi[v] \leftarrow u$ 
18:       PUT( $Q, v$ )
19:     end if
20:   end for
21:    $SZÍN[u] \leftarrow$  fekete
22: end while
end procedure
```

# Szélességi Keresés

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hálós adatszerkezet  
(gráf)

```
procedure ÚT_Kiír(G, s, v, π)
1: if v=s then
2:   PRINT(s)
3: else if π[v] = 0 then
4:   PRINT(„Nincs út” s „és” v „között”)
5: else
6:   ÚT_KIÍR(G, s, π[v], π)
7:   PRINT(v)
8: end if
end procedure
```



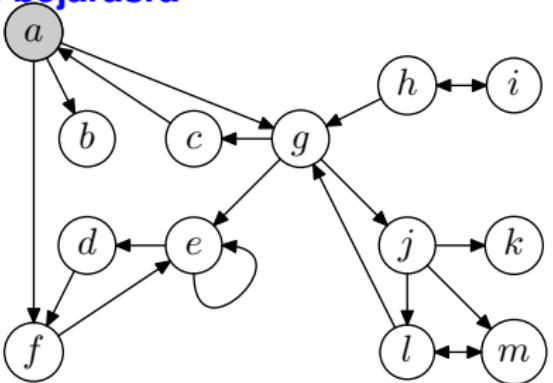
## Mélységi bejárás

### Mélységi bejárás

A bejárás során az adatelemeket fehér, szürke és fekete színűre fogjuk színezni, valamint egy **vermet** fogunk használni a szürke színű adatelemek tárolására. A bejárás során a feldolgozott elemekből feszítőfá(ka)t építünk fel.

- ① Színezzük a gráf adatelemeit fehér színűre, és hozzuk létre az üres vermet.
- ② Ha minden adatelem fekete színű, a bejárás véget ér.
- ③ Ha a verem üres, válasszunk tetszőlegesen egy fehér színű elemet, színezzük szürkére, és helyezzük el a veremben. Ennek az elemnek a feldolgozásakor új feszítőfát fogunk elkezdeni építeni.
- ④ Ha a veremben van elem, vegyük ki a verem első elemét, fehér színű gyermekéit szürkére festve helyezzük el a veremben, majd az adatelemet fessük feketére, és helyezzük el a megfelelő feszítőfában.
- ⑤ Folytassuk az algoritmust a 2. lépéssel.

## Példa mélységi bejárásra

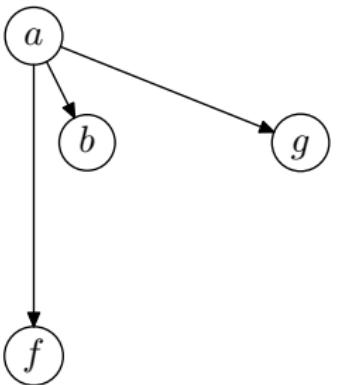
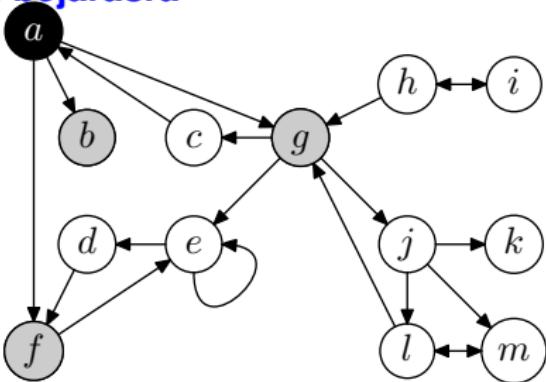


Verem: *a*

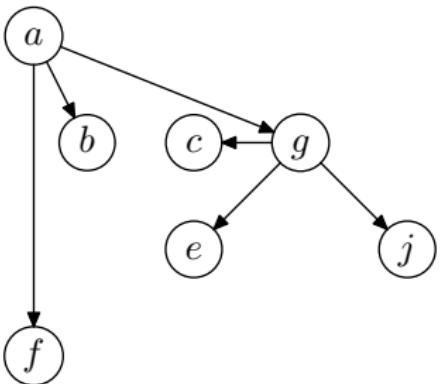
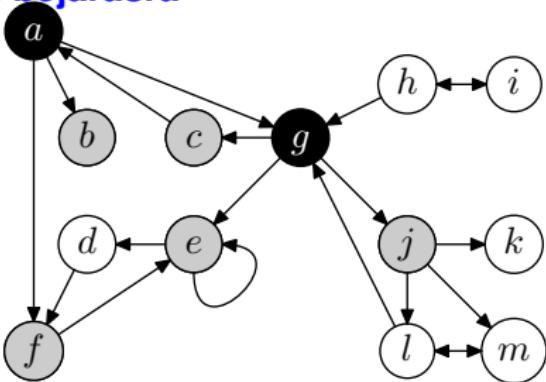
Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



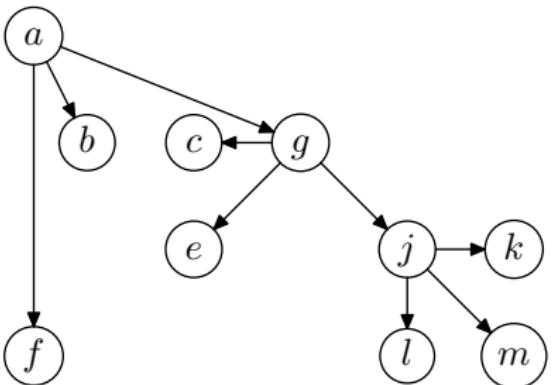
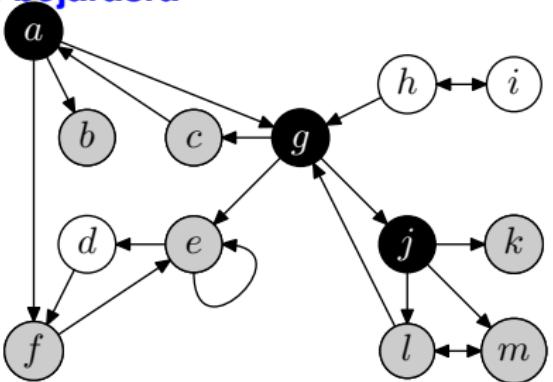
Hálós adatszerkezet  
(gráf)



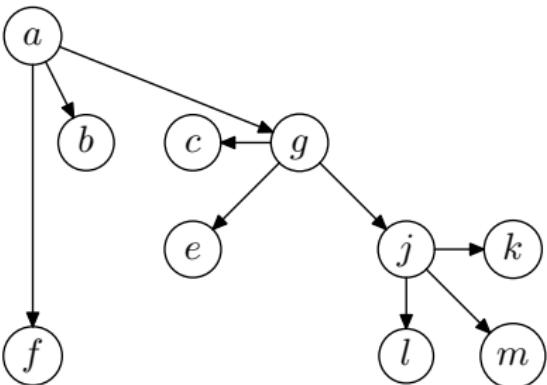
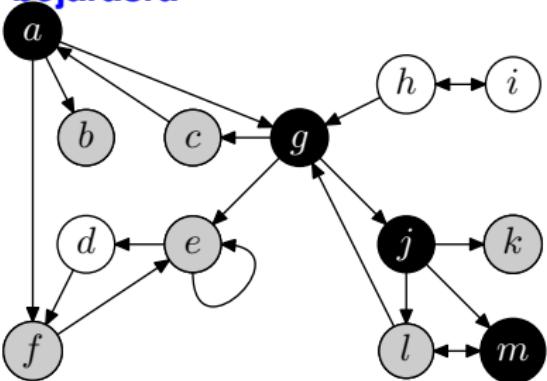
Verem:  $b, f, g$



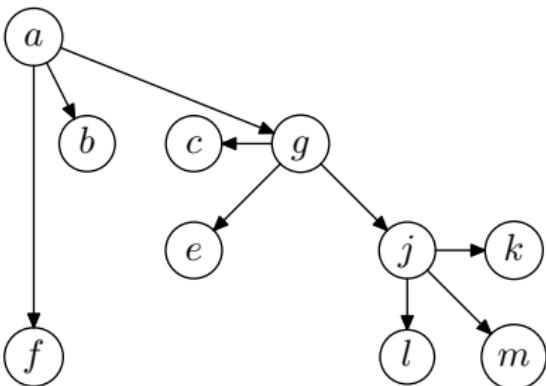
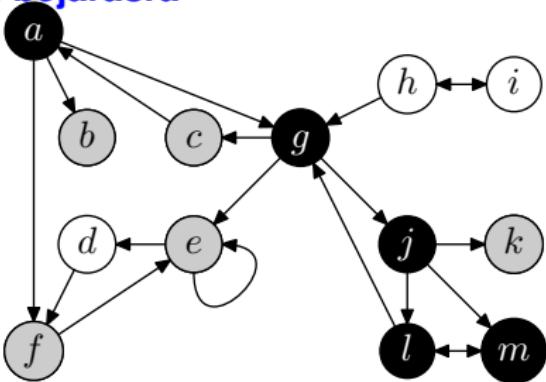
Verem:  $b, f, c, e, j$



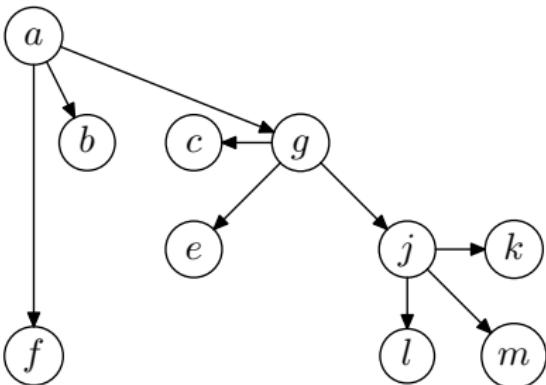
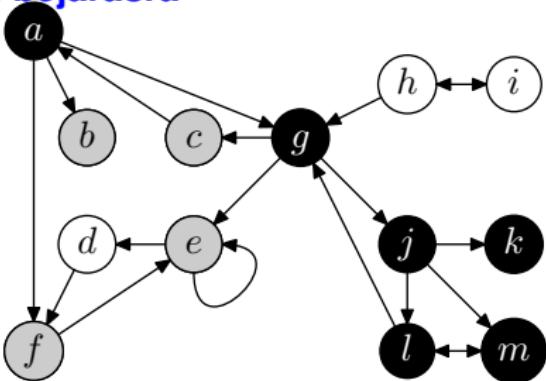
Verem:  $b, f, c, e, k, l, m$



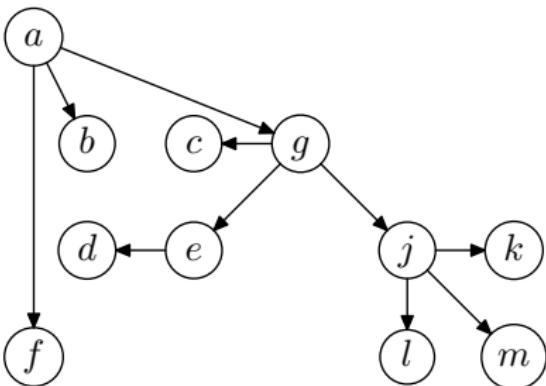
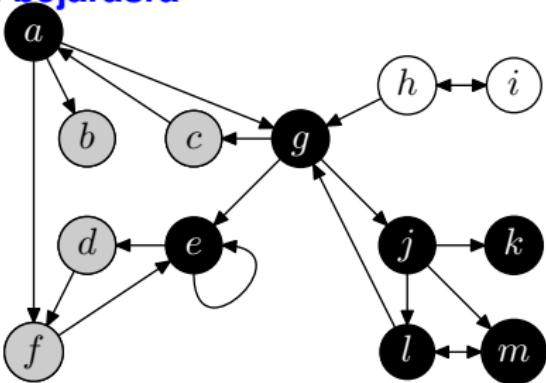
Verem:  $b, f, c, e, k, l$



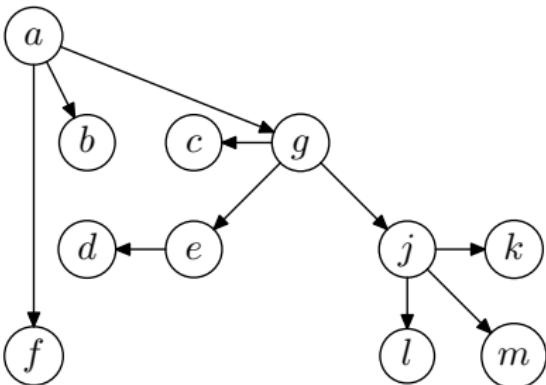
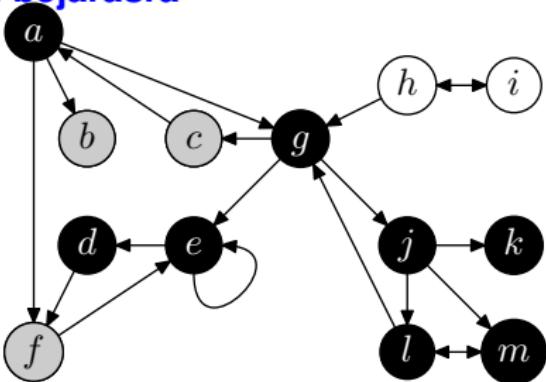
Verem:  $b, f, c, e, k$



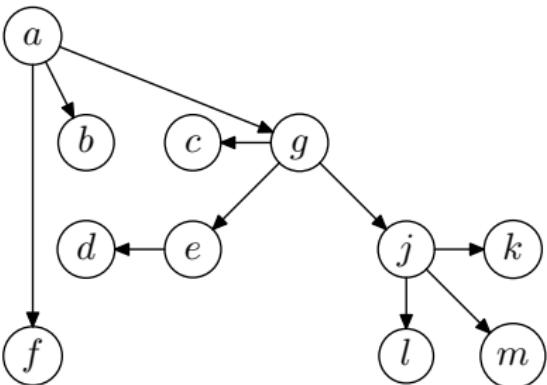
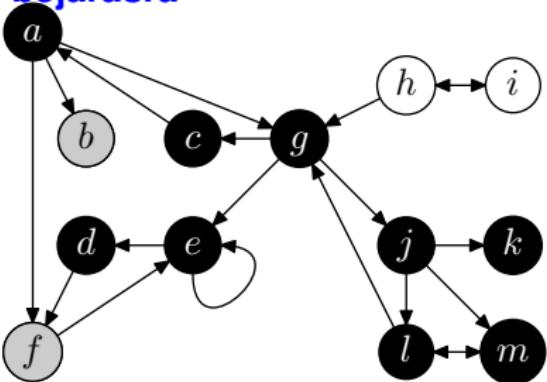
Verem:  $b, f, c, e$



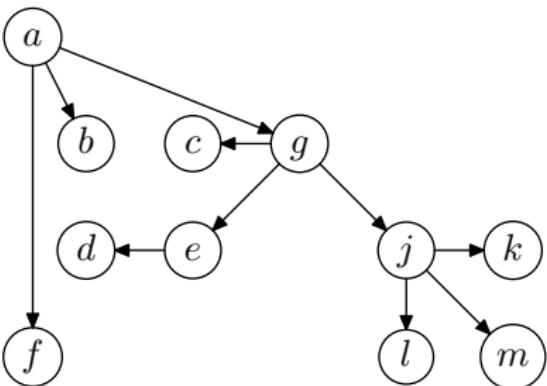
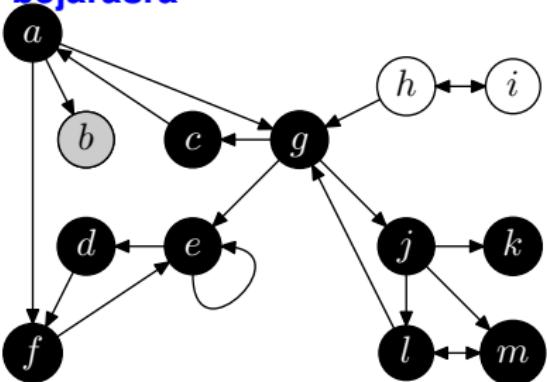
Verem:  $b, f, c, d$



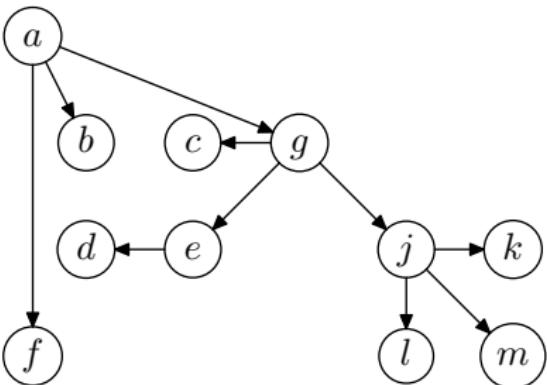
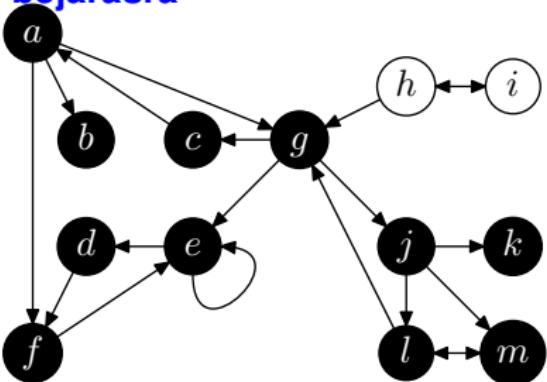
Verem:  $b, f, c$



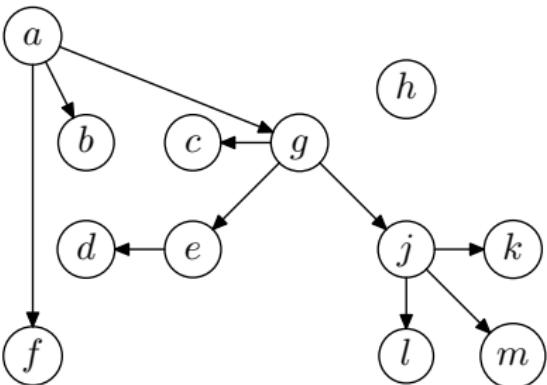
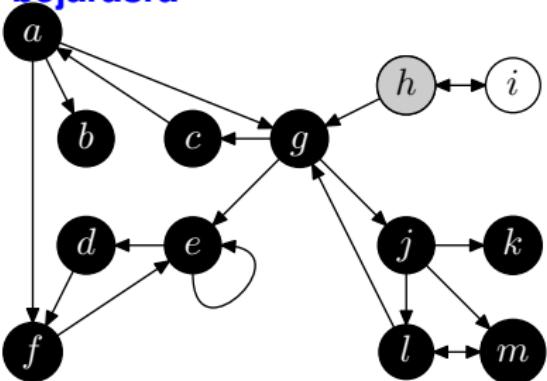
Verem:  $b, f$



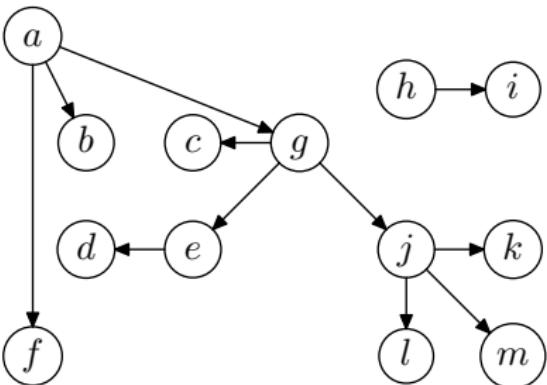
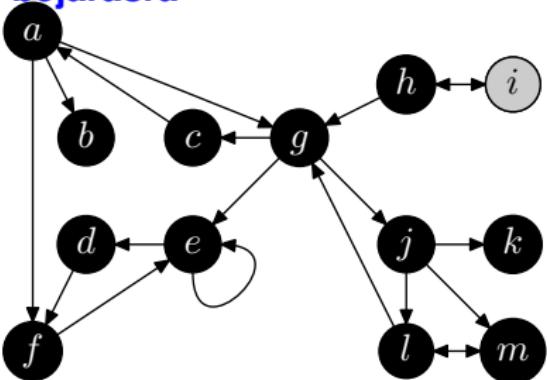
Verem: b



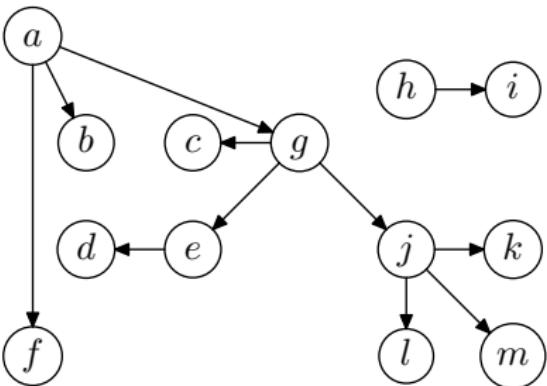
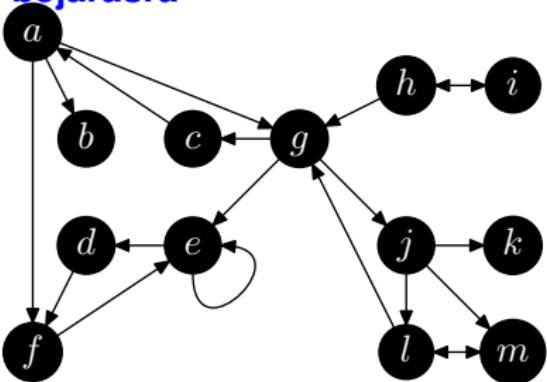
Verem: -



Verem: *h*



Verem: *i*



Verem: –



## Mélységi Keresés

### procedure

MÉLYSÉGI\_KERESÉS(G, s)

```

1: for i  $\leftarrow$  1 to méret(G) do
2:   SZÍN[i]  $\leftarrow$  fehér
3:    $\pi$ [i]  $\leftarrow$  0
4: end for
5: idő  $\leftarrow$  0
6: MK(G, s)
7: for i  $\leftarrow$  1 to méret(G) do
8:   if SZÍN[i] = fehér then
9:     MK(G, i)
10:  end if
11: end for
end procedure
```

-- D[u], F[u],  $\pi$ [u]: ??

### procedure MK(G, u)

```

1: SZÍN[u]  $\leftarrow$  szürke
2: D[u]  $\leftarrow$  idő  $\leftarrow$  idő+1
3: for i  $\leftarrow$  1 to
   méret(G[u].KÖV) do
4:   v  $\leftarrow$  G[u].KÖV[i]
5:   if SZÍN[v] = fehér then
6:      $\pi$ [v]  $\leftarrow$  u
7:     MK(G, v)
8:   end if
9: end for
10: SZÍN[u]  $\leftarrow$  fekete
11: F[u]  $\leftarrow$  idő  $\leftarrow$  idő+1
end procedure
```



## Mélységi Keresés

### procedure

MÉLYSÉGI\_KERESÉS( $G, s$ )

```

1: for  $i \leftarrow 1$  to méret( $G$ ) do
2:    $SZÍN[i] \leftarrow$  fehér
3:    $\pi[i] \leftarrow 0$ 
4: end for
5:  $idő \leftarrow 0$ 
6:  $MK(G, s)$ 
7: for  $i \leftarrow 1$  to méret( $G$ ) do
8:   if  $SZÍN[i] =$  fehér then
9:      $MK(G, i)$ 
10:   end if
11: end for
end procedure
```

--  $D[u], F[u], \pi[u]: ??$

### procedure MK( $G, u$ )

```

1:  $SZÍN[u] \leftarrow$  szürke
2:  $D[u] \leftarrow idő \leftarrow idő+1$ 
3: for  $i \leftarrow 1$  to
   méret( $G[u].KÖV$ ) do
4:    $v \leftarrow G[u].KÖV[i]$ 
5:   if  $SZÍN[v] =$  fehér then
6:      $\pi[v] \leftarrow u$ 
7:      $MK(G, v)$ 
8:   end if
9: end for
10:  $SZÍN[u] \leftarrow$  fekete
11:  $F[u] \leftarrow idő \leftarrow idő+1$ 
end procedure
```



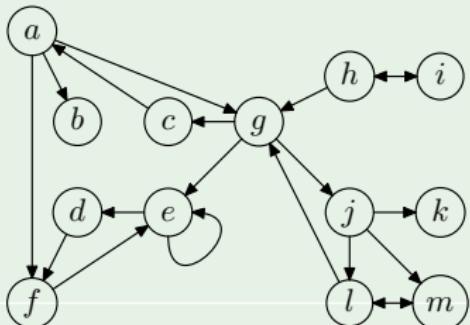
## A gráf reprezentációja

A gráf folytonosan **szomszédsági mátrixszal** (csúcsmátrixszal) reprezentálható. Egy  $n$  adatalemből álló gráf esetén ez egy  $n \times n$ -es logikai mátrix, amelynek a sorait és oszlopait az elemekkel címkézzük. Ha egy sor címkéjében szereplő elem megelőz egy oszlop címkéjében szereplő elemet, akkor az adott sor és oszlop metszetébe 1-et írunk, különben 0-t. A szomszédsági mátrixban egy adott elem megelőzőit az elem oszlopából, a rákövetkezőit pedig az elem sorából tudjuk kiolvasni.

Szétszórt reprezentáció is lehetséges, mégpedig **multilistával**. minden adatalem mellett megjelenik egy mutatótömb, ami annyi elemű, ahány rákövetkezője lehet maximálisan egy elemnek. Ez az érték legfeljebb annyi, ahány elemből a gráf állhat. Ha nem szeretnénk korlátozni a rákövetkezők számát, akkor a mutatókat elhelyezhetjük egy dinamikus vektorban, de felfűzhetjük őket egy egyirányban láncolt listába is.



Hálós adatszerkezet  
 (gráf)



	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>
<i>a</i>	0	1	0	0	0	1	1	0	0	0	0	0	0
<i>b</i>	0	0	0	0	0	0	0	0	0	0	0	0	0
<i>c</i>	1	0	0	0	0	0	0	0	0	0	0	0	0
<i>d</i>	0	0	0	0	0	1	0	0	0	0	0	0	0
<i>e</i>	0	0	0	1	1	0	0	0	0	0	0	0	0
<i>f</i>	0	0	0	0	1	0	0	0	0	0	0	0	0
<i>g</i>	0	0	1	0	1	0	0	0	0	1	0	0	0
<i>h</i>	0	0	0	0	0	0	1	0	1	0	0	0	0
<i>i</i>	0	0	0	0	0	0	0	1	0	0	0	0	0
<i>j</i>	0	0	0	0	0	0	0	0	0	0	1	1	1
<i>k</i>	0	0	0	0	0	0	0	0	0	0	0	0	0
<i>l</i>	0	0	0	0	0	0	1	0	0	0	0	0	1
<i>m</i>	0	0	0	0	0	0	0	0	0	0	1	0	0

## Gráfelméleti kérdések

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Hálós adatszerkezet  
(gráf)

### Gráfelméleti problémák:

- ▶ Annak eldöntése, hogy létezik-e út két adott csomópont között.
- ▶ Legrövidebb út keresése két adott csomópont között.
- ▶ Annak meghatározása, hogy egy adott csomópontból mely más csomópontokba vezet út.
- ▶ Maximális út (a gráf összes csomópontját tartalmazó út) keresése.
- ▶ Körút létezésének eldöntése, a leghosszabb körút keresése.
- ▶ Annak eldöntése, hogy egy gráf részhalmaza-e egy másik gráfnak.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



# 11. előadás

## Párhuzamosság

Több szálra futó algoritmusok, „*logikai*” párhuzamosság

*Adatszerkezetek és algoritmusok* előadás

2018. március 6.

Párhuzamosság  
Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP  
Fibonacci számok  
A többszázú végrehajtás modelllezése  
A párhuzamosítás hatékonysága  
Útemezés  
Párhuzamos ciklusok  
Versenyhelyzet

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

# Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

Párhuzamosság

Kósá Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Párhuzamosság

Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP  
Fibonacci számok  
A többszálú végrehajtás modelllezése  
A párhuzamosítás hatékonysága  
Ütemezés  
Párhuzamos ciklusok  
Versenyhelyzet



## Párhuzamosság

## Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Alapvető koncepciók

Eddig csak „soros” végrehajtású algoritmusokkal foglalkoztunk. De, a környezetünkben szinte elkerülhetetlenül egyre több sokprocesszoros, sokmagos processzor jelenik meg a legkülönbözőbb árárt és teljesítménnyel.

- ▶ A laptopjainkban már többmagos (multicore) processzorok dolgoznak.
  - Ezek a magok közös (shared) memóriát használnak.
- ▶ Szinte minden intézménynek van egy asztali gépekből álló klasztere/gridje, amely a felhasználók számára egyetlen gépnek látszik.
  - Ezek mindegyike saját független memóriát használ, és belső internet kapcsolat biztosítja az egyes gépek közötti feladat és adat megosztást.
- ▶ Az ú.n. szuper-komputerek speciális hardvereket használnak mind a processzorok, mind a hálózati kommunikáció hatékonyságának biztosítására.
  - Gyakran hibrid (hybrid) memória modellel dolgoznak.

Fontos, hogy ki tudjuk használni az ezek adta lehetőségeket. Ehhez meg kell teremteni az utasítások, programrészletek párhuzamos végrehajtásának vezérléséhez szükséges módszereket.



## Párhuzamosság

## Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Alapvető koncepciók

A párhuzamos működésre képes számítógépek valójában már évtizedek óta részei a világunknak.

Miközben a soros számolások „RAM” modellje történetileg gyorsan kialakult, a párhuzamos programozás terén sok eltérő modell működik a mai napig.

Ez elsősorban arra vezethető vissza, hogy a gyártók nem tudtak megegyezni a szabványosításról.

Példának okáért a memória területén egyaránt elterjedten használják

- ▶ a közös/megosztott (shared) memória és
- ▶ a elosztott (distributed) memória

modelleket.

Az asztali gépek és laptopok területén egyre nagyobb teret nyerő közös memória modell kapcsán talán ez lesz a végső befutó, de ezt csak az idő fogja eldöntení.



## Párhuzamosság

Alapvető koncepciók

**Statikus szálak**

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# Alapvető koncepciók

## Statikus szálak (static threads)

Az egyik elterjedt modell a közös memóriát használó processzorok esetére az ú.n. statikus szálak használata.

Ez szoftveresen absztrakcióval biztosít virtuális processzorokat vagy szálakat (threads), amelyek

- ▶ ugyanazt a memóriát érik el speciális utasítások nélkül,
- ▶ külön programszámlálójuk van,
- ▶ és egymástól függetlenül dolgoznak.

Ezeket a szálakat az operációs rendszer rendeli hozzá az egyes programokhoz, és kapcsolja át másokhoz igény szerint.

Bár megengedi a programozónak új szálak létrehozását, régiek lezárását,

- ▶ ez rendszerint egy lassú folyamat,
- ▶ és végül tipikusan a szálak végig lekötik az igényelt processzorokat

Ezért hívják ezt a módszert statikusnak.



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

## Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonyisége

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# Alapvető koncepciók

## Dinamikus szálak (Dynamic Multithreaded Programming – DMP)

- ▶ Közös memóriájú párhuzamos számítógépek közvetlen programozása a statikus szálak modelljében nehéz, és könnyen elhibázható dolog.
- ▶ Az egyik legnagyobb probléma, a feladatok felosztása közel azonos erőforrás igényű, várhatóan közel azonos ideig futó részfeladatokra.
- ▶ A legegyszerűbb estektől eltekintve komplex kommunikációs protokollokkal lehet megoldani a processzorok kiegyszűlyozott használatát.
- ▶ Ezek a körülmények vezettek versengő platformok (concurrency platforms) megalkotásához amelyek különböző szoftveres rétegeket biztosítanak az erőforrások eléréséhez, menedzselésehez.



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

## Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## ► Szálak: virtuális processzorok szoftveres absztrakciója

- A szálak közösen használják ugyanazt a memóriát
- Saját vermük és programszámlálójuk van
- Egymástól függetlenül hajtják végre a feladataikat
- Az operációs rendszer
  - tölti be a szálakat a processzorokba,
  - hajtatta végre a feladatokat,
  - vált más szálakra, ha szükséges

## ► A dinamikus többszálú programozási modell (DMP) megengedi a programozónak, hogy

- leírja a logikai párhuzamosságot
- és ne törődjön a kommunikációs protokollok vagy a betöltési egyensúly problémáival.

## ► A következőkben azt fogjuk tanulmányozni,

- hogyan írhatunk többszálú algoritmusokat a modell keretein belül
- és hogyan segítik a hatékony számolásokat a háttérben meghúzódó rendszer szintű támogatások



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Dinamikus többszálúság

- ▶ A pszeudo-nyelvünket csupán három utasítással fogjuk kiegészíteni:

- **parallel** (párhuzamos)
- **spawn** (ivadék, leszármazott processzus)
- **sync** (szinkronizálás)

Amennyiben töröljük ezeket a programból, egy helyesen működő „soros” programot kapunk.

- ▶ Elméleti szempontból egy tiszta és egyszerű analizálási lehetőséget kapunk a párhuzamosságról.
- ▶ A legtöbb beágyazottságra épülő többszálú algoritmus természetes módon kötődik az „oszd meg és uralkodj” elvéhez. Aminként ezek analízise rekurziókra vezetett, hasonlóan alakul ez itt is.
- ▶ Egyre több „versengő platform” támogatja a DMP modellt: Cilk, Cilk++, OpenMP, Task Parallel Library, Threading Building Blocks



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# Fibonacci számok

Mint azt „ mindenki tudja” a Fibonacci számok az alábbi rekurzióval definiálhatóak:

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_i &= F_{i-1} + F_{i-2} \quad \text{ha } i > 1. \end{aligned}$$

Ha számolnunk kell:

```
function Fib(n)
```

```
1: if n ≤ 1 then
```

```
2:   return n
```

```
3: end if
```

```
4: x ← Fib(n-1)
```

```
5: y ← Fib(n-2)
```

```
6: return x+y
```

```
end function
```

$$T(n) = T(n-1) + T(n-2) + \Theta(1).$$

$$T(n) = \Theta(F_n).$$



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszűrű végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Fibonacci számok

Mint azt „ mindenki tudja” a Fibonacci számok az alábbi rekurzióval definiálhatóak:

$$F_0 = 0,$$

$$F_1 = 1,$$

$$F_i = F_{i-1} + F_{i-2} \quad \text{ha } i > 1.$$

Ha számolnunk kell:

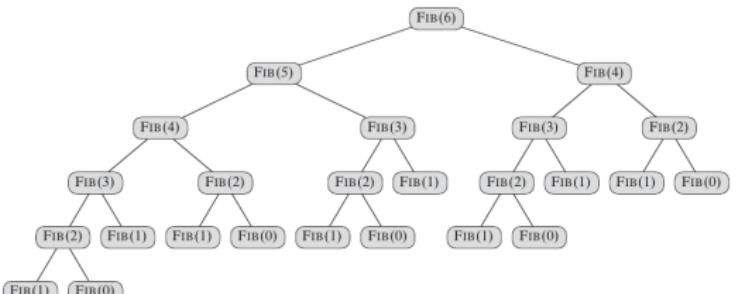
**function** FIB(n)
 1: **if** n ≤ 1 **then**

 2:   **return** n

 3: **end if**

4: x ← FIB(n-1)

5: y ← FIB(n-2)

 6: **return** x+y


$$T(n)=T(n-1)+T(n-2)+\Theta(1).$$

**end function**

$$T(n)=\Theta(F_n).$$



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszűrű végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Fibonacci számok

Mint azt „ mindenki tudja” a Fibonacci számok az alábbi rekurzióval definiálhatóak:

$$F_0 = 0,$$

$$F_1 = 1,$$

$$F_i = F_{i-1} + F_{i-2} \quad \text{ha } i > 1.$$

Ha számolnunk kell:

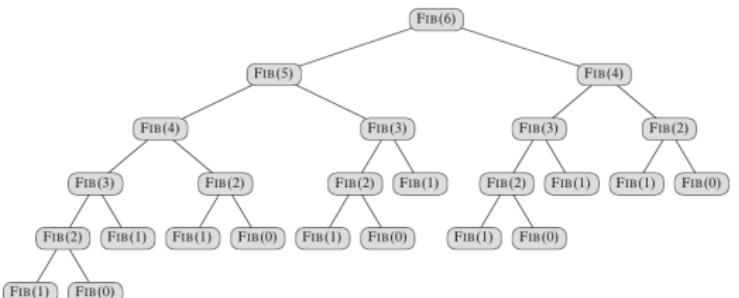
**function** FIB(n)
 1: **if** n ≤ 1 **then**

 2:   **return** n

 3: **end if**

4: x ← FIB(n-1)

5: y ← FIB(n-2)

 6: **return** x+y


$$T(n)=T(n-1)+T(n-2)+\Theta(1).$$

**end function**

$$T(n)=\Theta(F_n).$$



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## Fibonacci számok

Vegyük észre, hogy a 4. és 5. sorban szereplő függvényhívások egymástól függetlenek!

Ezt kihasználva párhuzamosíthatjuk az algoritmust a beágyazás **spawn** és **sync** kulcsszavaival:

```
function P_FIB(n)
```

- 1: **if** n ≤1 **then**

- 2:   **return** n

- 3: **end if**

- 4: x ← **spawn** P\_FIB(n-1)

- 5: y ← P\_FIB(n-2)

- 6: **sync**

- 7: **return** x+y

```
end function
```

- ▶ A beágyazás (nesting) a 4. sorban, a **spawn** kulcsszóval van aktiválva. Ennek hatására a „szülő szál” nem várja meg a P\_FIB(n-1) függvényhívás kiértékelését, amelyet átadott a „gyermek szálnak”, hanem azonnal elkezdi a 5. sor kiértékelését.
- ▶ A **sync** utasítás hatására az összes külön indított szálat össze fogja várni a vezérlő.
  - Implicit módon a **return** is ezt tenné.
  - x vagy y használata a **sync** előtt nem biztonságos
- ▶ Hány szál fog itt létrejönni?



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# Fibonacci számok

Vegyük észre, hogy a 4. és 5. sorban szereplő függvényhívások egymástól függetlenek!

Ezt kihasználva párhuzamosíthatjuk az algoritmust a beágyazás **spawn** és **sync** kulcsszavaival:

```
function P_FIB(n)
```

- 1: **if** n ≤1 **then**

- 2:   **return** n

- 3: **end if**

- 4: x ← **spawn** P\_FIB(n-1)

- 5: y ← P\_FIB(n-2)

- 6: **sync**

- 7: **return** x+y

```
end function
```

- ▶ A beágyazás (nesting) a 4. sorban, a **spawn** kulcsszóval van aktiválva. Ennek hatására a „szülő szál” nem várja meg a P\_FIB(n-1) függvényhívás kiértékelését, amelyet átadott a „gyermek szálnak”, hanem azonnal elkezdi a 5. sor kiértékelését.
- ▶ A **sync** utasítás hatására az összes külön indított szálat össze fogja várni a vezérlő.
  - Implicit módon a **return** is ezt tenné.
  - x vagy y használata a **sync** előtt nem biztonságos

▶ Hány szál fog itt létrejönni?



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

## Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# Fibonacci számok

Vegyük észre, hogy a 4. és 5. sorban szereplő függvényhívások egymástól függetlenek!

Ezt kihasználva párhuzamosíthatjuk az algoritmust a beágyazás **spawn** és **sync** kulcsszavaival:

```
function P_FIB(n)
```

- 1: **if** n ≤1 **then**

- 2:   **return** n

- 3: **end if**

- 4: x ← **spawn** P\_FIB(n-1)

- 5: y ← P\_FIB(n-2)

- 6: **sync**

- 7: **return** x+y

```
end function
```

- ▶ A beágyazás (nesting) a 4. sorban, a **spawn** kulcsszóval van aktiválva. Ennek hatására a „szülő szál” nem várja meg a P\_FIB(n-1) függvényhívás kiértékelését, amelyet átadott a „gyermek szálnak”, hanem azonnal elkezdi a 5. sor kiértékelését.
- ▶ A **sync** utasítás hatására az összes külön indított szálat össze fogja várni a vezérlő.
  - Implicit módon a **return** is ezt tenné.
  - x vagy y használata a **sync** előtt nem biztonságos
- ▶ Hány szál fog itt létrejönni?



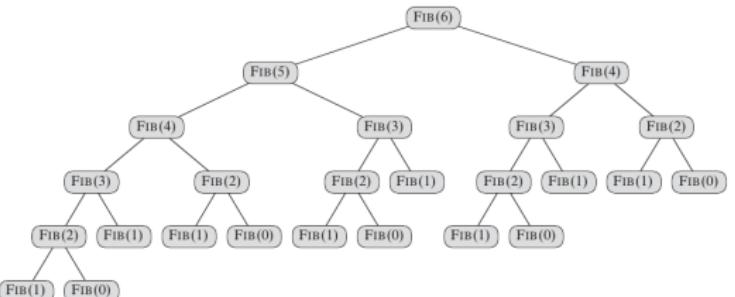
# Beágyzás

## ► Hány szál fog itt létrejönni?

- Ha  $n > 1$ , akkor minden szülő, minden gyermek újabb szálakat fog nyitni ...
- Belátható, hogy  $n > 1$  esetben  $F_{n+1}$  szál indul összesen.

## ► De mennyi fog közülük tényleg egyidőben futni?

- Akár az összes,
- vagy minden csak egy.
  - A spawn kulcsszó nem jelent kötelező párhuzamosságot,
  - csak a lehetőséget teremti meg a logikai párhuzamosság jelzésével.
  - A döntés az ütemező (scheduler) kezében van.
- Ha korlátlan számú, egymással ekvivalens processzor áll a rendelkezésünkre, és az ütemező minden időben a rendelkezésünkre is bocsátja őket, akkor 1 és kb.  $2^{n/2+1}$  közötti számú.



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszínű végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

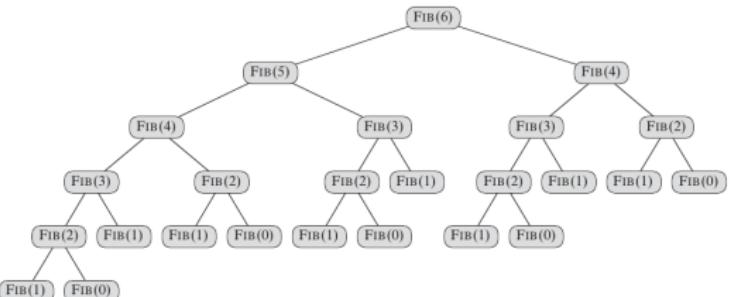
# Beágyzás

## ► Hány szál fog itt létrejönni?

- Ha  $n > 1$ , akkor minden szülő, minden gyermek újabb szálakat fog nyitni ...
- Belátható, hogy  $n > 1$  esetben  $F_{n+1}$  szál indul összesen.

## ► De mennyi fog közülük tényleg egyidőben futni?

- Akár az összes,
- vagy mindenkor csak egy.
  - A **spawn** kulcsszó nem jelent kötelező párhuzamosságot,
  - csak a lehetőséget teremti meg a logikai párhuzamosság jelzésével.
  - A döntés az ütemező (scheduler) kezében van.
- Ha korlátlan számú, egymással ekvivalens processzor áll a rendelkezésünkre, és az ütemező minden időben a rendelkezésünkre is bocsátja őket, akkor 1 és kb.  $2^{n/2+1}$  közötti számú.





## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

## A többszálú végrehajtás modellezése

A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

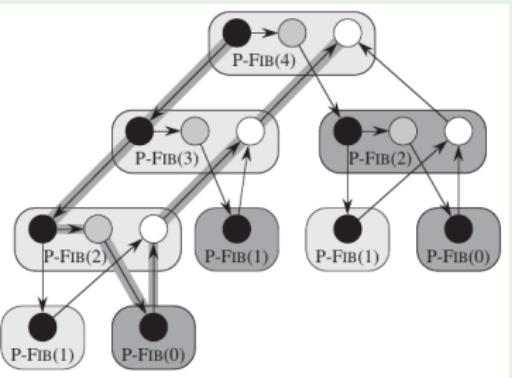
Versenyhelyzet

## A többszálú végrehajtás modellezése

Hasznos úgy tekinteni a többszálú számolásokra, amit modellezhetünk egy irányított, körmentes gráffal:

## G=(V, E) – számítási dag. (Directed Acyclic Graph)

- ▶ A csúcsok egy egy utasításnak felelnek meg
- ▶ Az élek pedig az utasítások közötti függőségeket mutatják.
  - A nyíl mutatja, melyik utasítást kell előbb végrehajtani.
  - Speciálisan a vízszintes nyíl...
- ▶ **spawn**
  - adott csúcsból több nyíl indul
- ▶ **sync**
  - ide több is mutat
- ▶ többi utasítás: *strand*



- ▶ Fekete – szürke – fehér
- ▶ Ha van irányított út két csúcs között, akkor azok (logikailag) sorosak
- ▶ Ha nincs, akkor párhuzamosak



A modellezés során feltesszük, hogy egy „ideális párhuzamos gépen” futtathatunk:

- ▶ Soros végrehajtás szempontjából konzisztens memória
  - az összes processzor által elérhető közös memória,
  - ami egyszerre több írást, olvasást tud kiszolgálni egyidejűleg
  - és az eredmény meg fog egyezni azzal, amit ugyanezen író, olvasó utasítások valamelyen sorrendben történő soros végrehajtásával kapnánk
    - Ha egyszerre 10 utasítást hajtunk végre, az eredmény meg fog egyezni a  $10!$  lehetséges sorrend valamelyike után várható eredménnyel, de nem tudjuk melyikével.
    - Az ütemezőtől függően, ugyanazon algoritmus kétszeri végrehajtása az egyedi utasítások eltérő sorrendjét eredményezheti, még az is eltérhet, hogy mely utasítások futnak logikailag egyidejűleg
- ▶ Azonos teljesítményű processzorok
- ▶ Az ütemezés időigénye elhanyagolható

## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded

Programming – DMP

Fibonacci számok

A többszálú végrehajtás  
modellezése

A párhuzamosítás  
hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

## A párhuzamosítás hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## A párhuzamosítás hatékonysága

A hatékonyság méréséhez két fontos fogalmat fogunk használni

**work** (munka), A számolások összes utasításának ideje/száma. Magyarul, ha egyetlen processzort használnánk, ennyi ideig tartana, ennyi utasítást hajtana végre a program futása.

**span** (feszítés) A leghosszabb (a kritikus) irányított út hossza a gráfban – a párhuzamos program minimális futási ideje (azonos végrehajtási időt feltételezve minden csúcshoz).

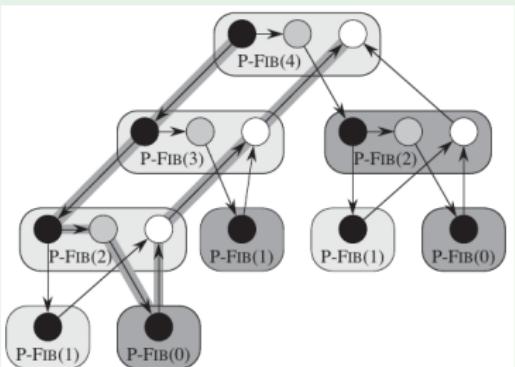
## Itt például:

$$\text{work} = 17$$

$$\text{span} = 8$$

$$(\text{work} = \Theta(F_n))$$

$$(\text{span} = 2^n)$$





## Párhuzamosság

Alapvető koncepciók  
 Statikus szálak  
 Dinamikus szálak  
 Dynamic Multithreaded Programming – DMP  
 Fibonacci számok  
 A többszázú végrehajtás modelllezése  
 A párhuzamosítás hatékonysága

Útemezés  
 Párhuzamos ciklusok  
 Versenyhelyzet

# A párhuzamosítás hatékonysága

A hatékonyság méréséhez két fontos fogalmat fogunk használni

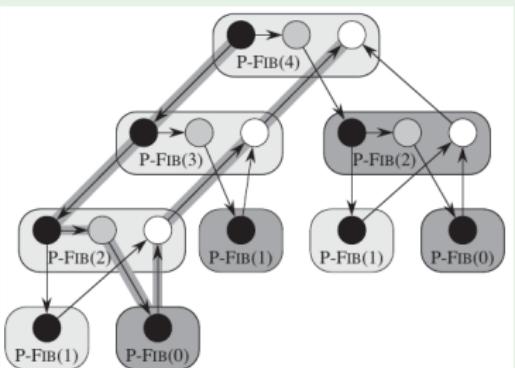
**work** (munka), A számolások összes utasításának ideje/száma. Magyarul, ha egyetlen processzort használnánk, ennyi ideig tartana, ennyi utasítást hajtana végre a program futása.

**span** (feszítés) A leghosszabb (a kritikus) irányított út hossza a gráfban – a párhuzamos program minimális futási ideje (azonos végrehajtási időt feltételezve minden csúcshoz).

## Itt például:

work = 17  
 span = 8

( $\text{work} = \Theta(F_n)$ )  
 ( $\text{span} = 2^n$ )





## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás  
hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

## A párhuzamosítás hatékonysága

- ▶ A tényleges futási idő „tökéletes ütemezés” mellett is függ a rendelkezésre álló processzorok számától.
- ▶ Ha  $P$  darab processzort tud kiosztani a mi programunk számára az ütemező, akkor a futási időt jelölje  $T_P$ .
- ▶ Néhány egyszerű összefüggés:
  - $T_1 = \text{work}$
  - $T_\infty = \text{span}$
  - Work law:  $T_P \geq T_1/P$
  - Span law:  $T_P \geq T_\infty$
- ▶ A párhuzamos program gyorsulását (speedup)  $P$  processzoron történő végrehajtás estére a  $T_1/T_P$  aránnyal definiálhatjuk.
  - A „munka törvény” alapján  $T_1/T_P \leq P$ .
    - Ha  $T_1/T_P = \Theta(P)$ , akkor lineáris gyorsulásról beszélünk,
    - ha  $T_1/T_P = P$ , akkor azt mondjuk, tökéletes lineáris gyorsulásunk van.
  - A  $T_1/T_\infty$  arányt az algoritmusunk párhuzamosságának (parallelism) nevezzük. Ez az arány
    - Felső határt jelent a gyorsulásra nézve
    - Felső határt jelent a processzorok számára, amivel még lineáris gyorsulás érhető el (ha  $P > T_1/T_\infty \implies P > T_1/T_P$ )



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás  
hatékonysága

Útemezés

Párhuzamos ciklusok

Versenyhelyzet

# A párhuzamosítás hatékonysága

- ▶ Párhuzamos „eltérés” (parallel slackness) legyen a párhuzamosság és a rendszerben elérhető processzorok számának aránya:  $(T_1 / T_\infty) / P = T_1 / (PT_\infty)$
- ▶ Ha ez kisebb 1-nél, nem is várhatunk tökéletes lineáris felgyorsulást:

$$T_1 / (PT_\infty) < 1 \implies T_1 / T_P \leq T_1 / T_\infty < P$$

- ▶ Ha viszont a párhuzamos eltérés nagyobb 1-nél, akkor a gyorsulást a munka törvény szerint a processzorok száma fogja korlátozni.
- ▶ Ha a párhuzamos eltérés lényegesen meghaladja az 1-et, akkor egy jó ütemező egyre közelebb tud kerülni a tökéletes lineáris gyorsuláshoz.

- ▶ A hatékony működés nem csak a munka és a feszítés minimalizálásától függ. Fontos, hogy a dag egyes csúcsai megfelelő hatékonysággal legyenek hozzárendelve az elérhető processzorokhoz.
- ▶ Bár a gyakorlatban többnyire az ütemező az egyes „strand”-okat statikus szálakhoz rendeli hozzá, és az operációs rendszer ad processzort a szálaknak, mi úgy fogjuk elképzelni, mintha az ütemező közvetlenül processzorokra osztaná ki az egyes csúcsokhoz tartozó utasításokat.
- ▶ Mivel az ütemező működésére a programozónak korlátozott ráhatása van (mi azt fogjuk feltételezni, hogy semmilyen), így az ütemező semmit nem tud arról, mi lenne jó a programunk hatékony végrehajtása érdekében, így „on-line” kell döntéseket hoznia.
- ▶ Léteznek bizonyítottan hatékony „elosztott” (distributed) ütemezők, de ezek elemzése bonyolult.
- ▶ Mi egy központosított (centralized) ütemezőt fogunk megvizsgálni, ami minden időben ismeri az aktuális globális állapotát a számolásainknak.



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

## Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

## Ütemezés (Scheduling)

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



- ▶ A centralizáltakon belül is mi egy egyszerűnek tekinthető változatot fogunk vizsgálni.
- ▶ A mohó (greedy) ütemező annyi processzort rendel hozzá szálakhoz minden időpillanatban, amennyi éppen elérhető. (Nem tervez semmilyen szempontból.)
- ▶ Ha egy adott időlépésben legalább  $P$  szál van előkészítve a futásra, akkor azt mondjuk az adott időlépés teljes (complete step)
  - Ilyenkor a mohó ütemező kiosztja az összes processzort valamelyik  $P$  darab szálnak az előkészítettek közül.
    - De hát mi lehetne ezzel a baj?
- ▶ Ha adott időben  $P$ -nél kevesebb szál van előkészítve, akkor nem-teljes időlépésről (incomplete step) beszélünk, és a mohó ütemező minden strand-hoz hozzárendel egy saját processzort.
- ▶ A munka ill. a feszítési törvény alapján van két alsó korlátunk a futási időre:
  - $T_P \geq T_1/P$
  - $T_P \geq T_\infty$
- ▶ Mohó ütemezés estén belátható, hogy  $T_P \leq T_1/P + T_\infty$

Párhuzamosság

Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP

Fibonacci számok  
A többszázú végrehajtás modelllezése  
A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok  
Versenyhelyzet

## Theorem (Textbook: 27.1)

Egy  $P$  processzoros ideális párhuzamos gépen a mohó ütemező a  $T_1$  munkát tartalmazó,  $T_\infty$  feszítésű programot legfeljebb  $T_P \leq \lfloor T_1/P \rfloor + T_\infty$  idő alatt hajtja végre.

Kós Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Bizonyítás.

- ▶ Teljes lépések száma nem lehet nagyobb  $\lfloor T_1/P \rfloor$ -nél.
- ▶ Nem-teljes lépések esetén
  - Jelölje
    - $G$ , a számítási dag-ot,
    - $G'$ , valamely nem-teljes lépés kezdeténél még végrehajtandó utasítások részgráfját, és
    - $G''$ , az ugyanezen nem-teljes lépés után még végrehajtandó utasítások részgráfját.
  - Mivel a  $G'$ -t feszítő irányított út első utasítása is végrehajtásra kerül a nem-teljes lépésekben, így a  $G''$ -t feszítő kritikus út feltétlenül csökken 1-el.

Más szóval, legfeljebb  $T_\infty$  nem-teljes lépésünk lesz.

- ▶ Innen:  $T_P \leq \lfloor T_1/P \rfloor + T_\infty \leq T_1/P + T_\infty$

## Párhuzamosság

Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP  
Fibonacci számok  
A többszálú végrehajtás modelllezése  
A párhuzamosítás hatékonysága

## Ütemezés

Párhuzamos ciklusok  
Versenyhelyzet



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

## Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

## Ütemezés (Scheduling)

## Corollary (Textbook: 27.2)

Egy  $P$  processzoros ideális párhuzamos gépen a mohó ütemező legfeljebb egy kettes faktorral lassabb az elérhető optimális sebességnél.

## Bizonyítás.

Legyen

- ▶  $T_1$  a program munkája
- ▶  $T_\infty$  a program feszítése
- ▶  $T_P^*$  az az idő, amely alatt egy tökéletes ütemező  $P$  processzoron végre tudná hajtani a programot.

A munka és feszítési törvények alapján tudjuk, hogy

$$T_P^* \geq \text{MAX}((T_1/P, T_\infty))$$

A 27.1-es téTEL alapján:

$$T_P \leq T_1/P + T_\infty \leq 2 \cdot \text{MAX}((T_1/P, T_\infty)) \leq 2 \cdot T_P^*$$



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

## Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

## Ütemezés (Scheduling)

## Corollary (Textbook: 27.3)

Egy  $P$  processzoros ideális párhuzamos gépen a  $T_1$  munkát tartalmazó,  $T_\infty$  feszítésű program estén,

ha  $P \ll T_1/T_\infty$ , akkor a mohó ütemező által elért gyorsulás megközelítőleg  $P$  lesz:

$$T_1/T_P \approx P.$$

## Bizonyítás.

A feltételünk ( $P \ll T_1/T_\infty$ ) egyenes következménye, hogy  $T_\infty \ll T_1/P$ . Ezt felhasználva a 27.1-es téTELben:

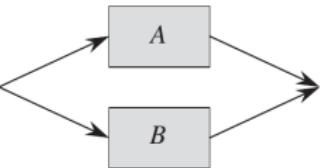
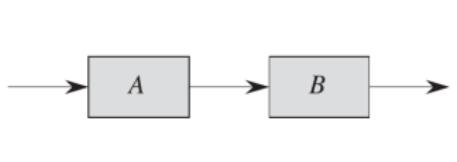
$$\begin{aligned} T_P &\leq T_1/P + T_\infty \\ &\approx T_1/P \end{aligned}$$

ahonnan egyszerű átrendezéssel kapjuk:

$$T_1/T_P \approx P$$

# A „párhuzamos” Fibonacci hatékonysága

Párhuzamosság



Work:  $T_1(A \cup B) = T_1(A) + T_1(B)$

Work:  $T_1(A \cup B) = T_1(A) + T_1(B)$

Span:  $T_\infty(A \cup B) = T_\infty(A) + T_\infty(B)$

Span:  $T_\infty(A \cup B) = \max(T_\infty(A), T_\infty(B))$

- ▶  $T_1(n) = T_1(n - 1) + T_1(n - 2) + \Theta(1) = \Theta(F_n) = \Theta(\phi^n)$
- ▶  $T_\infty(n) = \text{Max}(T_\infty(n - 1), T_\infty(n - 2)) + \Theta(1) = T_\infty(n - 1) + \Theta(1) = \Theta(n)$
- ▶ A párhuzamosság:  $T_1(n)/T_\infty(n) = \Theta(\phi^n/n)$ , ami drasztikusan növekszik  $n$ -el.
- ▶ Így még a legkomolyabb párhuzamos gépek esetén is –nem túl nagy  $n$ -ekre is– közel tökéletes lineáris gyorsulás érhető el a gyorsan növekvő párhuzamos eltérés  $((T_1(n)/T_\infty(n))/P)$  miatt.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Párhuzamosság

Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszűrű végrehajtás modelllezése

A párhuzamosítás hatékonysága

## Útemezés

Párhuzamos ciklusok  
Versenyhelyzet

- Bár a **spawn** és **sync** utasításokkal is dolgozhatnánk, kényelmesebb, ha egy új programozói eszközt vetünk be.
- Ez az új eszköz a **parallel** kulcsszó lesz, amit a **for** ciklusok elé írhatunk.

Tekintsük az  $n \times n$  méretű  $A = (a_{ij})$  mátrix és az  $n$  méretű  $X = (x_i)$  vektor összeszorzásának feladatát:  $y_i = \sum_{j=1}^n a_{ij} \cdot x_j$ .

**function** MAT\_VEC(A, X, n)

```
1: parallel for i ← 1 to n do
2:   Y[i] ← 0
3: end for
4: parallel for i ← 1 to n do
5:   for j ← 1 to n do
6:     Y[i] ← Y[i]+A[i,j]*X[j]
7:   end for
8: end for
9: return Y
end function
```

```
function M_V(A, X, Y, n,i,k)
1: if i=k then
2:   for j ← 1 to n do
3:     Y[i] ← Y[i]+A[i,j]*X[j]
4:   end for
5: else
6:   m ← [(i + k) / 2]
7:   spawn M_V(A,X,Y,n,i,m)
8:   M_V(A, X, Y, n, m+1,k
9:   sync
10: end if
end function
```



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

## Párhuzamos ciklusok

Versenyhelyzet

# Párhuzamos ciklusok

Párhuzamosság

- Bár a **spawn** és **sync** utasításokkal is dolgozhatnánk, kényelmesebb, ha egy új programozói eszközt vetünk be.
- Ez az új eszköz a **parallel** kulcsszó lesz, amit a **for** ciklusok elé írhatunk.

Tekintsük az  $n \times n$  méretű  $A = (a_{ij})$  mátrix és az  $n$  méretű  $X = (x_i)$  vektor összeszorzásának feladatát:  $y_i = \sum_{j=1}^n a_{ij} \cdot x_j$ .

**function** MAT\_VEC2(A, X, n)

1: **parallel for** i  $\leftarrow 1$  to n **do**

2:   Y[i]  $\leftarrow 0$

3: **end for**

4: **M\_V(A, X, Y, n, 1, n)**

5: **return** Y

**end function**

**function** M\_V(A, X, Y, n,i,k)

1: **if** i=k **then**

2:   **for** j  $\leftarrow 1$  to n **do**

3:     Y[i]  $\leftarrow Y[i] + A[i,j] \cdot X[j]$

4:   **end for**

5: **else**

6:   m  $\leftarrow \lfloor (i+k)/2 \rfloor$

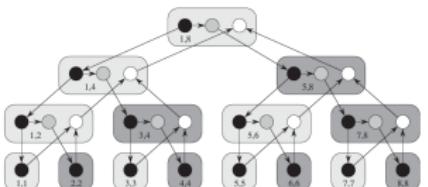
7:   **spawn** M\_V(A,X,Y,n,i,m)

8:   M\_V(A, X, Y, n, m+1,k)

9:   **sync**

10: **end if**

**end function**



Kós Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszínű végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

# A szorzás hatékonysága

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Párhuzamosság

- ▶  $T_1(n) = \Theta(n^2)$  ....
- ▶  $T_\infty(n) = \Theta(\lg n) + \max_{1 \leq i \leq n} t_\infty(i) = \Theta(\lg n) + n = \Theta(n)$
- ▶ Párhuzamosság:

$$T_1(n)/T_\infty(n) = \Theta(n^2)/\Theta(n) = \Theta(n)$$

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszálú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

## Versenyhelyzet (race condition)

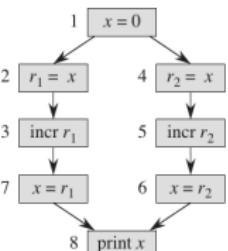
- ▶ Egy többszálú algoritmust determinisztikusnak (deterministic) nevezünk, ha az ütemező nem tudja befolyásolni a végeredményt
- ▶ Nem-determinisztikusnak nevezzük az algoritmust, ha a dag-ból kiolvasható szabályok megtartása melletti lehetséges eltérő utasítás végrehajtási sorrendek némelyike más eredményre vezetnek
- ▶ Determinisztikusnak tervezett algoritmusok gyakran válnak nem-determinisztikussá, mert úgynevezett versenyhelyzet (determinacy race) alakul ki bennük.
  - Ezek lényege: logikailag párhuzamos utasítások ugyanazt a memória területet érik el, és közülük legalább egy ír is oda.

**procedure** RACE\_EXAMPLE()

```

1: x ← 0
2: parallel for i ← 1 to 2 do
3:   x ← x+1
4: end for
5: print x

```

**end procedure**


(a)

step	x	r <sub>1</sub>	r <sub>2</sub>
1	0	–	–
2	0	0	–
3	0	1	–
4	0	1	0
5	0	1	1
6	1	1	1
7	1	1	1

(b)



## Párhuzamosság

Alapvető koncepciók  
Statikus szálak  
Dinamikus szálak  
Dynamic Multithreaded Programming – DMP  
Fibonacci számok  
A többszálú végrehajtás modelllezése  
A párhuzamosítás hatékonysága  
Útemezés  
Párhuzamos ciklusok  
**Versenyhelyzet**

## Versenyhelyzet (race condition)

Ha szeretnénk, hogy a programunk determinisztikus legyen, akkor

- ▶ biztosítanunk kell, hogy a párhuzamosan működő strandok függetlenek legyenek
  - ne legyen közöttük versengés
- ▶ parallel for esetén az iterációknak kell függetlennek lenniük egymástól
- ▶ spawn és sync között a leszármazott gyermek szálaknak kell függetlenek lenniük egymástól és a szülőtől, beleértve ebbe az ott indított újabb szálakat is



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

# Egy megtörtént eset

Adott egy program, amit  $P_1 = 32$  processzoros gépen fejlesztettek:

Majd a tényleges felhasználáshoz átültettek egy  $P_2 = 512$  processzoros gépre:

► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_1} \approx T_1/P_1 + T_\infty = 65$

► Optimalizálás utáni:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_1} \approx T'_1/P_1 + T'_\infty = 40$

„Hurrá!”

► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_2} \approx T_1/P_2 + T_\infty = 5$

► Optimalizálás utáni:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_2} \approx T'_1/P_2 + T'_\infty = 10$

„Hoppá!”



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

# Egy megtörtént eset

Adott egy program, amit  $P_1 = 32$  processzoros gépen fejlesztettek:

► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_1} \approx T_1/P_1 + T_\infty = 65$

► Optimalizálás utás:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_1} \approx T'_1/P_1 + T'_\infty = 40$

„Hurrá!”

Majd a tényleges felhasználáshoz átültettek egy  $P_2 = 512$  processzoros gépre:

► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_2} \approx T_1/P_2 + T_\infty = 5$

► Optimalizálás utás:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_2} \approx T'_1/P_2 + T'_\infty = 10$

„Hoppá!”



## Párhuzamosság

Alapvető koncepciók

Statikus szálak

Dinamikus szálak

Dynamic Multithreaded Programming – DMP

Fibonacci számok

A többszázú végrehajtás modelllezése

A párhuzamosítás hatékonysága

Ütemezés

Párhuzamos ciklusok

Versenyhelyzet

## Egy megtörtént eset

Adott egy program, amit  $P_1 = 32$  processzoros gépen fejlesztettek:

## ► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_1} \approx T_1/P_1 + T_\infty = 65$

## ► Optimalizálás utás:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_1} \approx T'_1/P_1 + T'_\infty = 40$

„Hurrá!”

Majd a tényleges felhasználáshoz átültettek egy  $P_2 = 512$  processzoros gépre:

## ► Első változat:

- $T_1 = 2048$
- $T_\infty = 1$
- $T_{P_2} \approx T_1/P_2 + T_\infty = 5$

## ► Optimalizálás utás:

- $T'_1 = 1024$
- $T'_\infty = 8$
- $T'_{P_2} \approx T'_1/P_2 + T'_\infty = 10$

„Hoppá!”

# 12. előadás

## Párhuzamosság

Párhuzamosítható algoritmusok,  
párhuzamosított algoritmusok analízise

*Adatszerkezetek és algoritmusok előadás*

2018. március 6.

Kósa Márk, Pánovics János,  
Szathmáry László és Halász Gábor

Debreceni Egyetem  
Informatikai Kar

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésűlő rendezés

Párhuzamos összefésűlések

Batcher-féle páros-páratlan összefésűlések



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

**Összefűsülő rendezés**

Párhuzamos összefűsülés

Batcher-féle páros-páratlan összefűsülés

## Általános tudnivalók

Ajánlott irodalom:

- ▶ Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein:  
Új algoritmusok, Scolar Informatika, 2003.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
1. (Alapvető algoritmusok), Műszaki Könyvkiadó, 1994.
- ▶ Donald E. Knuth: A számítógépprogramozás művészete  
3. (Keresés és rendezés), Műszaki Könyvkiadó, 1994.
- ▶ Seymour Lipschutz: Adatszerkezetek,  
Panem-McGraw-Hill, Budapest, 1993.
- ▶ Rónyai Lajos, Ivanyos Gábor, Szabó Réka: Algoritmusok,  
Typotex, Budapest, 2008.

Félév teljesítésének feltételei:

- ▶ Gyakorlati aláírás
  - 2 ZH
- ▶ Írásbeli vizsga, aminek az értékelésébe ...

További részletek:

<http://hallg.inf.unideb.hu/~halasz>

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## „Legegyszerűbb” változat

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**procedure** MATRIX\_SZOR(A, B, C, n)

```
1: for i ← 1 to n do
2:   for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure
```

Hány műveletet kell végrehajtani a sikeres számolás érdekében?

$$T(n) = \Theta(n^3)$$

Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## „Legegyszerűbb” változat

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**procedure** MATRIX\_SZOR(A, B, C, n)

```
1: for i ← 1 to n do
2:   for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure
```

Hány műveletet kell végrehajtani a sikeres számolás érdekében?

$$T(n) = \Theta(n^3)$$

Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## „Legegyszerűbb” változat

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**procedure** MATRIX\_SZOR(A, B, C, n)

```
1: for i ← 1 to n do
2:   for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure
```

Hány műveletet kell végrehajtani a sikeres számolás érdekében?

$$T(n) = \Theta(n^3)$$

Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés



## Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

## Összefésűlő rendezés

Párhuzamos összefésűlések

Batcher-féle páros-páratlan összefésűlések

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} & a_{18} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} & a_{28} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} & a_{38} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} & a_{48} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} & a_{58} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} & a_{67} & a_{68} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} & a_{77} & a_{78} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} & a_{87} & a_{88} \end{pmatrix}$$

$$\dots A_{12} = \begin{pmatrix} a_{15} & a_{16} & a_{17} & a_{18} \\ a_{25} & a_{26} & a_{27} & a_{28} \\ a_{35} & a_{36} & a_{37} & a_{38} \\ a_{45} & a_{46} & a_{47} & a_{48} \end{pmatrix} \dots$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}; B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}; C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C = A \cdot B = \begin{pmatrix} A_{11} \cdot B_{11} + A_{12} \cdot B_{21} & A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \\ A_{21} \cdot B_{11} + A_{22} \cdot B_{21} & A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \end{pmatrix};$$

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

```
1: if n=1 then
2:    $C_{11} \leftarrow C_{11} + a_{11} \cdot b_{11}$ 
3:   return
4: end if
5:  $n \leftarrow n/2$ 
6: M_S( $A_{11}, B_{11}, C_{11}, n$ )
7: M_S( $A_{12}, B_{21}, C_{11}, n$ )
8: M_S( $A_{11}, B_{12}, C_{12}, n$ )
9: M_S( $A_{12}, B_{22}, C_{12}, n$ )
10: M_S( $A_{21}, B_{11}, C_{21}, n$ )
11: M_S( $A_{22}, B_{21}, C_{21}, n$ )
12: M_S( $A_{21}, B_{12}, C_{22}, n$ )
13: M_S( $A_{22}, B_{22}, C_{22}, n$ )
end procedure
```

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés  
Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

- 1: **if**  $n=1$  **then**
- 2:    $C_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$
- 3:   **return**
- 4: **end if**
- 5:  $n \leftarrow n/2$
- 6: M\_S( $A_{11}, B_{11}, C_{11}, n$ )
- 7: M\_S( $A_{12}, B_{21}, C_{11}, n$ )
- 8: M\_S( $A_{11}, B_{12}, C_{12}, n$ )
- 9: M\_S( $A_{12}, B_{22}, C_{12}, n$ )
- 10: M\_S( $A_{21}, B_{11}, C_{21}, n$ )
- 11: M\_S( $A_{22}, B_{21}, C_{21}, n$ )
- 12: M\_S( $A_{21}, B_{12}, C_{22}, n$ )
- 13: M\_S( $A_{22}, B_{22}, C_{22}, n$ )

**end procedure**

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés  
Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

- 1: **if**  $n=1$  **then**
- 2:    $C_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$
- 3:   **return**
- 4: **end if**
- 5:  $n \leftarrow n/2$
- 6: M\_S( $A_{11}, B_{11}, C_{11}, n$ )
- 7: M\_S( $A_{12}, B_{21}, C_{11}, n$ )
- 8: M\_S( $A_{11}, B_{12}, C_{12}, n$ )
- 9: M\_S( $A_{12}, B_{22}, C_{12}, n$ )
- 10: M\_S( $A_{21}, B_{11}, C_{21}, n$ )
- 11: M\_S( $A_{22}, B_{21}, C_{21}, n$ )
- 12: M\_S( $A_{21}, B_{12}, C_{22}, n$ )
- 13: M\_S( $A_{22}, B_{22}, C_{22}, n$ )

**end procedure**

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$

3: **return**

4: **end if**

5:  $n \leftarrow n/2$

6: M\_S( $A_{11}, B_{11}, C_{11}, n$ )

7: M\_S( $A_{12}, B_{21}, C_{11}, n$ )

8: M\_S( $A_{11}, B_{12}, C_{12}, n$ )

9: M\_S( $A_{12}, B_{22}, C_{12}, n$ )

10: M\_S( $A_{21}, B_{11}, C_{21}, n$ )

11: M\_S( $A_{22}, B_{21}, C_{21}, n$ )

12: M\_S( $A_{21}, B_{12}, C_{22}, n$ )

13: M\_S( $A_{22}, B_{22}, C_{22}, n$ )

**end procedure**

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$

3: **return**

4: **end if**

5:  $n \leftarrow n/2$

6: M\_S( $A_{11}, B_{11}, C_{11}, n$ )

7: M\_S( $A_{12}, B_{21}, C_{11}, n$ )

8: M\_S( $A_{11}, B_{12}, C_{12}, n$ )

9: M\_S( $A_{12}, B_{22}, C_{12}, n$ )

10: M\_S( $A_{21}, B_{11}, C_{21}, n$ )

11: M\_S( $A_{22}, B_{21}, C_{21}, n$ )

12: M\_S( $A_{21}, B_{12}, C_{22}, n$ )

13: M\_S( $A_{22}, B_{22}, C_{22}, n$ )

**end procedure**

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyszerű” Oszd-meg-és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow c_{11} + a_{11} \cdot b_{11}$

3: **return**

4: **end if**

5:  $n \leftarrow n/2$

6: M\_S( $A_{11}, B_{11}, C_{11}, n$ )

7: M\_S( $A_{12}, B_{21}, C_{11}, n$ )

8: M\_S( $A_{11}, B_{12}, C_{12}, n$ )

9: M\_S( $A_{12}, B_{22}, C_{12}, n$ )

10: M\_S( $A_{21}, B_{11}, C_{21}, n$ )

11: M\_S( $A_{22}, B_{21}, C_{21}, n$ )

12: M\_S( $A_{21}, B_{12}, C_{22}, n$ )

13: M\_S( $A_{22}, B_{22}, C_{22}, n$ )

**end procedure**

$$C \leftarrow C + A \cdot B$$

$$n \neq 2^k?$$

Műveletek száma?

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(1)$$

$$T(n) = \Theta(n^3)$$

$$T(1) = \Theta(1)$$

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

Párhuzamosság

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor

Információs

szisztemák

Débrecenti Egyetem

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

„Egyeszerű” Oszd-meg-és-uralkodj változat

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



**procedure** M\_S(A, a, x, B, b, y, C, c, z, n)

1: **if** n=1 **then**

2:   C[c,z]  $\leftarrow$  C[c,z]+A[a,x]\*B[b,y]

3:   **return**

4: **end if**

5: n  $\leftarrow$  n/2

6: M\_S(A, a, x, B, b, y, C, c, z, n)

7: M\_S(A, a, x+n, B, b+n, y, C, c, z, n)

8: M\_S(A, a, x, B, b, y+n, C, c, z+n, n)

9: M\_S(A, a, x+n, B, b+n, y+n, C, c, z+n, n)

10: M\_S(A, a+n, x, B, b, y, C, c+n, z, n)

11: M\_S(A, a+n, x+n, B, b+n, y, C, c+n, z, n)

12: M\_S(A, a+n, x, B, b, y+n, C, c+n, z+n, n)

13: M\_S(A, a+n, x+n, B, b+n, y+n, C, c+n, z+n, n)

**end procedure**

Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésűlő rendezés

Párhuzamos összefésűlések

Batcher-féle páros-páratlan összefésűlések

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## Strassen algoritmusa

- ① Osszuk fel az A és B mátrixokat  $\frac{n}{2} \times \frac{n}{2}$  méretű részmátrixokra.
- ② Hozzunk létre 10 darab, szintén  $\frac{n}{2} \times \frac{n}{2}$  méretű új mátrixot az előbb definiált részmátrixokkal meghatározva:

$$S_1 = B_{12} - B_{22}, \quad S_2 = A_{11} + A_{12}, \quad S_3 = A_{21} + A_{22},$$

$$S_4 = B_{21} - B_{11}, \quad S_5 = A_{11} + A_{22}, \quad S_6 = B_{11} + B_{22},$$

$$S_7 = A_{12} - A_{22}, \quad S_8 = B_{21} + B_{22}, \quad S_9 = A_{11} - A_{21},$$

$$S_{10} = B_{11} + B_{12}.$$

- ③ Ezek felhasználásával számoljunk ki újabb 7 mátrixot:

$$P_1 = A_{11} \cdot S_1, \quad P_2 = S_2 \cdot B_{22}, \quad P_3 = S_3 \cdot B_{11}, \quad P_4 = A_{22} \cdot S_4,$$

$$P_5 = S_5 \cdot S_6, \quad P_6 = S_7 \cdot S_8, \quad P_7 = S_9 \cdot S_{10}.$$

- ④ És végül határozzuk meg a C (szintén  $\frac{n}{2} \times \frac{n}{2}$  méretű) részmátrixait:

$$C_{11} = P_5 + P_4 - P_2 + P_6, \quad C_{12} = P_1 + P_2,$$

$$C_{21} = P_3 + P_4, \quad C_{22} = P_5 + P_1 - P_3 - P_7.$$

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## Strassen algoritmusa

Párhuzamosság

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

- ▶ Ami biztos:
  - Ez „sokkal” bonyolultabb, mint az előzőek
- ▶ Amit nem nyilvánvaló
  - Helyes ez egyáltalán?
  - Megéri a bonyodalmat?Azaz hatékonyabb, mint a korábbi algoritmusok?

$$S(n) = 7S(n/2) + \Theta(n^2)$$

$$S(n) = \Theta(n^{\lg 7})$$

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## Strassen algoritmusa

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

- ▶ Ami biztos:
  - Ez „sokkal” bonyolultabb, mint az előzőek
- ▶ Amit nem nyilvánvaló
  - Helyes ez egyáltalán?
  - Megéri a bonyodalmat?  
Azaz hatékonyabb, mint a korábbi algoritmusok?

$$S(n) = 7S(n/2) + \Theta(n^2)$$

$$S(n) = \Theta(n^{3.5})$$

# Mátrix-szorzás: Soros (egy-szálú) algoritmussal

## Strassen algoritmusa

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

- ▶ Ami biztos:
  - Ez „sokkal” bonyolultabb, mint az előzőek
- ▶ Amit nem nyilvánvaló
  - Helyes ez egyáltalán?
  - Megéri a bonyodalmat?  
Azaz hatékonyabb, mint a korábbi algoritmusok?

$$S(n) = 7S(n/2) + \Theta(n^2)$$

$$S(n) = \Theta(n^{\lg 7})$$



## Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

## Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

## „Legegyszerűbb” változat

```

procedure MATRIX_SZOR(A, B, C, n)
1: parallel for i ← 1 to n do
2:   parallel for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure

```

- ▶ work:  
 $T_1(n) = \Theta(n^3)$
- ▶ span:  
 $T_\infty(n) = \Theta(n)$   
 $(2\Theta(\lg n) + \Theta(n))$
- ▶ párhuzamosság:  
 $\Theta(n^2)$
- ▶ Ez tovább javítható  $\Theta(n^3 / \lg n)$ -re, ha párhuzamosítjuk a 4.–6. sor ciklusát. DE ...



## Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

## Párhuzamos algoritmus

## Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

## „Legegyszerűbb” változat

```

procedure MATRIX_SZOR(A, B, C, n)
1: parallel for i ← 1 to n do
2:   parallel for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure

```

**▶ work:**

$$T_1(n) = \Theta(n^3)$$

**▶ span:**

$$T_\infty(n) = \Theta(n) \\ (2\Theta(\lg n) + \Theta(n))$$

**▶ párhuzamosság:**  
 $\Theta(n^2)$ 

**▶ Ez tovább javítható**  
 $\Theta(n^3 / \lg n)$ -re, ha párhuzamosítjuk a 4.–6. sor ciklusát. DE ...



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

## „Legegyszerűbb” változat

```

procedure MATRIX_SZOR(A, B, C, n)
1: parallel for i ← 1 to n do
2:   parallel for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure

```

▶ work:

$$T_1(n) = \Theta(n^3)$$

▶ span:

$$T_\infty(n) = \Theta(n) \\ (2\Theta(\lg n) + \Theta(n))$$

▶ párhuzamosság:  
 $\Theta(n^2)$

▶ Ez tovább javítható  
 $\Theta(n^3 / \lg n)$ -re, ha párhuzamosítjuk a 4.–6. sor ciklusát. DE ...

# Párhuzamos algoritmus

## „Legegyszerűbb” változat

```
procedure MATRIX_SZOR(A, B, C, n)
1: parallel for i ← 1 to n do
2:   parallel for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure
```

▶ work:

$$T_1(n) = \Theta(n^3)$$

▶ span:

$$T_\infty(n) = \Theta(n) \\ (2\Theta(\lg n) + \Theta(n))$$

▶ párhuzamosság:  
 $\Theta(n^2)$

▶ Ez tovább javítható  
 $\Theta(n^3 / \lg n)$ -re, ha párhuzamosítjuk a 4.-6. sor ciklusát. DE ...

Párhuzamosság

Kós Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

## „Legegyszerűbb” változat

```
procedure MATRIX_SZOR(A, B, C, n)
1: parallel for i ← 1 to n do
2:   parallel for j ← 1 to n do
3:     C[i,j] ← 0
4:     for k ← 1 to n do
5:       C[i,j] ← C[i,j]+A[i,k]*B[k,j]
6:     end for
7:   end for
8: end for
end procedure
```

- ▶ work:  
 $T_1(n) = \Theta(n^3)$
- ▶ span:  
 $T_\infty(n) = \Theta(n)$   
 $(2\Theta(\lg n) + \Theta(n))$
- ▶ párhuzamosság:  
 $\Theta(n^2)$
- ▶ Ez tovább javítható  
 $\Theta(n^3 / \lg n)$ -re, ha párhuzamosítjuk a 4.–6. sor ciklusát. DE ...





Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow C_{11} + a_{11} \cdot b_{11}$

3: **else**

4:   n  $\leftarrow n/2$

5:   **spawn** M\_S( $A_{11}$ ,  $B_{11}$ ,  $C_{11}$ , n)

6:   **spawn** M\_S( $A_{12}$ ,  $B_{21}$ ,  $C_{11}$ , n)

7:   **spawn** M\_S( $A_{11}$ ,  $B_{12}$ ,  $C_{12}$ , n)

8:   **spawn** M\_S( $A_{12}$ ,  $B_{22}$ ,  $C_{12}$ , n)

9:   **spawn** M\_S( $A_{21}$ ,  $B_{11}$ ,  $C_{21}$ , n)

10:   **spawn** M\_S( $A_{22}$ ,  $B_{21}$ ,  $C_{21}$ , n)

11:   **spawn** M\_S( $A_{21}$ ,  $B_{12}$ ,  $C_{22}$ , n)

12:   M\_S( $A_{22}$ ,  $B_{22}$ ,  $C_{22}$ , n)

13:   **sync**

14: **end if**

**end procedure**

$C \leftarrow C + A \cdot B$

Helyes ez?

Sajnos nem:  
versenyhelyzet



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow C_{11} + a_{11} \cdot b_{11}$

3: **else**

4:    $n \leftarrow n/2$

5:   **spawn** M\_S( $A_{11}$ ,  $B_{11}$ ,  $C_{11}$ , n)

6:   **spawn** M\_S( $A_{12}$ ,  $B_{21}$ ,  $C_{11}$ , n)

7:   **spawn** M\_S( $A_{11}$ ,  $B_{12}$ ,  $C_{12}$ , n)

8:   **spawn** M\_S( $A_{12}$ ,  $B_{22}$ ,  $C_{12}$ , n)

9:   **spawn** M\_S( $A_{21}$ ,  $B_{11}$ ,  $C_{21}$ , n)

10:   **spawn** M\_S( $A_{22}$ ,  $B_{21}$ ,  $C_{21}$ , n)

11:   **spawn** M\_S( $A_{21}$ ,  $B_{12}$ ,  $C_{22}$ , n)

12:   M\_S( $A_{22}$ ,  $B_{22}$ ,  $C_{22}$ , n)

13:   **sync**

14: **end if**

**end procedure**

$C \leftarrow C + A \cdot B$

Helyes ez?

Sajnos nem:  
versenyhelyzet



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefésűlő rendezés

Párhuzamos összefésűlések

Batcher-féle páros-páratlan összefésűlések

# Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat

**procedure** M\_S(A, B, C, n)

1: **if** n=1 **then**

2:    $C_{11} \leftarrow C_{11} + a_{11} \cdot b_{11}$

3: **else**

4:    $n \leftarrow n/2$

5:   **spawn** M\_S( $A_{11}$ ,  $B_{11}$ ,  $C_{11}$ , n)

6:   **spawn** M\_S( $A_{12}$ ,  $B_{21}$ ,  $C_{11}$ , n)

7:   **spawn** M\_S( $A_{11}$ ,  $B_{12}$ ,  $C_{12}$ , n)

8:   **spawn** M\_S( $A_{12}$ ,  $B_{22}$ ,  $C_{12}$ , n)

9:   **spawn** M\_S( $A_{21}$ ,  $B_{11}$ ,  $C_{21}$ , n)

10:   **spawn** M\_S( $A_{22}$ ,  $B_{21}$ ,  $C_{21}$ , n)

11:   **spawn** M\_S( $A_{21}$ ,  $B_{12}$ ,  $C_{22}$ , n)

12:   M\_S( $A_{22}$ ,  $B_{22}$ ,  $C_{22}$ , n)

13: **sync**

14: **end if**

**end procedure**

$C \leftarrow C + A \cdot B$

Helyes ez?

Sajnos nem:  
versenyhelyzet



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

## Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

## Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat  
**procedure** M\_S(A, B, C, n)

```

1: if n=1 then
2:    $C_{11} \leftarrow C_{11} + A_{11} \cdot B_{11}$ 
3: else
4:    $n \leftarrow n/2$ 
5:   spawn M_S( $A_{11}, B_{11}, C_{11}, n$ )
6:   spawn M_S( $A_{12}, B_{21}, T_{11}, n$ )
7:   spawn M_S( $A_{11}, B_{12}, C_{12}, n$ )
8:   spawn M_S( $A_{12}, B_{22}, T_{12}, n$ )
9:   spawn M_S( $A_{21}, B_{11}, C_{21}, n$ )
10:  spawn M_S( $A_{22}, B_{21}, T_{21}, n$ )
11:  spawn M_S( $A_{21}, B_{12}, C_{22}, n$ )
12:  M_S( $A_{22}, B_{22}, T_{22}, n$ )
13:  sync
14:  M_ADD(C, T, n)
15: end if
end procedure
```

**procedure** M\_ADD(C, T, n)

```

1: parallel for  $i \leftarrow 1$  to  $n$  do
2:   parallel for  $j \leftarrow 1$  to  $n$  do
3:      $C[i,j] \leftarrow C[i,j] + T[i,j]$ 
4:   end for
5: end for
end procedure
```

▶ work:  $M_1(n) = \Theta(n^3)$   
 $(8M_1(n/2) + \Theta(n^2))$

▶ span:  
 $M_\infty(n) = \Theta(\lg^2 n)$   
 $(M_\infty(n/2) + \Theta(\lg n))$

▶ párhuzamosság:  
 $\Theta(n^3 / \lg^2 n)$   
 Ez jobb, mint az  
 „egyszerű”  
 algoritmus  
 párhuzamossága.



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat  
**procedure** M\_S(A, B, C, n)

```

1: if n=1 then
2:    $C_{11} \leftarrow C_{11} + A_{11} \cdot B_{11}$ 
3: else
4:    $n \leftarrow n/2$ 
5:   spawn M_S( $A_{11}, B_{11}, C_{11}, n$ )
6:   spawn M_S( $A_{12}, B_{21}, T_{11}, n$ )
7:   spawn M_S( $A_{11}, B_{12}, C_{12}, n$ )
8:   spawn M_S( $A_{12}, B_{22}, T_{12}, n$ )
9:   spawn M_S( $A_{21}, B_{11}, C_{21}, n$ )
10:  spawn M_S( $A_{22}, B_{21}, T_{21}, n$ )
11:  spawn M_S( $A_{21}, B_{12}, C_{22}, n$ )
12:  M_S( $A_{22}, B_{22}, T_{22}, n$ )
13:  sync
14:  M_ADD(C, T, n)
15: end if
end procedure
```

**procedure** M\_ADD(C, T, n)

```

1: parallel for i  $\leftarrow 1$  to n do
2:   parallel for j  $\leftarrow 1$  to n do
3:      $C[i,j] \leftarrow C[i,j] + T[i,j]$ 
4:   end for
5: end for
end procedure
```

► work:  $M_1(n) = \Theta(n^3)$   
 $(8M_1(n/2) + \Theta(n^2))$

► span:  
 $M_\infty(n) = \Theta(\lg^2 n)$   
 $(M_\infty(n/2) + \Theta(\lg n))$

► párhuzamosság:  
 $\Theta(n^3 / \lg^2 n)$   
 Ez jobb, mint az  
 „egyszerű”  
 algoritmus  
 párhuzamossága.



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

## Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

## Párhuzamos algoritmus

„Egyszerű” Oszd-meg- és-uralkodj változat  
**procedure** M\_S(A, B, C, n)

```

1: if n=1 then
2:    $C_{11} \leftarrow C_{11} + A_{11} \cdot B_{11}$ 
3: else
4:    $n \leftarrow n/2$ 
5:   spawn M_S( $A_{11}, B_{11}, C_{11}, n$ )
6:   spawn M_S( $A_{12}, B_{21}, T_{11}, n$ )
7:   spawn M_S( $A_{11}, B_{12}, C_{12}, n$ )
8:   spawn M_S( $A_{12}, B_{22}, T_{12}, n$ )
9:   spawn M_S( $A_{21}, B_{11}, C_{21}, n$ )
10:  spawn M_S( $A_{22}, B_{21}, T_{21}, n$ )
11:  spawn M_S( $A_{21}, B_{12}, C_{22}, n$ )
12:  M_S( $A_{22}, B_{22}, T_{22}, n$ )
13:  sync
14:  M_ADD(C, T, n)
15: end if
end procedure
```

**procedure** M\_ADD(C, T, n)

```

1: parallel for i  $\leftarrow 1$  to n do
2:   parallel for j  $\leftarrow 1$  to n do
3:      $C[i,j] \leftarrow C[i,j] + T[i,j]$ 
4:   end for
5: end for
end procedure
```

- ▶ work:  $M_1(n) = \Theta(n^3)$   
 $(8M_1(n/2) + \Theta(n^2))$

- ▶ span:  
 $M_\infty(n) = \Theta(\lg^2 n)$   
 $(M_\infty(n/2) + \Theta(\lg n))$

- ▶ párhuzamosság:  
 $\Theta(n^3 / \lg^2 n)$   
 Ez jobb, mint az  
 „egyszerű”  
 algoritmus  
 párhuzamossága.



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

## Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

## Párhuzamos algoritmus

„Egyszerű” Oszd-meg-és-uralkodj változat  
**procedure** M\_S(A, B, C, n)

```

1: if n=1 then
2:    $C_{11} \leftarrow C_{11} + A_{11} \cdot B_{11}$ 
3: else
4:    $n \leftarrow n/2$ 
5:   spawn M_S( $A_{11}, B_{11}, C_{11}, n$ )
6:   spawn M_S( $A_{12}, B_{21}, T_{11}, n$ )
7:   spawn M_S( $A_{11}, B_{12}, C_{12}, n$ )
8:   spawn M_S( $A_{12}, B_{22}, T_{12}, n$ )
9:   spawn M_S( $A_{21}, B_{11}, C_{21}, n$ )
10:  spawn M_S( $A_{22}, B_{21}, T_{21}, n$ )
11:  spawn M_S( $A_{21}, B_{12}, C_{22}, n$ )
12:  M_S( $A_{22}, B_{22}, T_{22}, n$ )
13:  sync
14:  M_ADD(C, T, n)
15: end if
end procedure
```

**procedure** M\_ADD(C, T, n)

```

1: parallel for i  $\leftarrow 1$  to n do
2:   parallel for j  $\leftarrow 1$  to n do
3:      $C[i,j] \leftarrow C[i,j] + T[i,j]$ 
4:   end for
5: end for
end procedure
```

- ▶ work:  $M_1(n) = \Theta(n^3)$   
 $(8M_1(n/2) + \Theta(n^2))$

- ▶ span:  
 $M_\infty(n) = \Theta(\lg^2 n)$   
 $(M_\infty(n/2) + \Theta(\lg n))$

- ▶ párhuzamosság:  
 $\Theta(n^3 / \lg^2 n)$   
 Ez jobb, mint az  
 „egyszerű”  
 algoritmus  
 párhuzamossága.

$$M_\infty(n) = M_\infty(n/2) + \Theta(\lg n)$$

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



## Theorem (Mester téTEL)

Legyenek  $a \geq 1$ ,  $b > 1$  állandók,  $f(n)$  egy függvény,  $T(n)$  pedig a nemnegatív egészeken a

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

rekurzív egyenlettel definiált függvény, ahol  $n/b$  jelentheti akár az  $\lfloor n/b \rfloor$  egészrészét, akár az  $\lceil n/b \rceil$  egészrészét. Ekkor:

- ① Ha  $f(n) = O(n^{\log_b a - \epsilon})$ , egy  $\epsilon > 0$  állandóval, akkor
$$\longrightarrow T(n) = \Theta(n^{\log_b a}).$$
- ② Ha  $f(n) = \Theta(n^{\log_b a})$ , akkor
$$\longrightarrow T(n) = \Theta(n^{\log_b a} \lg n) (= \Theta(f(n) \lg n)).$$
- ③ Ha  $f(n) = \Omega(n^{\log_b a + \epsilon})$  és  $a \cdot f(n/b) \leq c \cdot f(n)$  valamely  $\epsilon > 0$  és  $c < 1$  állandóra és elég nagy  $n$ -re, akkor
$$\longrightarrow T(n) = \Theta(f(n)).$$

Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

$$M_\infty(n) = M_\infty(n/2) + \Theta(\lg n)$$

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés

Batcher-féle páros-páratlan összefűsítés

## Theorem (Tankönyv 4.6-2 feladat)

Ha  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , ahol  $k \geq 0$ , akkor a mester rekurzió megoldása:

$$T(n) = \Theta\left(n^{\log_b a} \lg^{k+1} n\right).$$

Esetünkben:  $a=1$ ,  $b=2$ ,  $k=1$ :  $T(n) = \Theta(n^{\lg 1} \lg^2 n) = \Theta(\lg^2 n)$



Mátrix-szorzás

Soros algoritmus  
 Strassen algoritmus

Párhuzamos algoritmus

Összefűsítő rendezés

Párhuzamos összefűsítés  
 Batcher-féle páros-páratlan összefűsítés

# Párhuzamos algoritmus

## Strassen algoritmusa

- ① Osszuk fel az A és B mátrixokat  $\frac{n}{2} \times \frac{n}{2}$  méretű részmátrixokra. work=span=Θ(1)
- ② Hozzunk létre 10 darab, szintén  $\frac{n}{2} \times \frac{n}{2}$  méretű új mátrixot az előbb definiált részmátrixokkal meghatározva:

$$\begin{aligned} S_1 &= B_{12} - B_{22}, & S_2 &= A_{11} + A_{12}, & S_3 &= A_{21} + A_{22}, \\ S_4 &= B_{21} - B_{11}, & S_5 &= A_{11} + A_{22}, & S_6 &= B_{11} + B_{22}, \\ S_7 &= A_{12} - A_{22}, & S_8 &= B_{21} + B_{22}, & S_9 &= A_{11} - A_{21}, \\ S_{10} &= B_{11} + B_{12}. & \text{work: } \Theta(n^2), & \text{span: } \Theta(\lg n) \end{aligned}$$

- ③ Ezek felhasználásával számoljunk ki újabb 7 mátrixot:

$$\begin{aligned} P_1 &= A_{11} \cdot S_1, & P_2 &= S_2 \cdot B_{22}, & P_3 &= S_3 \cdot B_{11}, & P_4 &= A_{22} \cdot S_4, \\ P_5 &= S_5 \cdot S_6, & P_6 &= S_7 \cdot S_8, & P_7 &= S_9 \cdot S_{10}. \end{aligned}$$

work:  $\Theta(n^{\lg 7})$ , span:  $\Theta(\lg^2 n)$

- ④ És végül határozzuk meg a C (szintén  $\frac{n}{2} \times \frac{n}{2}$  méretű) részmátrixait:

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6, & C_{12} &= P_1 + P_2, \\ C_{21} &= P_3 + P_4, & C_{22} &= P_5 + P_1 - P_3 - P_7. \end{aligned}$$

work:  $\Theta(n^2)$ , span:  $\Theta(\lg n)$

- ▶ work:  $\Theta(n^{\lg 7})$ , span:  $\Theta(\lg^2 n)$ ,
- ▶ párhuzamosság:  $\Theta(n^{\lg 7} / \lg^2 n)$

# Összefésülő rendezés

**procedure** RENDEZ'(A, p, r)

1: **if**  $p < r$  **then**

2:    $q \leftarrow \lfloor (p + q)/2 \rfloor$

3:   **spawn** RENDEZ'(A, p, q)

4:   RENDEZ'(A, q+1, r)

5:   **sync**

6:   ÖSSZEFÉSÜL(A, p, q, r)

7: **end if**

**end procedure**

**procedure** ÖSSZEFÉSÜL(A,p,q,r)

1:  $n_1 \leftarrow q-p+1$

2:  $n_2 \leftarrow r-q$

3: **for**  $i \leftarrow 1$  **to**  $n_1$  **do**

4:    $L[i] \leftarrow A[p+i-1]$

5: **end for**

6: **for**  $j \leftarrow 1$  **to**  $n_2$  **do**

7:    $R[j] \leftarrow A[q+j]$

8: **end for**

9:  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$

10:  $i \leftarrow j \leftarrow 1$

11: **for**  $k \leftarrow p$  **to**  $r$  **do**

12:   **if**  $L[i] \leq R[j]$  **then**

13:      $A[k] \leftarrow L[i]$

14:      $i \leftarrow i + 1$

15:   **else**

16:      $A[k] \leftarrow R[j]$

17:      $j \leftarrow j + 1$

18:   **end if**

19: **end for**

**end procedure**

Párhuzamosság

Kósa Márk

Pánovics János

Szathmáry László

Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

**Összefésülő rendezés**

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Összefésülő rendezés

**procedure** RENDEZ'(A, p, r)1: **if**  $p < r$  **then**2:    $q \leftarrow \lfloor (p+q)/2 \rfloor$ 3:   **spawn** RENDEZ'(A, p, q)

4:   RENDEZ'(A, q+1, r)

5:   **sync**

6:   ÖSSZEFÉSÜL(A, p, q, r)

7: **end if****end procedure****procedure** ÖSSZEFÉSÜL(A,p,q,r)1:  $n_1 \leftarrow q-p+1$ 2:  $n_2 \leftarrow r-q$ 3: **for**  $i \leftarrow 1$  **to**  $n_1$  **do**4:    $L[i] \leftarrow A[p+i-1]$ 5: **end for**6: **for**  $j \leftarrow 1$  **to**  $n_2$  **do**7:    $R[j] \leftarrow A[q+j]$ 8: **end for**9:  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$ 
 10:  $i \leftarrow j \leftarrow 1$   
 11: **for**  $k \leftarrow p$  **to**  $r$  **do**  
 12:   **if**  $L[i] \leq R[j]$  **then**  
 13:      $A[k] \leftarrow L[i]$   
 14:      $i \leftarrow i + 1$   
 15:   **else**  
 16:      $A[k] \leftarrow R[j]$   
 17:      $j \leftarrow j + 1$   
 18:   **end if**  
 19: **end for**
**end procedure**

▶ work:

$$R'_1(n) = \Theta(n \lg n)$$

$$(2R'_1(n/2) + \Theta(n))$$

▶ span:

$$R'_{\infty}(n) = \Theta(n)$$

$$(R'_{\infty}(n/2) + \Theta(n))$$

▶ párhuzamosság:

$$\Theta(\lg n)$$

Ez elég „sovány”.

Nincs annyi adat, hogy néhány száz processzort érdemes legyen használni a rendezéshez.

▶ Lehetne-e az összefésülést is párhuzamosítani?



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

**Összefésülő rendezés**

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Összefésülő rendezés

**procedure** RENDEZ'(A, p, r)1: **if**  $p < r$  **then**2:    $q \leftarrow \lfloor (p+q)/2 \rfloor$ 3:   **spawn** RENDEZ'(A, p, q)

4:   RENDEZ'(A, q+1, r)

5:   **sync**

6:   ÖSSZEFÉSÜL(A, p, q, r)

7: **end if****end procedure****procedure** ÖSSZEFÉSÜL(A,p,q,r)1:  $n_1 \leftarrow q-p+1$ 2:  $n_2 \leftarrow r-q$ 3: **for**  $i \leftarrow 1$  **to**  $n_1$  **do**4:    $L[i] \leftarrow A[p+i-1]$ 5: **end for**6: **for**  $j \leftarrow 1$  **to**  $n_2$  **do**7:    $R[j] \leftarrow A[q+j]$ 8: **end for**9:  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$ 
 10:  $i \leftarrow j \leftarrow 1$   
 11: **for**  $k \leftarrow p$  **to**  $r$  **do**  
 12:   **if**  $L[i] \leq R[j]$  **then**  
 13:      $A[k] \leftarrow L[i]$   
 14:      $i \leftarrow i + 1$   
 15:   **else**  
 16:      $A[k] \leftarrow R[j]$   
 17:      $j \leftarrow j + 1$   
 18:   **end if**  
 19: **end for**
**end procedure**

▶ work:

$$R'_1(n) = \Theta(n \lg n)$$

$$(2R'_1(n/2) + \Theta(n))$$

▶ span:

$$R'_{\infty}(n) = \Theta(n)$$

$$(R'_{\infty}(n/2) + \Theta(n))$$

▶ párhuzamosság:

$$\Theta(\lg n)$$

Ez elég „sovány”.

Nincs annyi adat, hogy néhány száz processzort érdemes legyen használni a rendezéshez.

▶ Lehetne-e az összefésülést is párhuzamosítani?



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

**Összefésülő rendezés**

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Összefésülő rendezés

**procedure** RENDEZ'(A, p, r)1: **if**  $p < r$  **then**2:    $q \leftarrow \lfloor (p+q)/2 \rfloor$ 3:   **spawn** RENDEZ'(A, p, q)

4:   RENDEZ'(A, q+1, r)

5:   **sync**

6:   ÖSSZEFÉSÜL(A, p, q, r)

7: **end if****end procedure****procedure** ÖSSZEFÉSÜL(A,p,q,r)1:  $n_1 \leftarrow q-p+1$ 2:  $n_2 \leftarrow r-q$ 3: **for**  $i \leftarrow 1$  **to**  $n_1$  **do**4:    $L[i] \leftarrow A[p+i-1]$ 5: **end for**6: **for**  $j \leftarrow 1$  **to**  $n_2$  **do**7:    $R[j] \leftarrow A[q+j]$ 8: **end for**9:  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$ 
 10:  $i \leftarrow j \leftarrow 1$   
 11: **for**  $k \leftarrow p$  **to**  $r$  **do**  
 12:   **if**  $L[i] \leq R[j]$  **then**  
 13:      $A[k] \leftarrow L[i]$   
 14:      $i \leftarrow i + 1$   
 15:   **else**  
 16:      $A[k] \leftarrow R[j]$   
 17:      $j \leftarrow j + 1$   
 18:   **end if**  
 19: **end for**
**end procedure**

▶ work:

$$R'_1(n) = \Theta(n \lg n)$$

$$(2R'_1(n/2) + \Theta(n))$$

▶ span:

$$R'_{\infty}(n) = \Theta(n)$$

$$(R'_{\infty}(n/2) + \Theta(n))$$

▶ párhuzamosság:

$$\Theta(\lg n)$$

EZ elég „sovány”.

Nincs annyi adat, hogy néhány száz processzort érdemes legyen használni a rendezéshez.

▶ Lehetne-e az összefésülést is párhuzamosítani?



Mátrix-szorzás

Soros algoritmus

Strassen algoritmusa

Párhuzamos algoritmus

**Összefésülő rendezés**

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Összefésülő rendezés

**procedure** RENDEZ'(A, p, r)1: **if** p < r **then**2:   q  $\leftarrow \lfloor (p+q)/2 \rfloor$ 3:   **spawn** RENDEZ'(A, p, q)

4:   RENDEZ'(A, q+1, r)

5:   **sync**

6:   ÖSSZEFÉSÜL(A, p, q, r)

7: **end if****end procedure****procedure** ÖSSZEFÉSÜL(A,p,q,r)1:  $n_1 \leftarrow q-p+1$ 2:  $n_2 \leftarrow r-q$ 3: **for** i  $\leftarrow 1$  **to**  $n_1$  **do**4:   L[i]  $\leftarrow A[p+i-1]$ 5: **end for**6: **for** j  $\leftarrow 1$  **to**  $n_2$  **do**7:   R[j]  $\leftarrow A[q+j]$ 8: **end for**9:  $L[n_1+1] \leftarrow R[n_2+1] \leftarrow \infty$ 
 10: i  $\leftarrow j \leftarrow 1$   
 11: **for** k  $\leftarrow p$  **to** r **do**  
 12:   **if** L[i]  $\leq R[j]$  **then**  
 13:     A[k]  $\leftarrow L[i]$   
 14:     i  $\leftarrow i + 1$   
 15:   **else**  
 16:     A[k]  $\leftarrow R[j]$   
 17:     j  $\leftarrow j + 1$   
 18:   **end if**  
 19: **end for**

**end procedure**

**► work:**

$$R'_1(n) = \Theta(n \lg n)$$

$$(2R'_1(n/2) + \Theta(n))$$

**► span:**

$$R'_\infty(n) = \Theta(n)$$

$$(R'_\infty(n/2) + \Theta(n))$$

**► párhuzamosság:**

$$\Theta(\lg n)$$

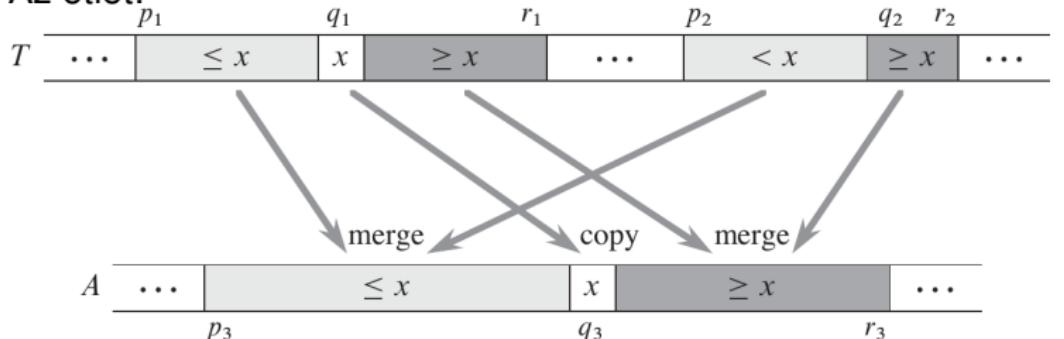
EZ elég „sovány”.

Nincs annyi adat,  
hogy néhány száz  
processzort  
érdekes legyen  
használni a  
rendezéshez.

**► Lehetne-e az  
összefésülést is  
párhuzamosítani?**

# Párhuzamos összefésülés

Az ötlet:



```

procedure BINÁRIS_KERES
    (x, A, p, r)
1: alsó  $\leftarrow$  p
2: felső  $\leftarrow$  MAX(p, r+1)
3: while alsó < felső do
4:     középső  $\leftarrow$ 
         $\lfloor (\text{alsó}+\text{felső})/2 \rfloor$ 
5:     if  $x \leq A[\text{középső}]$  then
6:         felső  $\leftarrow$  középső
7:     else
8:         alsó  $\leftarrow$  középső+1
9:     end if
10: end while
11: return felső
end procedure

```

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

Összefésűlő rendezés

Párhuzamos összefésűlés  
Batcher-féle páros-páratlan összefésűlés



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

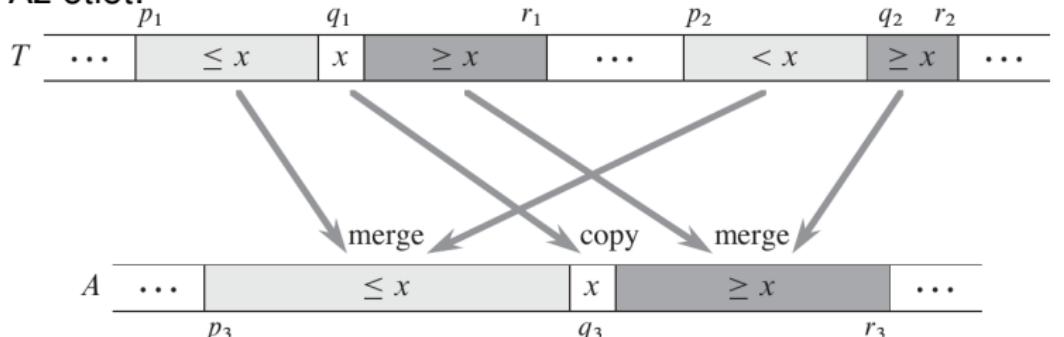
Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

Az ötlet:



```

procedure BINÁRIS_KERES
    (x, A, p, r)
    1: alsó  $\leftarrow$  p
    2: felső  $\leftarrow$  MAX(p, r+1)
    3: while alsó < felső do
    4:   középső  $\leftarrow$ 
         $\lfloor (\text{alsó}+\text{felső})/2 \rfloor$ 
    5:   if x  $\leq$  A[középső] then
    6:     felső  $\leftarrow$  középső
    7:   else
    8:     alsó  $\leftarrow$  középső+1
    9:   end if
    10: end while
    11: return felső
end procedure
  
```

# Párhuzamos összefésülés

Párhuzamosság

**procedure** P\_ÖSSZEFÉSÜL(T, p1, r1, p2, r2, A, p3)

- 1:  $n1 \leftarrow r1-p1+1$
- 2:  $n2 \leftarrow r2-p2+1$
- 3: **if**  $n1 < n2$  **then**
- 4:   ( $p1, r1, n1$ ) és ( $p2, r2, n2$ ) felcserélése
- 5: **end if**
- 6: **if**  $n1>0$  **then**
- 7:    $q1 \leftarrow \lfloor (p1+p2)/2 \rfloor$
- 8:    $q2 \leftarrow \text{BINÁRIS\_KERES}(T[q1], T, p2, r2)$
- 9:    $q3 \leftarrow p3+(q1-p1)+(q2-p2)$
- 10:    $A[q3] \leftarrow T[q1]$
- 11:   **spawn** P\_ÖSSZEFÉSÜL( $T, p1, q1-1, p2, q2-1, A, p3$ )
- 12:   P\_ÖSSZEFÉSÜL( $T, q1+1, r1, q2, r2, A, q3+1$ )
- 13:   **sync**
- 14: **end if**

**end procedure**

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésűlő rendezés

Párhuzamos összefésűlés

Batcher-féle páros-páratlan összefésűlés

# Párhuzamos összefésülés

Párhuzamosság

**procedure** P\_ÖSSZEFEΣÜL(T, p1, r1, p2, r2, A, p3)

```
1: n1 ← r1-p1+1
2: n2 ← r2-p2+1
3: if n1 < n2 then
4:   (p1, r1, n1) és (p2, r2, n2) felcserélése
5: end if
6: if n1>0 then
7:   q1 ← ⌊(p1+p2)/2⌋
8:   q2 ← BINÁRIS_KERES(T[q1], T, p2, r2)
9:   q3 ← p3+(q1-p1)+(q2-p2)
10:  A[q3] ← T[q1]
11:  spawn P_ÖSSZEFEΣÜL
        (T, p1, q1-1, p2, q2-1, A, p3)
12:  P_ÖSSZEFEΣÜL(T, q1+1, r1, q2, r2, A, q3+1)
13:  sync
14: end if
end procedure
```

► work:  $F_1(n) = \Theta(n)$   
 $(F_1(\alpha n) + F_1((1 - \alpha)n))$   
 $+ \Theta(\lg n))$

$$0.25 \leq \alpha \leq 0.75$$

► span:  
 $F_\infty(n) = \Theta(\lg^2 n)$   
 $(F_\infty(3n/4) + \Theta(\lg n))$

► párhuzamosság:  
 $\Theta(n/\lg^2 n)$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás  
Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

## Összefésűlő rendezés

Párhuzamos összefésűlés  
Batcher-féle páros-páratlan összefésűlés

# Párhuzamos összefésülés

Párhuzamosság

**procedure** P\_ÖSSZEFEΣÜL(T, p1, r1, p2, r2, A, p3)

```
1: n1 ← r1-p1+1
2: n2 ← r2-p2+1
3: if n1 < n2 then
4:   (p1, r1, n1) és (p2, r2, n2) felcserélése
5: end if
6: if n1>0 then
7:   q1 ← ⌊(p1+p2)/2⌋
8:   q2 ← BINÁRIS_KERES(T[q1], T, p2, r2)
9:   q3 ← p3+(q1-p1)+(q2-p2)
10:  A[q3] ← T[q1]
11:  spawn P_ÖSSZEFEΣÜL
        (T, p1, q1-1, p2, q2-1, A, p3)
12:  P_ÖSSZEFEΣÜL(T, q1+1, r1, q2, r2, A, q3+1)
13:  sync
14: end if
end procedure
```

► work:  $F_1(n) = \Theta(n)$   
 $(F_1(\alpha n) + F_1((1 - \alpha)n))$   
 $+ \Theta(\lg n))$

$$0.25 \leq \alpha \leq 0.75$$

► span:  
 $F_\infty(n) = \Theta(\lg^2 n)$   
 $(F_\infty(3n/4) + \Theta(\lg n))$

► párhuzamosság:  
 $\Theta(n/\lg^2 n)$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás  
Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

## Összefésülő rendezés

Párhuzamos összefésülés  
Batcher-féle páros-páratlan összefésülés

# Párhuzamos összefésülés

Párhuzamosság

**procedure** P\_ÖSSZEFEΣÜL(T, p1, r1, p2, r2, A, p3)

```
1: n1 ← r1-p1+1
2: n2 ← r2-p2+1
3: if n1 < n2 then
4:   (p1, r1, n1) és (p2, r2, n2) felcserélése
5: end if
6: if n1>0 then
7:   q1 ← ⌊(p1+p2)/2⌋
8:   q2 ← BINÁRIS_KERES(T[q1], T, p2, r2)
9:   q3 ← p3+(q1-p1)+(q2-p2)
10:  A[q3] ← T[q1]
11:  spawn P_ÖSSZEFEΣÜL
        (T, p1, q1-1, p2, q2-1, A, p3)
12:  P_ÖSSZEFEΣÜL(T, q1+1, r1, q2, r2, A, q3+1)
13:  sync
14: end if
end procedure
```

► work:  $F_1(n) = \Theta(n)$

$$(F_1(\alpha n) + F_1((1 - \alpha)n) + \Theta(\lg n))$$

$$0.25 \leq \alpha \leq 0.75$$

► span:

$$F_\infty(n) = \Theta(\lg^2 n)$$

$$(F_\infty(3n/4) + \Theta(\lg n))$$

► párhuzamosság:

$$\Theta(n/\lg^2 n)$$

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

**procedure** RENDEZ(A, p, r, B, s)

```
1: n ← r-p+1
2: if n = 1 then
3:   B[s] ← A[p]
4: else
5:   Legyen T[1..n] új tömb
6:   q1 ← ⌊(p+r)/2⌋
7:   q2 ← q1-p+1
8:   spawn RENDEZ(A, p, q1, T, 1)
9:   RENDEZ(A, q1+1, r, T, q2+1)
10:  sync
11:  P_ÖSSZEFÉSÜL
      (T, 1, q2, q2+1, n, B, s)
12: end if
end procedure
```

▶ work:

$$R_1(n) = \Theta(n \lg n)$$
$$(2R_1(n/2) + \Theta(n))$$

▶ span:

$$R_\infty(n) = \Theta(\lg^3 n)$$
$$(R_\infty(n/2) + \Theta(\lg^2 n))$$

▶ párhuzamosság:

$$\Theta(n / \lg^2 n)$$

Itt már egy masszív  
párhuzamosítást  
lehet elérni.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés



## Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

## Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

## Összefésülő rendezés

**procedure** RENDEZ(A, p, r, B, s)

```

1: n ← r-p+1
2: if n = 1 then
3:   B[s] ← A[p]
4: else
5:   Legyen T[1..n] új tömb
6:   q1 ← ⌊(p+r)/2⌋
7:   q2 ← q1-p+1
8:   spawn RENDEZ(A, p, q1, T, 1)
9:   RENDEZ(A, q1+1, r, T, q2+1)
10:  sync
11:  P_ÖSSZEFÉSÜL
      (T, 1, q2, q2+1, n, B, s)
12: end if
end procedure

```

## ▶ work:

$$R_1(n) = \Theta(n \lg n)$$

$$(2R_1(n/2) + \Theta(n))$$

## ▶ span:

$$R_\infty(n) = \Theta(\lg^3 n)$$

$$(R_\infty(n/2) + \Theta(\lg^2 n))$$

## ▶ párhuzamosság:

$$\Theta(n / \lg^2 n)$$

Itt már egy masszív párhuzamosítást lehet elérni.

**procedure** RENDEZ(A, p, r, B, s)

```
1: n ← r-p+1
2: if n = 1 then
3:   B[s] ← A[p]
4: else
5:   Legyen T[1..n] új tömb
6:   q1 ← ⌊(p+r)/2⌋
7:   q2 ← q1-p+1
8:   spawn RENDEZ(A, p, q1, T, 1)
9:   RENDEZ(A, q1+1, r, T, q2+1)
10:  sync
11:  P_ÖSSZEFÉSÜL
      (T, 1, q2, q2+1, n, B, s)
12: end if
end procedure
```

► work:

$$R_1(n) = \Theta(n \lg n)$$
$$(2R_1(n/2) + \Theta(n))$$

► span:

$$R_\infty(n) = \Theta(\lg^3 n)$$
$$(R_\infty(n/2) + \Theta(\lg^2 n))$$

► párhuzamosság:

$$\Theta(n / \lg^2 n)$$

Itt már egy masszív  
párhuzamosítást  
lehet elérni.

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Összefésülő rendezés

**procedure** RENDEZ(A, p, r, B, s)

```
1: n ← r-p+1
2: if n = 1 then
3:   B[s] ← A[p]
4: else
5:   Legyen T[1..n] új tömb
6:   q1 ← ⌊(p+r)/2⌋
7:   q2 ← q1-p+1
8:   spawn RENDEZ(A, p, q1, T, 1)
9:   RENDEZ(A, q1+1, r, T, q2+1)
10:  sync
11:  P_ÖSSZEFÉSÜL
      (T, 1, q2, q2+1, n, B, s)
12: end if
end procedure
```

► work:

$$R_1(n) = \Theta(n \lg n)$$
$$(2R_1(n/2) + \Theta(n))$$

► span:

$$R_\infty(n) = \Theta(\lg^3 n)$$
$$(R_\infty(n/2) + \Theta(\lg^2 n))$$

► párhuzamosság:

$$\Theta(n / \lg^2 n)$$

Itt már egy masszív  
párhuzamosítást  
lehet elérni.

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefésülő rendezés

Párhuzamos összefésülés

Batcher-féle páros-páratlan összefésülés

# Batcher-féle páros-páratlan összefésülés

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

Összefűző rendezés

Párhuzamos összefésülés  
Batcher-féle páros-páratlan összefésülés

Az ötlet:

$A = \{ \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{9}, \underline{12}, \underline{14}, \underline{19}, \underline{20} \};$   
 $B = \{ \underline{6}, \underline{7}, \underline{8}, \underline{10}, \underline{11}, \underline{13}, \underline{15}, \underline{16}, \underline{17}, \underline{18} \};$

$u = \{ \underline{1}, \underline{3}, \underline{5}, \underline{7}, \underline{10}, \underline{12}, \underline{13}, \underline{16}, \underline{18}, \underline{19} \};$   
 $v = \{ \underline{2}, \underline{4}, \underline{6}, \underline{8}, \underline{9}, \underline{11}, \underline{14}, \underline{15}, \underline{17}, \underline{20} \};$

# Batcher-féle páros-páratlan összefésülés

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmus  
Párhuzamos algoritmus

Összefűző rendezés

Párhuzamos összefésülés  
Batcher-féle páros-páratlan összefésülés

Az ötlet:

$A = \{ \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{9}, \underline{12}, \underline{14}, \underline{19}, \underline{20} \}$ ;

$B = \{ \underline{6}, \underline{7}, \underline{8}, \underline{10}, \underline{11}, \underline{13}, \underline{15}, \underline{16}, \underline{17}, \underline{18} \}$ ;

$u = \{ \underline{1}, \underline{3}, \underline{5}, \underline{7}, \underline{10}, \underline{12}, \underline{13}, \underline{16}, \underline{18}, \underline{19} \}$ ;

$v = \{ \underline{2}, \underline{4}, \underline{6}, \underline{8}, \underline{9}, \underline{11}, \underline{14}, \underline{15}, \underline{17}, \underline{20} \}$ ;

# Batcher-féle páros-páratlan összefésülés

Párhuzamosság

Kósa Márk  
Pánovics János  
Szathmáry László  
Halász Gábor



Mátrix-szorzás

Soros algoritmus  
Strassen algoritmusa  
Párhuzamos algoritmus

Összefűző rendezés

Párhuzamos összefésülés  
Batcher-féle páros-páratlan összefésülés

Az ötlet:

$$A = \{ \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{9}, \underline{12}, \underline{14}, \underline{19}, \underline{20} \};$$

$$B = \{ \underline{6}, \underline{7}, \underline{8}, \underline{10}, \underline{11}, \underline{13}, \underline{15}, \underline{16}, \underline{17}, \underline{18} \};$$

$$u = \{ \underline{1}, \underline{3}, \underline{5}, \underline{7}, \underline{10}, \underline{12}, \underline{13}, \underline{16}, \underline{18}, \underline{19} \};$$

$$v = \{ \underline{2}, \underline{4}, \underline{6}, \underline{8}, \underline{9}, \underline{11}, \underline{14}, \underline{15}, \underline{17}, \underline{20} \};$$



Mátrix-szorzás

Soros algoritmus

Strassen algoritmus

Párhuzamos algoritmus

Összefűző rendezés

Párhuzamos összefűzés

Batcher-féle páros-páratlan összefűzés

# Batcher-féle páros-páratlan összefűzés

**procedure** B\_ÖSSZEFÉSÜL( $T, p1, r1, p2, r2, A, p3, k$ )

```

1: n1 ← ⌈(r1-p1+1)/k⌉
2: n2 ← ⌈(r2-p2+1)/k⌉
3: if p1>r1 then
4:   parallel for i ← 0 to n2-1 do
5:     A[p3+k*i] ← T[p2+k*i]
6:   end for
7: else if p2>r2 then
8:   parallel for i ← 0 to n1-1 do
9:     A[p3+k*i] ← T[p1+k*i]
10:  end for
11: end if
12: else
13:   kk ← k+k
14:   spawn B_ÖSSZEFÉSÜL( $T, p1, r1, p2+k, r2, A, p3, kk$ )
15:   B_ÖSSZEFÉSÜL( $T, p1+k, r1, p2, r2, A, p3, kk$ )
16:   sync
17:   parallel for i ← 1 to ⌊(n1+n2)/2⌋ do
18:     if A[p3+i*kk]>A[p3+i*kk] then
19:       A[p3+i*kk-kk] és A[p3+i*kk] felcserélése
20:     end if
21:   end for
22: end if
end procedure
```

Az ötlet:

 $A = \{ \underline{1}, \underline{2}, \underline{3}, \underline{4}, \underline{5}, \underline{9}, \underline{12}, \underline{14}, \underline{19}, \underline{20} \}$ ; $B = \{ \underline{6}, \underline{7}, \underline{8}, \underline{10}, \underline{11}, \underline{13}, \underline{15}, \underline{16}, \underline{17}, \underline{18} \}$ ; $U = \{ \underline{1}, \underline{3}, \underline{5}, \underline{7}, \underline{10}, \underline{12}, \underline{13}, \underline{16}, \underline{18}, \underline{19} \}$ ; $V = \{ \underline{2}, \underline{4}, \underline{6}, \underline{8}, \underline{9}, \underline{11}, \underline{14}, \underline{15}, \underline{17}, \underline{20} \}$ ;