SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Dr. Farkas Richárd

SzTE TTIK, Számítógépes Algoritmusok és Mesterséges Intelligencia tanszék

www.inf.u-szeged.hu/~rfarkas

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Miért tanuljak “Algát”?

2020. szeptember

1. hét – 1. videó

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok

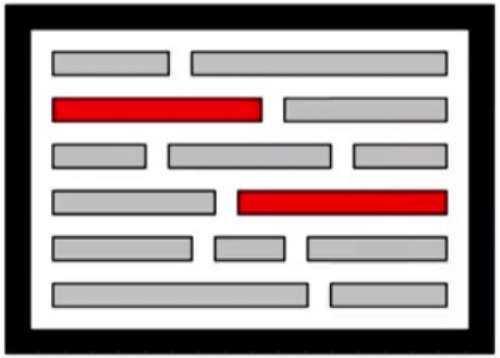
Algoritmusnak nevezünk bármilyen ***jól definiált számítási eljárás***t, amely ***bemenetként*** bizonyos értéket vagy értékeket kap és ***kimenetként*** bizonyos értéket vagy értékeket állít elő.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmus?

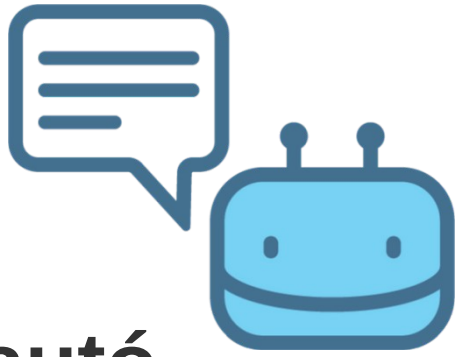
- **Jeleníts meg egy képet a weblapon**
  - **túl triviális, nem érdekes itt...**

- **Egy adott szó szerepel-e egy fájlban**
  - **Ha sebesség fontos, okos megoldás kell!**



Algoritmus?

- **Chatrobot**



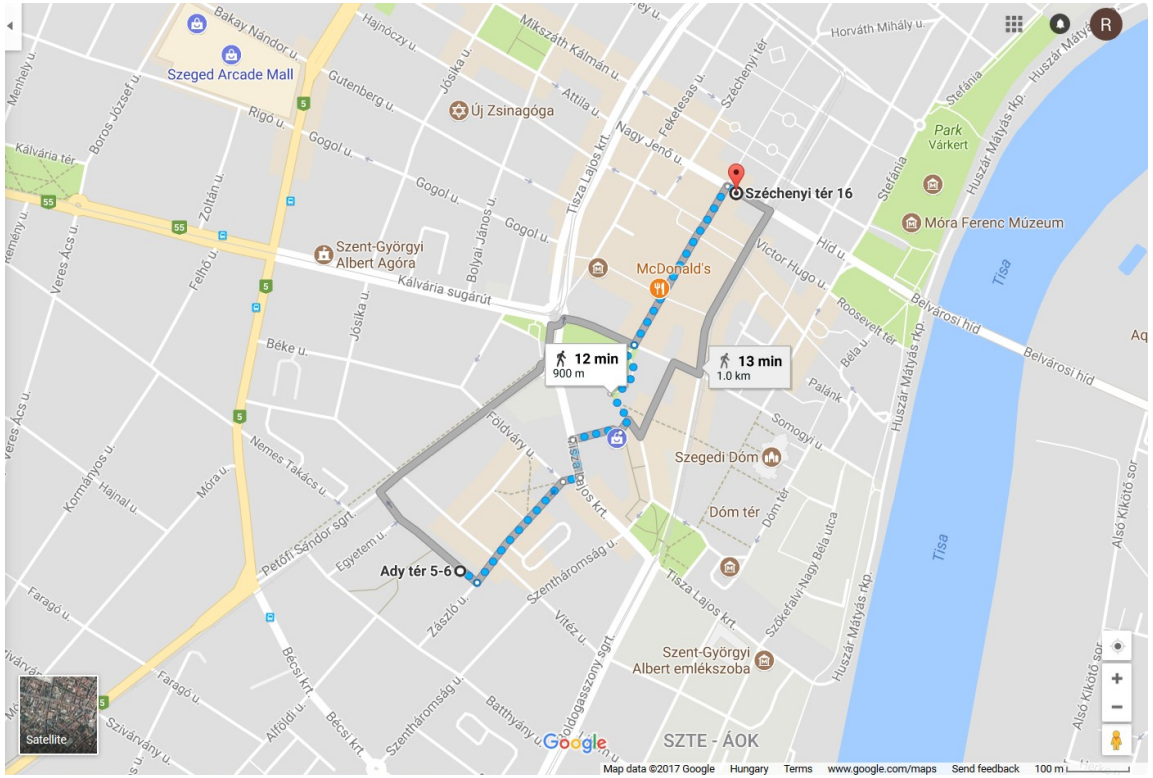
- **Önvezető autó**

  - **Nem jól definiált!**
  - **Mesterséges Intelligencia**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmus!

- **Legrövidebb út keresése**



- **Nem triviális a megoldás**
- **Egyszerű megoldás túl lassú**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Adatszerkezetek

Az ***adatszerkezet*** adatok tárolására és szervezésére szolgáló módszer, amely lehetővé teszi a hozzáférést és módosításokat

egfelelő algoritmushoz

egfelelő adatszerkezetet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Miért tanuljak algoritmusokat?

- **Mindenki fogja használni!**
- **BigData – skálázódás fontos!**

Miért tanuljak algoritmusokat?

- **Algoritmikus gondolkodás!**
  - Algoritmus eddig megoldatlan problémára?
  - Megfelelő algoritmusok és adatszerkezetek kiválasztása
  - Gondoljuk végig a helyességet és hatékonyságot!

Miért tanuljak algoritmusokat?

- Nyelvfüggetlen programozói szemlélet

> “You can practice really hard for two years to become a great programmer and you can practice for 10 years to become an excellent programmer. Or you can practice for two years and take an algorithms course and become an excellent programmer

- Absztraktabb gondolkodás

[How to: Work at Google — Example Coding/Engineering Interview]

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Algoritmusok helyesség és futásidő elemzése

2020. szeptember

1. hét – 2. videó
S01E02

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok tervezése

- Értsük meg mélyen a feladatot!
- Nincs általános módszertan algoritmizálásra
- A félév folyamán
  - megismerünk hasznos technikákat
  - látunk számtalan algoritmust különböző problémákra

ezek mintául szolgálhatnak a jövőben.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok elemzése

- Helyesség
- Hatékonyság:
  - előre meg akarjuk mondani, milyen erőforrásokra lesz szüksége az algoritmusnak
  - ***számítási idő***, memória, sávszélesség
- Cél: algoritmusok összehasonlítása

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Futási idő

- Milyen hardver?
- CPU? GPU? Felhő?

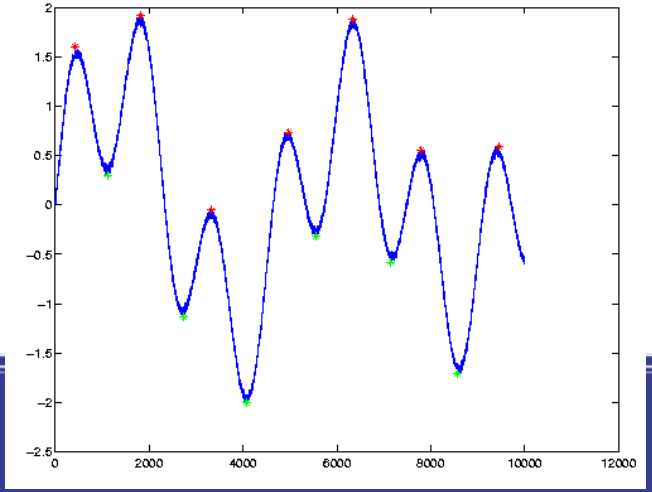
- ***Futási idő***: egy bizonyos bemenetre a végrehajtott (gépfüggetlen) alapműveletek vagy ”lépések” száma
- Feltesszük, hogy egy kód egy sorának végrehajtásához konstans mennyiségű idő szükséges

Feladat: csúcs keresés

Bemenet: egész számok tömbje

Találjunk és adjunk vissza ***egy*** „csúcs”ot ha létezik!

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb



Forrás: MIT Introduction to Algorithms, Lecture 1.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Feladat: csúcs keresés

1	3	4	3	5	1	3

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb

Létezik mindig csúcs?

„nem kisebb” helyett „nagyobb” egy másik algoritmust igényelhet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső alg

- Balról jobbra vizsgáljunk meg minden elemet és ha csúcsot találunk térjünk vissza

1	3	4	3	5	1	3

- Pszeudokód:

for *j* ← 1 **to** *hossz*[*A*]
  do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]
    return j

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső alg

```
if hossz[A] < 1
  return nil
if hossz[A] = 1
  return 1
else if A[1] ≥ A[2]
  return 1
else if A[hossz[A]] ≥ A[hossz[A]-1]
  return hossz[A]
for j ← 2 to hossz[A]-1
  do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]
        return j
```

Egyszerű csúcs kereső alg

```
public static Integer find_a_peak(int[] a) throws IllegalArgumentException {
  //határesetek kezelése:
  if(a == null)
    throw new IllegalArgumentException();
  if(a.length < 1)
    return null;
  if(a.length == 1)
    return 0;
  if(a[0] >= a[1])
    return 0;
  if(a[a.length-1] >= a[a.length-2])
    return a.length-1;

  //algoritmus:
  for(int j = 2 ; j < a.length-1; ++j)
    if((a[j] >= a[j-1])  && (a[j] >= a[j+1]))
      return j;

  //soha nem érhetünk ide:
  return null;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok (és implementáció) helyességének tesztelése

- **Tesztesetek (unit test)**
- **Először elvárt bemenetekre**

```
find_a_peak(new int[]{1,3,4,3,5,1,3});
```

- **Aztán határesetekre is!**

```
find_a_peak(new int[]{7,3,4,3,5,1,3});
find_a_peak(new int[]{1,3,4,3,5,1,5});
find_a_peak(new int[]{1,3});
find_a_peak(new int[]{1});
find_a_peak(new int[]{});
find_a_peak(null);
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső algoritmus elemzése

	költség	végrehajtási szám
if *hossz*[A] < 1	c_1	1
return nil	c_2	1
if *hossz*[A] = 1	c_3	1
return 1	c_4	1
else if *A*[1] ≥ *A*[2]	c_5	1
return 1	c_6	1
else if A[*hossz*[A]] ≥ A[*hossz*[A]-1]	c_7	1
return *hossz*[A]	c_8	1
for *j* ← 2 **to** *hossz*[*A*]-1	c_9	n-2
do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]	c_{10}	n-2
return j	c_{11}	1

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Legjobb, átlagos esetek elemzése

- Bemenet mérete konstans ***n***
- Algoritmus össz futás idejét **T(n)**-el jelöljük
- Legjobb esetben az első elem csúcs, ekkor T(n)=2 minden *n*-re
- Az átlagos vagy várható futásidőt nagyon nehéz kiszámolni… (valószínűségi elemzés)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Legrosszabb eset elemzése

- Inkább legyünk pesszimisták!
- Az algoritmus legrosszabb futási ideje bármilyen bemenetre a futási idő felső korlátja.
  - Gyakran előfordul a legrosszabb eset
  - Az átlagos eset gyakran nagyjából ugyanolyan rossz, mint a legrosszabb eset.

Egyszerű csúcs kereső algoritmus elemzése

	költség	végrehajtási szám
if *hossz*[A] < 1	c_1	1
return nil	c_2	1
if *hossz*[A] = 1	c_3	1
return 1	c_4	1
else if *A*[1] ≥ *A*[2]	c_5	1
return 1	c_6	1
else if A[*hossz*[A]] ≥ A[*hossz*[A]-1]	c_7	1
return *hossz*[A]	c_8	1
for *j* ← 2 **to** *hossz*[*A*]-1	c_9	n-2
do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]	c_{10}	n-2
return j	c_{11}	1

Legrosszabb esetben:

$$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső algoritmus elemzése

Legrosszabb esetben:

$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$

$T(n) = (c_9 + c_{10})(n-2) + c_1 + c_3 + c_5 + c_7 + c_{11}$

- Tényleges futásidők helyett *c* konstansok
- Nagy *n* esetén $c_1 + c_3 + c_5 + c_7 + c_{11}$ elhanyagolható
- Hatékonyság szempontjából minket az érdekel, hogy hogyan skálázódik az algoritmus, azaz milyen a ***futásidő növekedési sebessége***
- Elég csak a fő tagot figyelembe venni: $(c_9 + c_{10})(n-2)$
- Nagyságrendileg T(n) *n* lineáris függvénye

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok stressz tesztelése

```
public static void running_time(){
  int size = 100000000;
  int[] ints = new int[size];
  ints[size-1] = 0;
  for(int i=0;i<size-1;++i)
    ints[i]=i;
  long time = System.currentTimeMillis();
  find_a_peak(ints);
  System.out.println((System.currentTimeMillis()-
time) + " msec");
}
```

```
stdout 1M:  1 msec
stdout 10M:  9 msec
stdout 100M: 78 msec
```

Feladat: csúcs keresés

- Feladat ugyanaz!
- Tudunk hatékonyabb megoldást találni?

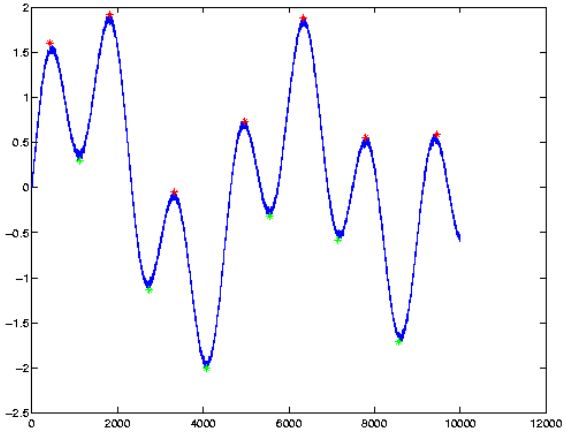
- Gondoltam egy számra 1 és 2^{32} közt

- **Oszd meg és uralkodj!**

Felező csúcskereső algoritmus

Vizsgáljuk meg a középső elemet. Ha nem csúcs akkor egyik szomszéd nagyobb, vizsgáljuk a bemenet felét a ezen szomszéd irányába!

1	3	4	3	5	1	3



Helyes?

SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Felező csúcskereső algoritmus

```
public static Integer find_a_peak(int[] a){
  // határesetek kezelése ugyanaz, mint a lassú verzióban!

  // algoritmus:
  int lo = 1;
  int hi = a.length - 2;
  while (lo <= hi) {
    //kell lennie csucsnak az a[lo..hi]-ban
    int mid = lo + (hi - lo) / 2;
    if (a[mid] < a[mid - 1])
      hi = mid - 1;
    else if (a[mid] < a[mid + 1])
      lo = mid + 1;
    else
      return mid;
  }

  // Soha nem érhetünk ide:
  return null;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Felező csúcskereső algoritmus futási ideje

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

$T(1) = c$

$T(16) = T(8) + c = T(4) + c + c = T(2) + c + c + c = T(1) + c + c + c + c = c + c + c + c + c = 5c$

$T(n) = (\log_2 n + 1)\, c$

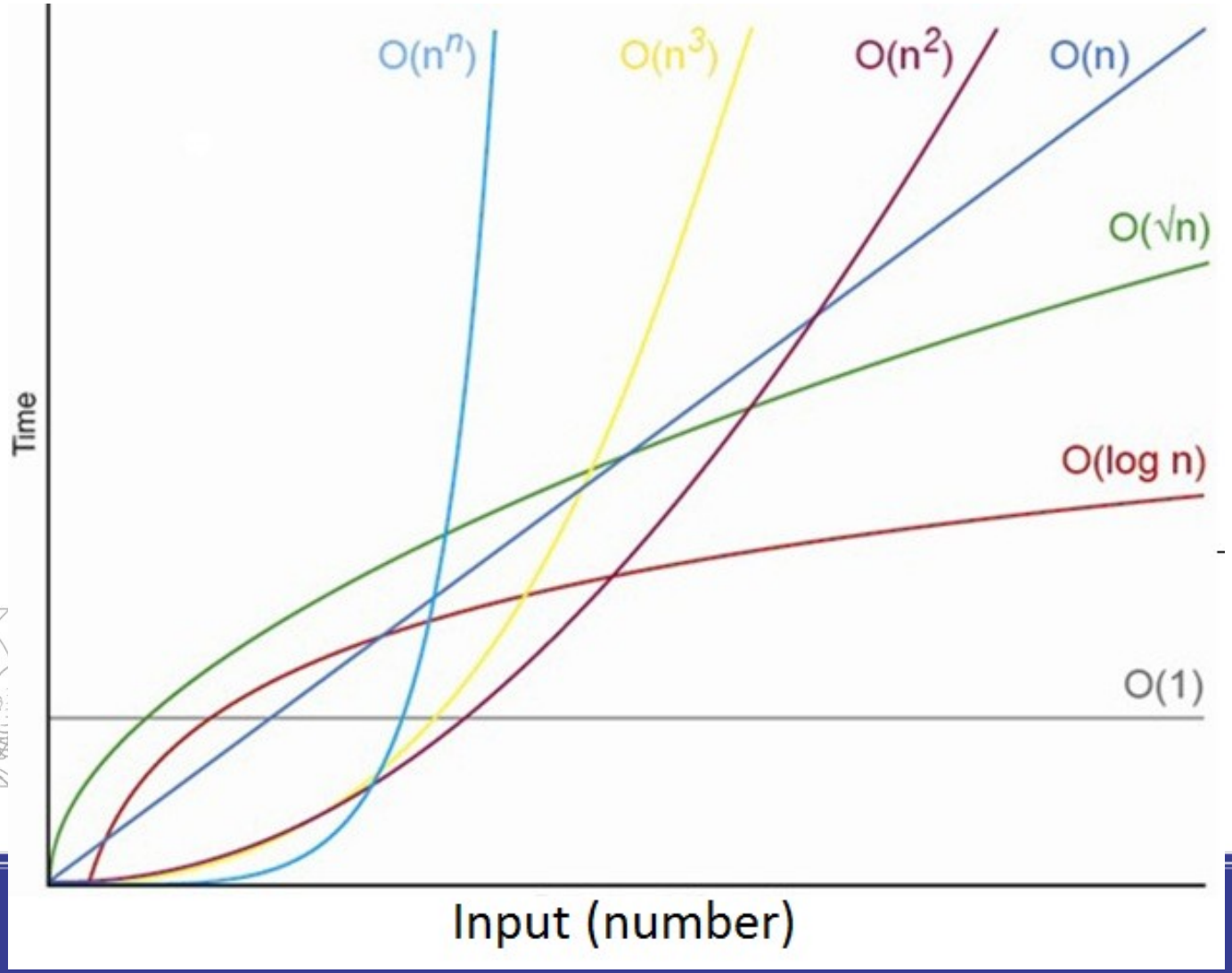
Legrosszabb esetben is T(n) *n* függvényében logaritmikus növekedési sebességű

```
stdout 100M: 0 msec
```

Megj: $\log_a b = \log_c b / \log_c a$ miatt a logaritmus alapja nem számít, hiszen az „csak” konstans szorzó. Nem írjuk ki a kurzuson, hanem mindig 2-es alapú logaritmussal számolunk.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Tipikus aszimptotikus felső korlátok



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Feladat: 2D csúcs keresés

Bemenet: egész számok két dimenziós tömbje (mátrix)

Találjunk és adjunk vissza ***egy*** „csúcs”ot ha létezik!

Csúcs: olyan elem a 2D tömbben ami minden szomszédjánál nem kisebb

Feladat: 2D csúcs keresés
Értsük meg a feladatot!

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

méret: n x n

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Mohó hegymászó 2D csúcskereső

Induljunk valahonnan (középről vagy egyik sarokból), minden lépésben lépjünk az egyik nagyobb szomszédra. Ha nincs nagyobb szomszéd csúcsot találtunk.

Minden n mérethez kezdőponthoz és lépési startégiához lehet adni olyan bemenetet amire az egész mátrixot be fogja járni (legrosszabb eset) azaz T(n) n^2 növekedési sebességű.

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés 1D csúcskeresésre visszavezetve

Válasszuk a középső oszlopot. Keressünk 1D csúcsot ebben. A talált 1D csúcs sorában keressünk újra 1D csúcsot. Ez 2D csúcs lesz.

Legrosszabb esetben is log*n* növekedési sebességű, hiszen log*n* idő alatt találunk 1D csúcsot

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés 1D csúcskeresésre visszavezetve

8	4	1
7	5	3
2	3	1

Egy helyes de nem hatékony algoritmus ér valamit, nem úgy mint egy helytelen de hatékony ◀◀

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés felezéssel

Válasszuk a középső oszlopot. Keressünk meg egy maximális elemet ebben. Ha ennek bal vagy jobb szomszédja nagyobb, mint az elem ismételjük meg az eljárást ebben a fél mátrixban. Ha bal és jobb szomszédok nem kisebbek 2D csúcsot találtunk.

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés felezéssel

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

Helyes?

2D csúcskeresés felezéssel futási ideje

Ha csak egy oszlop van a maximum elem keresés legrosszabb időben n lépést igényel.

$T(n, 1) = cn$

$T(n, m) = T(n, m/2) + cn$

$T(n, m) = \log n \cdot cn$

Legrosszabb esetben $n \cdot \log n$ növekedési sebességű az algoritmus

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Feladat: 2D csúcs keresés

Mohó hegymászó ✓ n^2

Visszavezetés 1D-re ✗ $\log n$

Mátrixfelezés ✓ $n \cdot \log n$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összegzés

- Algoritmusok tervezése
  - Értsük meg a problémát/feladatot
  - Legegyszerűbb megoldást elemezzük
  - Ha kell tervezzünk hatékonyabb algoritmust!
- Algoritmusok elemzése
  - helyesség
  - hatékonyság (skálázódás)

eszköze: legrosszabb eset futásidő növekedési sebesség

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Az Ordó jelölés

2020. szeptember

1. hét – 3. videó
S01E03

Algoritmusok futási idő elemzése

- Bemenet mérete konstans ***n***
- Algoritmus össz futás idejét **T(n)**-el jelöljük

$$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITATIS SCIENTIARUM SZEGEDIENSIS

Aszimptotikus hatékonyság

Ha a bemenet mérete elég nagy, akkor az algoritmus futási idejének csak a nagyságrendje lényeges

Theta:

$$\Theta(g(n)) = \{f(n) : \text{ létezik } c_1, c_2 \text{ és } n_0 \text{ pozitív állandó úgy, hogy } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ teljesül minden } n \ge n_0 \text{ esetén}\}.$$

Theta:

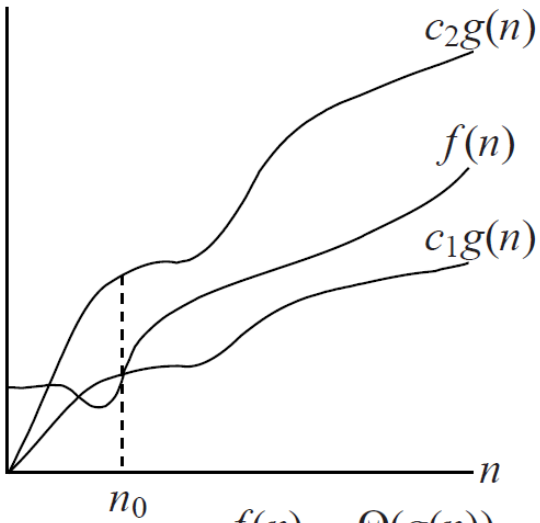
$\Theta(g(n)) = \{f(n) :$ létezik c_1, c_2 és n_0 pozitív állandó úgy, hogy
$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ teljesül minden $n \ge n_0$ esetén$\}$.

Ordó:

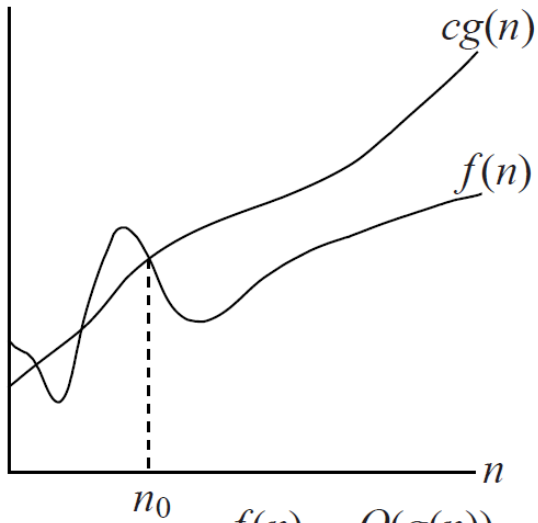
$O(g(n)) = \{f(n) :$ létezik c és n_0 pozitív állandó úgy, hogy
$0 \le f(n) \le cg(n)$ teljesül minden $n \ge n_0$ esetén$\}$.

Omega:

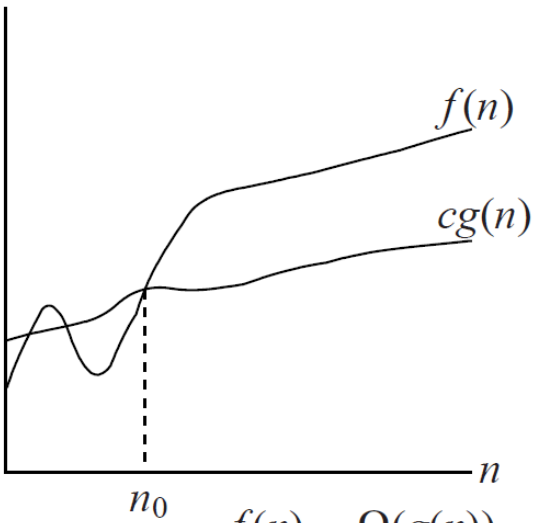
$\Omega(g(n)) = \{f(n) :$ létezik c és n_0 pozitív konstans úgy, hogy
$0 \le cg(n) \le f(n)$ teljesül minden $n \ge n_0$ esetén$\}$.



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



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



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

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Aszimptotikus felső korlát

Ordó

$$O(g(n)) = \{f(n) : \text{ létezik } c \text{ és } n_0 \text{ pozitív állandó úgy, hogy } 0 \le f(n) \le cg(n) \text{ teljesül minden } n \ge n_0 \text{ esetén}\}.$$

Ha nem mondunk mást $O(n)$ azt jelenti, hogy a vizsgált algoritmus ***legrosszabb*** esetben is aszimptotikusan lineáris időben lefut.

Megj. egy lineáris fgv. is $O(n^2)$-ben van

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

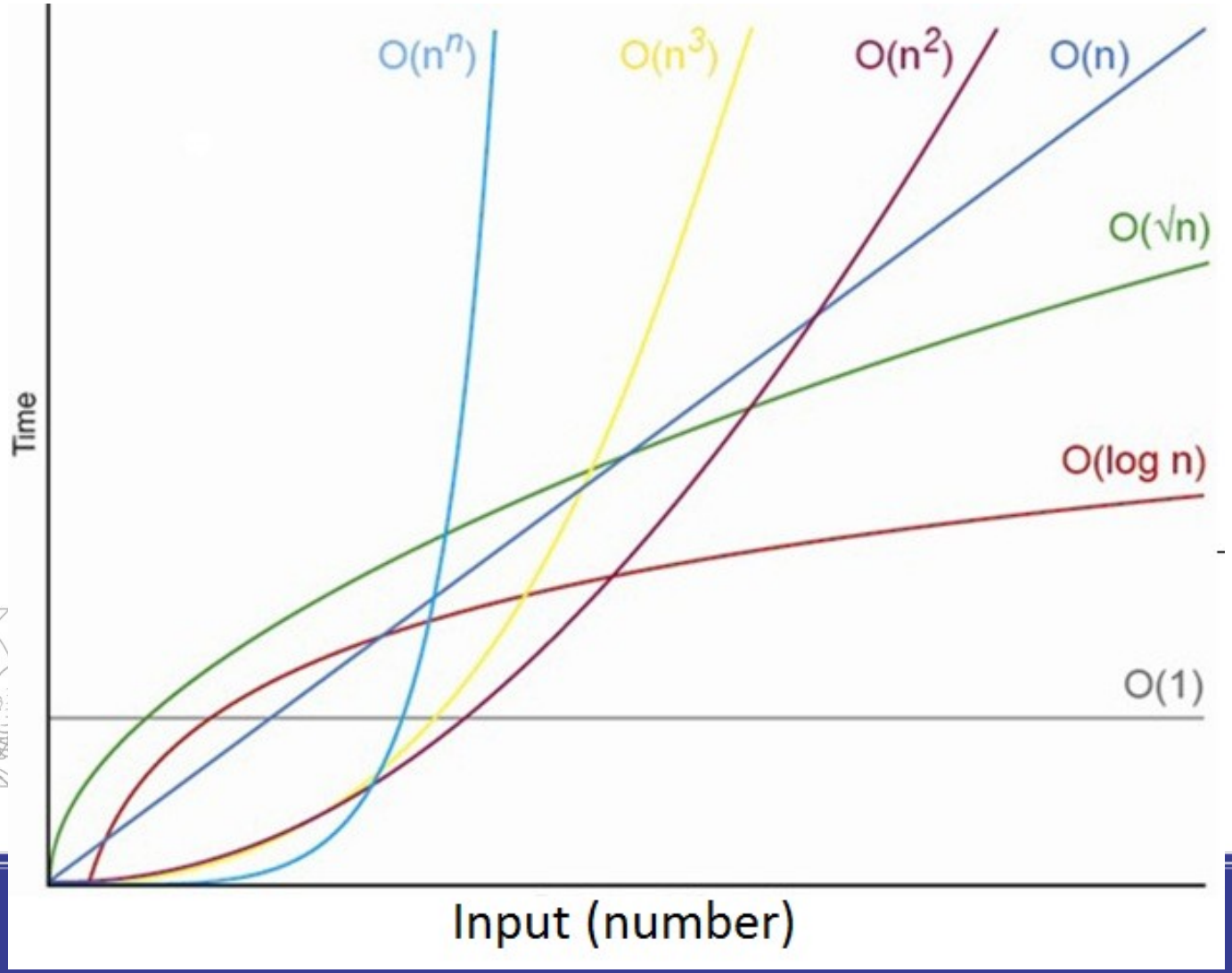
Aszimptotikus felső korlát

$T(n)=9999n^3 + \sin n + 78n\log n = O(n^3)$

- Architektúrától független
- Fontos konstans szorzókat elfed!
- Kényelmes használni, de ne feledkezzünk meg róla, hogy ez csak ***aszimptotikus*** korlát!

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Tipikus aszimptotikus felső korlátok



SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Oszd meg és uralkodj!

2020. szeptember

2. hét – 1. videó
S02E01

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Gondoltam egy számra [1..100]

50< ?

Nagyon hasonló,
kisebb részprobléma

Igen

Gondoltam egy számra [51..100]

oszd-meg-és-uralkodj megközelítés

A feladatot ***több részfeladat***ra osztjuk, amelyek hasonlóak az eredeti feladathoz, de méretük kisebb, rekurzív módon megoldjuk a részfeladatokat, majd összevonjuk ezeket a megoldásokat, hogy az eredeti feladatra megoldást adjanak.

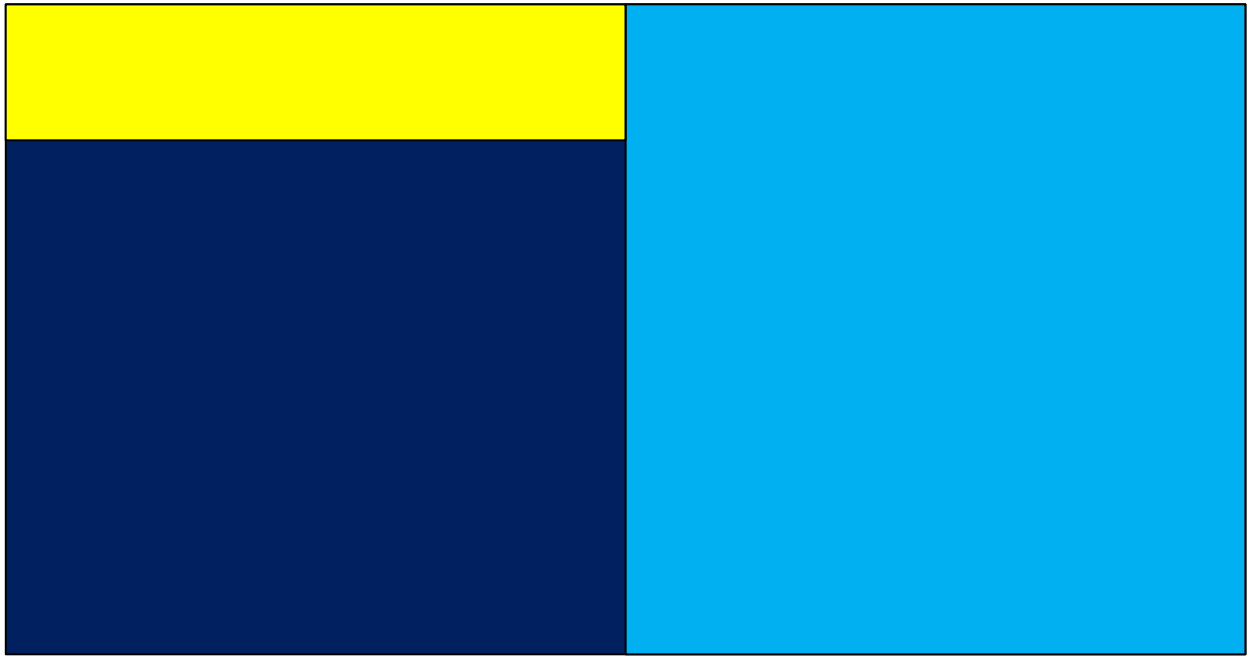
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Oszd meg és uralkodj

Feladat

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

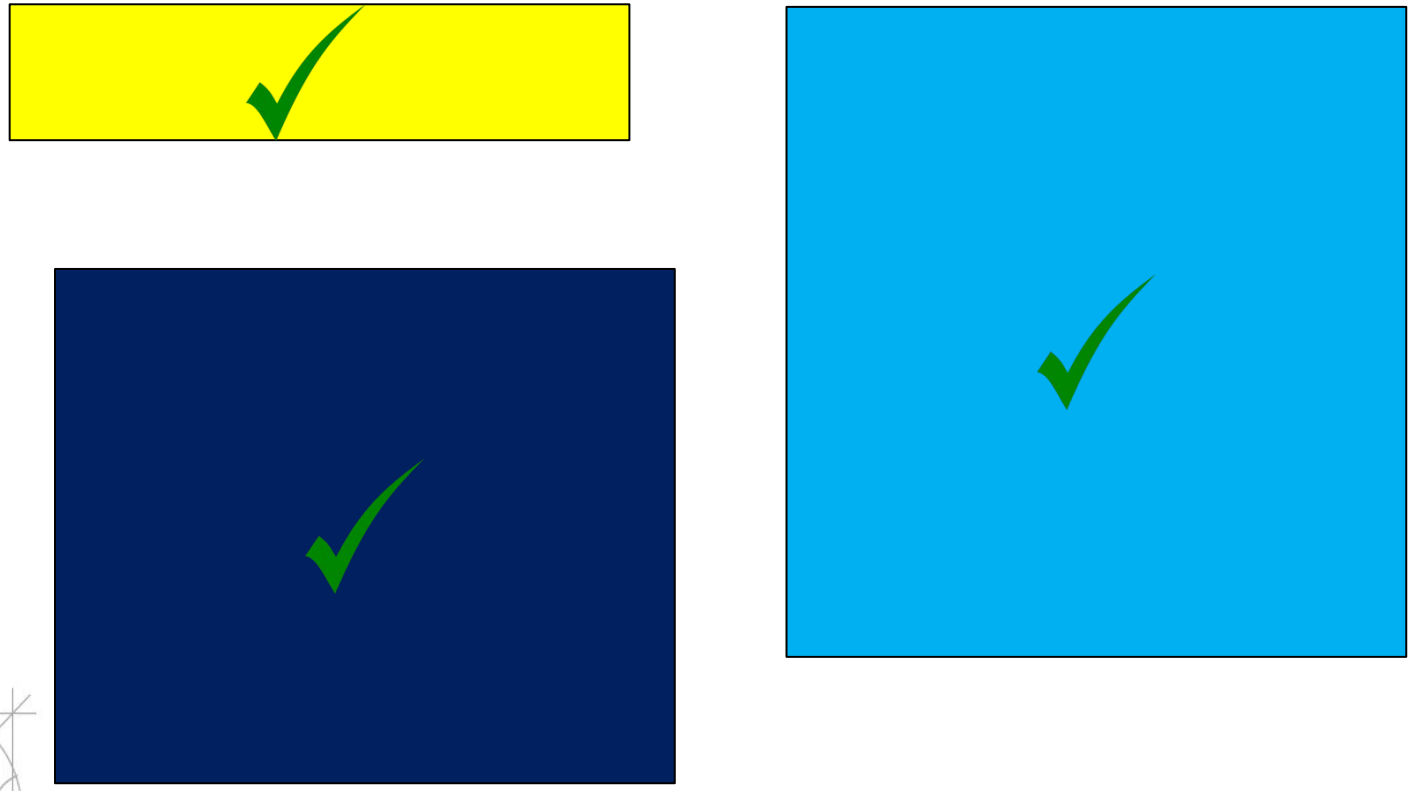
Oszd meg és uralkodj



… több részfeladatra osztjuk …

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

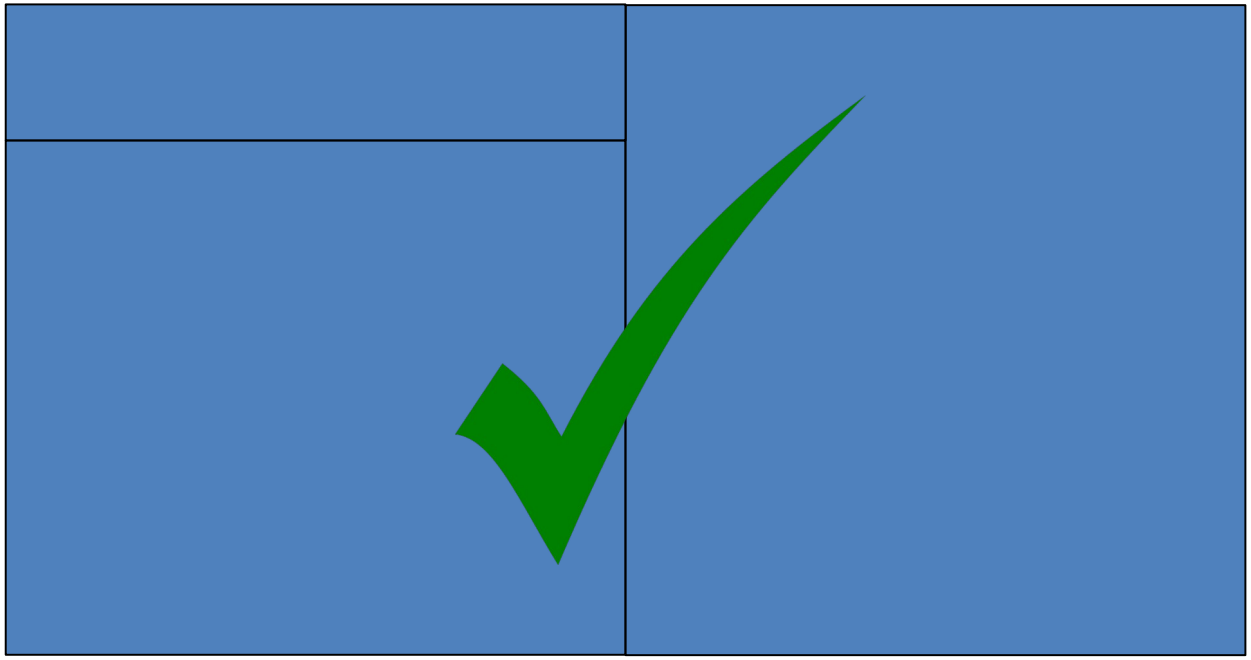
Oszd meg és uralkodj



… megoldjuk a részfeladatokat …

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

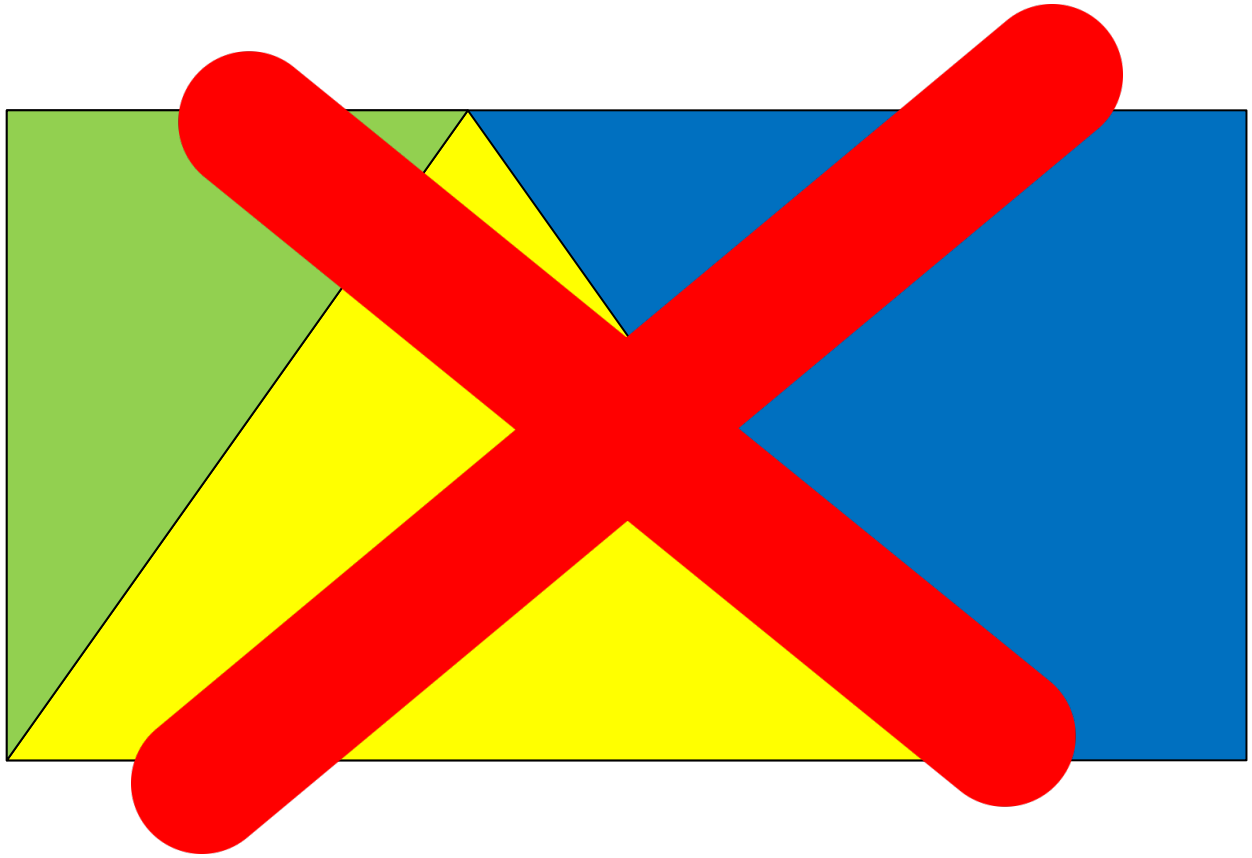
Oszd meg és uralkodj



… majd összevonjuk ezeket a megoldásokat, hogy az eredeti feladatra megoldást adjanak

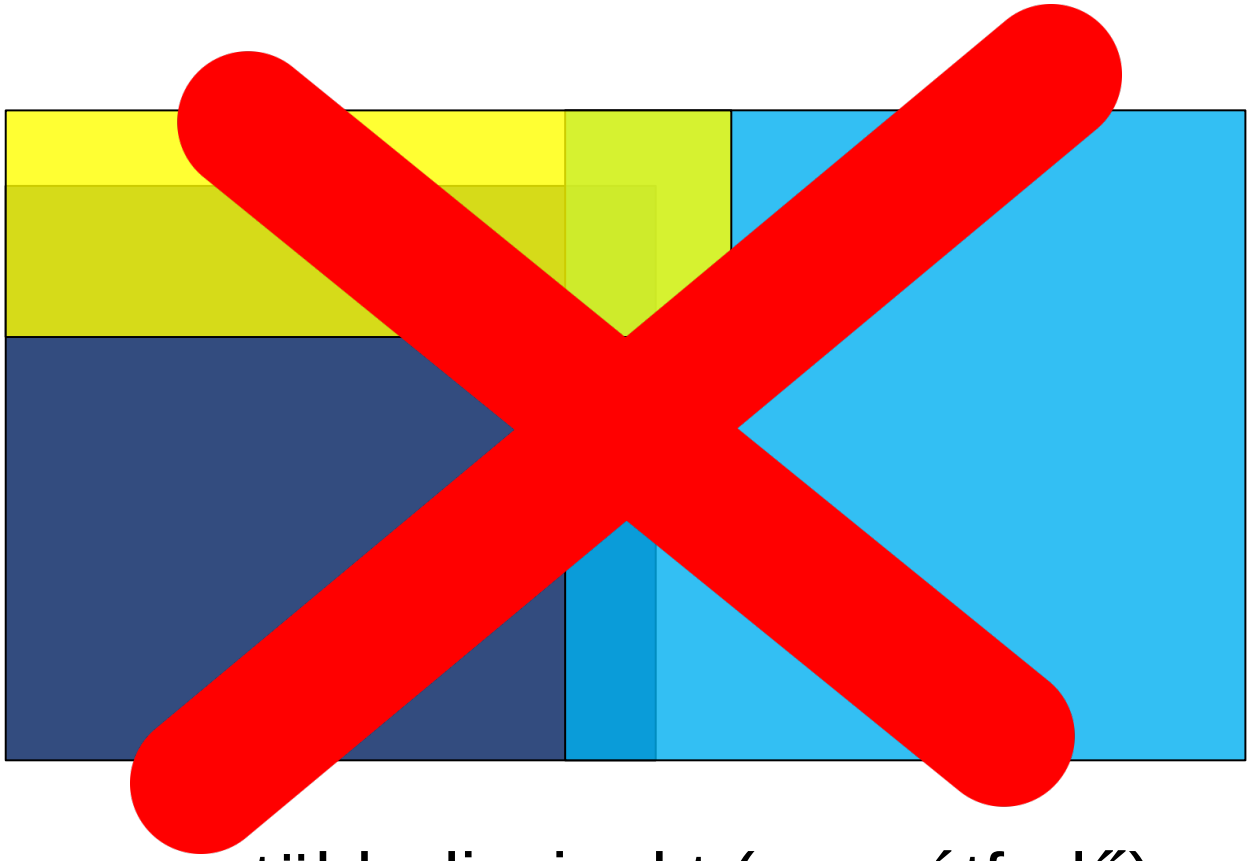
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Oszd meg és uralkodj?



… amelyek hasonlóak az eredeti feladathoz …

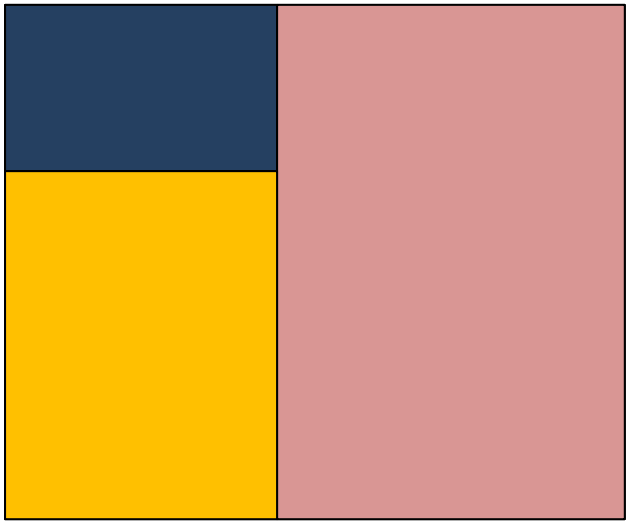
Oszd meg és uralkodj?



… több diszjunkt (nem átfedő) részfeladatra osztjuk …

Oszd meg és uralkodj

Hogyan oldjuk meg a részfeladatokat?



… ***rekurzív módon*** megoldjuk a részfeladatokat …

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Oszd meg és uralkodj lépései

- **Felosztás**: hogyan osztjuk a feladatot több részfeladatra
- **Uralkodás**: a részfeladatokat rekurzív módon megoldjuk. Ha a részfeladatok mérete elég kicsi, akkor közvetlenül megoldja a részfeladatokat.
- **Összevonás**: a részfeladatok megoldásait összevonjuk az eredeti feladat megoldásává

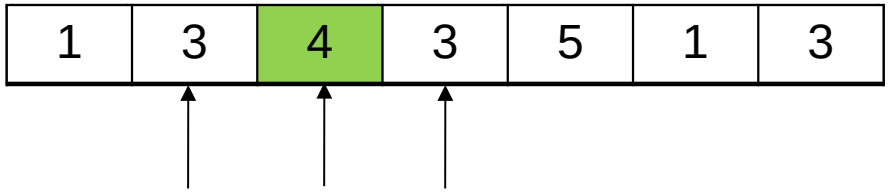
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

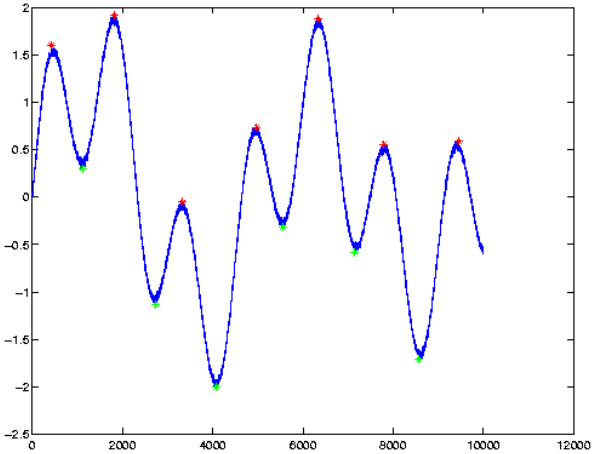
Felező csúcskereső algoritmus – egy oszd-meg-és-uralkodj algoritmus

Felosztás: n elemű sorozatot felosztjuk két (n-1)/2 elemű részsorozatra

Uralkodás: elég az egyik részsorozatban csúcsot keresni, tegyük rekurzívan

Összevonás: ha csúcsot találtunk adjuk vissza





SZEGEDI TUDOMÁNYEGYETEM
SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Rekurzió

2020. szeptember

2. hét – 2. videó
S02E02

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

oszd-meg-és-uralkodj algoritmus

$\neq$

rekurzió

Felező csúcskereső algoritmus – iteratív implementáció

```
public static Integer find_a_peak(int[] a){
  // határesetek kezelése ugyanaz, mint a lassú verzióban!

  // algoritmus:
  int lo = 1;
  int hi = a.length - 2;
  while (lo <= hi) {
    //kell lennie csucsnak az a[lo..hi]-ban
    int mid = lo + (hi - lo) / 2;
    if (a[mid] < a[mid - 1])
      hi = mid - 1;
    else if (a[mid] < a[mid + 1])
      lo = mid + 1;
    else
      return mid;
  }

  // Soha nem érhetünk ide:
  return null;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

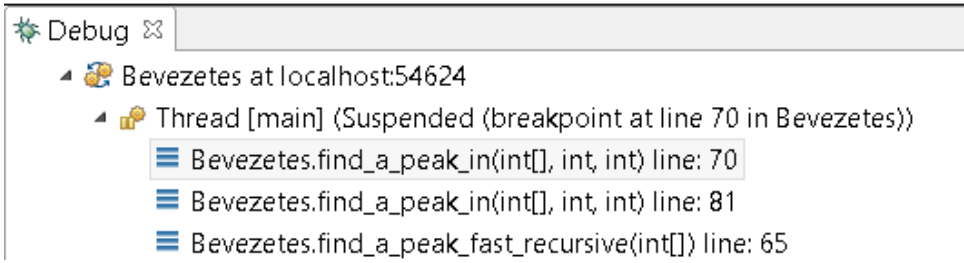
Felező csúcskereső algoritmus – rekurzív implementáció

```
protected static Integer find_a_peak_in(int[] a, int lo, int hi)
{
  if (hi-lo <= 1)
    return lo;
  if (a[lo] >= a[lo+1])
    return lo;
  if (a[hi] >= a[hi - 1])
    return hi;

// algoritmus:
// kell lennie csucsnak az a[lo..hi]-ban
  int mid = lo + (hi - lo) / 2;
  if (a[mid] < a[mid - 1])
    return find_a_peak_in(a, lo, mid - 1);
  else if (a[mid] < a[mid + 1])
    return find_a_peak_in(a, mid + 1, hi);
  return mid;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Felező csúcskereső algoritmus – rekurzív implementáció

```
public static Integer find_a_peak_fast_recursive(int[] a) {
  if (a == null)
    throw new IllegalArgumentException();
  if (a.length < 1)
    return null;

  return find_a_peak_in(a, 0, a.length-1);
}

find_a_peak_fast_recursive(new int[]{1,3,4,3,5,1,0});
find_a_peak_in(a,0,6)
find_a_peak_in(a,0,2)
```



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Oszd meg és uralkodj algoritmusok iteratív vs rekurzív implementáció

- (Elméletben) minden oszd-meg-és-uralkodj algoritmust le lehet iteratívan és rekurzívan is programozni
- Rekurzív implementáció előnye:
  - olvashatóbb kód
  - intuitív megvalósítás
- Rekurzív implementáció hátrányai:
  - memóriallokáció eljáráshívásokra (stack)
  - nehezebb debugolni

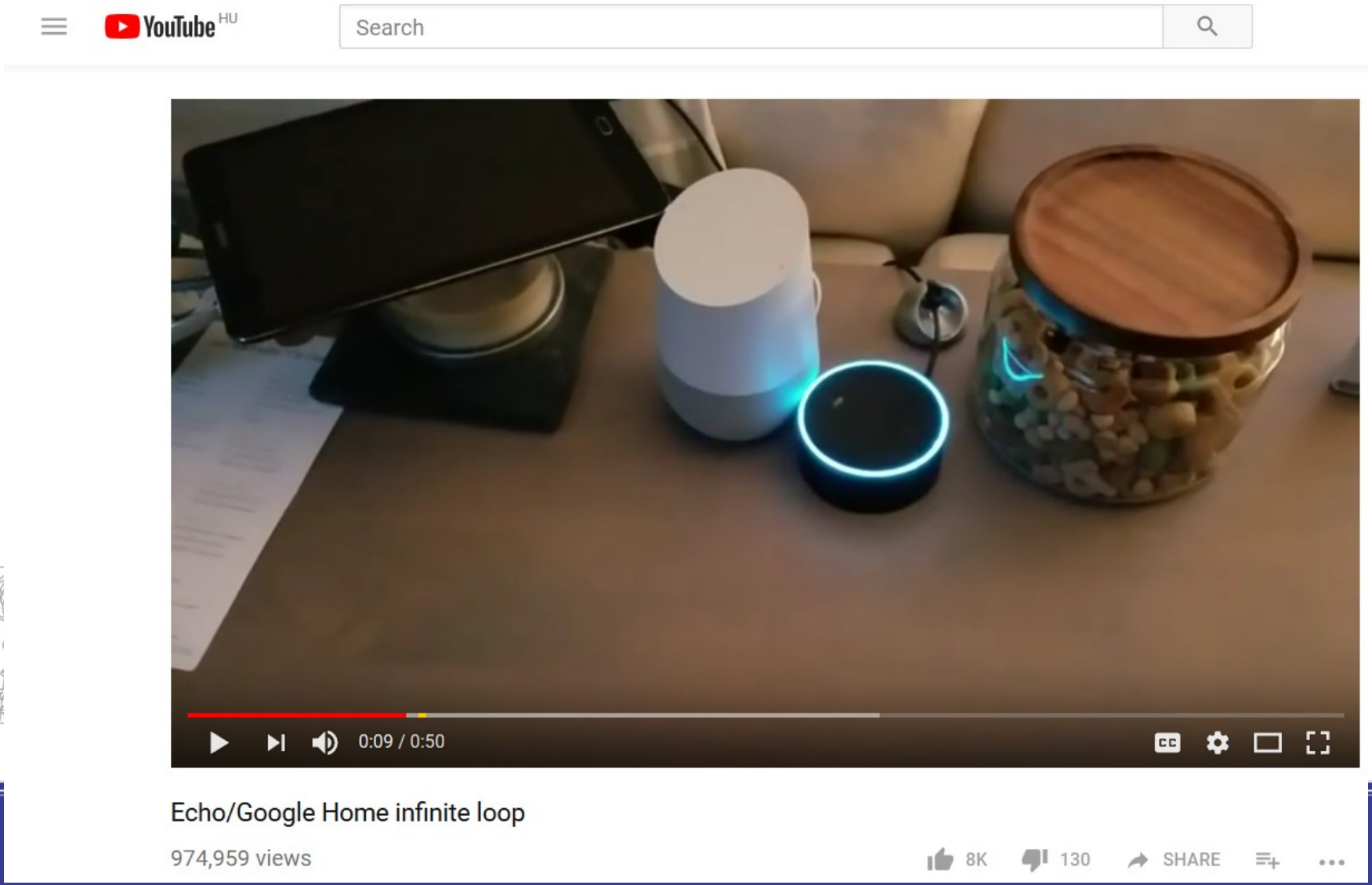
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Végtelen rekurzió

```
public static void
self_recursion(){
  self_recursion();
}
```

```
Exception in thread "main"
java.lang.StackOverflowError
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
    at Bevezetes.self_recursion(Bevezetes.java:97)
```

Kölcsönös rekurzió



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Rekurzió

„A ***rekurzió*** a matematikában, valamint a számítógép-tudományban egy olyan művelet, mely végrehajtáskor a saját maga által definiált műveletet, vagy műveletsort hajtja végre, ezáltal önmagát ismétli”

- Programozás: önmagát hívó eljárás
- Matematika: függvény definíciója a függvénnyel magával
- Algoritmusok: a részfeladatok ugyanazon módszerrel bonthatóak további részfeladatokra majd kerülnek összevonásra, mint az eredeti teljes feladat

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Felező csúcskereső algoritmus futási ideje

Felosztás $\Theta(1)$

Uralkodás $T(n/2)$

Összevonás $\Theta(1)$

$T(n) =$ ha $n>2$: $\Theta(1) + T(n/2) + \Theta(1)$

egyébként : $\Theta(1)$

$O(\log n)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Rendezési feladat

2020. szeptember

2. hét – 3. videó
S02E03

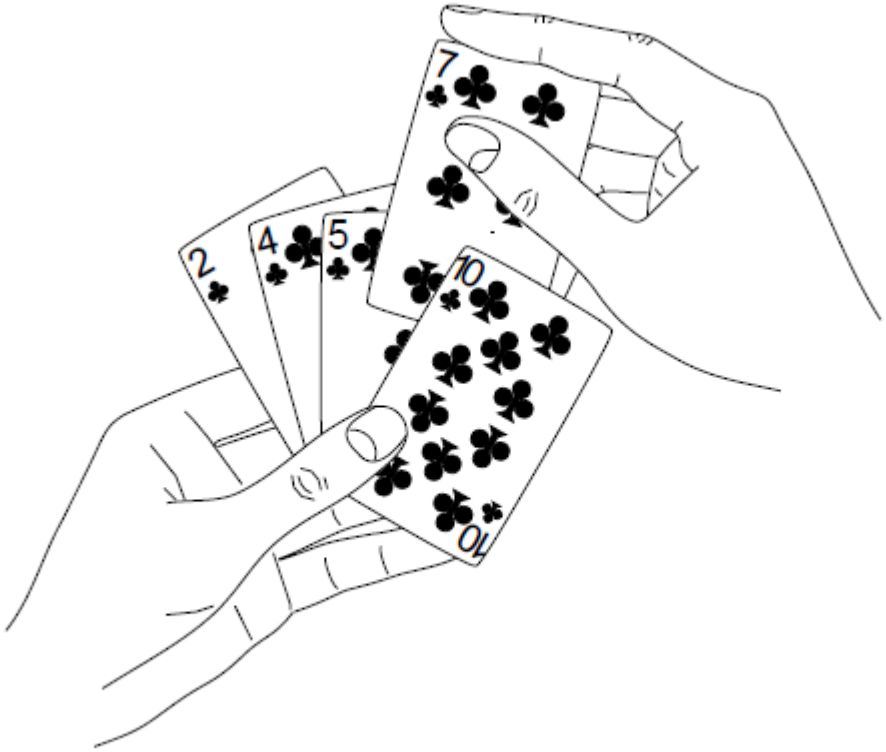
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Rendezési feladat

- **Bemenet:** egy n számból álló a_1, a_2, . . . , a_n sorozat.
- **Kimenet:** a bemenő sorozat elemeinek olyan a_1, a_2, . . . , a_n permutációja (átrendezése), amelyre $a_1 \leq a_2 \leq \ldots \leq a_n$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Beszúró rendezés



2.1. ábra. Kézben tartott lapok rendezése beszúró rendezéssel.

$O(n^2)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Összefésülő rendezés

(egy oszd-meg-és-uralkodj algoritmus)

2020. szeptember

2. hét – 4. videó
S02E04

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összefésülő rendezés

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.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Összefésülő rendezés

ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, r)

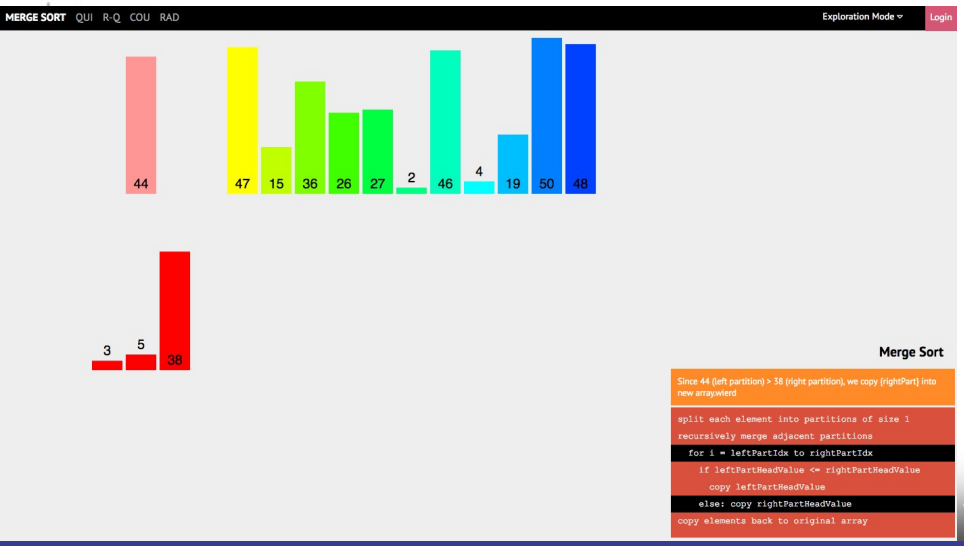
```
if p < r
    then q ← ⌊(p + r)/2⌋
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, q)
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, q + 1, r)
         ÖSSZEFÉSÜL(A, p, q, r)
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

ÖSSZEFÉSÜL(A, p, q, r)

```
n1 ← q − p + 1
n2 ← r − q
az L[1 .. n1 + 1] és R[1 .. n2 + 1] tömbök létrehozása
for i ← 1 to n1
    do L[i] ← A[p + i − 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i + 1
          else A[k] ← R[j]
               j ← j + 1
```

6 5 3 1 8 7 2 4



https://visualgo.net/en/sorting https://www.youtube.com/watch?v=EeQ8pwjQxTM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Összefésülő rendezés helyessége

2020. szeptember

2. hét – 5. videó
S02E05

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Összefésülő rendezés

Helyes?

ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, r)

```
if p < r
    then q ← ⌊(p + r)/2⌋
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, q)
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, q + 1, r)
         ÖSSZEFÉSÜL(A, p, q, r)
```

Algoritmusok helyességének bizonyítására egy módszer

- A ***ciklusinvariánsok***at arra használjuk, hogy megértsük, miért helyes egy algoritmus. Egy ciklusinvariánsról három dolgot kell megmutatnunk:
- **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.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

ÖSSZEFÉSÜL(A, p, q, r)

```
n1 ← q − p + 1
n2 ← r − q
az L[1 . . n1 + 1] és R[1 . . n2 + 1] tömbök létrehozása
for i ← 1 to n1
    do  L[i] ← A[p + i − 1]
for j ← 1 to n2
    do  R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then  A[k] ← L[i]
                i ← i + 1
          else  A[k] ← R[j]
                j ← j + 1
```

Helyes?

ÖSSZEFÉSÜL() helyessége

ciklusinvariáns:

- A 12–17. sorokbeli **for** ciklus minden iterációjának kezdetén
- az $A[p \,.\,.\, k - 1]$ résztömb az $L[1 \,.\,.\, n_1 + 1]$ és $R[1 \,.\,.\, n_2 + 1]$ tömbök $k - p$ darab legkisebb elemét tartalmazzák, rendezetten.
- $L[i]$ és $R[j]$ a megfelelő tömbök legkisebb olyan elemei, amelyek még nincsenek visszamásolva A-ba

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

ÖSSZEFÉSÜL() helyessége

Teljesül: k=p és i=j=1, L és R rendezett

Megmarad: ha L[i] ≤ R[j] akkor L[i] a legkisebb A[p … k − 1]-n kívüli elem, ezért az A[p… k] résztömb a $k - p + 1$ legkisebb elemet fogja tartalmazni

Befejeződik: k=r+1, A[p … r] rendezett

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Összefésülő rendezés helyessége

2020. szeptember

2. hét – 5. videó
S02E05

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Összefésülő rendezés futási ideje

Felosztás $\Theta(1)$

Uralkodás $2T(n/2)$

Összevonás $\Theta(n)$

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

Általános megoldások rekurzió feloldására

Például $T(n) = \begin{cases} \Theta(1), & \text{ha } n = 1, \\ 2T(n/2) + \Theta(n), & \text{ha } n > 1 \end{cases}$?

- **Rekurziós fa módszer**
- **Helyettesítő módszer**

- **Mester tétel**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Általános megoldások rekurzió feloldására

1. Sejtsük meg a megoldást. (pl. rekurziós fa módszer)

2. Helyettesítő módszer: Teljes indukcióval határozzuk meg az állandókat és igazoljuk a sejtés helyességét.

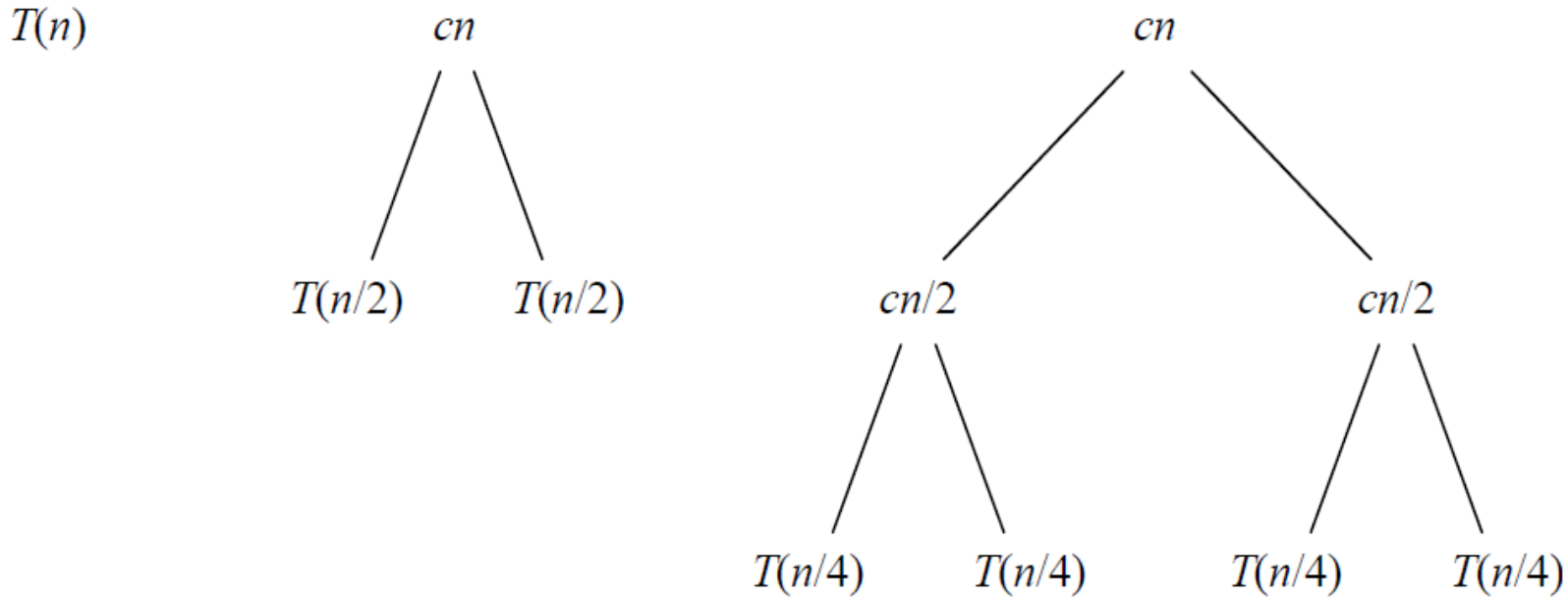
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Rekurziós fa módszer

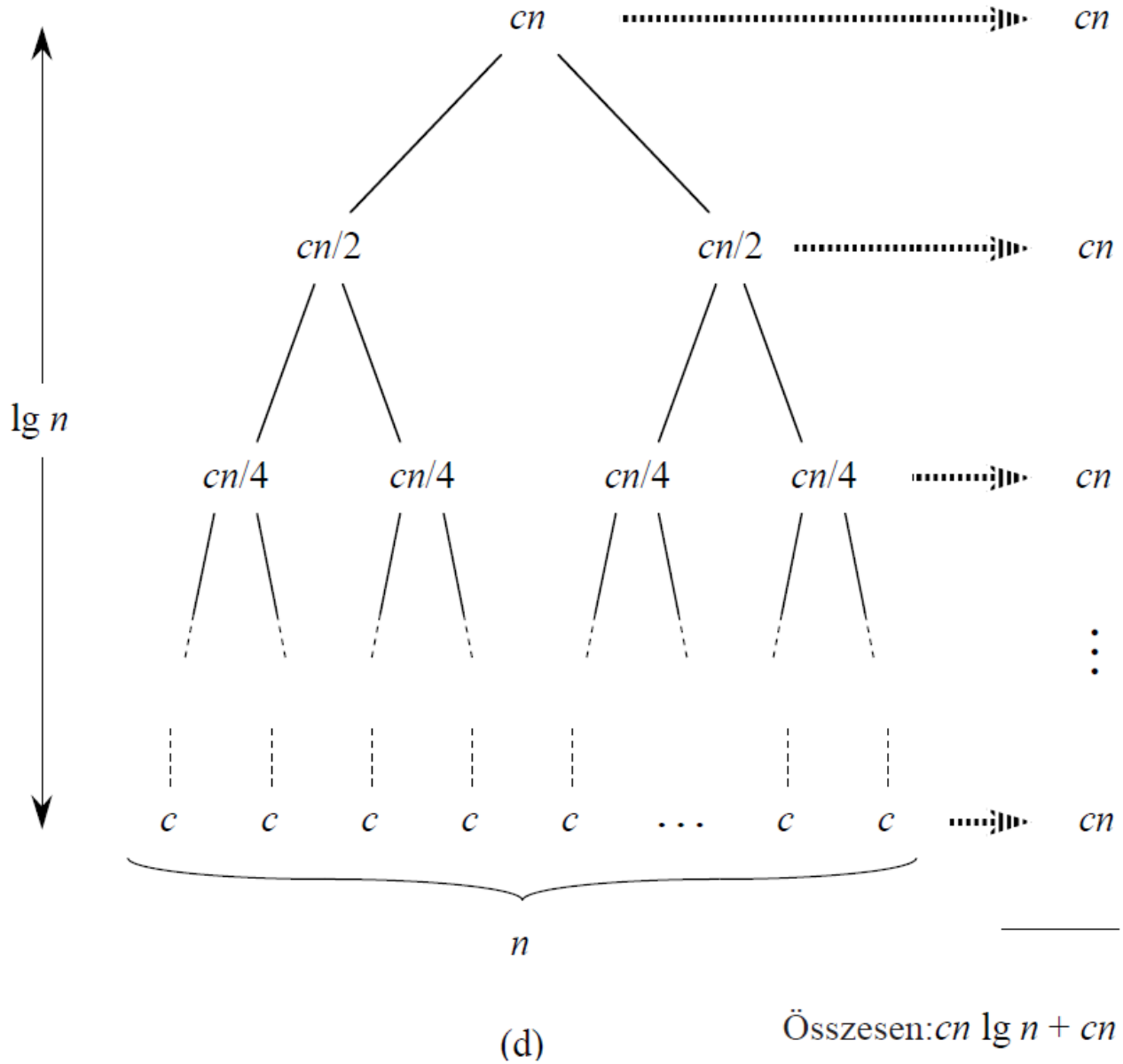
- Hány szintje van a hívási fának?
- Egy szintnek mekkora a futási ideje?

- Jó eszköz megsejteni a futásidőt majd helyettesítő módszerrel bizonyítjuk…
- Nagyvonalúan lehet számolni

Összefésülő rendezés futási ideje – rekurziós fa módszer



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS



2.5. ábra. Rekurziós fa építése a $T(n) = 2T(n/2) + cn$ rekurzióhoz. Az **(a)** rész $T(n)$-t mutatja, amelyet fokozatosan terjesztünk ki a **(b)–(d)** részekben rekurziós fává. A teljesen kiterjesztett fának a (d) részben $\lg n + 1$ szintje van (azaz a magassága $\lg n$, amint azt az ábra mutatja), és minden szint cn-nel járul a teljes költséghez. Ezért az összes költség $cn \lg n + cn$, azaz $\Theta(n \lg n)$.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Összefésülő rendezés futási ideje – Helyettesítő módszer

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

Sejtés: $O(n \log n)$

Teljes indukció

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

T(2)=4
T(3)=5

ha $c \ge 2$ akkor
$T(2) \le c\ 2 \log 2$ és
$T(3) \le c\ 3 \log 3$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Általános megoldások rekurzió feloldására

1. **Rekurziós fa módszer**ével sejtsük meg T(n)-t

2. A sejtést bizonyítsuk **helyettesítő módszer**rel

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Pénzváltási feladat

(struktúra optimalizálási feladat)

2020. szeptember

3. hét – 1. videó
S03E01

Pénzváltási feladat

Feladat:

Adott $P_1, P_2, \ldots P_n$ típusú pénzérmék
(mindegyikből végtelen darabszámú áll rendelkezésre)
mellett a lehető legkevesebb érmét használva hogyan fizethető ki F forint?

Ha nem fizethető ki pontosan, a kimenet legyen -1.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat

$P_1=1$, $P_2=5$, $P_3=10$, $P_4=25$, $P_5=50$

$F=40$

optimális megoldás: 25+10+5 (3db)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat

$P_1=1$, $P_2=5$, $P_3=10$, $P_4=20$, $P_5=25$, $P_6=50$

$F=40$

optimális megoldás: 20+20 (2db)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

(struktúra) Optimalizálási feladat

Sok megengedett megoldása lehet. Mindegyiknek van értéke, és mi az optimális (minimális vagy maximális) értékűt kívánjuk megtalálni.

Megj: egy ilyen megoldást *egy optimális megoldásnak* nevezzük, nem pedig *az optimális megoldásnak,* mivel más megoldások is elérhetik az optimális értéket.

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Pénzváltási feladat triviális megoldásai

2020. szeptember

3. hét – 2. videó
S03E02

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat

Egyszerű (mohó) megoldás:

Mindig vegyük a legnagyobb pénzérmét ami még használható (nem lépjük túl a fizetendő összeget).

Helyes?

$P_1=1$, $P_2=5$, $P_3=10$, $P_4=20$, $P_5=25$, $P_6=50$

$F=40$

Egyszerű algoritmus kimenete: 25+10+5

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat

Nyers erő módszerrel (brute force):

Soroljunk fel minden kombinációt, ahol az érmék összege F és ezek közül válasszunk egyet ahol a felhasznált érmék száma minimális.

Helyes?

Helyes, de nagyon lassú: $O(F^n)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Pénzváltási feladat rekurzív megoldása

2020. szeptember

3. hét – 3. videó
S03E03

Pénzváltási feladat – rekurzív megoldás

Bontsuk részproblémákra!

$P_1=1$, $P_2=5$, $P_3=6$, $F=9$

$$\text{minPenz}(9)=\min\begin{cases}\text{minPenz}(8)+1\\ \text{minPenz}(4)+1\\ \text{minPenz}(3)+1\end{cases}$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat – rekurzív megoldás

$$minPenz(9)=\min\begin{cases} minPenz(8)+1 \\ minPenz(4)+1 \\ minPenz(3)+1 \end{cases}$$

$$minPenz(8)=\min\begin{cases} minPenz(7)+1 \\ minPenz(3)+1 \\ minPenz(2)+1 \end{cases}$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat – rekurzív megoldás

PenzValtas(F,P)

$m \leftarrow \infty$

for $i \leftarrow 1$ **to** n

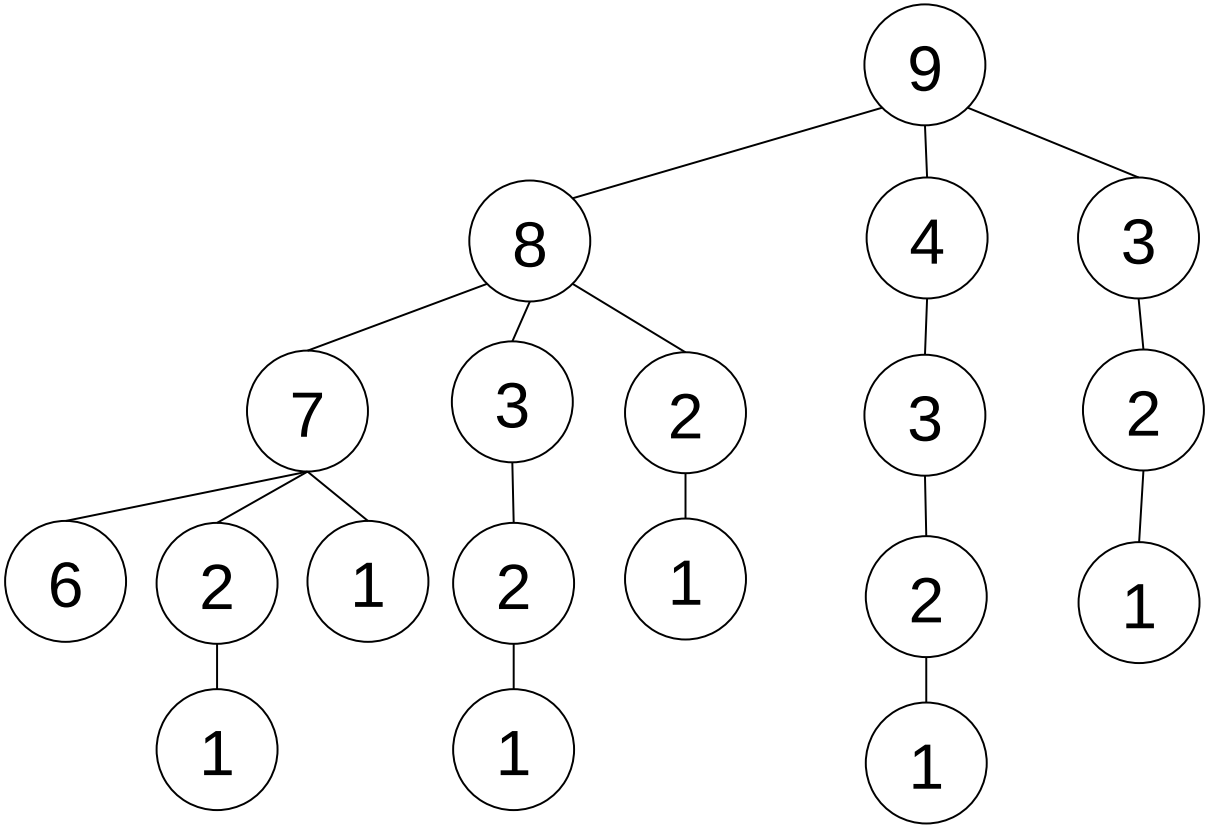
$m \leftarrow \min(m, \text{PenzValtas}(F - P_i))$

return $m + 1$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

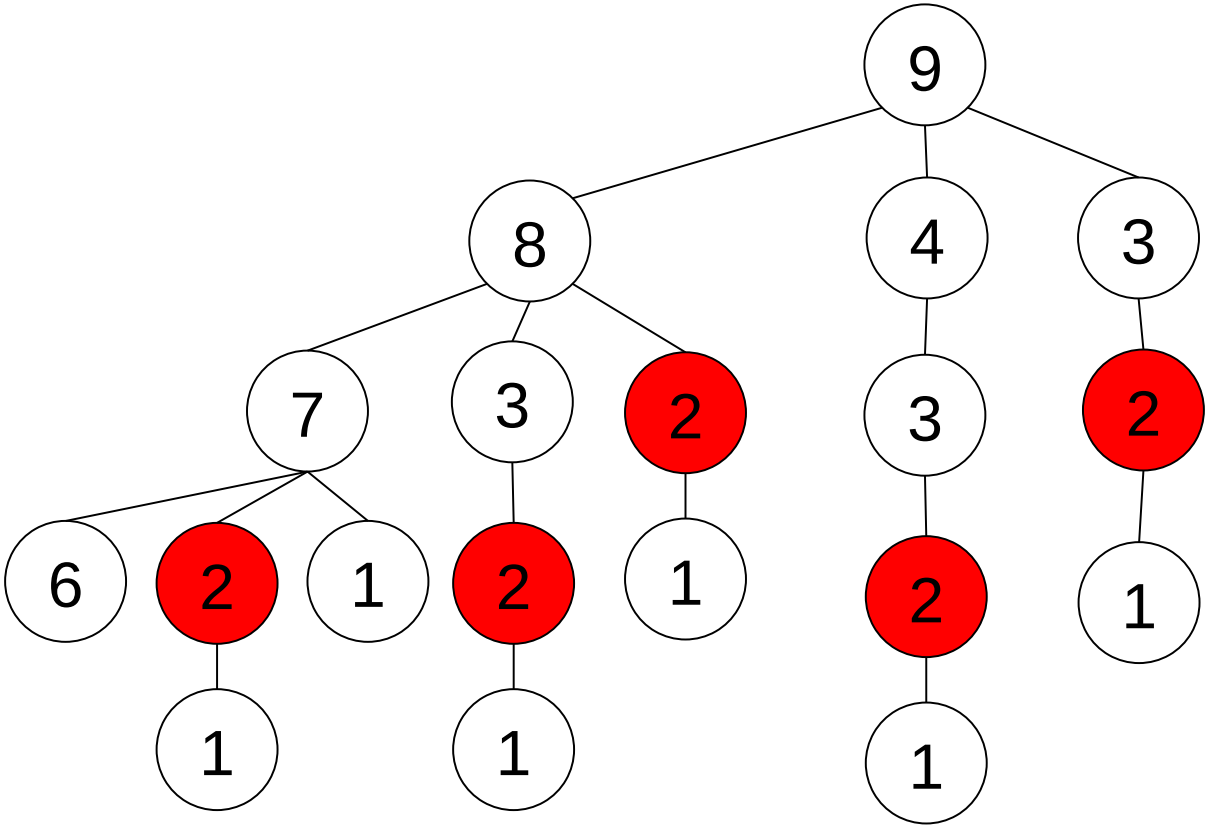
Pénzváltási feladat – rekurzív megoldás

$P_1=1$, $P_2=5$, $P_3=6$, $F=9$



Pénzváltási feladat – rekurzív megoldás

$P_1=1$, $P_2=5$, $P_3=6$, $F=9$



sejtés: $O(n^F)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Pénzváltási feladat

Dinamikus Programozási (DP) megoldása

2020. szeptember

3. hét – 4. videó
S03E04

Dinamikus Programozás (DP)

- Részfeladatokra való bontással oldjuk meg a problémát.
- DP – és nem oszd-meg-és-uralkodj – ha a részproblémák nem függetlenek, azaz ***közös részproblémáik*** vannak (optimalizálási feladatoknál tipikus!).
- Alapgondolat: a már megoldott részproblémák optimális megoldásának értékét ***memorizáljuk*** és ha még egyszer fel kell használni felidézzük

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Dinamikus Programozás (DP)

DP minden egyes részfeladatot és annak minden részfeladatát pontosan egyszer oldja meg, ***az eredményt egy táblázatban tárolja***, és ezáltal elkerüli az ismételt számítást, ha a részfeladat megint felmerül.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Pénzváltási feladat – dinamikus programozási megoldás

```
minPenz[0] ← 0
for p ← 1 to F
   minPenz[p] ← ∞
   for j ← 1 to n
      if p ≥ P_j
         minPenz[p] ← min(minPenz[p],
                          minPenz[p- P_j]+1)
return minPenz[F]
```

futásidő: $O(Fn)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Dinamikus Programozás

- **Iteratív megvalósítás (táblázatkitöltés):**
  - Minden részmegoldást kiszámolunk
  - Alulról felfele (bottom-up) építkezünk
- **Rekurzív megvalósítás memorizálással:**
  - Részmegoldásokat kulcs-érték formájában tároljuk (feljegyzéses módszer)
  - Felülről lefele (top-down) építkezünk
  - Csak akkor használjuk ha nem kell minden megoldást kiszámolni!

    (pl. minden érme 10 többszöröse)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Dinamikus Programozás

$P_1=1$, $P_2=5$, $P_3=6$, $F=9$

- **Rekurzív megvalósítás memorizálással**

  Felülről-lefelé (top-down):

0	1	2	3	4		1	2	3	4

- **Iteratív megvalósítás (táblázatkitöltés)**

  Alulról-felfelé (bottom-up):

0	1	2	3	4	1	1	2	3	4

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Hátizsák problémák

2020. szeptember

3. hét – 5. videó
S03E05

TV reklámidő feltöltés:

Válasszuk ki a reklámok egy halmazát (mindegyiknek van költsége és ideje) úgy, hogy maximalizáljuk a bevételt, de az összidő beleférjen a szünetbe

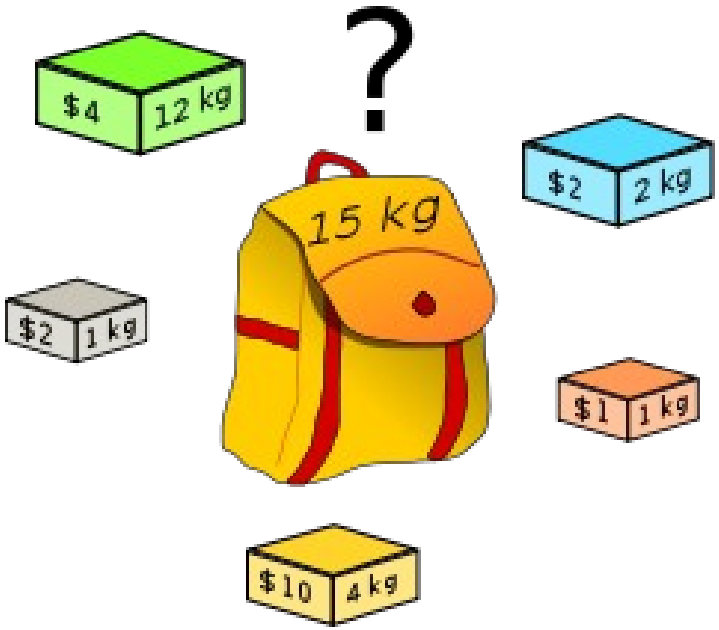
Szerverpark kiépítés:

Vásároljunk gépeket egy költségvetésből, úgy hogy az összteljesítmény maximális legyen

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák probléma

Adott egy hátizsák **kapacitás**a és *n* tárgy, mindegyik **érték**kel és **súlly**al megadva. Mekkora a **legnagyobb összérték** amit a hátizsákba tehetünk?



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák probléma

Tfh. **tárgyak nem darabolhatóak**

Két különböző feladat:

- minden tárgyból pontosan 1db van
- a tárgyakat lehet (tetszőleges számban) ismételni

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Hátizsák ismétlődő tárgyakkal

Bemenet:

- hátizsák S kapacitása
- n tárgy, amelyek súlyát $S_1,\ldots,S_n$ ill. értékét $E_1,\ldots,E_n$ jelöli

Kimenet:

Tárgyakat nem darabolva, minden tárgy tetszőleges számban rendelkezésre állva, mi a legnagyobb $\sum E$ ami S kapacitásba belefér

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Hátizsák problémák

2020. szeptember

3. hét – 5. videó
S03E05

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Hátizsák ismétlődő tárgyakkal

Bemenet:

- hátizsák S kapacitása
- n tárgy, amelyek súlyát $S_1,\ldots,S_n$ ill. értékét $E_1,\ldots,E_n$ jelöli

Kimenet:

Tárgyakat nem darabolva, minden tárgy tetszőleges számban rendelkezésre állva, mi a legnagyobb $\sum E$ ami S kapacitásba belefér

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődő tárgyakkal

Részfeladatra bontás:

- vegyünk az optimális megoldást valamely *s* kapacitásra
- ha valamely *i* tárgyat kivesszük az optimális megoldásból egy $(s-S_i)$-re optimális megoldáshoz jutunk

$$\text{ertek}(s)=\max(\text{ertek}(s-S_i)+E_i)$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődő tárgyakkal

```
ertek[0] ← 0
for s ← 1 to S
   ertek[s] ← 0
   for j ← 1 to n
      if s ≥ S_j
         ertek[s] ← max( ertek[s],
                         ertek[s- S_j] + E_j )
return ertek[S]
```

futásidő: $O(Sn)$

Hátizsák ismétlődő tárgyakkal - példa

S=10

0	0	9	14	18	23	30	32	39	44	48

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Ismétlés nélküli hátizsák feladat DP megoldása

2020. szeptember

3. hét – 7. videó
S03E07

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődő tárgyakkal

Bemenet:

- hátizsák S kapacitása
- n tárgy, amelyek súlyát $S_1,\ldots,S_n$ ill. értékét $E_1,\ldots,E_n$ jelöli

Kimenet:

Tárgyakat nem darabolva, minden tárgyból 1 darab áll rendelkezésre, mi a legnagyobb $\sum E$ ami S kapacitásba belefér

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

Nyers erő:

lehetséges megoldások száma: 2^n

Ismétlődéses részproblémára bontása nem működik! Ha egy optimális megoldásból kiveszünk egy tárgyat a megmaradt kapacitásra nem biztos, hogy optimális megoldás összeállítható $s-S_i$ kapacitásra

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

Részproblémára bontás:

- vegyünk az optimális megoldást valamely *s* kapacitásra
- ha az *n.* tárgy része az optimális megoldásnak akkor $(s-S_n)$-re optimális megoldáshoz jutunk 1..(n-1) tárgyak felhasználásával
- egyébként s-re optimális megoldás kapható 1..(n-1) tárgyak felhasználása

$$ertek(s,i)=max(ertek(s-S_i,i-1)+E_i,\ ertek(s,i-1))$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

for $s \leftarrow 0$ **to** S
  ertek[s,0] $\leftarrow$ 0
for $j \leftarrow 1$ **to** n
  ertek[0,j] $\leftarrow$ 0
  for $s \leftarrow 1$ **to** S
    ertek[s, j] $\leftarrow$ ertek[s, j-1]
    if $s \geq S_j$
      ertek[s, j] $\leftarrow$ max(ertek[s, j-1],
                ertek[s- S_j , j-1] + E_j)
return ertek[S,n]

futásidő: $O(Sn)$

Hátizsák ismétlődés nélkül

S=10

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

S=10

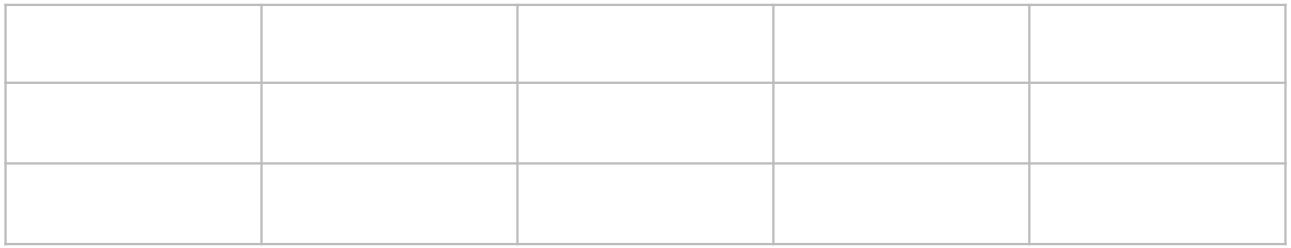
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

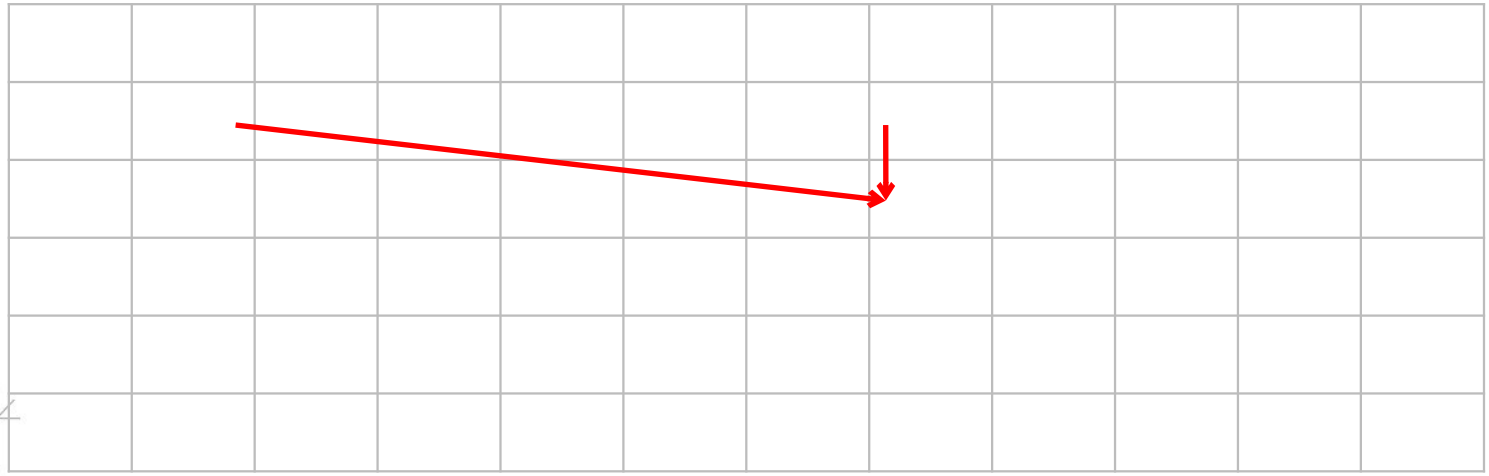
S=10

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

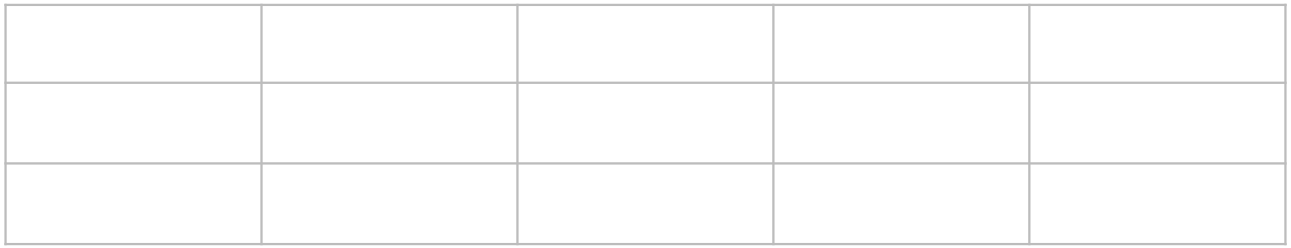


S=10



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

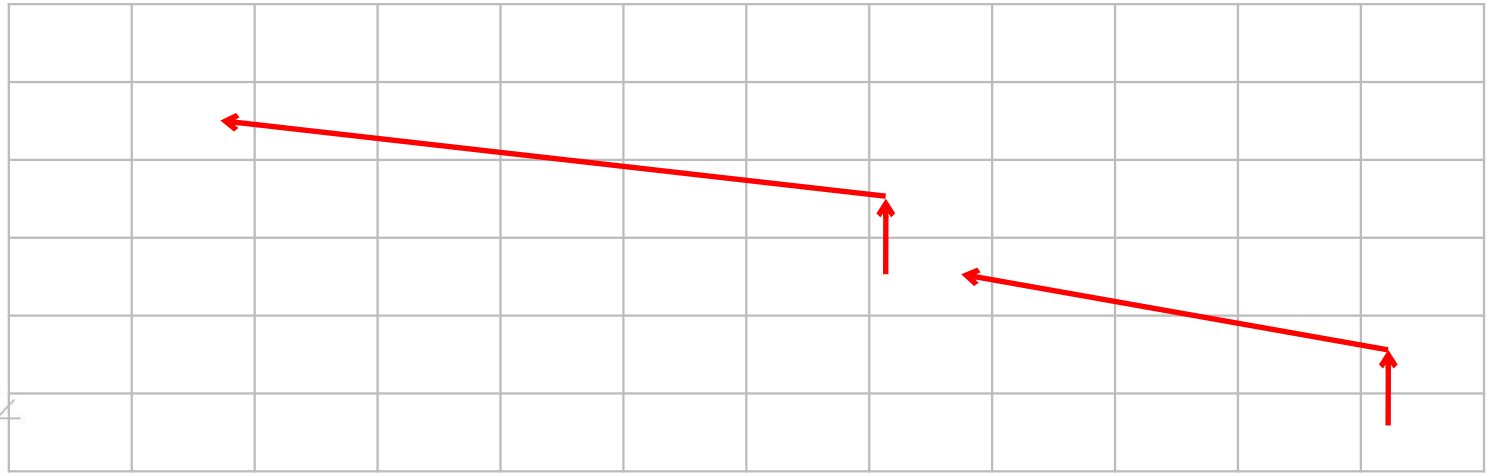


S=10

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák ismétlődés nélkül

S=10



Optimális megoldás megszerkesztése:

1	0	1	0

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egy optimális megoldás előállítása

- Mindig a teljes megoldásból visszafejtéssel
- Megjegyezhetjük, hogy melyik döntéssel jutottunk az aktuális részmegoldáshoz (backpointerek)
- VAGY minden részmegoldásnál újra kiszámoljuk, hogy melyik részmegoldásból jutottunk ide

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Dinamikus Programozás

2020. szeptember

3. hét – 8. videó
S03E08

Dinamikus Programozás elemei

- DP alkalmazásához szükséges tulajdonságok:
  - Optimális részstruktúra = helyesség
  - Ismétlődő részfeladatok
- Egy optimális megoldás megszerkesztése

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Optimális részstruktúra

- Egy feladat ***optimális részstruktúrájú,*** ha a probléma egy optimális megoldása önmagán belül a részfeladatok optimális megoldásait tartalmazza.
- Biztosítanunk kell, hogy a részfeladatok általunk vizsgált halmaza tartalmazza azokat, amelyekből az optimális megoldás áll.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Ismétlődő részfeladatok

A részfeladatok halmaza ”kicsi” legyen abban az értelemben, hogy a probléma rekurzív algoritmusa ismételten ugyanazokat a részfeladatokat oldja meg ahelyett, hogy mindig új részfeladatot állítana elő.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Tár-idő átváltás

Nagyon sok esetben adható az algoritmusnak egy módosított változata ami kevesebb/több futásidőt igényel aminek ára a nagyobb/kisebb tár (trade-off)

- Az egész DP-nek ez a lényege…
- Egy optimális megoldás megszerkesztésének két módszere is jó példa erre

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Dr. Farkas Richárd

SzTE TTIK, Számítógépes Algoritmusok és Mesterséges Intelligencia tanszék

www.inf.u-szeged.hu/~rfarkas

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Miért tanuljak “Algát”?

2020. szeptember

1. hét – 1. videó

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok

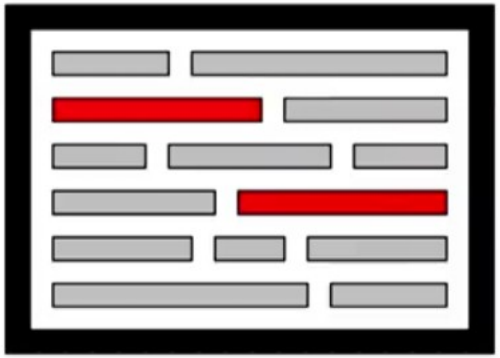
Algoritmusnak nevezünk bármilyen ***jól definiált számítási eljárás***t, amely ***bemenetként*** bizonyos értéket vagy értékeket kap és ***kimenetként*** bizonyos értéket vagy értékeket állít elő.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmus?

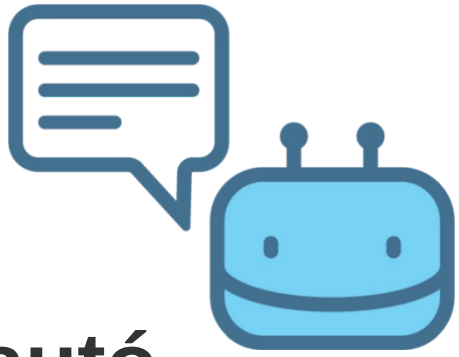
- **Jeleníts meg egy képet a weblapon**
  - **túl triviális, nem érdekes itt...**

- **Egy adott szó szerepel-e egy fájlban**
  - **Ha sebesség fontos, okos megoldás kell!**



Algoritmus?

- **Chatrobot**



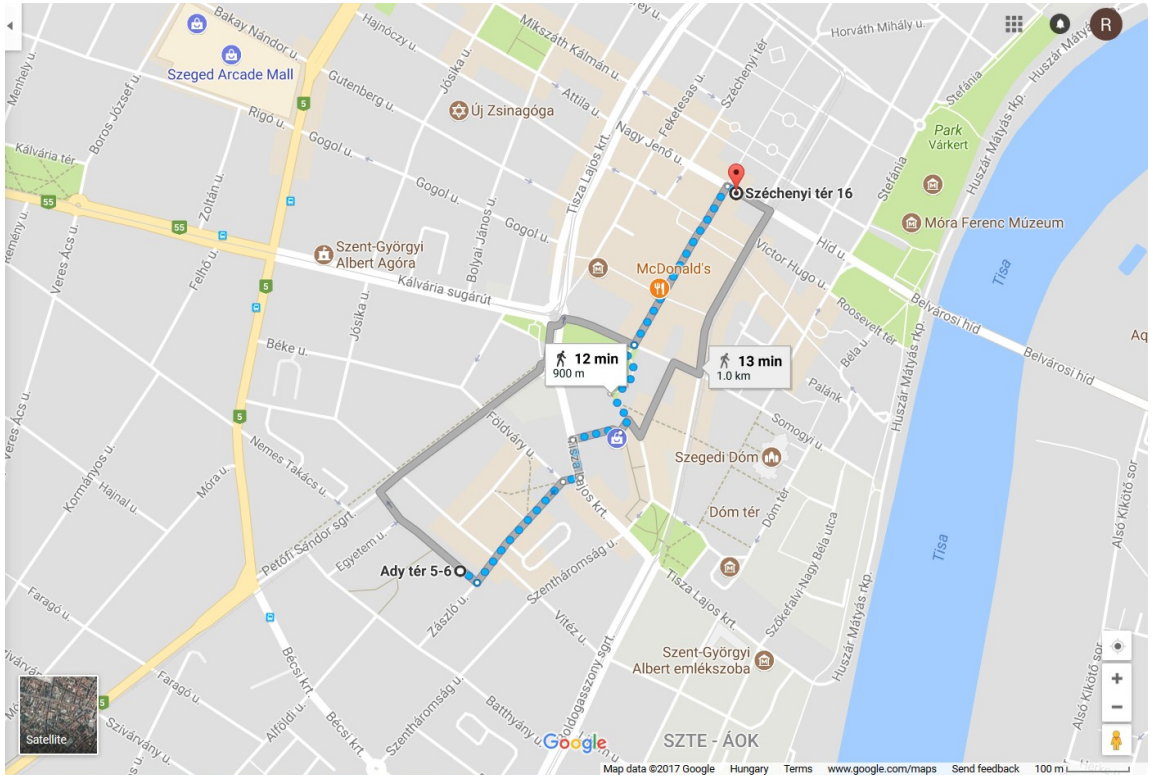
- **Önvezető autó**

  - **Nem jól definiált!**
  - **Mesterséges Intelligencia**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmus!

- **Legrövidebb út keresése**



- **Nem triviális a megoldás**
- **Egyszerű megoldás túl lassú**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Adatszerkezetek

Az ***adatszerkezet*** adatok tárolására és szervezésére szolgáló módszer, amely lehetővé teszi a hozzáférést és módosításokat

egfelelő algoritmushoz

egfelelő adatszerkezetet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Miért tanuljak algoritmusokat?

- **Mindenki fogja használni!**
- **BigData – skálázódás fontos!**

Miért tanuljak algoritmusokat?

- **Algoritmikus gondolkodás!**
  - Algoritmus eddig megoldatlan problémára?
  - Megfelelő algoritmusok és adatszerkezetek kiválasztása
  - Gondoljuk végig a helyességet és hatékonyságot!

Miért tanuljak algoritmusokat?

- Nyelvfüggetlen programozói szemlélet

> “You can practice really hard for two years to become a great programmer and you can practice for 10 years to become an excellent programmer. Or you can practice for two years and take an algorithms course and become an excellent programmer

- Absztraktabb gondolkodás

[How to: Work at Google — Example Coding/Engineering Interview]

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Töredékes hátizsák feladat mohó megoldása

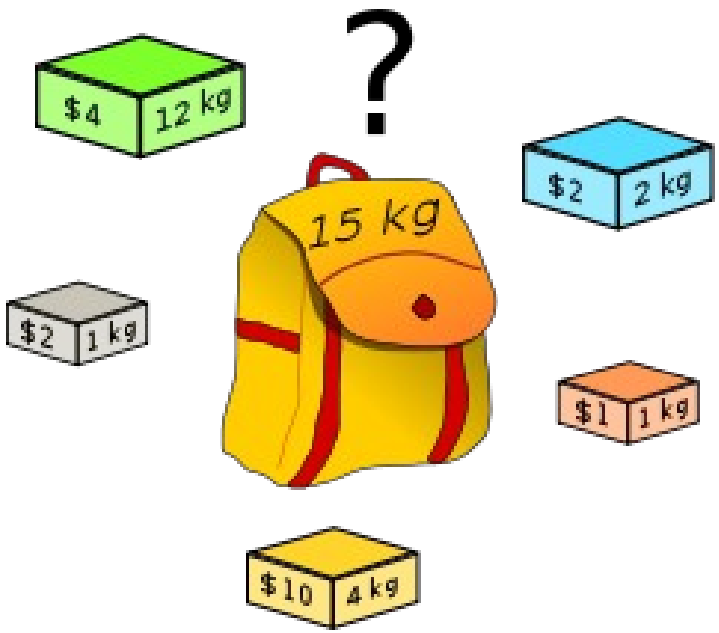
2020. szeptember

4. hét – 1. videó
S04E01

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hátizsák probléma

Adott egy hátizsák **kapacitás**a és *n* tárgy, mindegyik **érték**kel és **súlly**al megadva. Mekkora a **legnagyobb összérték** amit a hátizsákba tehetünk?



Töredékes hátizsák probléma

Tfh. **a tárgyak feldarabolhatóak**

Bemenet:
hátizsák S kapacitása
n tárgy, amelyek súlyát $S_1,\ldots,S_n$ ill. értékét $E_1,\ldots,E_n$ jelöli

Kimenet:
Minden tárgyból 1db áll rendelkezésre, de az darabolható. Mi a legnagyobb $\sum E$ ami S kapacitásba belefér?

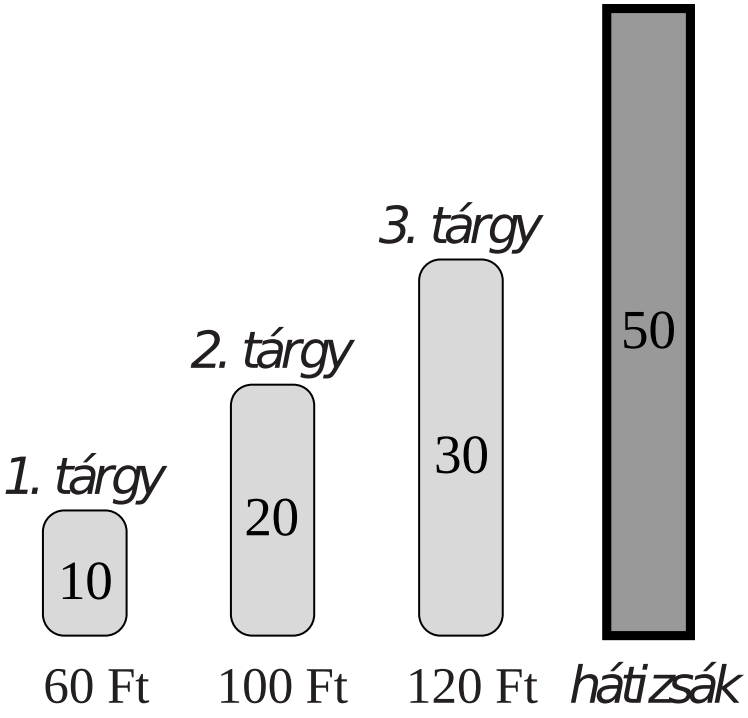
SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

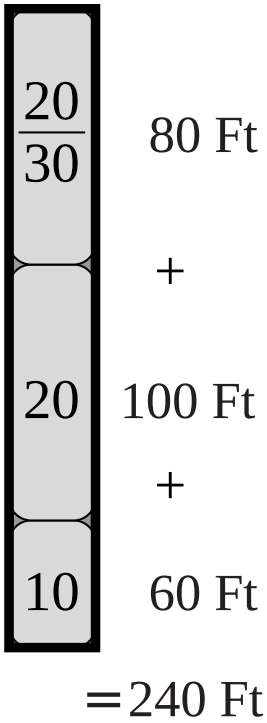
Mohó algoritmus a töredékes hátizsák problémára

- számoljuk ki minden tárgyra az E_i/S_i arányt
- tegyük a hátizsákba a legnagyobb, még rendelkezésre álló, E_i/S_i tárgyat. Ha a szabad kapacitás engedi akkor az egész tárgyat, ha nem, akkor akkora darabot vágjunk le, ami kitölti a szabad kapacitást.

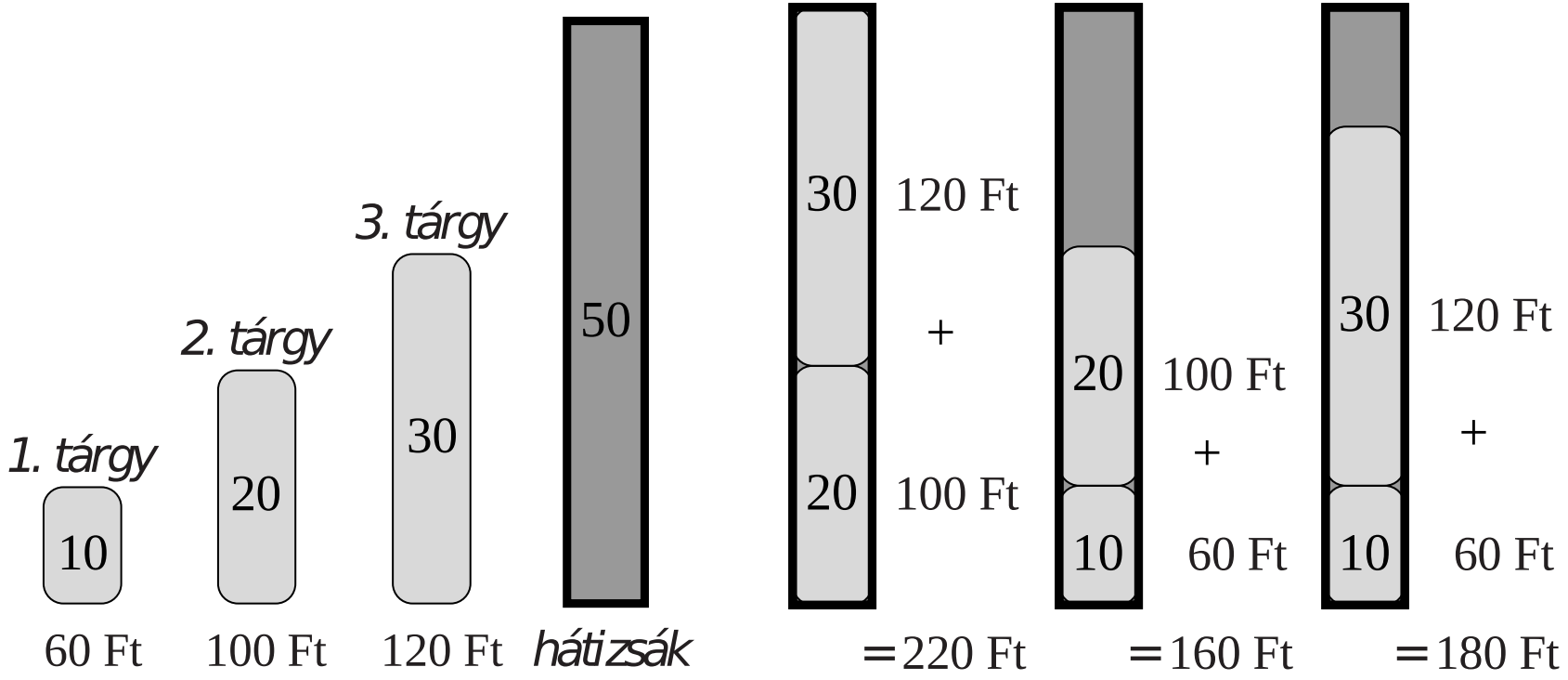
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Töredékes hátizsák feladat





mohó algoritmus ismétlés nélküli hátizsák feladatra



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Mohó algoritmus

- **Részfeladatra bontás**
- **Optimalizálás**

A ***mohó algoritmus*** mindig az adott lépésben optimálisnak látszó választást teszi.

Nem minden problémára adható mohó megoldás!

De ha létezik mohó megoldás az nagyon hatékony

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Algoritmusok helyesség és futásidő elemzése

2020. szeptember

1. hét – 2. videó
S01E02

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok tervezése

- Értsük meg mélyen a feladatot!
- Nincs általános módszertan algoritmizálásra
- A félév folyamán
  - megismerünk hasznos technikákat
  - látunk számtalan algoritmust különböző problémákra

ezek mintául szolgálhatnak a jövőben.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok elemzése

- Helyesség
- Hatékonyság:
  - előre meg akarjuk mondani, milyen erőforrásokra lesz szüksége az algoritmusnak
  - ***számítási idő***, memória, sávszélesség
- Cél: algoritmusok összehasonlítása

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Futási idő

- Milyen hardver?
- CPU? GPU? Felhő?

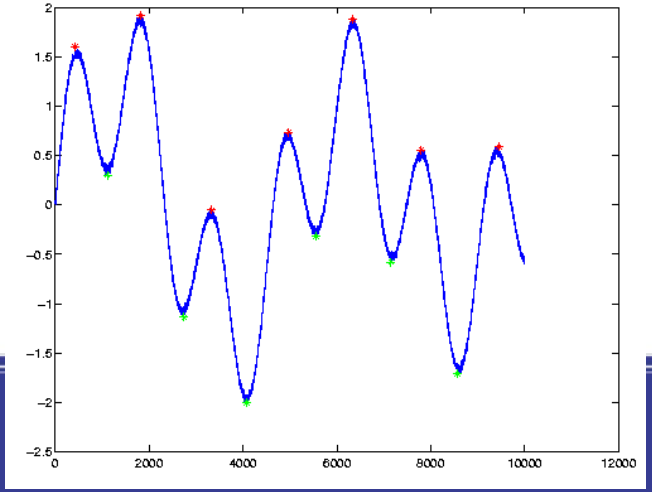
- ***Futási idő***: egy bizonyos bemenetre a végrehajtott (gépfüggetlen) alapműveletek vagy ”lépések” száma
- Feltesszük, hogy egy kód egy sorának végrehajtásához konstans mennyiségű idő szükséges

Feladat: csúcs keresés

Bemenet: egész számok tömbje

Találjunk és adjunk vissza ***egy*** „csúcs”ot ha létezik!

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb



Forrás: MIT Introduction to Algorithms, Lecture 1.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Feladat: csúcs keresés

1	3	4	3	5	1	3

Csúcs: olyan elem a tömbben ami minden szomszédjánál nem kisebb

Létezik mindig csúcs?

„nem kisebb” helyett „nagyobb” egy másik algoritmust igényelhet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső alg

- Balról jobbra vizsgáljunk meg minden elemet és ha csúcsot találunk térjünk vissza

1	3	4	3	5	1	3

- Pszeudokód:

for *j* ← 1 **to** *hossz*[*A*]
  do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]
    return j

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső alg

```
if hossz[A] < 1
  return nil
if hossz[A] = 1
  return 1
else if A[1] ≥ A[2]
  return 1
else if A[hossz[A]] ≥ A[hossz[A]-1]
  return hossz[A]
for j ← 2 to hossz[A]-1
  do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]
        return j
```

Egyszerű csúcs kereső alg

```
public static Integer find_a_peak(int[] a) throws IllegalArgumentException {
  //határesetek kezelése:
  if(a == null)
    throw new IllegalArgumentException();
  if(a.length < 1)
    return null;
  if(a.length == 1)
    return 0;
  if(a[0] >= a[1])
    return 0;
  if(a[a.length-1] >= a[a.length-2])
    return a.length-1;

  //algoritmus:
  for(int j = 2 ; j < a.length-1; ++j)
    if((a[j] >= a[j-1])  && (a[j] >= a[j+1]))
      return j;

  //soha nem érhetünk ide:
  return null;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok (és implementáció) helyességének tesztelése

- **Tesztesetek (unit test)**
- **Először elvárt bemenetekre**

```
find_a_peak(new int[]{1,3,4,3,5,1,3});
```

- **Aztán határesetekre is!**

```
find_a_peak(new int[]{7,3,4,3,5,1,3});
find_a_peak(new int[]{1,3,4,3,5,1,5});
find_a_peak(new int[]{1,3});
find_a_peak(new int[]{1});
find_a_peak(new int[]{});
find_a_peak(null);
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső algoritmus elemzése

	költség	végrehajtási szám
if *hossz*[A] < 1	c_1	1
return nil	c_2	1
if *hossz*[A] = 1	c_3	1
return 1	c_4	1
else if *A*[1] ≥ *A*[2]	c_5	1
return 1	c_6	1
else if A[*hossz*[A]] ≥ A[*hossz*[A]-1]	c_7	1
return *hossz*[A]	c_8	1
for *j* ← 2 **to** *hossz*[*A*]-1	c_9	n-2
do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]	c_{10}	n-2
return j	c_{11}	1

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Legjobb, átlagos esetek elemzése

- Bemenet mérete konstans ***n***
- Algoritmus össz futás idejét **T(n)**-el jelöljük
- Legjobb esetben az első elem csúcs, ekkor T(n)=2 minden *n*-re
- Az átlagos vagy várható futásidőt nagyon nehéz kiszámolni… (valószínűségi elemzés)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Legrosszabb eset elemzése

- Inkább legyünk pesszimisták!
- Az algoritmus legrosszabb futási ideje bármilyen bemenetre a futási idő felső korlátja.
  - Gyakran előfordul a legrosszabb eset
  - Az átlagos eset gyakran nagyjából ugyanolyan rossz, mint a legrosszabb eset.

Egyszerű csúcs kereső algoritmus elemzése

	költség	végrehajtási szám
if *hossz*[A] < 1	c_1	1
return nil	c_2	1
if *hossz*[A] = 1	c_3	1
return 1	c_4	1
else if *A*[1] ≥ *A*[2]	c_5	1
return 1	c_6	1
else if A[*hossz*[A]] ≥ A[*hossz*[A]-1]	c_7	1
return *hossz*[A]	c_8	1
for *j* ← 2 **to** *hossz*[*A*]-1	c_9	n-2
do if A[j] ≥ A[j-1] és A[j] ≥ A[j+1]	c_{10}	n-2
return j	c_{11}	1

Legrosszabb esetben:

$$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű csúcs kereső algoritmus elemzése

Legrosszabb esetben:

$T(n) = c_1 + c_3 + c_5 + c_7 + c_9(n-2) + c_{10}(n-2) + c_{11}$

$T(n) = (c_9 + c_{10})(n-2) + c_1 + c_3 + c_5 + c_7 + c_{11}$

- Tényleges futásidők helyett *c* konstansok
- Nagy *n* esetén $c_1 + c_3 + c_5 + c_7 + c_{11}$ elhanyagolható
- Hatékonyság szempontjából minket az érdekel, hogy hogyan skálázódik az algoritmus, azaz milyen a ***futásidő növekedési sebessége***
- Elég csak a fő tagot figyelembe venni: $(c_9 + c_{10})(n-2)$
- Nagyságrendileg T(n) *n* lineáris függvénye

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Algoritmusok stressz tesztelése

```
public static void running_time(){
  int size = 100000000;
  int[] ints = new int[size];
  ints[size-1] = 0;
  for(int i=0;i<size-1;++i)
    ints[i]=i;
  long time = System.currentTimeMillis();
  find_a_peak(ints);
  System.out.println((System.currentTimeMillis()-
time) + " msec");
}
```

```
stdout 1M:  1 msec
stdout 10M:  9 msec
stdout 100M: 78 msec
```

Feladat: csúcs keresés

- Feladat ugyanaz!
- Tudunk hatékonyabb megoldást találni?

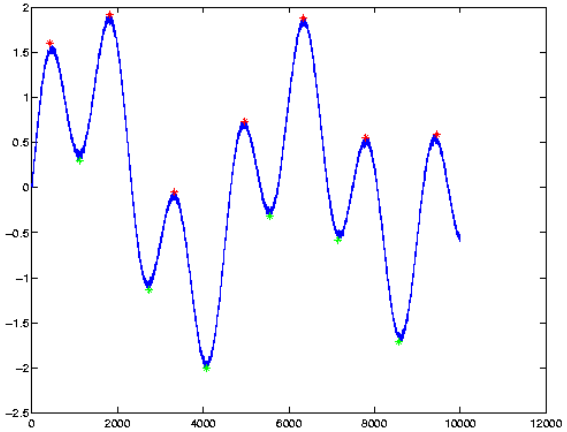
- Gondoltam egy számra 1 és 2^{32} közt

- **Oszd meg és uralkodj!**

Felező csúcskereső algoritmus

Vizsgáljuk meg a középső elemet. Ha nem csúcs akkor egyik szomszéd nagyobb, vizsgáljuk a bemenet felét a ezen szomszéd irányába!

1	3	4	3	5	1	3



Helyes?

SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Felező csúcskereső algoritmus

```
public static Integer find_a_peak(int[] a){
  // határesetek kezelése ugyanaz, mint a lassú verzióban!

  // algoritmus:
  int lo = 1;
  int hi = a.length - 2;
  while (lo <= hi) {
    //kell lennie csucsnak az a[lo..hi]-ban
    int mid = lo + (hi - lo) / 2;
    if (a[mid] < a[mid - 1])
      hi = mid - 1;
    else if (a[mid] < a[mid + 1])
      lo = mid + 1;
    else
      return mid;
  }

  // Soha nem érhetünk ide:
  return null;
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Felező csúcskereső algoritmus futási ideje

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

$T(1) = c$

$T(16) = T(8) + c = T(4) + c + c = T(2) + c + c + c = T(1) + c + c + c + c = c + c + c + c + c = 5c$

$T(n) = (\log_2 n + 1)\, c$

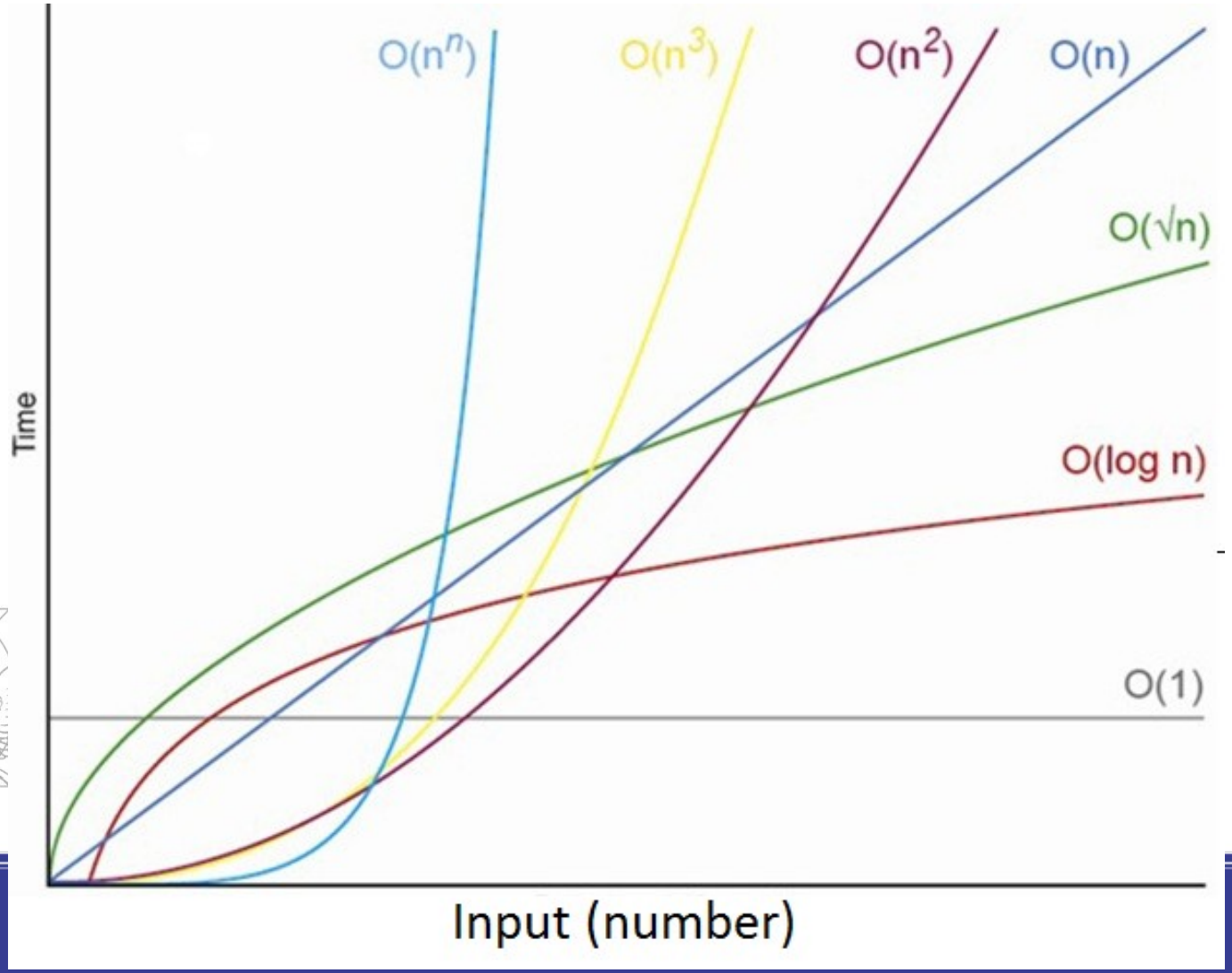
Legrosszabb esetben is T(n) *n* függvényében logaritmikus növekedési sebességű

```
stdout 100M: 0 msec
```

Megj: $\log_a b = \log_c b / \log_c a$ miatt a logaritmus alapja nem számít, hiszen az „csak” konstans szorzó. Nem írjuk ki a kurzuson, hanem mindig 2-es alapú logaritmussal számolunk.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Tipikus aszimptotikus felső korlátok



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Feladat: 2D csúcs keresés

Bemenet: egész számok két dimenziós tömbje (mátrix)

Találjunk és adjunk vissza ***egy*** „csúcs”ot ha létezik!

Csúcs: olyan elem a 2D tömbben ami minden szomszédjánál nem kisebb

Feladat: 2D csúcs keresés
Értsük meg a feladatot!

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

méret: n x n

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Mohó hegymászó 2D csúcskereső

Induljunk valahonnan (középről vagy egyik sarokból), minden lépésben lépjünk az egyik nagyobb szomszédra. Ha nincs nagyobb szomszéd csúcsot találtunk.

Minden n mérethez kezdőponthoz és lépési startégiához lehet adni olyan bemenetet amire az egész mátrixot be fogja járni (legrosszabb eset) azaz T(n) n^2 növekedési sebességű.

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés 1D csúcskeresésre visszavezetve

Válasszuk a középső oszlopot. Keressünk 1D csúcsot ebben. A talált 1D csúcs sorában keressünk újra 1D csúcsot. Ez 2D csúcs lesz.

Legrosszabb esetben is log*n* növekedési sebességű, hiszen log*n* idő alatt találunk 1D csúcsot

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés 1D csúcskeresésre visszavezetve

8	4	1
7	5	3
2	3	1

Egy helyes de nem hatékony algoritmus ér valamit, nem úgy mint egy helytelen de hatékony ◀◀

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés felezéssel

Válasszuk a középső oszlopot. Keressünk meg egy maximális elemet ebben. Ha ennek bal vagy jobb szomszédja nagyobb, mint az elem ismételjük meg az eljárást ebben a fél mátrixban. Ha bal és jobb szomszédok nem kisebbek 2D csúcsot találtunk.

Helyes?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

2D csúcskeresés felezéssel

0	0	9	0	0	0	0
0	0	0	0	0	0	0
0	1	0	0	0	0	0
0	2	0	0	0	0	0
0	3	0	0	0	0	0
0	5	0	0	0	0	0
0	4	7	0	0	0	0

Helyes?

2D csúcskeresés felezéssel futási ideje

Ha csak egy oszlop van a maximum elem keresés legrosszabb időben n lépést igényel.

$T(n, 1) = cn$

$T(n, m) = T(n, m/2) + cn$

$T(n, m) = \log n \cdot cn$

Legrosszabb esetben $n \cdot \log n$ növekedési sebességű az algoritmus

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Feladat: 2D csúcs keresés

Mohó hegymászó ✓ n^2

Visszavezetés 1D-re ✗ $\log n$

Mátrixfelezés ✓ $n \cdot \log n$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összegzés

- Algoritmusok tervezése
  - Értsük meg a problémát/feladatot
  - Legegyszerűbb megoldást elemezzük
  - Ha kell tervezzünk hatékonyabb algoritmust!
- Algoritmusok elemzése
  - helyesség
  - hatékonyság (skálázódás)

eszköze: legrosszabb eset futásidő növekedési sebesség

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

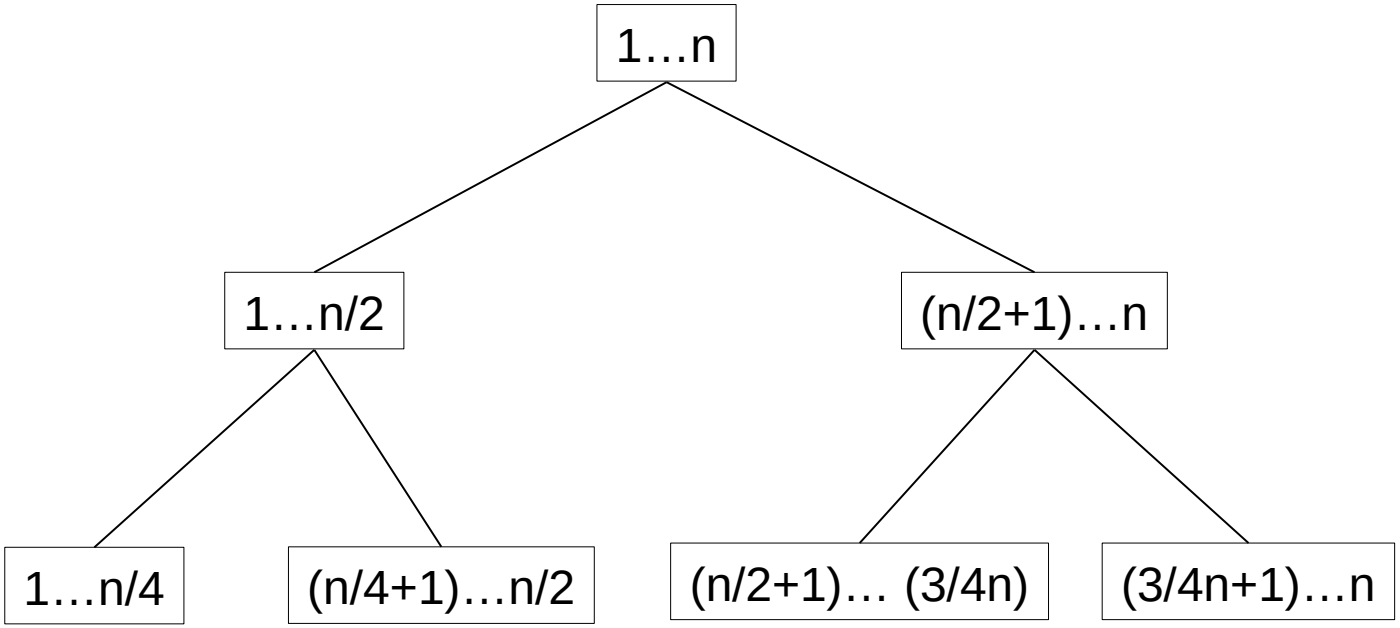
Algoritmusok és Adatszerkezetek I.

Rekurzív részproblémára bontási stratégiák összehasonlítása

2020. szeptember

4. hét – 3. videó
S04E03

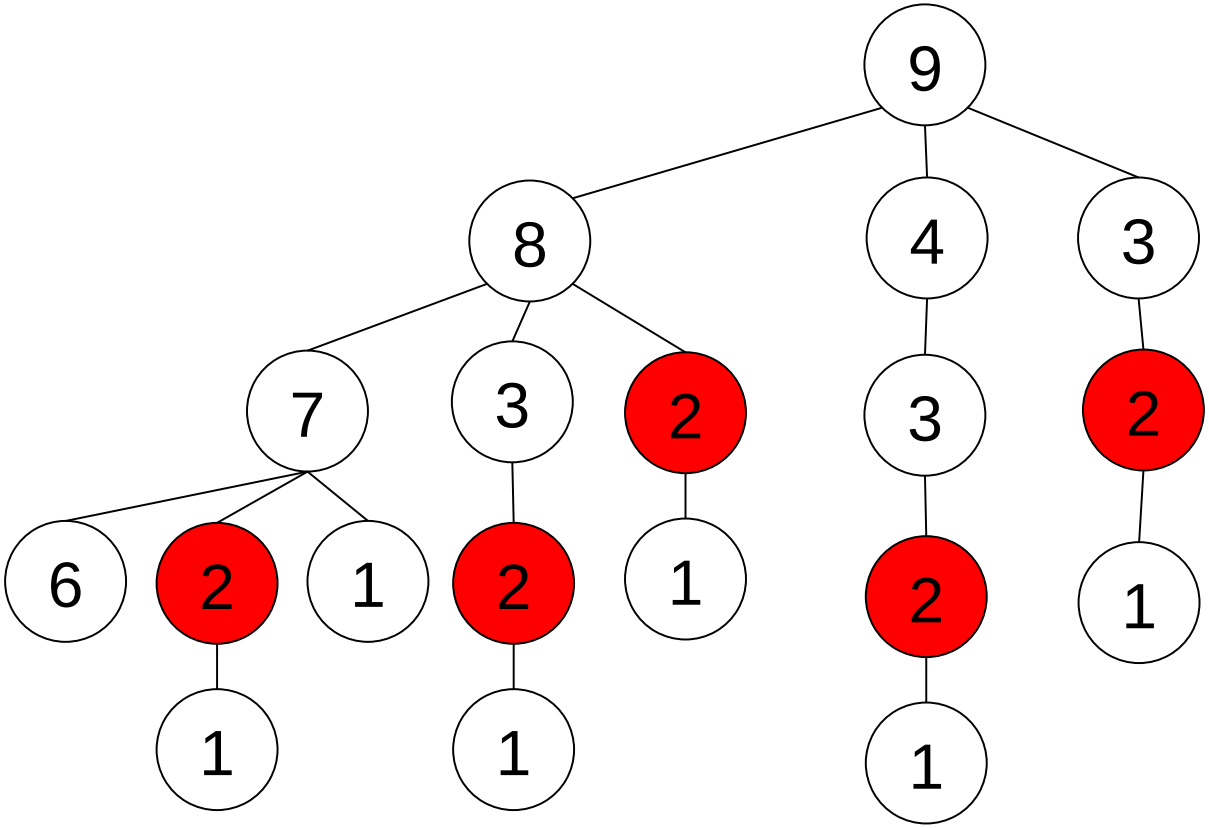
Összefésülő rendezés



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

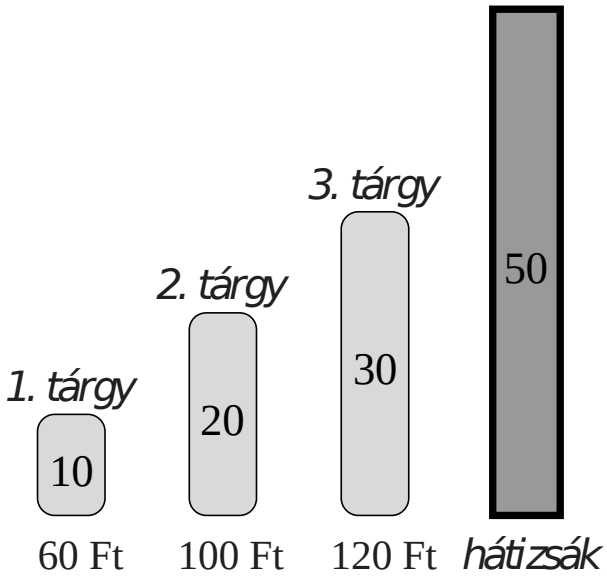
Pénzváltási feladat – rekurzív megoldás

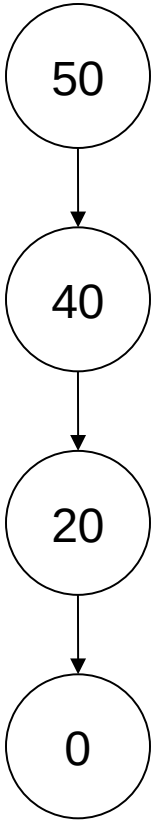
$P_1=1$, $P_2=5$, $P_3=6$, $F=9$



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Töredékes hátizsák feladat





Részfeladatokra bontási technikák

- Oszd-meg-és-uralkodj:
  - független részfeladatok
- Dinamikus Programozás:
  - ismétlődő részfeladatok
  - tipikusan optimalizálási feladatok
- Mohó algoritmusok:
  - ha adható mohó stratégia az opt feladatra

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Részfeladatokra bontási technikák

	✓	✗	✓
		>>	

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Részfeladatokra bontási technikák

- Mindhárom technika esetén lehet
  - ***rekurzív*** és ***iteratív*** implementációt is adni
  - rekurzív összefüggéssel adott futásidőket fel tudjuk oldani
- Helyesség igazolás speciális technikákat igényel

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Tömörítési feladat

2020. október

5. hét – 1. videó
S05E01

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Tömörítési alkalmazás

Cél: egy adatállomány (karaktersorozat) hatékony tömörítése adatvesztés nélkül

Intuíció: emberi nyelv szavainak kódolása

a	65868724
az	24383179
és	19288013
hogy	11902102
is	10097022
nem	9918516
de	3103363
…	
alacsonyabbrendûségi	10
beleegyezéseddel	10
megszentségtelenítésének	10

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Karaktersorozatok bináris kódolása

	a	b	c	d	e	f
Gyakoriság (ezrekben)	45	13	12	16	9	5
Fix hosszú kódszó	000	001	010	011	100	101
Változó hosszú kódszó	0	101	100	111	1101	1100

- fix hosszú kód (300K bit)
- változó hosszú kód (224K bit)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Prefix-kód

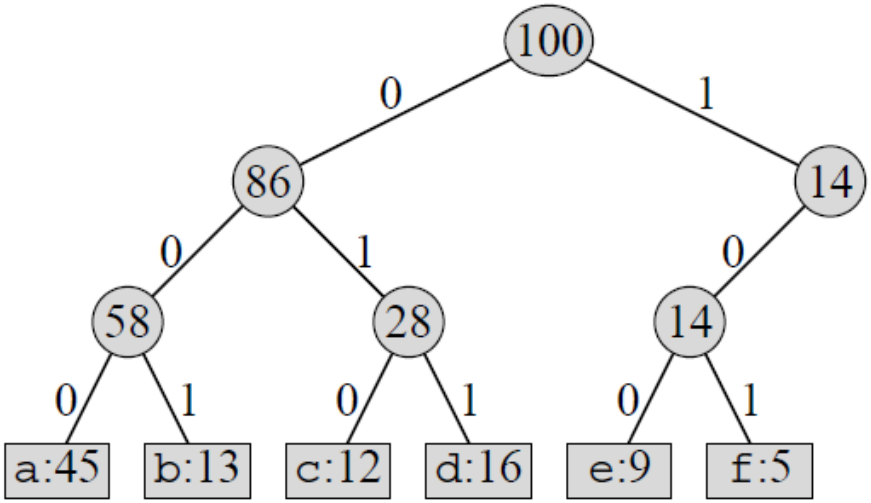
- egyik kód sem kezdőszelete egyik másik kódnak sem
- bizonyítható, hogy prefix-kóddal adható optimális tömörítés

	a	b	c	d	e	f
Változó hosszú kódszó	0	101	100	111	1101	1100

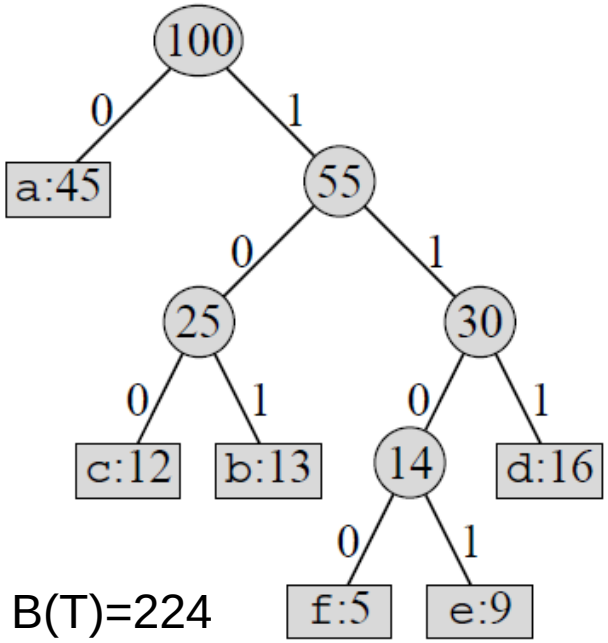
0101100 001011101

ab c aab e

prefix(-kód) fa



B(T)=300



B(T)=224

Az optimális kód teljes bináris fa: minden nem levél csúcsnak két gyereke van

T fa ***költsége:*** $B(T) = \sum_{c \in C} f(c) d_T(c).$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Tömörítési feladat

- **Bemenet:** C abécé és f(c) gyakoriságok
- **Kimenet**: minimális költségű prefix-fa

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Huffman kódolási algoritmusa

(„Huffman kód”)

2020. október

5. hét – 2. videó
S05E02

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Tömörítési feladat

- **Bemenet:** C abécé és f(c) gyakoriságok

- **Kimenet**: minimális költségű prefix-fa

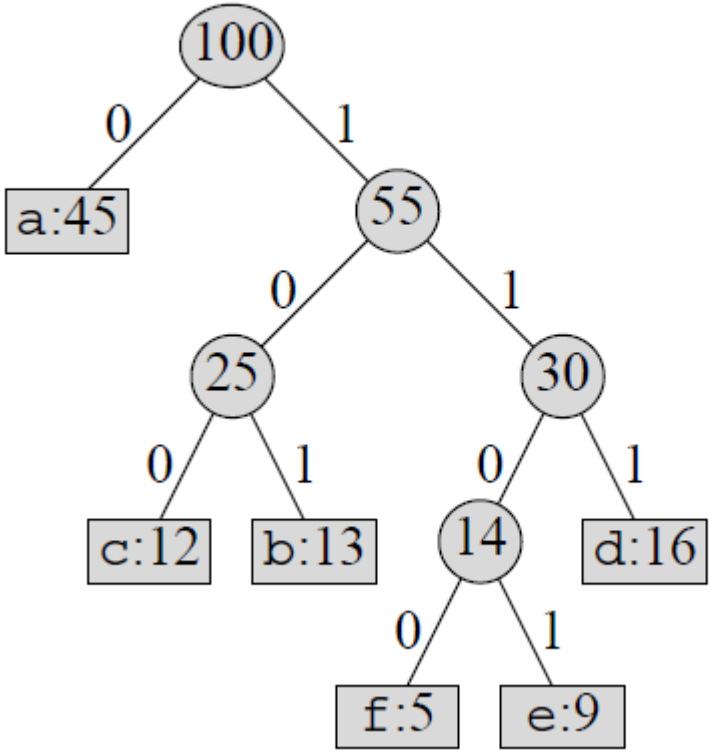
(*teljes bináris fa*, azaz |C| levél és |C|-1 belső csúcs)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Huffman kódolási algoritmusa

```
HUFFMAN(C)
n ← |C|
Q ← C
for i ← 1 to n − 1
    do új z csúcs létesítése
        bal[z] ← x ← KIVESZ-MIN(Q)
        jobb[z] ← y ← KIVESZ-MIN(Q)
        f[z] ← f[x] + f[y]
        BESZÚR(Q, z)
return KIVESZ-MIN(Q)
```

Huffman algoritmus



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Huffman-algoritmus helyessége

biztonságos:

16.2. lemma. *Legyen C tetszőleges karakterhalmaz, és legyen $f[c]$ a $c \in C$ karakter gyakorisága. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Ekkor létezik olyan optimális prefix-kód, amely esetén az x-hez és y-hoz tartozó kódszó hossza megegyezik, és a két kódszó csak az utolsó bitben különbözik.*

optimális részstruktúrák:

16.3. lemma. *Legyen C tetszőleges ábécé, és minden $c \in C$ karakter gyakorisága $f[c]$. Legyen x és y a két legkisebb gyakoriságú karakter C-ben. Tekintsük azt a C' ábécét, amelyet C-ből úgy kapunk, hogy eltávolítjuk az x és y karaktert, majd hozzáadunk egy új z karaktert, tehát $C' = C - \{x, y\} \cup \{z\}$. Az f gyakoriságok C'-re megegyeznek a C-beli gyakoriságokkal, kivéve z esetét, amelyre $f[z] = f[x] + f[y]$. Legyen T' olyan fa, amely a C' ábécé egy optimális prefix-kódját ábrázolja. Ekkor az a T fa, amelyet úgy kapunk, hogy a z levélcsúcshoz hozzákapcsoljuk gyerek csúcsként x-et és y-t, olyan fa lesz, amely a C ábécé optimális prefix-kódját ábrázolja.*

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Huffman-algoritmus futásideje

HUFFMAN(C)

$n \leftarrow |C|$
$Q \leftarrow C$
for $i \leftarrow 1$ **to** $n - 1$
    do új z csúcs létesítése
        $bal[z] \leftarrow x \leftarrow$ KIVESZ-MIN(Q)
        $jobb[z] \leftarrow y \leftarrow$ KIVESZ-MIN(Q)
        $f[z] \leftarrow f[x] + f[y]$
        BESZÚR(Q, z)
return KIVESZ-MIN(Q)

Futásidő: $O(n \log n)$

$O(\log n)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Adatszerkezetek

2020. október

6. hét – 1. videó
S06E01

Adatszerkezetek

Introduction to Data Structures

English Dictionary

City map

ABC Hardware
Cash Book - 03/01/2013 to 03/31/2013

S. no.	Date	Particulars	Debit	Credit
1	03/01/2013	Opening balance		50000
2	03/02/2013	Transport bill	2000	
3	03/07/2013	Goods sales		1500
4	03/08/2013	Bank Loan		5000
5	03/15/2013	Goods sales		1000
6	03/17/2013	Electiricty bill	1200	
7	03/21/2013	Good sales		1200
8	03/25/2013	Hardware purchase	500	
9	03/29/2013	Employee salary	20000	
10	03/31/2013	Closing Balance	35000	
		Total	58,700	58,700

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Adatszerkezetek

Az ***adatszerkezet*** adatok tárolására és szervezésére szolgáló módszer, amely lehetővé teszi a hatékony hozzáférést és módosításokat.

- Az algoritmushoz válasszunk adatszerkezetet!
- Sőt van, hogy az algoritmus a megfelelő adatszerkezeten alapul.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Absztrakt adatszerkezetek

- Adatok „dinamikus halmaza” (H)
- kulcsmező (k)
- kísérő adatok is társulhatnak hozzá, de ezzel általában nem foglalkozunk

- Műveletei definiálják az absztrakt adatszerkezetet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Absztrakt adatszerkezetek

Leggyakoribb műveletek:

KERES(H,k)
BESZÚR(H,k)
TÖRÖL(H,k)
MIN(H)
MAX(H)
ELŐZŐ(H,k)
KÖV(H,k)

módosító műveletek

lekérdező műveletek

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Adatszerkezetek

Adatszerkezet: absztrakt adatszerkezet konkrét megvalósítása

- Egy programozási nyelven is lehet több megvalósítás
- bizonyos műveletek hatékonyabbak
- megfelelő absztrakt adatszerkezet megválasztása fontos az algoritmushoz,
- megfelelő adatszerkezet megválasztása kulcs az implementáció optimális futásidejéhez!

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Listák

2020. október

6. hét – 2. videó
S06E02

Lista – absztrakt adatszerkezet

Az adatok lineáris sorrendben követik egymást. Egy kulcs többször is előfordulhat.

Műveletei:

ÉRTÉK(H,i): i. pozíción (index) a kulcs, ÉRTÉKAD(H,i,k)

KERES(H,k): első k értékű elem pozíciója

BESZÚR(H,k, i): i. pozíció után beszúr

TÖRÖL(H,k): első k értékű elem törlése

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Lista – közvetlen eléréssel

Adatok összefüggő memóriaterületen, minden index közvetlen elérésű, azaz közvetlenül olvasható/írható

ÉRTÉK(H,i) $O(1)$

KERES(H,k) $O(n)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Lista – közvetlen eléréssel

BESZÚR és TÖRÖL esetén változik a méret és át kell mozgatni elemeket!

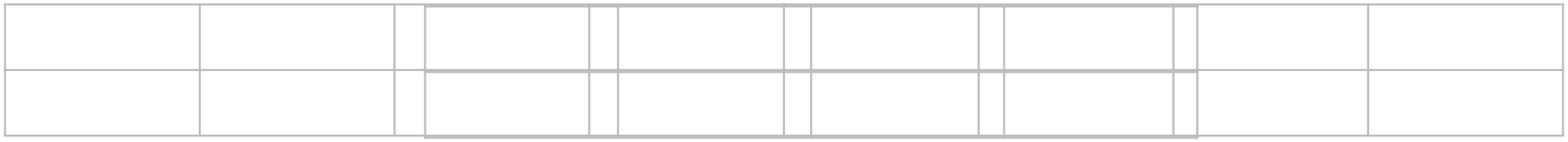
BESZÚR(H,25,0)

növeljük/csökkentsük a tömb méretét 1-el?

minden beszúrás új memóriaallokáció és minden *n* elemet át kell másolni az új helyre

BESZÚR(H,k,i)=O(n) TÖRÖL(H,k)=O(n)

Lista – közvetlen eléréssel



Cél: átméretezés minél ritkábban kelljen!

Tömb méret = kapacitás

Ha tele van a tömb duplázzuk meg a kapacitást

Ha negyedére csökken felezzük meg a kapacitást

Így nem kell mindig az egész tömböt másolni

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

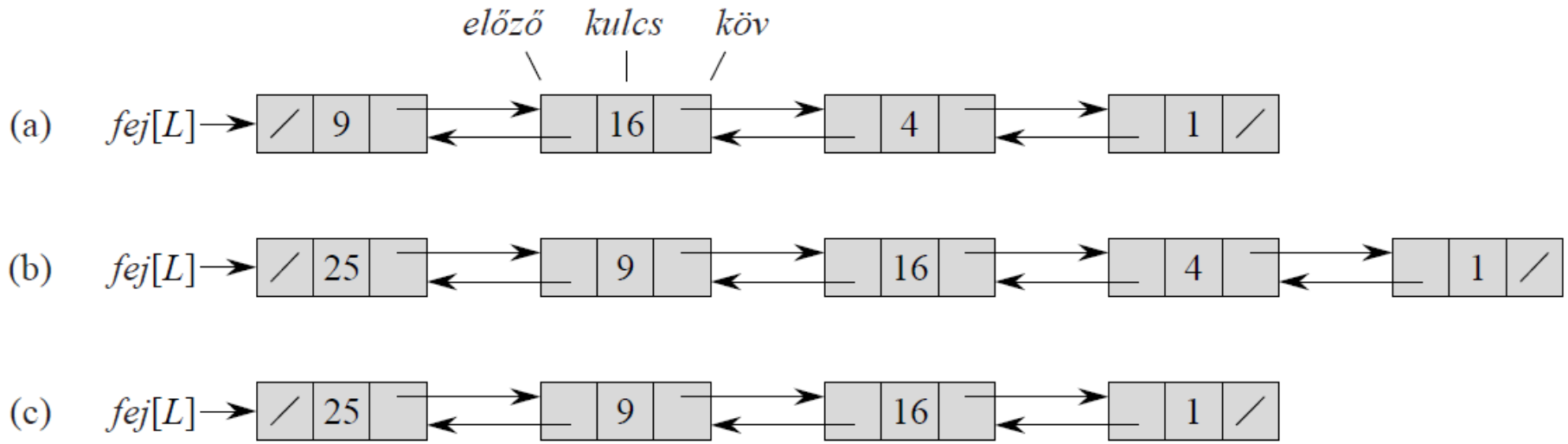
Lista – láncolt listás megvalósítás

Minden kulcs mellé mutatókat tárolunk a következő/megelőző elemre

(nem feltétlenül vannak egyazon memóriaterületen)

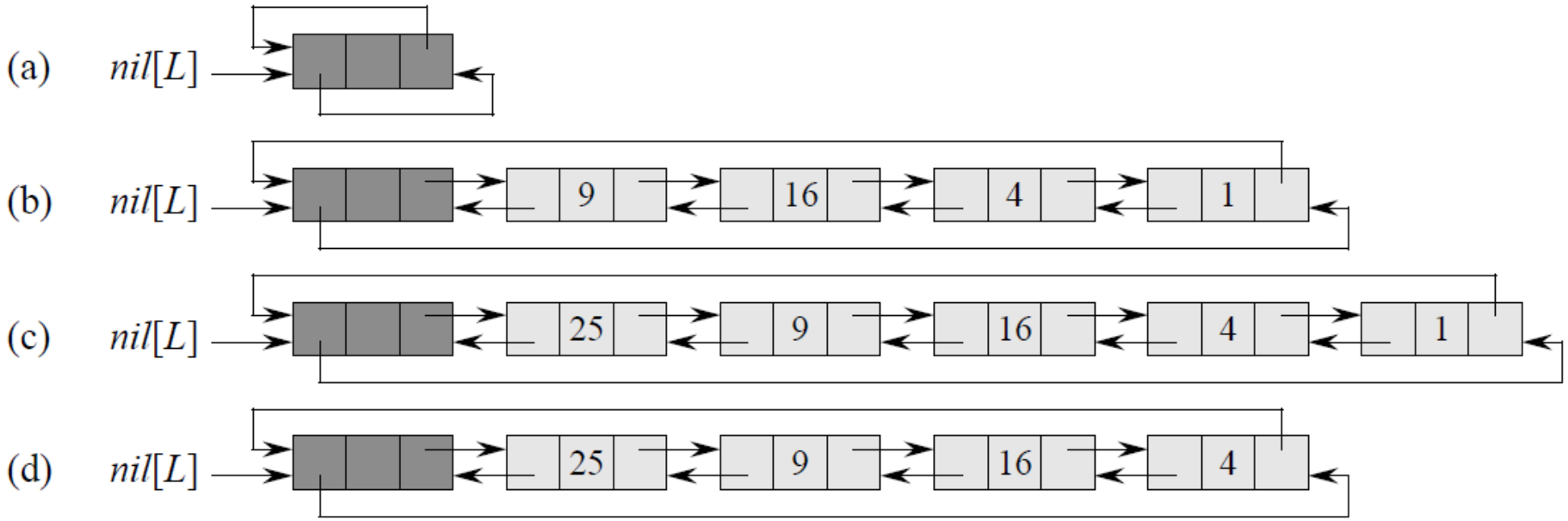
- Egyszeresen láncolt lista: csak a következő elemre mutat
- Kétszeresen láncolt lista: minden kulcshoz két mutató, következő és megelőző elemre
- ciklikus lista: utolsó elemre rákövetkezője az első elem
- őrszem: egy nil elem ami mindig a lista elejére (fej) mutat

Lista – láncolt listás megvalósítás



10.3. ábra. (a) Az $\{1, 4, 9, 16\}$ dinamikus halmaz ábrázolása kétszeresen láncolt L listával. A lista minden eleme egy kulcsmezőt és két mutató mezőt tartalmaz; a (nyilakkal ábrázolt) pointerek rendre a megelőző és a rákövetkező elemekre mutatnak. Az utolsó elem *köv* mutatója és a fej *előző* mutatója NIL, amit 'per'-jel (slash) jelöl. A *fej*[L] pointer a lista fejére mutat. **(b)** A LISTÁBA-BESZÚR(L, x) művelet végrehajtása után, ahol *kulcs*[x] = 25, a láncolt lista a 25 kulcsértékű új objektummal bővül, és ez lesz a lista új feje. Ez az új objektum a korábbi, 9 kulcsértékű fejre mutat. **(c)** A LISTÁBÓL-TÖRÖL(L, x) művelet végrehajtásának eredménye, ahol x a 4-es kulcsú elemre mutat.

Lista – láncolt listás megvalósítás



10.4. ábra. Egy ciklikus kétszeresen láncolt őrszemes lista. A *nil*[*L*] őrszem a lista feje és vége között helyezkedik el. A *fej*[*L*] attribútumra tovább nincs szükség, mivel a lista feje elérhető a *köv*[*nil*[*L*]] mutatóval. **(a)** Az üres lista. **(b)** A 10.3(a) ábrán szereplő lista őrszemmel kiegészített ciklikus változata; a lista fejében a kulcs értéke 9, a lista végének a kulcsa 1. **(c)** A lista a LISTÁBA-BESZÚR′(*L*, *x*) művelet végrehajtása után, ahol *kulcs*[*x*] = 25. A beszúrt új objektum lesz a lista feje. **(d)** A lista az 1-es kulcsú objektum törlése után. A lista új vége a 4-es kulcsú elem lesz.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Lista – láncolt listás megvalósítás

ÉRTÉK(H,i) $O(n)$

KERES(H,k) $O(n)$

BESZÚR(H,k,i) =ÉRTÉK()+$O(1)$

TÖRÖL(H,k) =ÉRTÉK()+$O(1)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Lista megvalósítások

Közvetlen elérésű megvalósítás:

- ÉRTÉK() konstans idejű
- módosító műveletek lassúak

Láncolt listás megvalósítás:

- ÉRTÉK() lassú, módosító műveletek gyorsak
- sok memória kell a mutatóknak

Lista – JAVA megvalósítások

```
import java.util.*;

List<Object> al = new ArrayList<>();
List<Object> ll = new LinkedList<>();
```

interface ami definiálja a szükséges Lista műveleteket
ABSZTRAKT ADATSZERKEZET

különböző megvalósítások
ADATSZERKEZET

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

```
private static void test_list(List<Object> l)
{
   long t = System.currentTimeMillis();
   for(int i=0;i<100000;++i)
      l.add(0, i);
   System.out.println("insert:"+
                (System.currentTimeMillis()-t)
+"ms");
   t=System.currentTimeMillis();
   for(int i=0;i<l.size();++i)
      l.get(i);
   System.out.println("get:"+
                (System.currentTimeMillis()-t)
+"ms");
}
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

```
test_list(new ArrayList<>());
insert: 1042ms
get: 2ms
```

```
test_list(new LinkedList<>());
insert: 5ms
get: 4157ms
```

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Verem és Sor

2020. október

6. hét – 3. videó
S06E03

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Verem(Stack) és Sor (Queue)

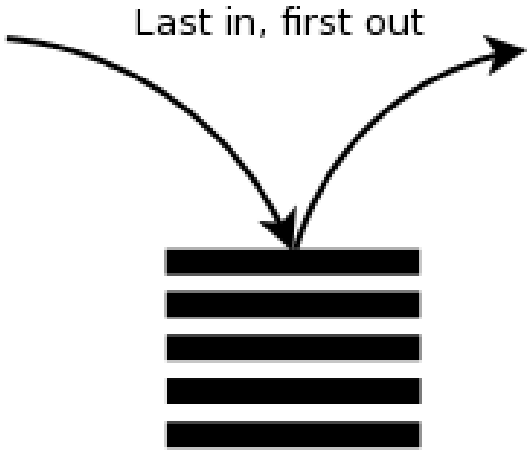
Olyan listák ahol a beszúrás és törlés csak meghatározott pozícióban történhet.

- Verem: legutoljára beszúrt elemet vehetjük csak ki (last-in, first-out = LIFO)
- Sor: legrégebbi elemet vehetjük csak ki (first-in, first-out = FIFO)

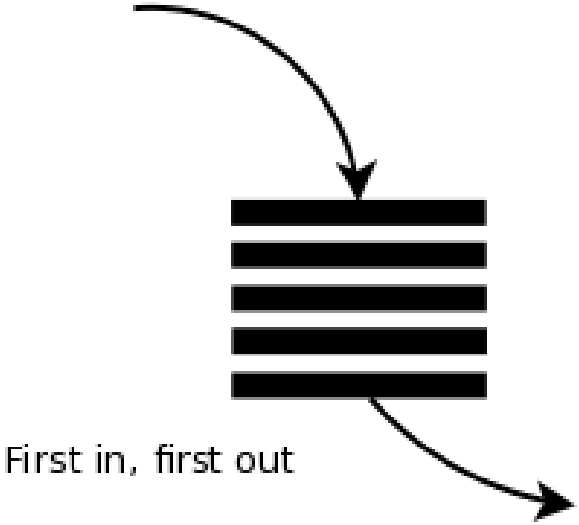
Speciális műveletigény miatt hatékonyabb (spec) megvalósítások!

Verem(Stack) és Sor (Queue)

Stack:



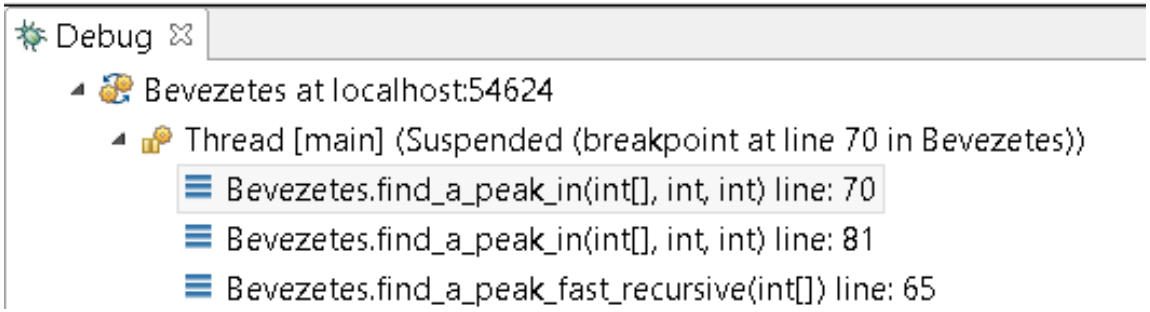
Queue:



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Verem alkalmazások

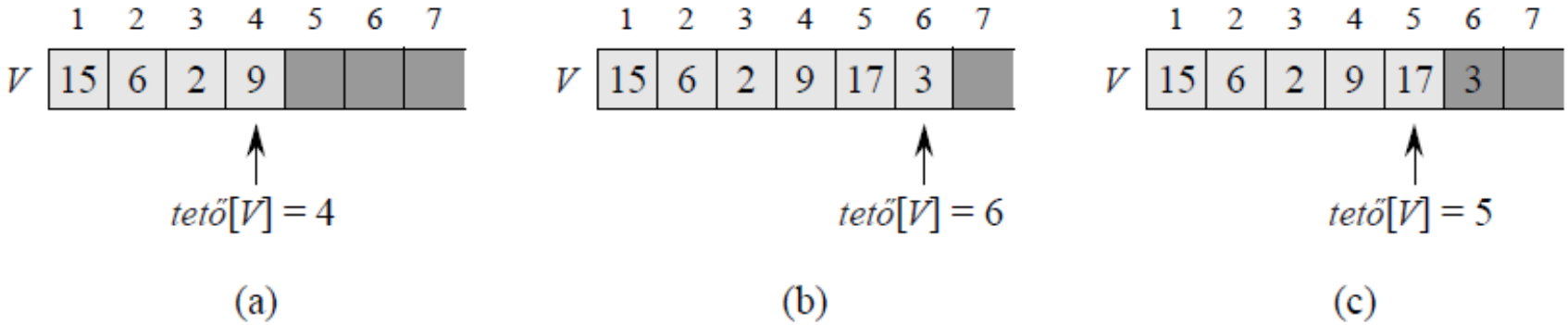


Fordítsunk meg egy szót:

alga → agla

Verem

Megvalósítás fix méretű tömbbel



10.1. ábra. A V verem tömbös megvalósítása. A verem elemeit a világos mezők tartalmazzák. **(a)** A V verem 4 elemet tartalmaz. A tető elem a 9. **(b)** A V verem a VEREMBE(V,17) és a VEREMBE(V, 3) műveletek végrehajtása után. **(c)** A V verem a VEREMBŐL(V) művelet meghívása után a 3-at adja vissza, amely az utoljára betett elem. Annak ellenére, hogy a 3 a tömbben marad, a veremnek már nem eleme; az új tetőelem a 17.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Verem

ÜRES-VEREM(V)

```
if tető[V] = 0
    then return IGAZ
    else return HAMIS
```

VEREMBE(V, x)

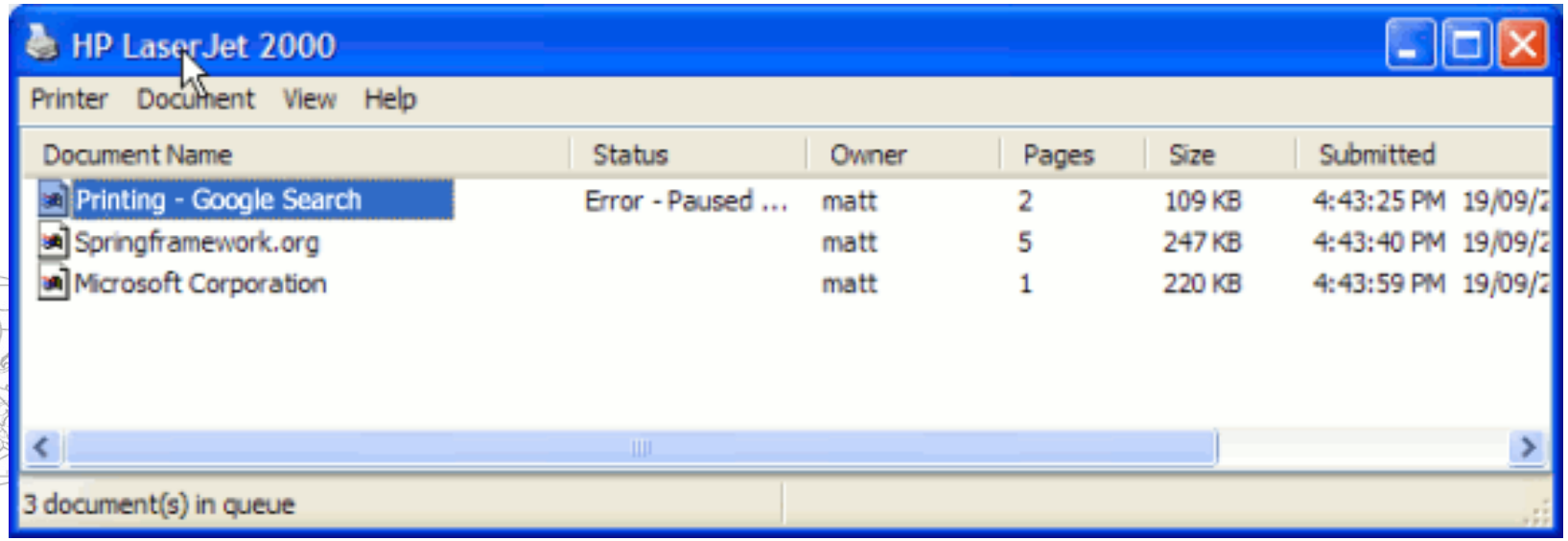
```
tető[V] ← tető[V] + 1
V[tető[V]] ← x
```

VEREMBŐL(V)

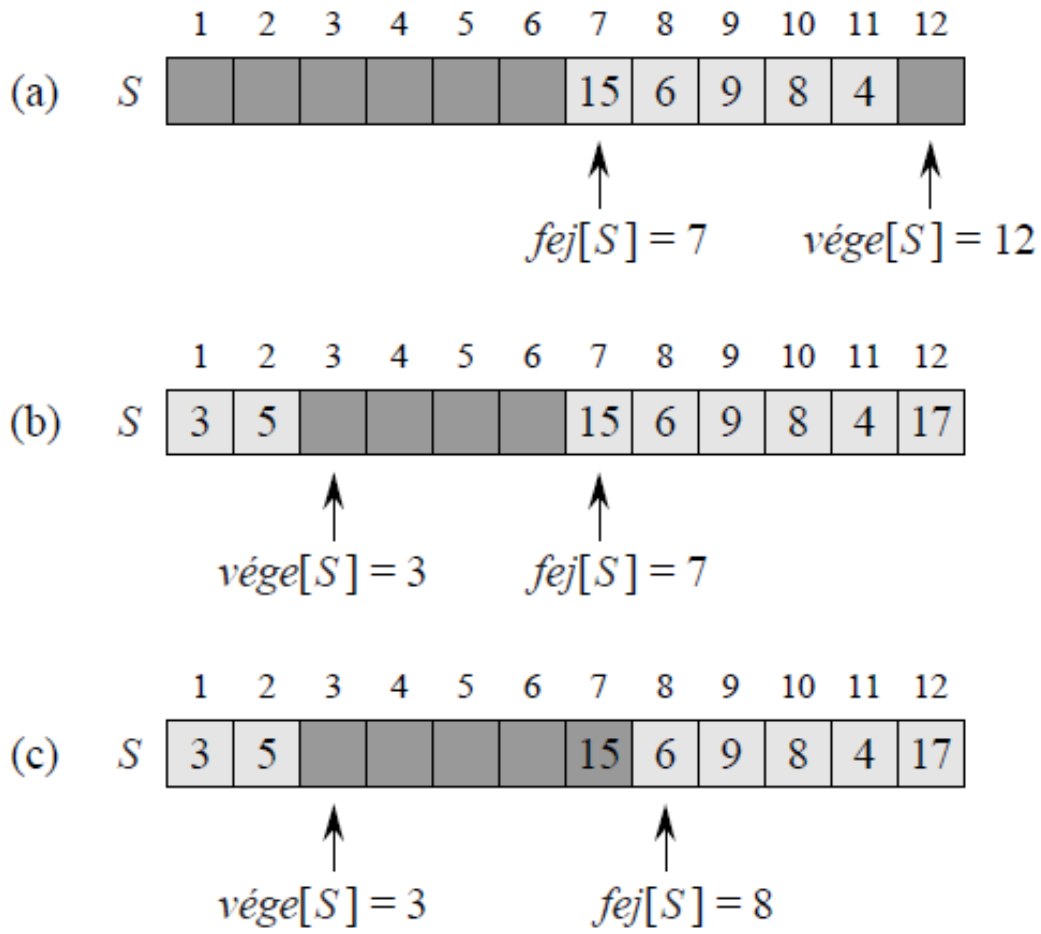
```
if ÜRES-VEREM(V)
    then error „alulcsordulás”
    else tető[V] ← tető[V] − 1
         return V[tető[V] + 1]
```

$O(1)$

Sor alkalmazások



Sor



10.2. ábra. Egy sor megvalósítása az $S[1..12]$ tömbbel. A sor elemeit a világos mezők tartalmazzák. **(a)** A sor öt eleme az $S[7..11]$ pozíciókban helyezkedik el. **(b)** A sor a SORBA(S, 17), a SORBA(S, 3) és a SORBA(S, 5) műveletek végrehajtása után. **(c)** A SORBÓL(S) meghívása a 15-öt adja vissza. A 15 a tömbben marad ugyan, de a sornak nem eleme, az új fej a 6-os kulcsértékű elem.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Sor

SORBA(S, x)

```
S[vége[S]] ← x
if vége[S] = hossz[S]
    then vége[S] ← 1
    else vége[S] ← vége[S] + 1
```

SORBÓL(S)

```
x ← S[fej[S]]
if fej[S] = hossz[S]
    then fej[S] ← 1
    else fej[S] ← fej[S] + 1
return x
```

$O(1)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Prioritási sor és Kupac

2020. október

6. hét – 4. videó
S06E04

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Prioritási (elsőbbségi) sor

Kulcsok érkezése sorrendje lényegtelen
mindig maximális (minimális) elemet akarjuk kivenni

Műveletek:

BESZÚR(H,k)

MAX(H)

KIVESZ-MAX(H)

Prioritási sor – naive megvalósítás

Verem? Sor?

- nem az érkezési sorrend számít most!

Listás megvalósítás:

- rendezetlen lista

  BESZÚR(H,k) – $O(1)$

  KIVESZ-MAX(H) – $O(n)$

- rendezett lista

  BESZÚR(H,k) – $O(n)$

  KIVESZ-MAX(H) – $O(1)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

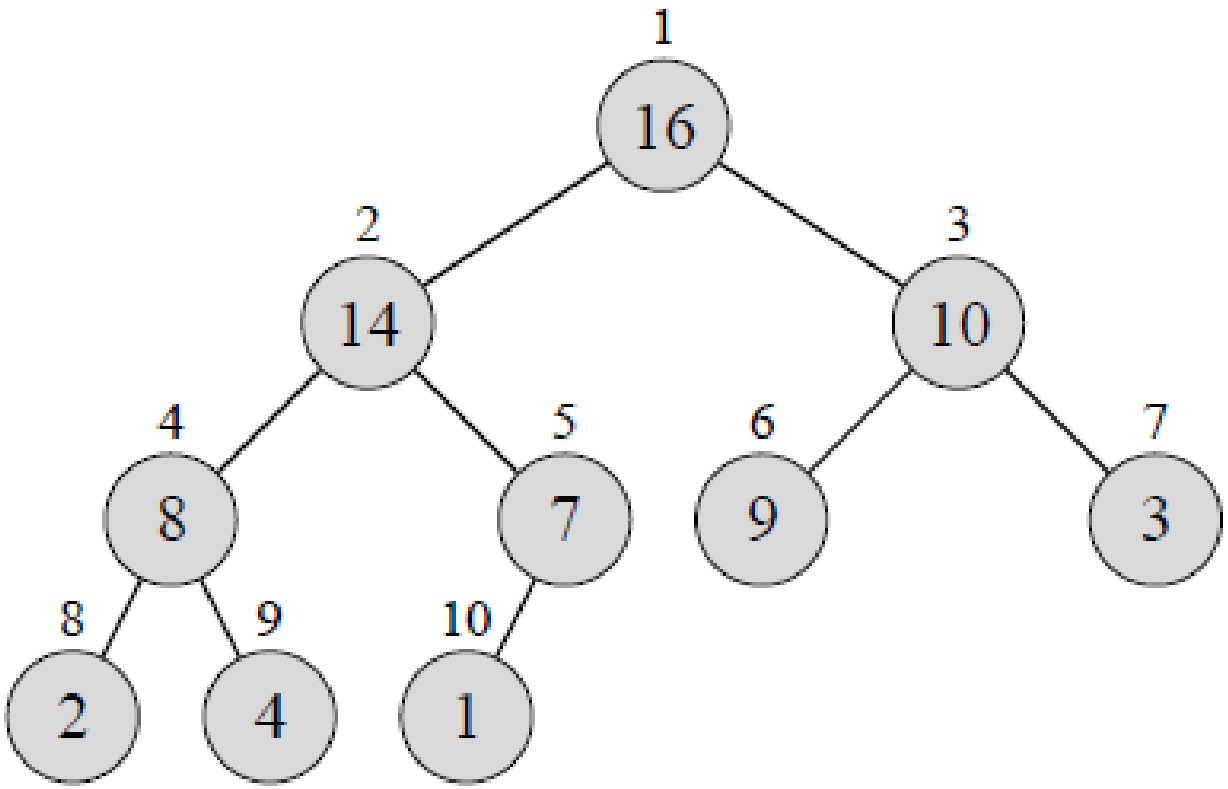
Prioritási sor hatékony megvalósítása

A **kupac** (heap) egy hatékony megvalósítása a prioritási sor absztrakt adatszerkezetnek.

A kupac egy ***majdnem teljes bináris fa***, amelynek minden csúcsa legalább akkora, mint gyerekei → maximális elem a gyökérben van

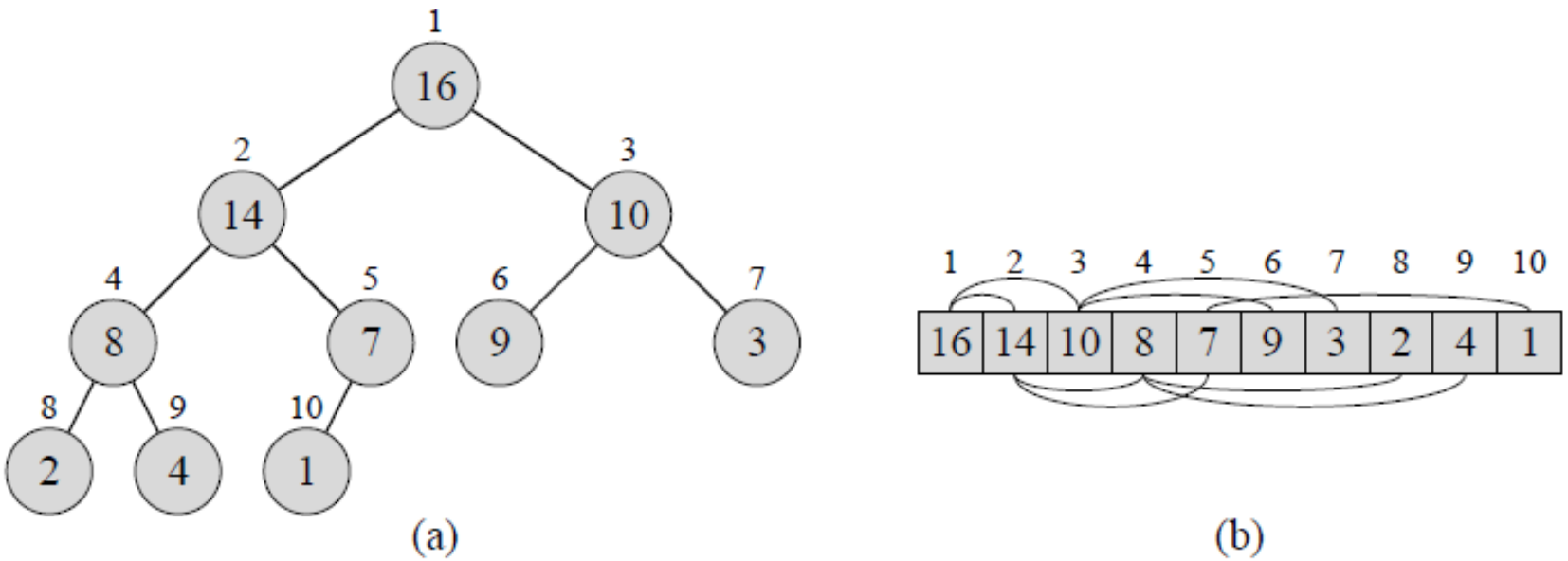
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Kupac



https://visualgo.net/en/heap

Kupac tömbös megvalósítása



6.1. ábra. A maximum-kupac mint bináris fa **(a)** és mint tömb **(b)**. A körök belsejébe írt szám a fa csúcsában tárolt érték, a kör fölé írt szám a csúcsnak megfelelő tömbelem indexe. A tömb alatti és feletti ívek mutatják a szülő-gyerek viszonyokat; a szülő mindig balra található a gyerekeihez képest. A fa magassága három; a 4 indexű csúcs (melynek 8 az értéke) egy magasságú.

SZÜLŐ(i)

1 **return** $\left\lfloor \frac{i}{2} \right\rfloor$

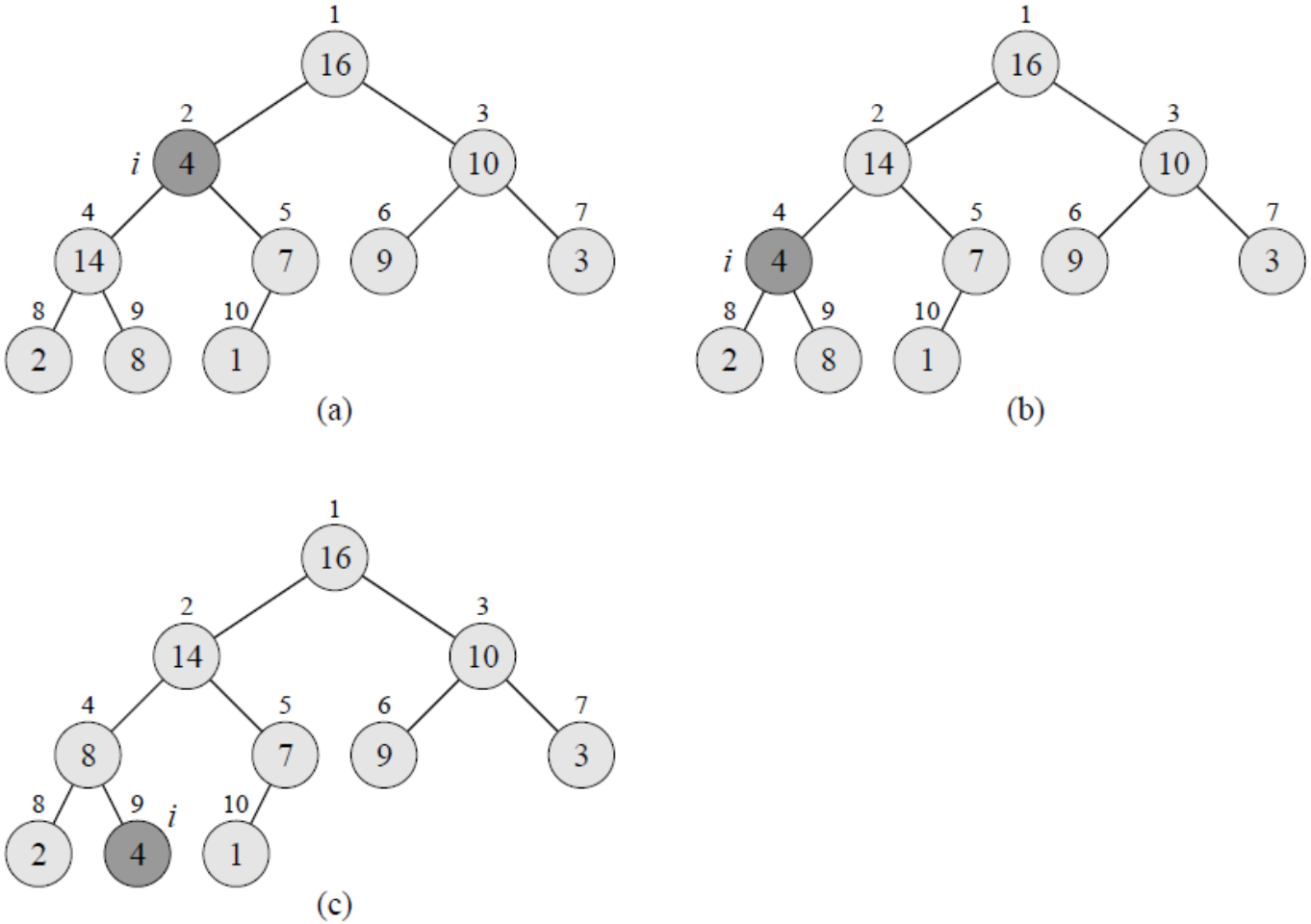
BAL(i)

1 **return** $2i$

JOBB(i)

1 **return** $2i + 1$

Kupactulajdonság fenntartása



6.2. ábra. A MAXIMUM-KUPACOL(A, 2) működése, ahol *kupac-méret*[A] = 10. **(a)** Kiinduláskor $A[2]$ az i = 2-es csúcsnál sérti a maximum-kupactulajdonságot, mivel nem nagyobb a gyerekeinél. A maximum-kupactulajdonságot helyreállítjuk a 2-es csúcsra nézve azzal, hogy $A[2]$ és $A[4]$ elemeket felcseréljük, de elrontjuk a 4-es csúcsra nézve **(b)**. A MAXIMUM-KUPACOL(A, 4) rekurzív hívása i-t 4-re állítja. $A[4]$ és $A[9]$ felcserélése után, ahogy azt **(c)** mutatja, a 4 értékű elem helyére kerül, és a MAXIMUM-KUPACOL(A, 9) rekurzív hívás már nem talál változtatni valót az adatszerkezeten.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Kupactulajdonság fenntartása

```
MAXIMUM-KUPACOL(A, i)
l ← BAL(i)
r ← JOBB(i)
if l ≤ kupac-méret[A] és A[l] > A[i]
    then legnagyobb ← l
    else legnagyobb ← i
if r ≤ kupac-méret[A] és A[r] > A[legnagyobb]
    then legnagyobb ← r
if legnagyobb ≠ i
    then A[i] ↔ A[legnagyobb] csere
         MAXIMUM-KUPACOL(A, legnagyobb)
```

$$T(n) \le T\left(\frac{2n}{3}\right) + \Theta(1) \xrightarrow[\text{Helyettesítő módszer}]{\text{Rekurziós fa módszer}} T(n) = O(\lg n).$$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Prioritási sor

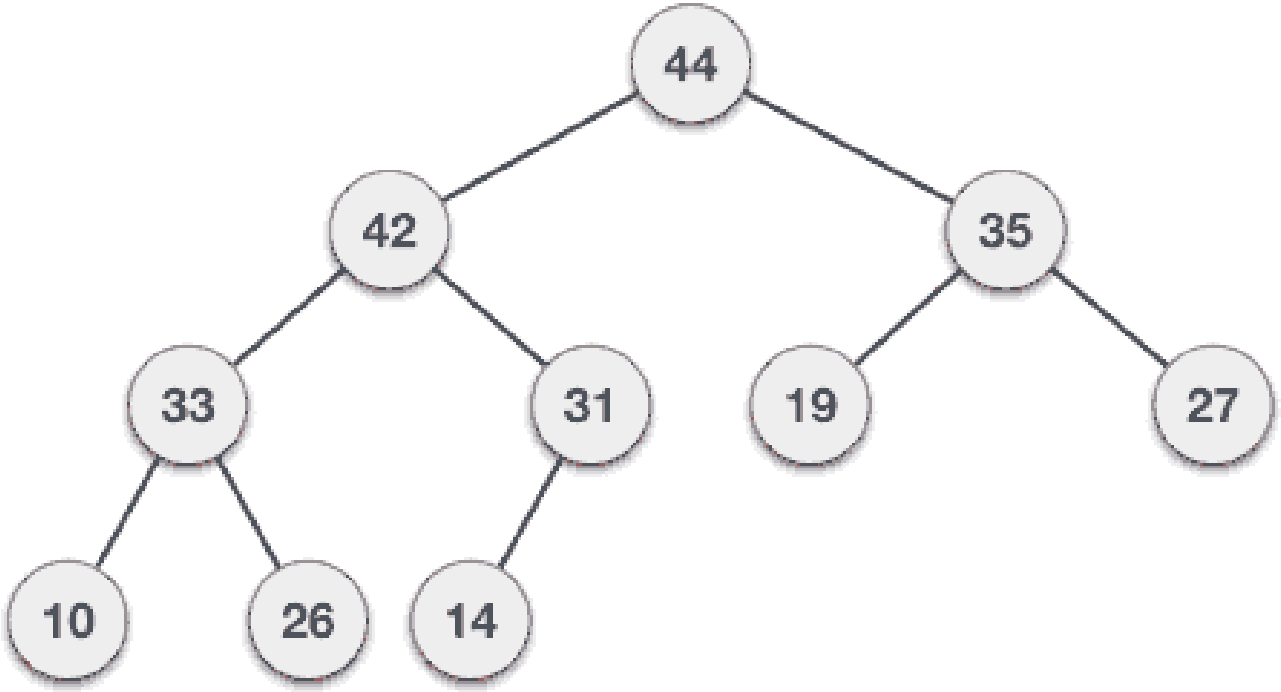
```
KUPAC-MAXIMUMA(A)
1  return A[1]
```

```
KUPACBÓL-KIVESZ-MAXIMUM(A)
1  if kupac-méret[A] < 1
2     then error „kupacméret alulcsordulás”
3  max ← A[1]
4  A[1] ← A[kupac-méret[A]]
5  kupac-méret[A] ← kupac-méret[A] – 1
6  MAXIMUM-KUPACOL(A, 1)
7  return max
```

***O*(log*n*)**

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Prioritási sor – kivesz max



Forrás: www.tutorialspoint.com

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Prioritási sor beszúrás

1. Szúrjuk be levelet (tömb végére)
2. Szülővel való cserével emeljük a helyére a levelet

Input 35 33 42 10 14 19 27 44 26 31

***O*(log*n*)**

a kupac magassága log*n*

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Prioritási sor kupac megvalósítással

BESZÚR(H,k)	$O(\log n)$
MAX(H)	$\Theta(1)$
KIVESZ-MAX(H)	$O(\log n)$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Elemi adatszerkezetek implementációi

2020. október

6. hét – 5. videó
S06E05

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

ArrayList vs LinkedList

- Általában ArrayList gyorsabb és kevesebb memóriát igényel
- DE ha a módosító műveletek vannak többségben válasszuk a LinkedList-et!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Java tömbök

Tömböket

```
int[] a = new int[100000];
```

is lehet használni, de csak akkor érdemes használni ha fix a méret

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Iterátorok

Az iterátor egy mutató az adatszerkezet egy elemére. Lista esetében előre(hátra) léptethetjük.

JAVA:

```
List<String> list = new ArrayList<>();
Iterator<String> i = list.iterator();
while(i.hasNext()){
    i.next();
}
```

vagy összes elem bejárása:

```
for(String s : list)
    s="hi";
```

```java
Stack<Integer> s = new Stack<>();
s.push(4);s.push(3);s.push(5);
while(!s.isEmpty())
   System.out.print(s.pop()+" ");

Queue<Integer> q = new ArrayDeque<>();
q.add(4);q.add(3);q.add(5);
while(!q.isEmpty())
   System.out.print(q.poll()+" ");

PriorityQueue<Integer> pq = new
PriorityQueue<>();
pq.add(4);pq.add(3);pq.add(5);
while(!pq.isEmpty())
   System.out.print(pq.poll()+" ");
```

Minden programozási nyelven megvalósítottak!!!

	Java	C++ std::	C#	python	Scala
Közvetlen elérésű lista	ArrayList	vector	List	--	ArrayBuffer
Láncolt lista	LinkedList	list	LinkedList	deque	LinkedList
Verem	Stack	stack	Stack	list.pop()	Stack
Sor	ArrayDeque	queue	Queue	queue. LifoQueue	Queue
Prioritási sor	PriorityQueue	priority_queue	SortedList	queue. PriorityQueue	PriorityQueue

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Leszállópálya nyilvántartási feladat

2020. október

7. hét – 1. videó
S07E01

Leszállópálya nyilvántartási probléma

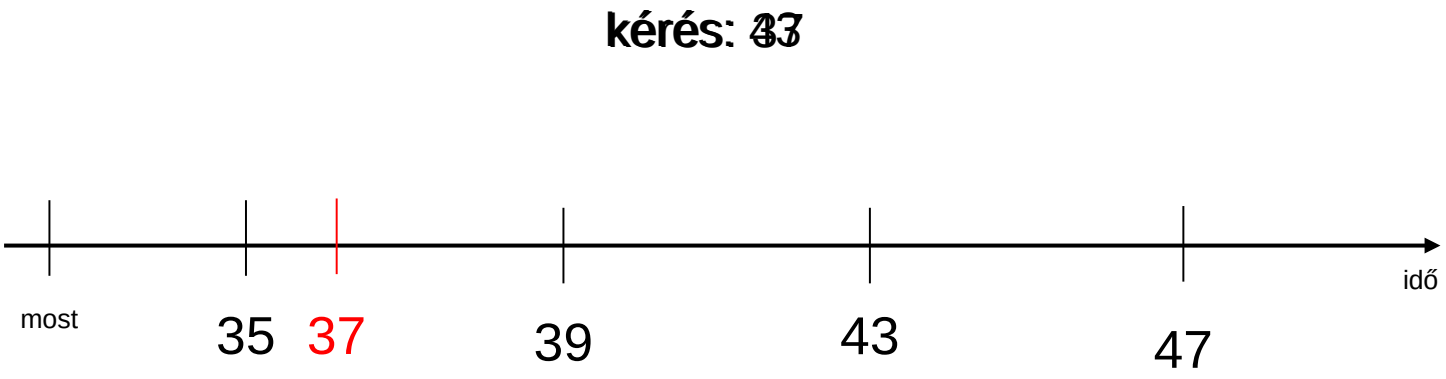
- Repterünknek egyetlen kifutópályája van.
- A landolási időket nyilvántartó rendszert fejlesztünk.

- Minden repülő elküld egy *t* időpillanatot amikor le szeretne szállni
- Ha biztonságos a leszállás, azaz ±3 percen belül nincs ütemezett leszállás akkor rögzítsük a kérést
- Akik már leszálltak azokat távolítsuk el a nyilvántartásból

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Feladat: https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-006-introduction-to-algorithms-fall-2011/lecture-videos/lecture-5-binary-search-trees-bst-sort/

Leszállópálya nyilvántartási probléma



1. KERESsük meg a legnagyobb elemet R-ben ami nem nagyobb, mint a *t* kérés
2. Ha ez 3 távolságon belül van *t*-től *return nil* egyébként keressük meg a RÁKÖVETKEZŐ elemet is
3. Ha a rákövetkező elem 3 távolságon belül van *t*-től *return nil* egyébként BESZÚRjuk *t*-t R-be

Másik use-case:
1. Ha most>MIN(R) akkor TÖRÖLjük MIN(R)-et

KERES, BESZÚR, TÖRÖL, MIN, RÁKÖVETKEZŐ legyen $O(\log|R|)$ idejű!

Milyen adatszerekzetet használjunk?

n=|R| T(BESZUR)=T(TORLES)

- rendezetlen (láncolt) lista:

  KERES $O(n)$, BESZÚR $O(1)$, KÖVETKEZŐ $O(n)$, MIN $O(n)$

- rendezett tömb (bináris keresés):

  KERES $O(\log n)$, BESZÚR $O(n)$, KÖVETKEZŐ $O(1)$, MIN $O(1)$

- kupac:

  KERES $O(n)$, BESZÚR $O(\log n)$, KÖVETKEZŐ $O(n)$, , MIN $O(1)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Fák számítógépes reprezentációja

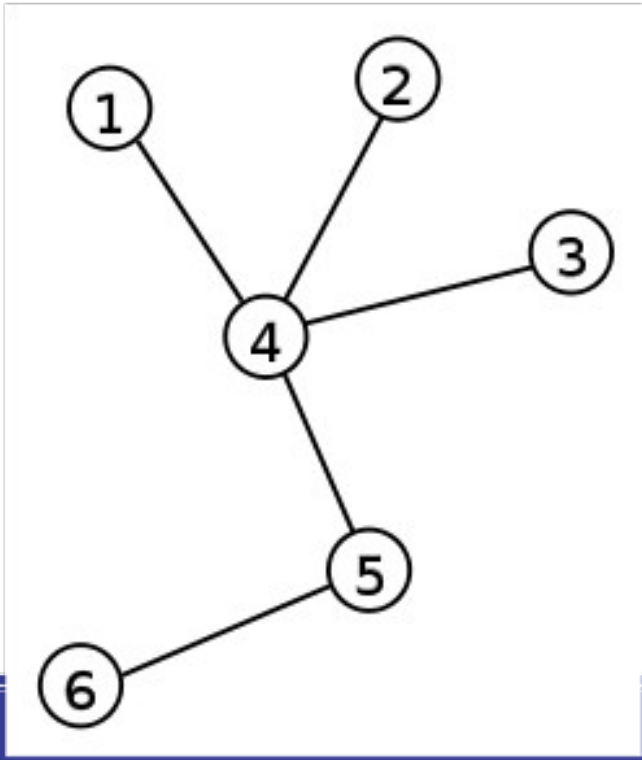
2020. október

7. hét – 2. videó
S07E02

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Fa

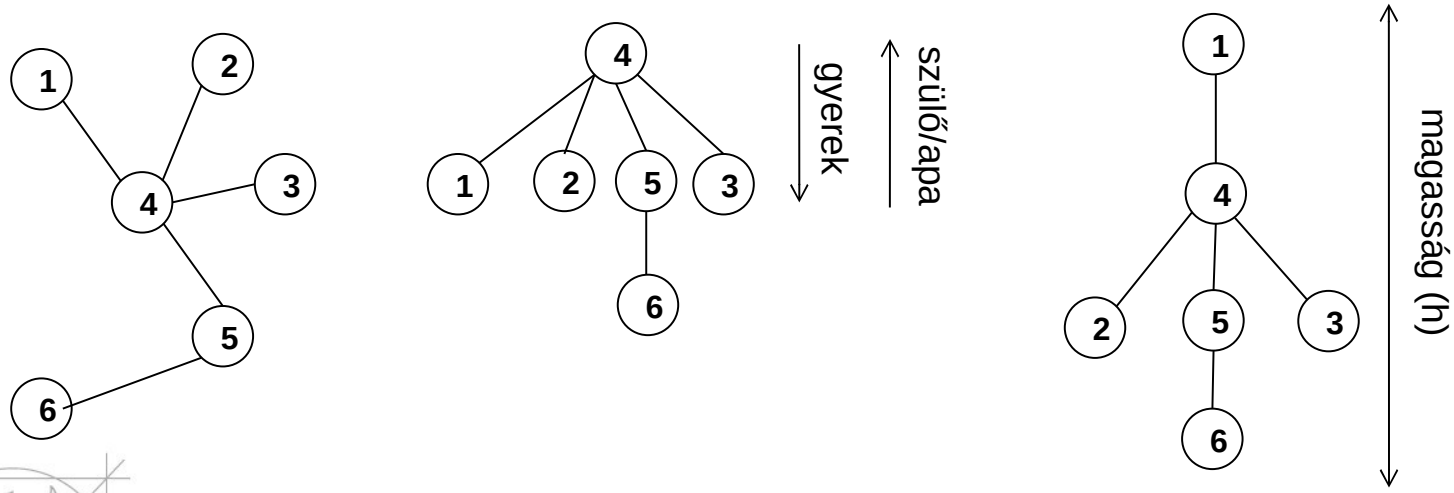
- Fa = összefüggő, körmentes **gráf**
- Bármely két csúcsát *pontosan egy* út köti össze
- A fa elsőfokú csúcsait **levél**nek hívjuk. Egy nem levél csúcs a fában **belső csúcs**.



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Bináris fa

Gyökeres fa: van egy kitűntetett csúcsa, a gyökér

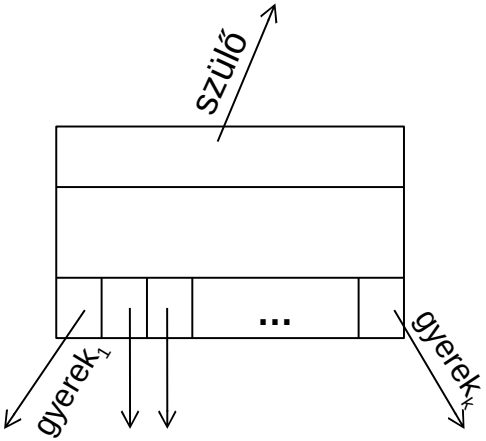


Bináris fa: gyökeres fa, ahol minden csúcsnak legfeljebb két gyereke van

Fák számítógépes reprezentációja

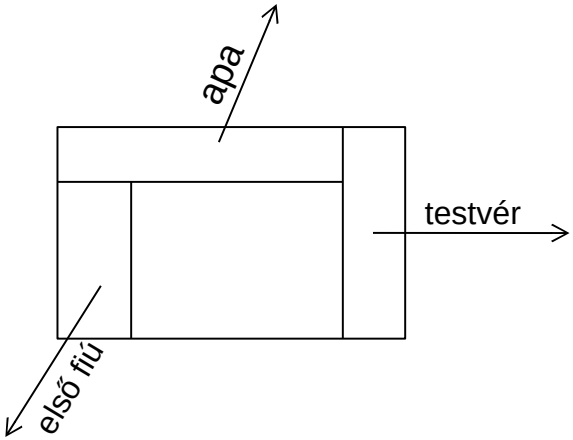
- csúcsokat és éleiket reprezentáljuk
- maga a fa egy mutató a gyökérre

gyerek állista



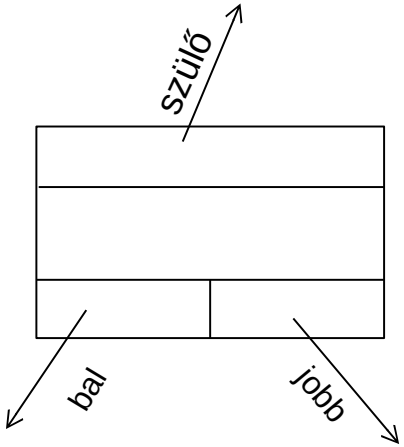

```
class Node{
  Object key;
  Node parent;
  List<Node> children;
}
```

első fiú, apa, testvér




```
class Node{
  Object key;
  Node parent;
  Node first_child;
  Node brother;
}
```

bináris fa




```
class Node{
  Object key;
  Node parent;
  Node left;
  Node right;
}
```

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Bináris keresőfák

(Binary Search Tree, BST)

2020. október

7. hét – 3. videó
S07E03

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Adatszerkezet

KERES

BESZÚR

TÖRÖL

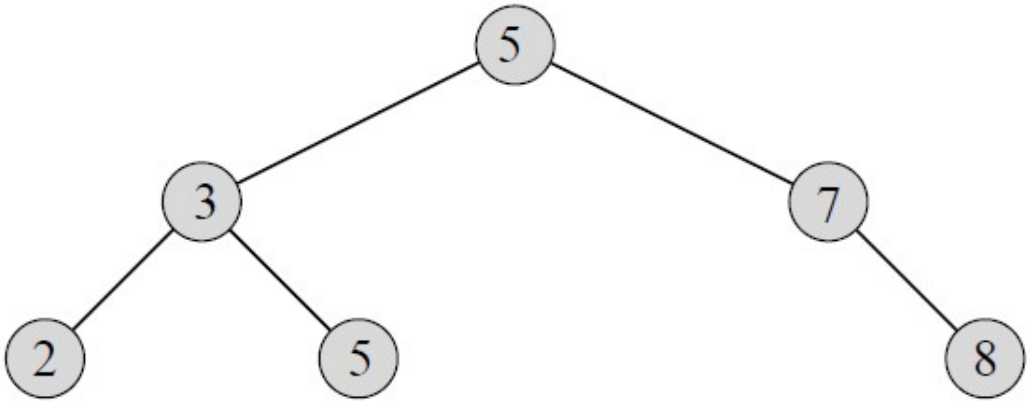
MIN/MAX

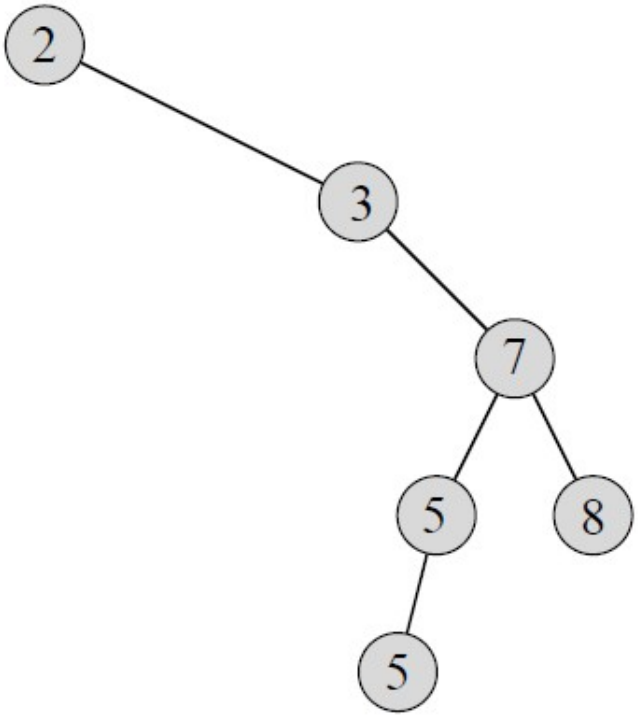
KÖVETKEZŐ/ELŐZŐ

mind legyen $O(\log n)$ idejű!

Bináris keresőfa

Legyen x az adott bináris keresőfa egy tetszőleges csúcsa. Ha y az x bal oldali részfájának egy csúcsa, akkor $kulcs[y] \leq kulcs[x]$. Ha y az x jobb oldalára esik, akkor $kulcs[x] \leq kulcs[y]$.

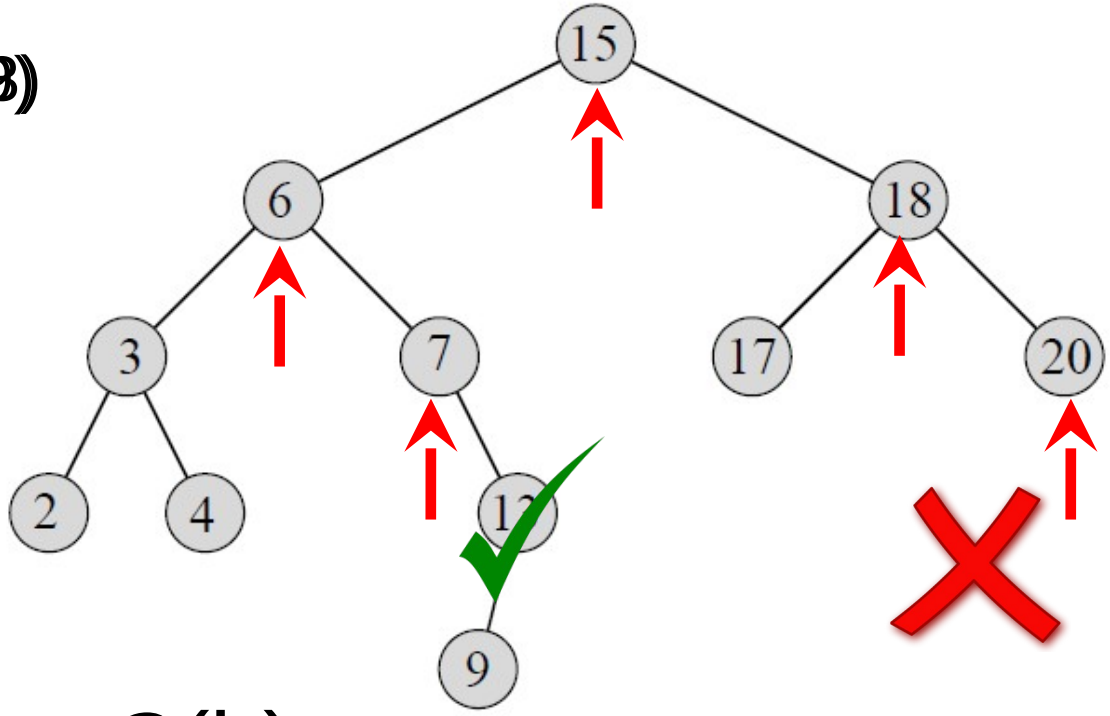




SZEGEDI TUDOMÁNYEGYETEM
SCIENTIARUM SZEGEDIENSIS

KERES

KERES(13)



$O(h)$

FÁBAN-KERES(x, k)

```
if x = NIL vagy k = kulcs[x]
    then return x
if k < kulcs[x]
    then return FÁBAN-KERES(bal[x], k)
    else return FÁBAN-KERES(jobb[x], k)
```

FÁBAN-ITERATÍVAN-KERES(x, k)

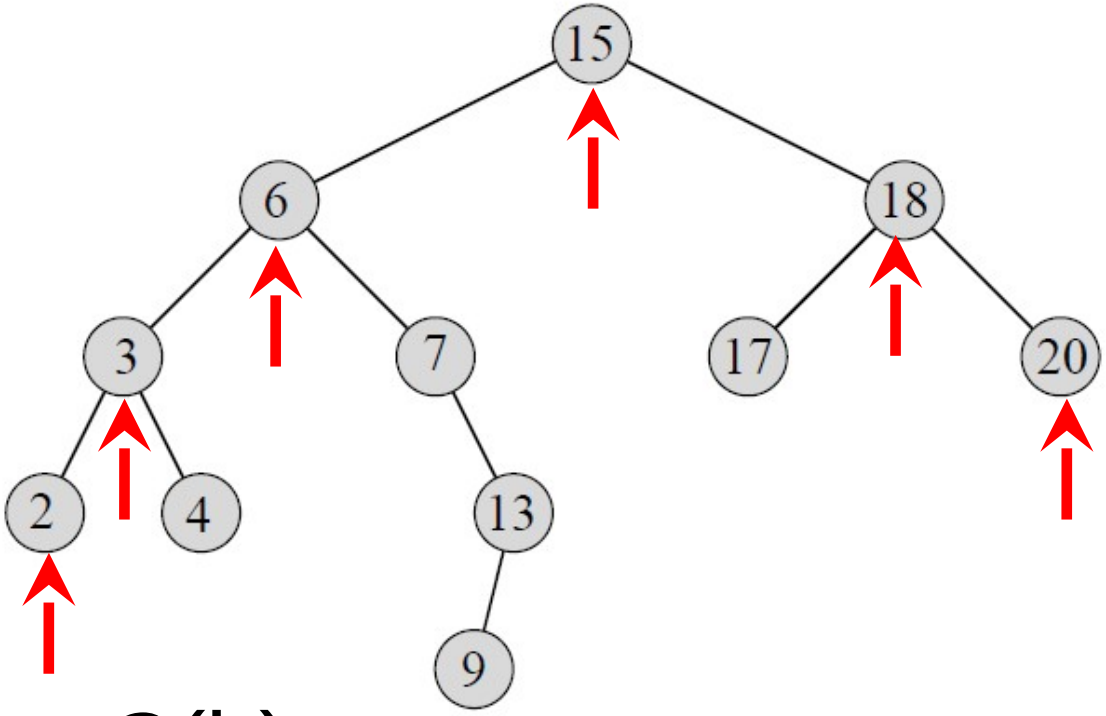
hibás a könyv!

```
while x ≠ NIL és k ≠ kulcs[x]
    do if k < kulcs[x]
        then x ← bal[x]
        else x ← jobb[x]
return x
```

MIN/MAX

MIN()

MAX()



$O(h)$

FÁBAN-MINIMUM(x)

```
while bal[x] ≠ NIL
        do x ← bal[x]
return x
```

FÁBAN-MAXIMUM(x)

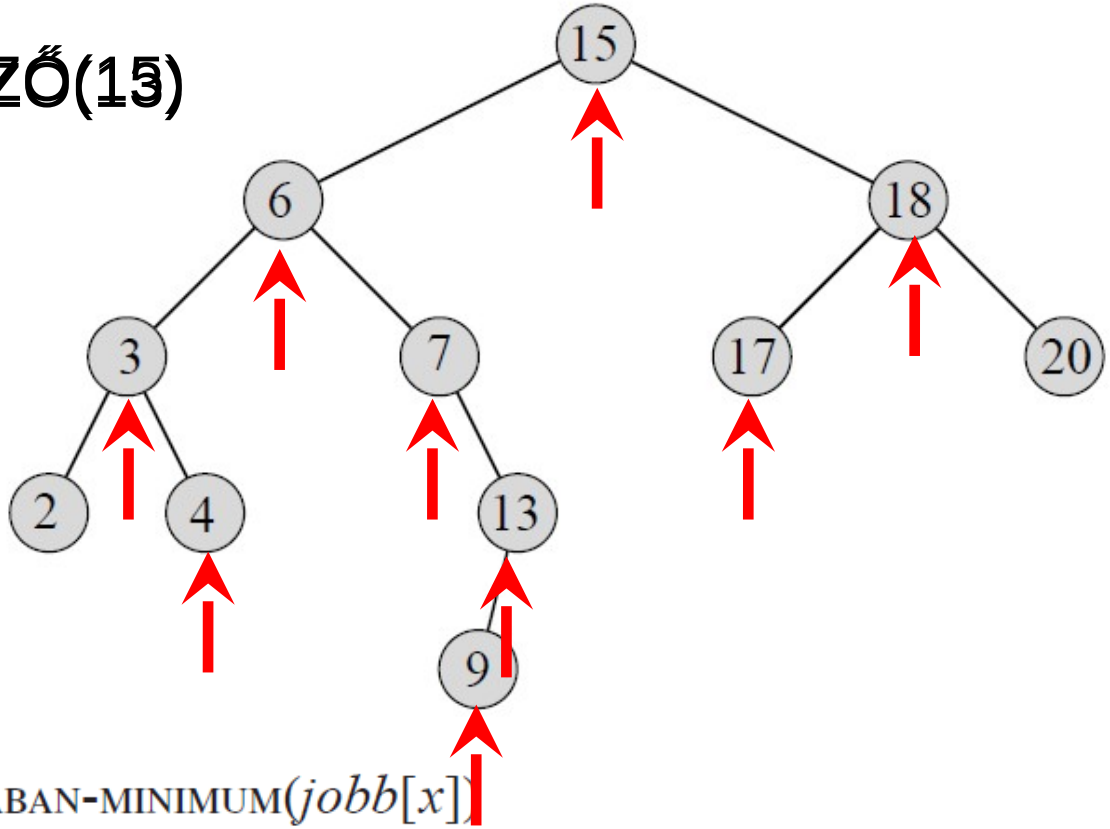
```
while jobb[x] ≠ NIL
        do x ← jobb[x]
return x
```

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

KÖVETKEZŐ/ELŐZŐ

KÖVETKEZŐ(15)

ELŐZŐ(15)



FÁBAN-KÖVETKEZŐ(x)

```
if jobb[x] ≠ NIL
    then return FÁBAN-MINIMUM(jobb[x])
y ← p[x]
while y ≠ NIL és x = jobb[y]
    do x ← y
        y ← p[y]
return y
```

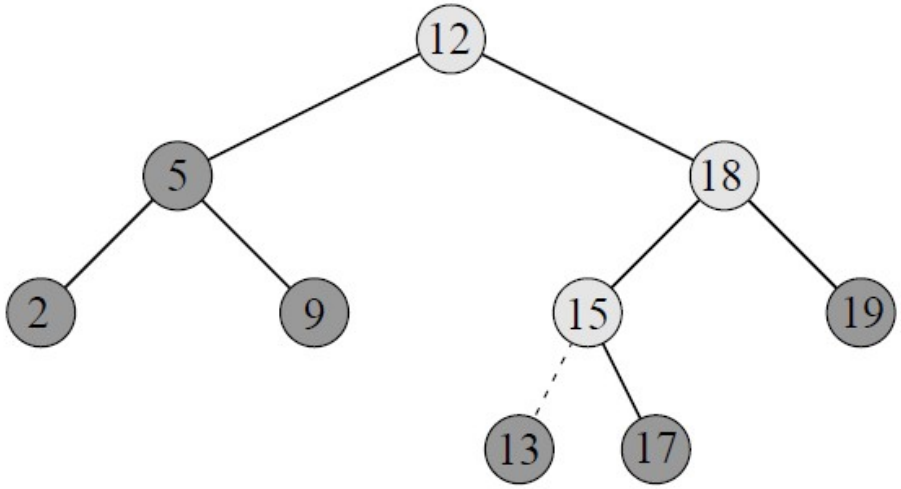
O(h)

SZEGEDI TUDOMÁNYEGYETEM
RUM SZEGEDIENSIS

BESZÚR

FÁBA-BESZÚR(T, z)

```
y ← NIL
x ← gyökér[T]
while x ≠ NIL
        do y ← x
            if kulcs[z] < kulcs[x]
                then x ← bal[x]
                else  x ←jobb[x]
p[z] ← y
if y = NIL
        then gyökér[T] ← z                ▷ Üres T fa esetén.
        else if kulcs[z] < kulcs[y]
                then bal[y] ← z
                else  jobb[y] ← z
```

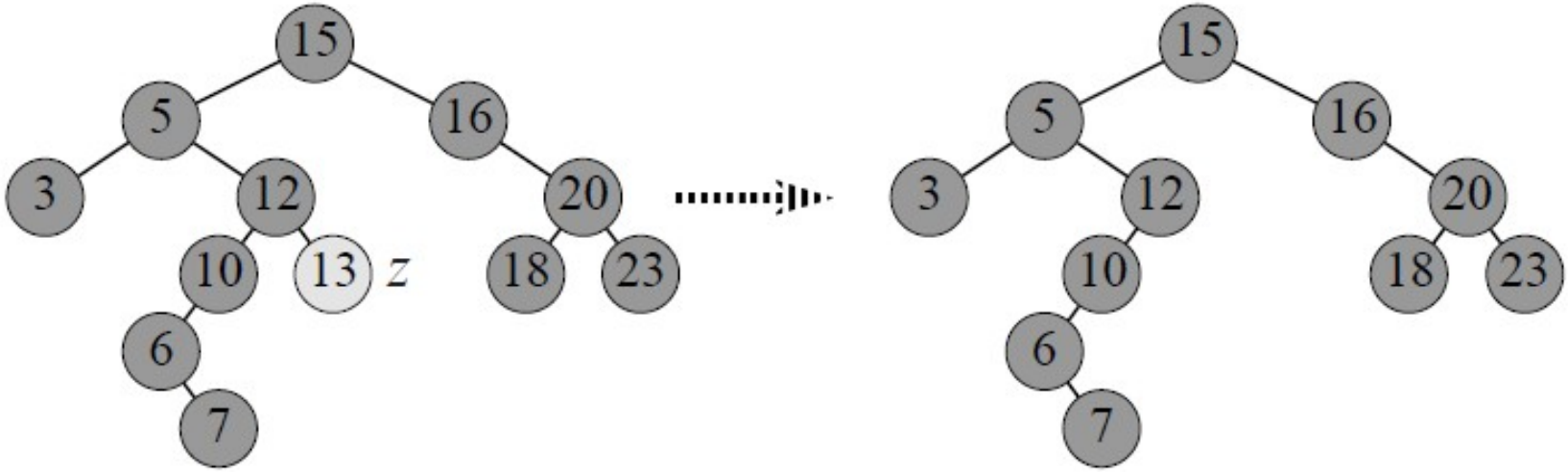


O(h)

TÖRÖL

Ha törlendő csúcs levél

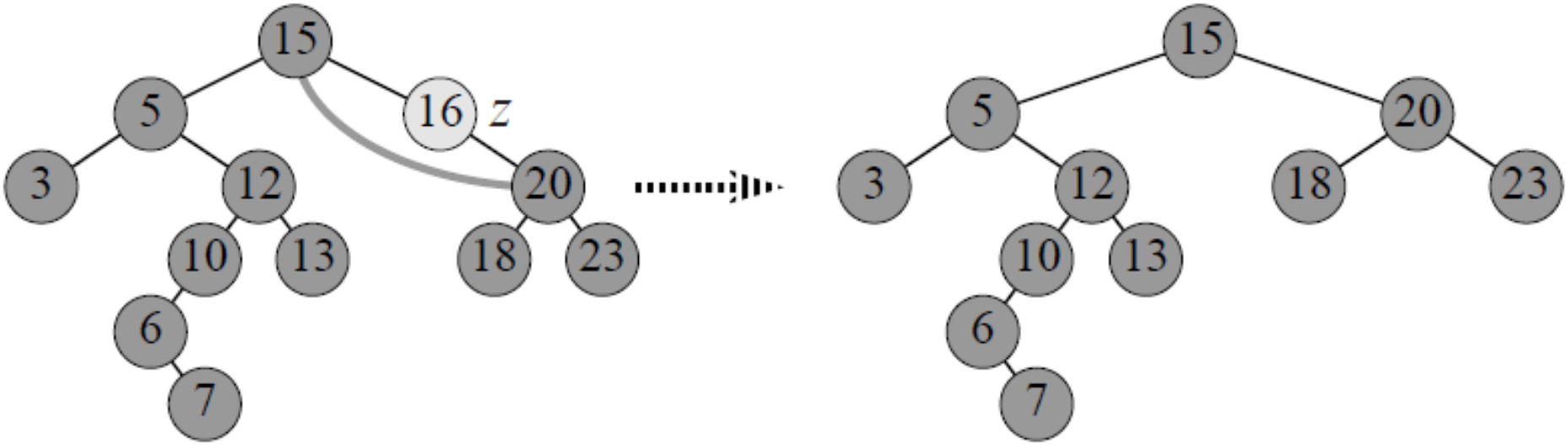
TÖRÖL(13)



TÖRÖL

Ha törlendő csúcsnak egy gyereke van

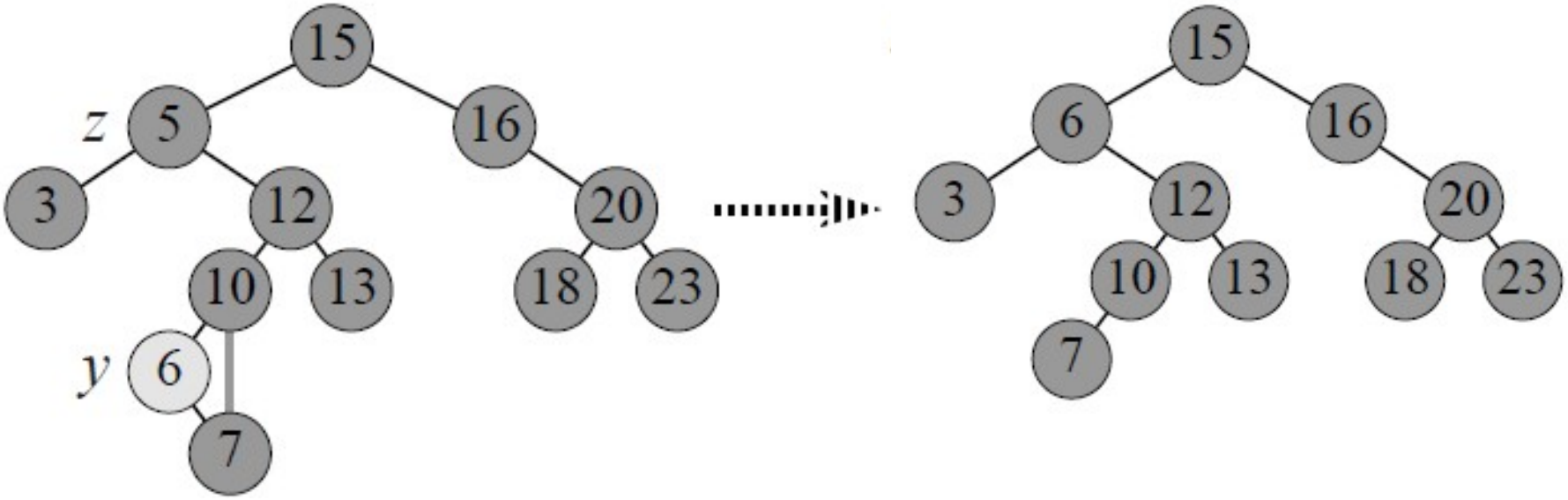
TÖRÖL(16)



TÖRÖL

Ha törlendő csúcsnak két gyereke van

TÖRÖL(5)



TÖRÖL

```
FÁBÓL-TÖRÖL(T, z)
if bal[z] = NIL vagy jobb[z] = NIL
    then y ← z
    else  y ← FÁBAN-KÖVETKEZŐ(z)
if bal[y] ≠ NIL
    then x ← bal[y]
    else  x ← jobb[y]
if x ≠ NIL
    then p[x] ← p[y]
if p[y] = NIL
    then gyökér[T] ← x
    else  if y = bal[p[y]]
              then bal[p[y]] ← x
              else  jobb[p[y]] ← x
if y ≠ z
    then kulcs[z] ←kulcs[y]
              ▷ ha y-nak más mezői is vannak, azok átmásolása
return y
```

$O(h)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Keresőfa vs Kupac

	Kupac	Kereső fa
KERES()	$O(n)$	$\boldsymbol{O(h)}$
MAX()	$\boldsymbol{O(1)}$	$O(h)$
BESZÚR()	$\boldsymbol{O(h)}$	$O(h)$
TÖRÖL()	$\boldsymbol{O(h)}$	$O(h)$
KÖVETKEZŐ()	$O(n)$	$\boldsymbol{O(h)}$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Leszállópálya nyilvántartási algoritmus

2020. október

7. hét – 4. videó
S07E04

Leszállópálya nyilvántartási algoritmus

```
y ← nil
x ← gyökér[T]
while x ≠ nil
    do y ← x
        if kulcs[z] < kulcs[x]
          then x ← bal[x]
          else x ←jobb[x]
if ABS(kulcs[z]  - kulcs[y]) < 3
  then return NIL
if kulcs[z] < kulcs[y]
   then if kulcs[z]  - ELÖZŐ(kulcs[y]) < 3
            then return NIL
            else bal[y] ← z
   else if KÖVETKEZŐ(kulcs[y]) - kulcs[z] < 3
            then return NIL
            else jobb[y] ← z
```

O(h)

Leszállópálya nyilvántartási algoritmus

másik use-case: TÖRÖL(MIN(R))

A minimális elemnek legfeljebb egy gyereke van, így a harmadik esetet nem is kell implementálni

$O(h)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Kiegyensúlyozott bináris keresőfák

2020. október

7. hét – 5. videó
S07E05

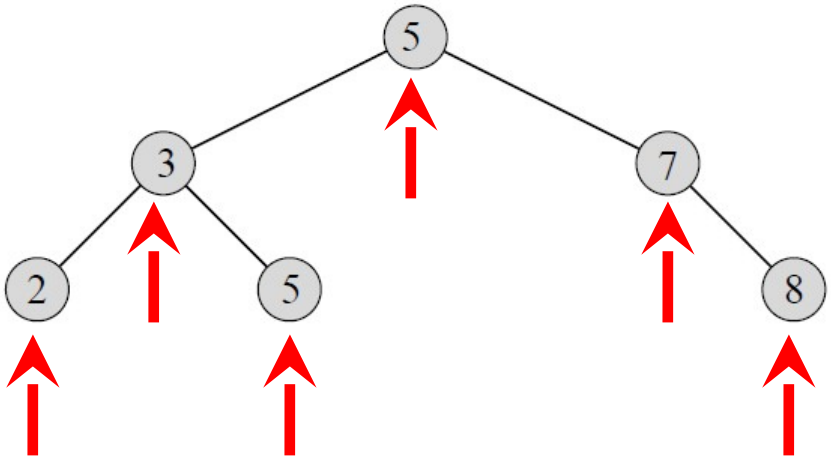
Inorder fabejárás

if *x*≠NIL

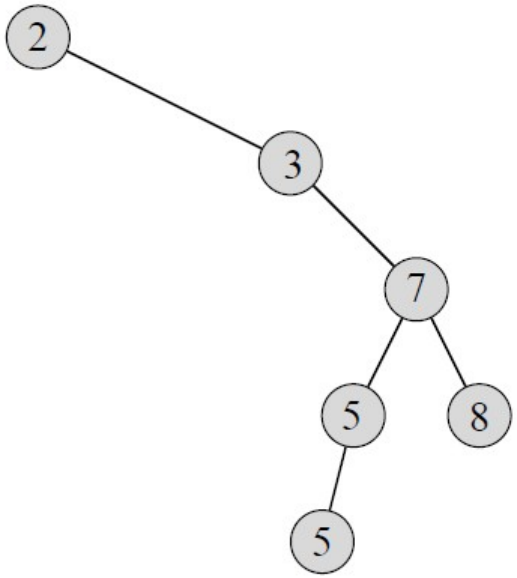
then FABEJÁRÁS(*bal*[x])

muvelet(*kulcs*[x])

FABEJÁRÁS(*jobb*[x])



2 3 5 5 7 8



2 3 5 5 7 8

Fa magassága (h)

- Általános bináris keresőfa esetén semmi sem biztosítja, hogy $h=\Theta(\log n)$

  Pl szúrjuk be sorban: 2,3,5,5,7,8
- A kupac egy **teljes** bináris fa, azaz legfeljebb egy belső csúcsa van aminek csak egy gyereke van.
- Egy teljes bináris fa magassága $\lceil \log_2(n+1) \rceil$
- A keresőfák nem teljesek!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

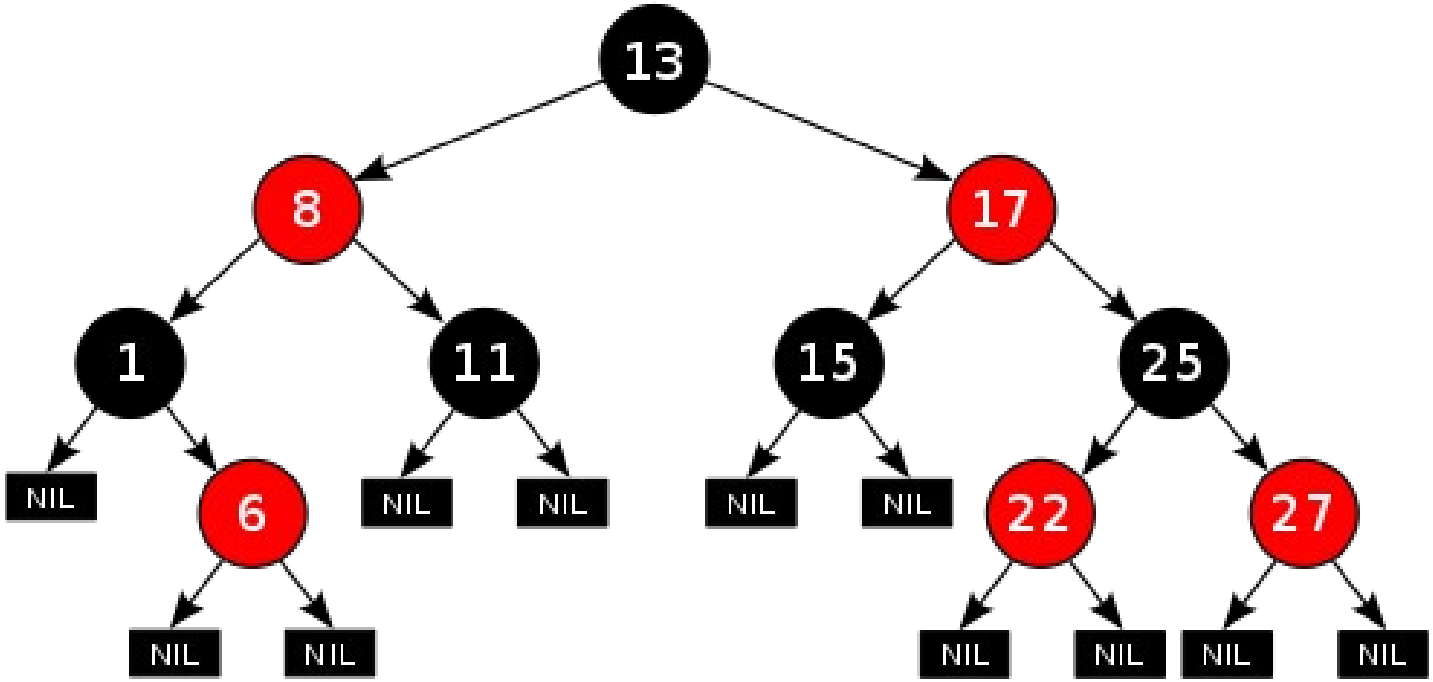
Fa magassága (h)

- Legrosszabb esetben a kereső fa magassága *n*
- Ugyanazokat az elemeket számtalan keresőfában tárolhatjuk
- **<u>Kiegyensúlyozott keresőfa:</u>** törekszünk arra, hogy a keresőfa magassága a lehető legkisebb maradjon, de műveletek továbbra is a magasság függvénye legyen

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Piros-fekete fák

Egy fajta kiegyensúlyozott keresőfa



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Piros-fekete fák

Bármely *n* belső csúcsot tartalmazó piros-fekete fa magassága legfeljebb $2\log_2(n+1)$.

→ <u>megközelítőleg kiegyensúlyozott</u>

1. Minden csúcs színe vagy piros, vagy fekete.
2. A gyökércsúcs színe fekete.
3. Minden levél (NIL) színe fekete.
4. Minden piros csúcsnak mindkét gyereke fekete.
5. Bármely csúcsból bármely levélig vezető úton ugyanannyi fekete csúcs van.

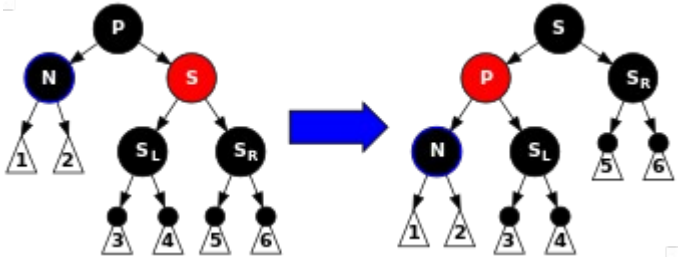
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

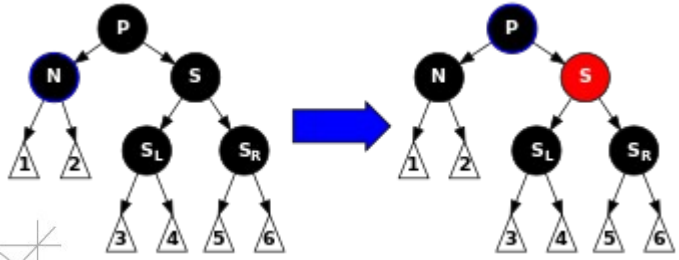
Piros-fekete fák

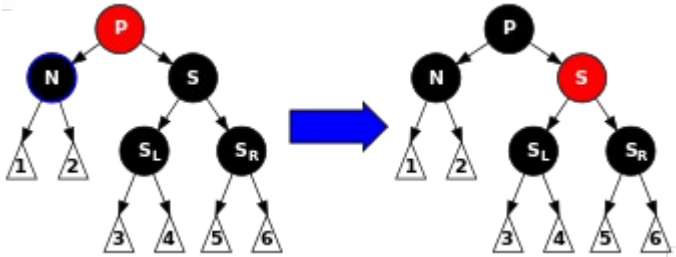
- KERES, MIN/MAX, KÖVETKEZŐ/ ELÖZŐ sima keresőfa műveletek
- Módosító műveleteknél (BESZÚR, TÖRÖL) fent kell tartani az 5 piros-fekete tulajdonságot!
- A tulajdonságok helyreállítása is megvalósítható $O(\log n)$ időben

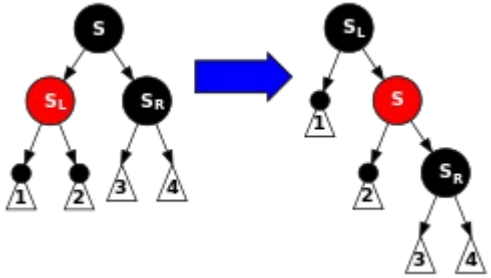
SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

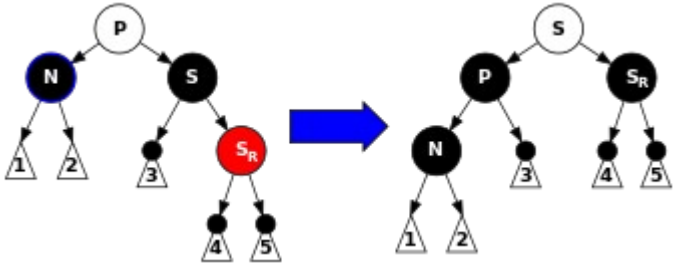
Törlés piros-fekete fából











SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Bináris keresőfa implementációk

2020. október

7. hét – 6. videó
S07E06

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Keresőfa Javában

Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	**clear**() Removes all of the elements from this set (optional operation).
boolean	**contains**(Object o) Returns true if this set contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	**equals**(Object o) Compares the specified object with this set for equality.
int	**hashCode**() Returns the hash code value for this set.
boolean	**isEmpty**() Returns true if this set contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this set.
boolean	**remove**(Object o) Removes the specified element from this set if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this set (its cardinality).
Object[]	**toArray**() Returns an array containing all of the elements in this set.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Java TreeSet

E	**ceiling(E e)** Returns the least element in this set greater than or equal to the given element, or `null` if there is no such element.
void	**clear()** Removes all of the elements from this set.
Object	**clone()** Returns a shallow copy of this `TreeSet` instance.
Comparator<? super E>	**comparator()** Returns the comparator used to order the elements in this set, or `null` if this set uses the **natural ordering** of its elements.
boolean	**contains(Object o)** Returns `true` if this set contains the specified element.
Iterator<E>	**descendingIterator()** Returns an iterator over the elements in this set in descending order.
NavigableSet<E>	**descendingSet()** Returns a reverse order view of the elements contained in this set.
E	**first()** Returns the first (lowest) element currently in this set.
E	**floor(E e)** Returns the greatest element in this set less than or equal to the given element, or `null` if there is no such element.
SortedSet<E>	**headSet(E toElement)** Returns a view of the portion of this set whose elements are strictly less than `toElement`.
NavigableSet<E>	**headSet(E toElement, boolean inclusive)** Returns a view of the portion of this set whose elements are less than (or equal to, if `inclusive` is true) `toElement`.
E	**higher(E e)** Returns the least element in this set strictly greater than the given element, or `null` if there is no such element.
boolean	**isEmpty()** Returns `true` if this set contains no elements.
Iterator<E>	**iterator()** Returns an iterator over the elements in this set in ascending order.
E	**last()** Returns the last (highest) element currently in this set.
E	**lower(E e)** Returns the greatest element in this set strictly less than the given element, or `null` if there is no such element.
E	**pollFirst()** Retrieves and removes the first (lowest) element, or returns `null` if this set is empty.
E	**pollLast()** Retrieves and removes the last (highest) element, or returns `null` if this set is empty.
boolean	**remove(Object o)** Removes the specified element from this set if it is present.
NavigableSet<E>	**subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive)** Returns a view of the portion of this set whose elements range from `fromElement` to `toElement`.
SortedSet<E>	**subSet(E fromElement, E toElement)** Returns a view of the portion of this set whose elements range from `fromElement`, inclusive, to `toElement`, exclusive.
SortedSet<E>	**tailSet(E fromElement)** Returns a view of the portion of this set whose elements are greater than or equal to `fromElement`.
NavigableSet<E>	**tailSet(E fromElement, boolean inclusive)** Returns a view of the portion of this set whose elements are greater than (or equal to, if `inclusive` is true) `fromElement`.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Keresőfák

Implementációk általában piros-fekete fát valósítanak meg:

C++	std::set
C#	TreeSet
Scala	collection.immutable.TreeSet

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Quora

Search for questions, people, and topics

Why does Python not have any data structures that store data in sorted order?

Promoted by Association for Computing Machinery

Write more than code. Write the future. Join ACM today.
Join a global network of your peers in making a positive impact on the computing profession and society.

Learn More At Acm.org

3 Answers

Grant Jenks
Answered Apr 3 2014

Python has these, but they aren't in the standard library. The Python Package Index (PyPI) is the go-to place for data structures like this. In fact, there's over half a dozen great implementations that you can quickly install with pip or easy_install.

The rationale for none of those implementations being in the standard library is that there's many ways to solve the problem. There's no "one way" to maintain an ordered list/set/dict. So if you're smart enough to know you need one then you're qualified to pick one of the half-dozen third-party implementations.

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Halmaz és szótár absztrakt adatszerkezetek

2020. október

8. hét – 1. videó
S08E01

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Halmaz

- Egy elem legfeljebb egyszer szerepelhet benne (ellentétben a sorozattal/listával/tömbbel)
- Alapvető műveletek:
  - KERES(k) (itt most TARTALMAZa-e igen/nem)
  - BESZÚR(k), TÖRÖL(k)
- Egyéb halmazműveletek:
  - METSZET, UNIÓ

Halmaz Javában

```
Set<Object> s = new
```

Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	**clear**() Removes all of the elements from this set (optional operation).
boolean	**contains**(Object o) Returns true if this set contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns true if this set contains all of the elements of the specified collection.
boolean	**equals**(Object o) Compares the specified object with this set for equality.
int	**hashCode**() Returns the hash code value for this set.
boolean	**isEmpty**() Returns true if this set contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this set.
boolean	**remove**(Object o) Removes the specified element from this set if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this set (its cardinality).
Object[]	**toArray**() Returns an array containing all of the elements in this set.
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

unió

metszet

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Java TreeSet

E	ceiling(E e) Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
void	clear() Removes all of the elements from this set.
Object	clone() Returns a shallow copy of this TreeSet instance.
Comparator<? super E>	comparator() Returns the comparator used to order the elements in this set, or null if this set uses the **natural ordering** of its elements.
boolean	contains(Object o) Returns true if this set contains the specified element.
Iterator<E>	descendingIterator() Returns an iterator over the elements in this set in descending order.
NavigableSet<E>	descendingSet() Returns a reverse order view of the elements contained in this set.
E	first() Returns the first (lowest) element currently in this set.
E	floor(E e) Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
SortedSet<E>	headSet(E toElement) Returns a view of the portion of this set whose elements are strictly less than toElement.
NavigableSet<E>	headSet(E toElement, boolean inclusive) Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
E	higher(E e) Returns the least element in this set strictly greater than the given element, or null if there is no such element.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set in ascending order.
E	last() Returns the last (highest) element currently in this set.
E	lower(E e) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst() Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast() Retrieves and removes the last (highest) element, or returns null if this set is empty.
boolean	remove(Object o) Removes the specified element from this set if it is present.
NavigableSet<E>	subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) Returns a view of the portion of this set whose elements range from fromElement to toElement.
SortedSet<E>	subSet(E fromElement, E toElement) Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
SortedSet<E>	tailSet(E fromElement) Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
NavigableSet<E>	tailSet(E fromElement, boolean inclusive) Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Szótárban keresési feladat

Név	Felhasználónév	E-mail
Aba Abdul	kismardar	aabdul92@freemail.hu
…		
Gibson Mel	madmax	mel@gibson.com
Gipsz Jakab	gyors_kotes	jani_cim_spameknek@gmail.com
…		
Zsuzsi Zsuzsa	enishateis97	abc@xyz.com

Mi „Gipsz Jakab” e-mailcíme?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Szótárban KERES() feladat

- **Szótár:** egy halmaza elemeihez (kulcs) egy érték adat tartozik
- **Bemenet:** egy szótár és egy keresett kulcs
- **Kimenet:**
  - Ha a *kulcs* szerepel a szótárban akkor a hozzá tartozó *érték*
  - Ha nem szerepel *kulcs* a szótárban akkor NIL

Szótár (dictionary) = = asszociatív tömb = map

- Minden kulcs legfeljebb egyszer szerepel (kulcsok halmaza), de egy érték tetszőleges számban előfordulhat
- “Asszociatív tömb”: egészek helyett bármilyen típussal indexelhetünk
- “map”: egy kulcs → érték leképezés

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Példák szótár használatára

- DNS szerver
- Gyakoriság leszámolás

  `freq["alma"]++;` (C++)
- Helyesírás ellenörző+javító

Bármilyen nagy adathalmaz, ahol a gyors keresés a kritikus!

python:

```
>>> tel = {'jack': 4098, 'sape': 4139}
>>> tel['guido'] = 4127
>>> tel
{'sape': 4139, 'guido': 4127, 'jack': 4098}
>>> tel['jack']
4098
>>> del tel['sape']
>>> tel['irv'] = 4127
>>> tel
{'guido': 4127, 'irv': 4127, 'jack': 4098}
>>> tel.keys()
['guido', 'irv', 'jack']
>>> 'guido' in tel
True
```

Java:

```
Map<String, Integer> tel = new HashMap<>();
tel.put("jack",4098);
tel.put("sape",4139);
tel.put("guido",4127);
System.out.println(tel);
> {guido=4127, sape=4139, jack=4098}
System.out.println(tel.get("jack"));
> 4098
tel.remove("sape");
tel.put("irv", 4127);
System.out.println(tel);
> {guido=4127, irv=4127, jack=4098}
System.out.println(tel.keySet());
> [guido, irv, jack]
System.out.println(tel.containsKey("guido"));
> true
```

Absztrakt Adatszerkezet választó

- Ha egy elem többször is előfordulhat, sorrendiség fontos:
  - lista, verem, sor, prioritási sor
- Halmaz, csak az számít, hogy tartalmaz-e egy elemet vagy sem:
  - halmaz (Set)
- Ha kulcs-érték párokat tárolunk:
  - szótár (dictionary, Map)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Hasító táblák

2020. október

8. hét – 2. videó
S08E02

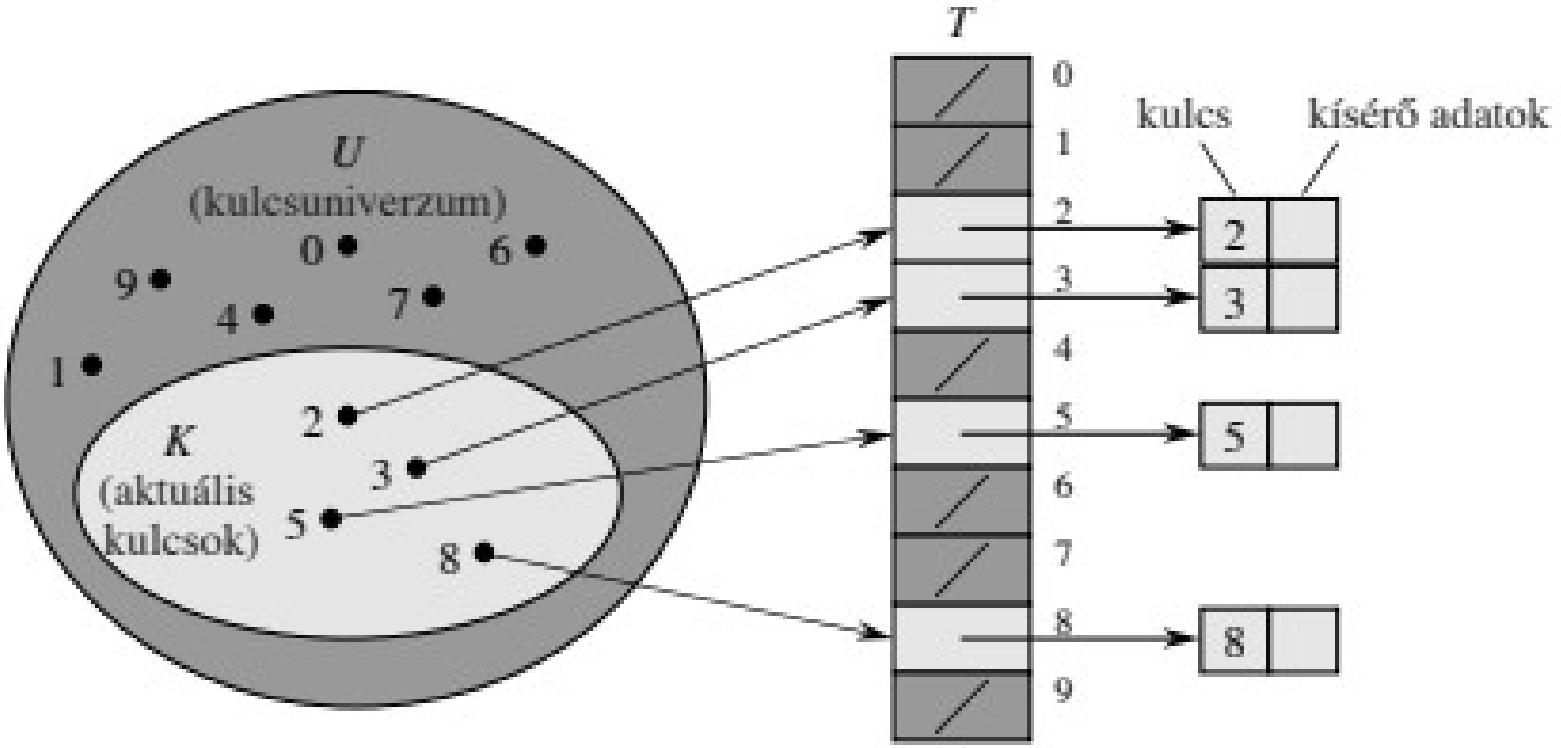
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hasító táblák

- Halmazok és szótárak hatékony megvalósítása
- KERES(), BESZUR(), TOROL() legyen hatékony!
- <u>átlagos eset</u>ben $O(1)$

Megj: keresőfa legrosszabb esetben $O(\log n)$ de(!) egyéb műveleteket biztosít: (KÖVETKEZŐ/ELŐZŐ, MIN/MAX)

Közvetlen címzésű táblázat



11.1. ábra. Egy dinamikus halmaz megvalósítása a T közvetlen címzésű táblázattal. Az $U = \{0, 1, \ldots, 9\}$ univerzum minden kulcsának megfelel egy index a táblázatban. Az aktuális kulcsok $K = \{2, 3, 5, 8\}$ halmaza meghatározza a táblázatban azokat a réseket, melyek az elemekre mutató pointereket tartalmazzák. A többi, sötéten árnyékolt rés NIL-t tartalmaz.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Közvetlen címzésű táblázat

Közvetlen címzésű táblázat:

KERES $O(1)$, BESZUR/TOROL $O(1)$

tár: $O(|\text{univerzum}|)$

Rendezett tömb (közvetlen elérésű memóriával):

KERES $O(\log n)$, BESZUR/TOROL $O(n)$
tár: $O(n)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hasító (hash) táblázatok

Általában a szótárban tárolt kulcsok halmaza (n) sokkal kisebb a lehetséges kulcsok univerzumánál…

h ***hasító függvény*** a kulcsok U univerzumát képezi le a T $[0 \ldots m-1]$ *hasító táblázat* ***rés***eire:

$$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

(a k kulcsú elem a $h(k)$ résre *képződik le*
$h(k)$ a k kulcs *hasított értéke)*

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

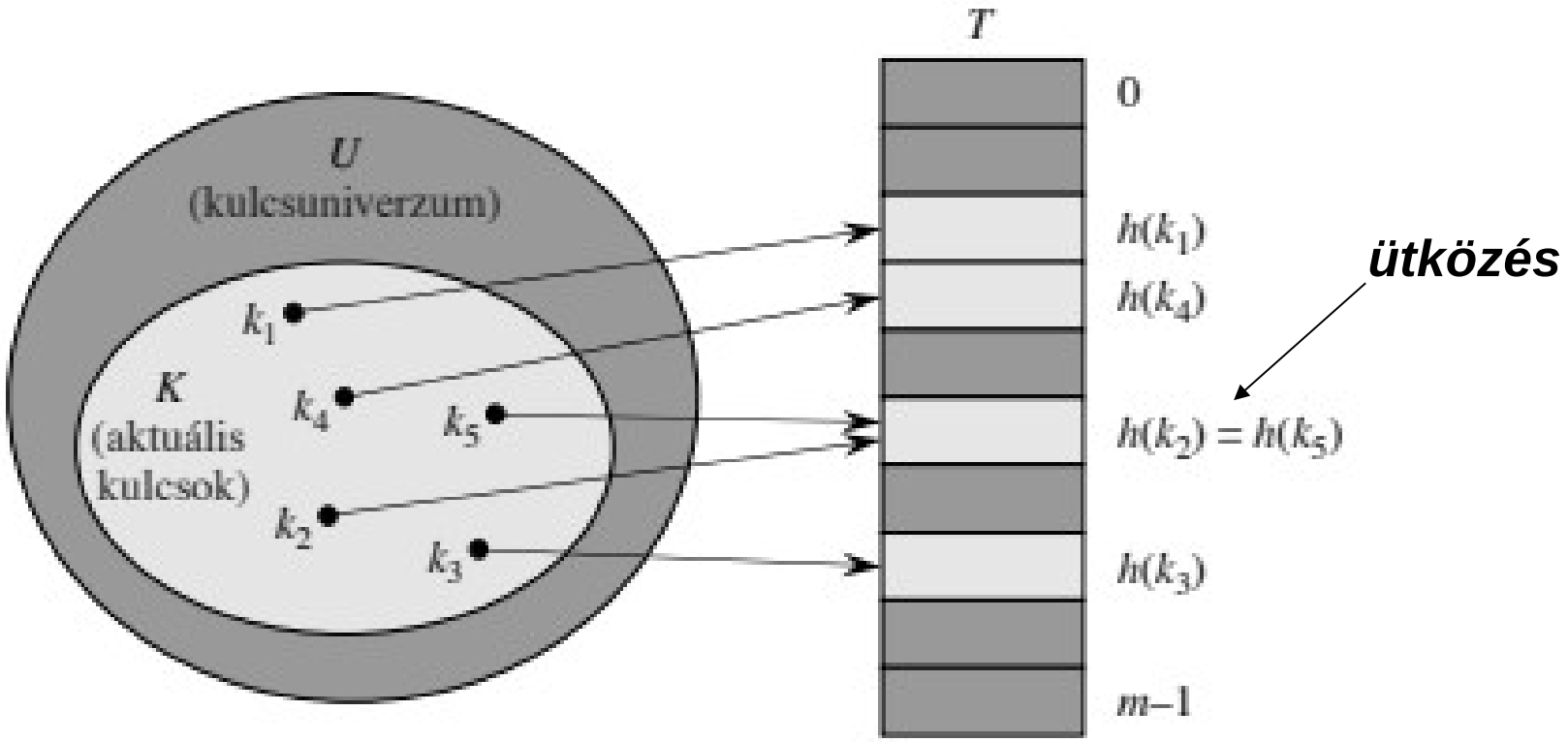
Példa hasító függvény

h(*k*) = *k* mod *m*

m a hasító táblázat mérete, azaz réseinek száma

pl. ha *m* = 12 akkor *h*(100) = 100 mod 12 = 4

Hasító táblázatok



11.2. ábra. A h hasító függvény felhasználása a kulcsoknak egy hasító táblázat réseire való leképezésére. A k_2 és k_3 kulcsok ugyanarra a résre képződnek le, tehát ütköznek.

Mivel |U|>m ezért az ütközés elkerülhetetlen!
Cél: ütközések számának minimalizálása

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

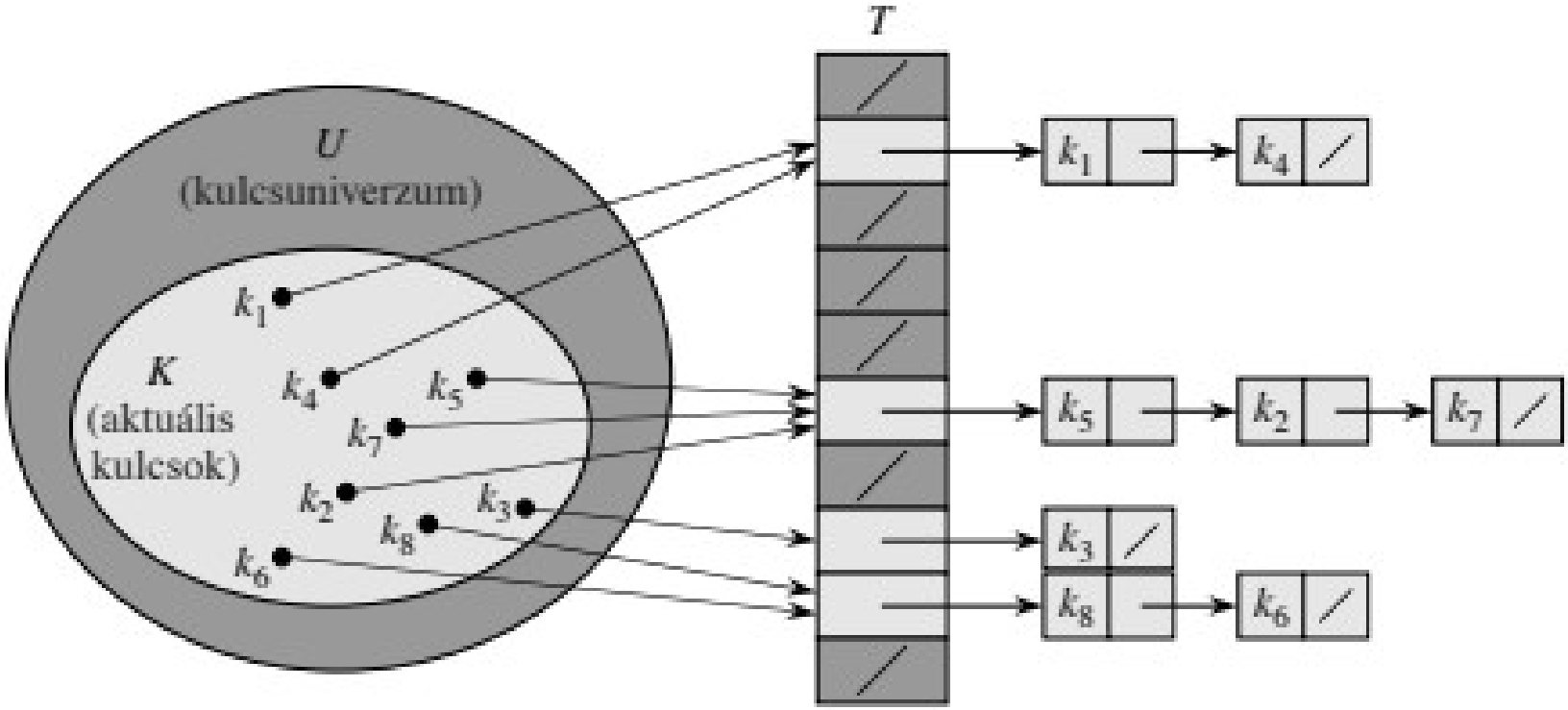
Algoritmusok és Adatszerkezetek I.

Ütközésfeloldás láncolással

2020. október

8. hét – 3. videó
S08E03

Ütközésfeloldás láncolással



11.3. ábra. Ütközésfeloldás láncolással. A hasító táblázat egy $T[j]$ rése azoknak a kulcsoknak a láncolt listáját tartalmazza, melyek hasított értéke pontosan j. Például $h(k_1) = h(k_4)$ és $h(k_5) = h(k_2) = h(k_7)$.

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Ütközésfeloldás láncolással

LÁNCOLT-HASÍTÓ-BESZÚRÁS(T, x)

1 beszúrás a $T[h(kulcs[x])]$ lista elejére

$O(1)$

LÁNCOLT-HASÍTÓ-KERESÉS(T, k)

1 a k kulcsú elem keresése a $T[h(k)]$ listában

???

legrosszabb esetben egy résben az n elem → $\Theta(n)$

LÁNCOLT-HASÍTÓ-TÖRLÉS

1 x törlése a $T[h(kulcs[x])]$ listából.

https://visualgo.net/en/hashtable

LÁNCOLT-HASÍTÓ-KERESÉS futásideje átlagos esetben

- m méretű T tömb, n elem
- ***kitöltési tényező*** $\alpha = n/m$

  (=láncok átlagos hossza)
- bármely j-re a T[j] lista hosszának várható értéke $E[n_j] = n/m = \alpha$
- $h(k)$ hasított érték $O(1)$ idő alatt számítható ki

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

LÁNCOLT-HASÍTÓ-KERESÉS futásideje átlagos esetben

egyszerű egyenletes hasítási feltétel: minden elem egyforma valószínűséggel képződik le bármely résre, függetlenül attól, hogy a többiek hova kerültek

11.1. tétel. *Ha egy hasító táblázatban az ütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikertelen keresés átlagos ideje* $\Theta(1+\alpha)$.

11.2. tétel. *Ha egy hasító táblázatban a kulcsütközések feloldására láncolást használunk és a hasítás egyszerű egyenletes, akkor a sikeres keresés átlagos ideje* $\Theta(1+\alpha)$.

ha $n=O(m) \rightarrow \alpha=n/m=O(m)/m=O(1)$

azaz a KERES, BESZUR, TOROL $O(1)$

SZEGEDI TUDOMÁNYEGYETEM

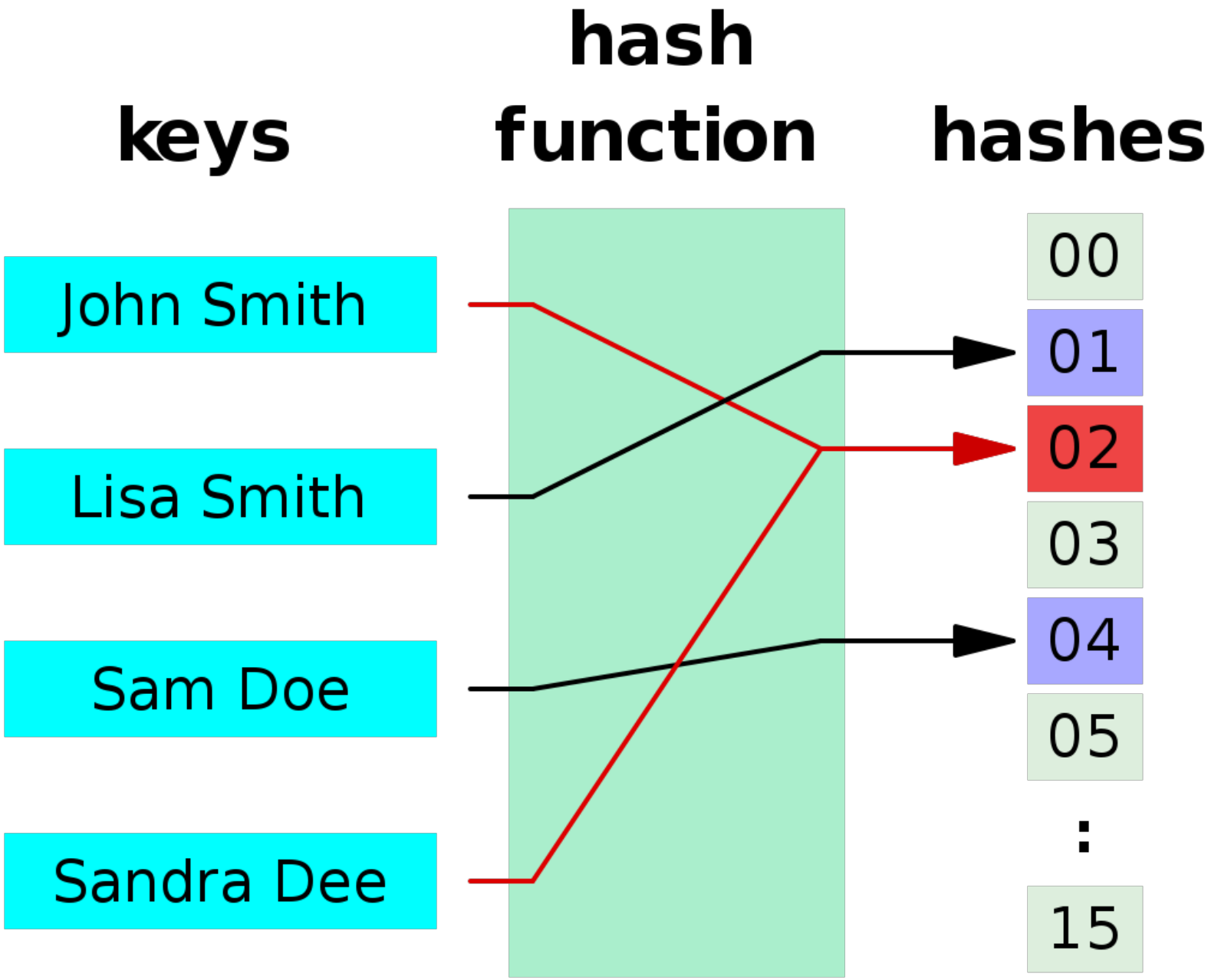
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Hasító függvények

2020. október

8. hét – 4. videó
S08E04


hash
keys
function
hashes
John Smith
Lisa Smith
Sam Doe
Sandra Dee
00
01
02
03
04
05
:
15

Hogyan válasszunk hasítófüggvényt?

- Egy jó hasító függvény (közelítőleg) kielégíti az „egyszerű egyenletességi feltételt”, azaz minden kulcs egyforma valószínűséggel képződikle az *m* rés bármelyikére
- saját hasítófüggvény kellhet új típusú adatokra
- várhatóan független legyen az adatokban esetleg meglévő mintáktól

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hogyan válasszunk hasítófüggvényt?

- heurisztika: “közel” lévő kulcsokhoz tartozó értékek távol legyenek egymástól
- Pl. vonalas telefonszámoknál

  +36-62-541234

  +36-62-549876

  Hasítás első számjegyek alpján nem egyszerű egyenletes, utolsó számjegyek alapján jobb!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Kulcsok természetes számokkal

- Ha nem természetes számok a kulcsok akkor célszerű először átalakítani természetes számra
- Pl. String esetén ascii karakterek 128-as számrendszerbeli számoknak tekinthetjük
- Természetes számokra számtalan hasító függvény lett kidolgozva…

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Osztásos módszer

$$\mathbf{h}(k) = k \bmod m$$

Jó általában ha *m* kettő hatványhoz nem túl közeli prímek

Pl. n~2000 és α=3 a cél akkor érdemes m=701-re választani (prím)

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Szorzásos módszer

- 0<A<1
- **h(k)=m(kA mod 1)**
- *m* nem kritikus, általában kettő hatvány → gyorsan számolható
- *A* választása még befolyásolja az egyenletességet, de nem annyira kritikus

$$A \approx \frac{\sqrt{5}-1}{2} = 0,6180339887\ldots$$

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

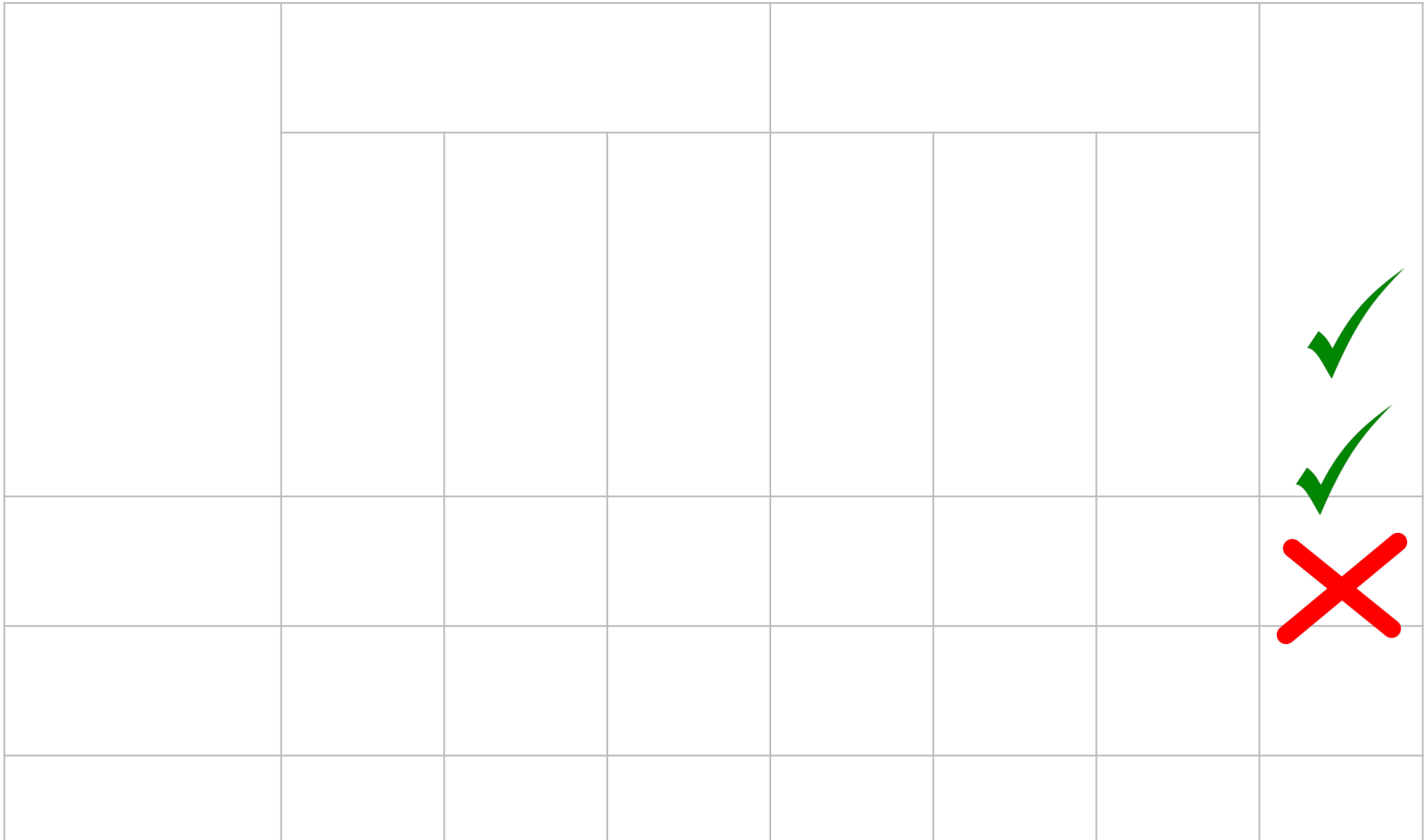
Algoritmusok és Adatszerkezetek I.

Bináris keresőfa vs Hasító tábla

2020. október

8. hét – 6. videó
S08E06

KeresőFa vs Hasító táblák



KeresőFa vs Hasító tábla Javában

```
Set<MyClass> s = new TreeSet<>();
    class MyClass implements Comparable<MyClass>
    int MyClass.compareTo(MyClass c)
Set<MyClass> s = new HashSet<>();

    boolean MyClass.equals(Object o)
    int MyClass.hashCode()
```

HashSet általában gyorsabb, de ha rendezett műveletekre is szükségünk van, akkor TreeSet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

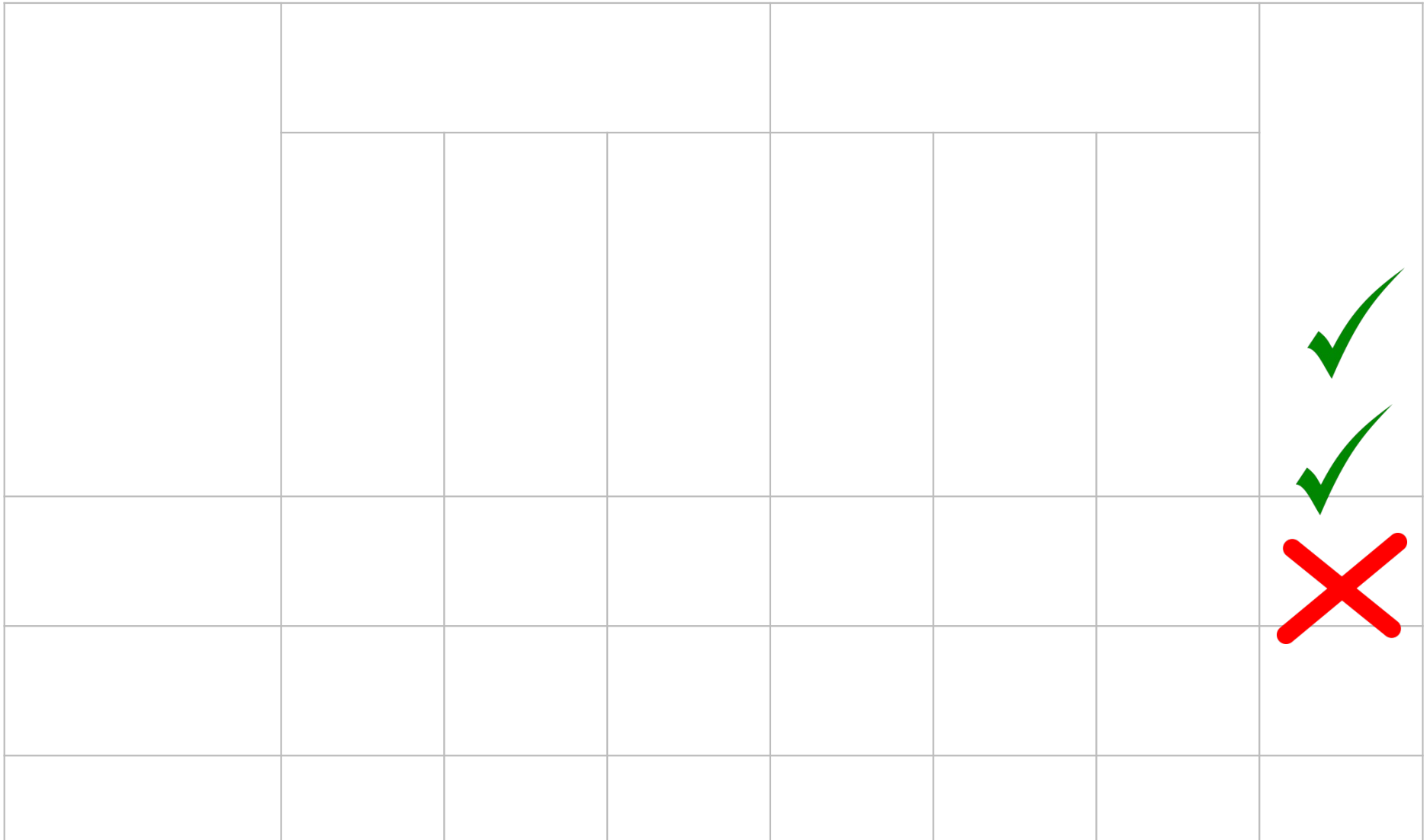
Algoritmusok és Adatszerkezetek I.

Bináris keresőfa vs Hasító tábla

2020. október

8. hét – 6. videó
S08E06

KeresőFa vs Hasító táblák



KeresőFa vs Hasító tábla Javában

```
Set<MyClass> s = new TreeSet<>();
    class MyClass implements Comparable<MyClass>
    int MyClass.compareTo(MyClass c)
Set<MyClass> s = new HashSet<>();

    boolean MyClass.equals(Object o)
    int MyClass.hashCode()
```

HashSet általában gyorsabb, de ha rendezett műveletekre is szükségünk van, akkor TreeSet!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Hasítás a kriptográfiában

2020. október

8. hét – 7. videó
S08E07

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Hasítás a kriptográfiában

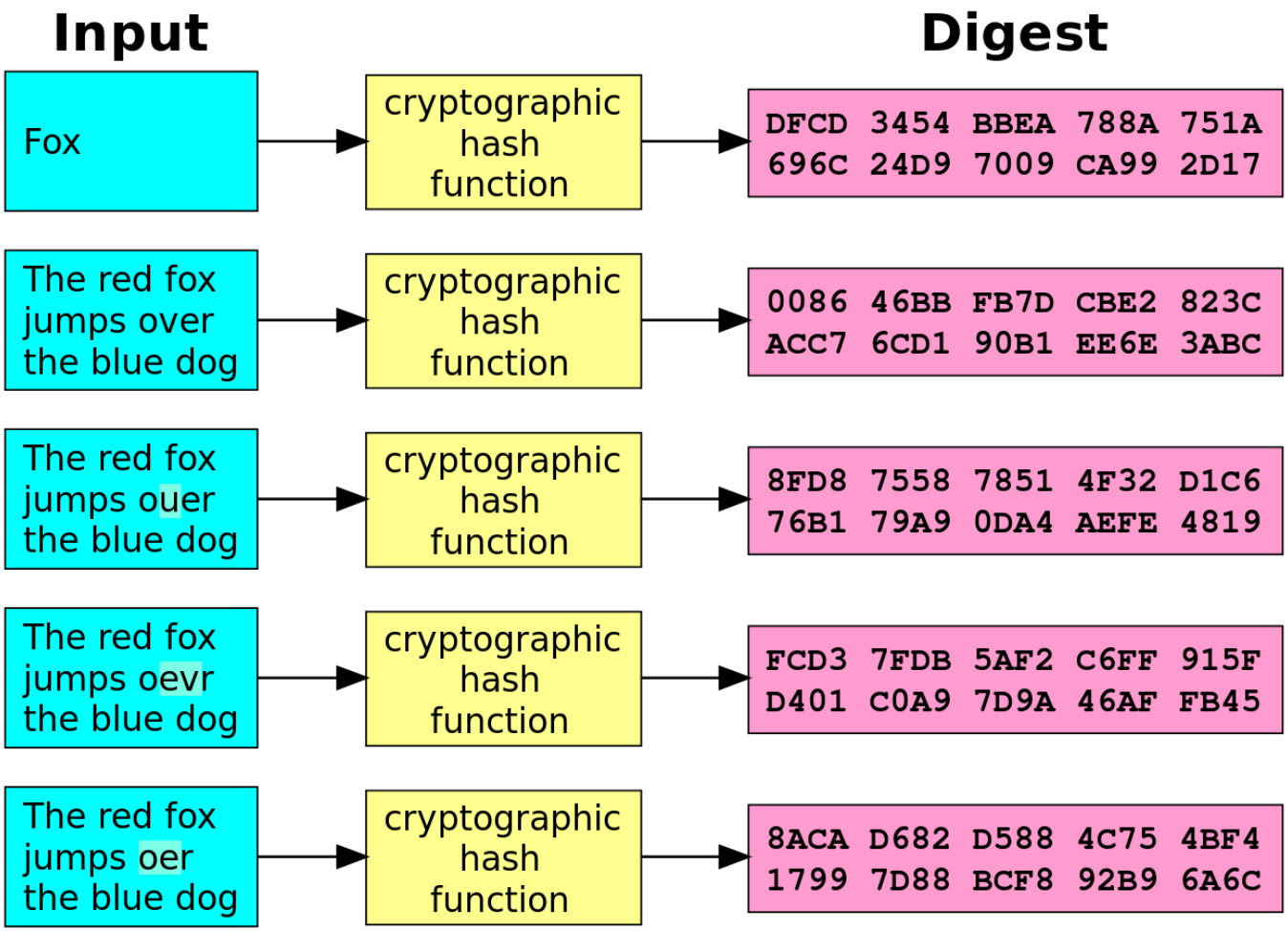
Egy rendszer felhasználónév-jelszó párjait hogyan tároljuk?

Ha a jelszavakat magukat eltároljuk akkor az komoly biztonsági rés!!!

Csak a jelszavak hasított értékét tároljuk. *Egyirányú hasító* *függvény*t használjunk!

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Hasítás a kriptográfiában



Forrás: https://en.wikipedia.org/wiki/Cryptographic_hash_function#/media/File:Cryptographic_Hash_Function.svg

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

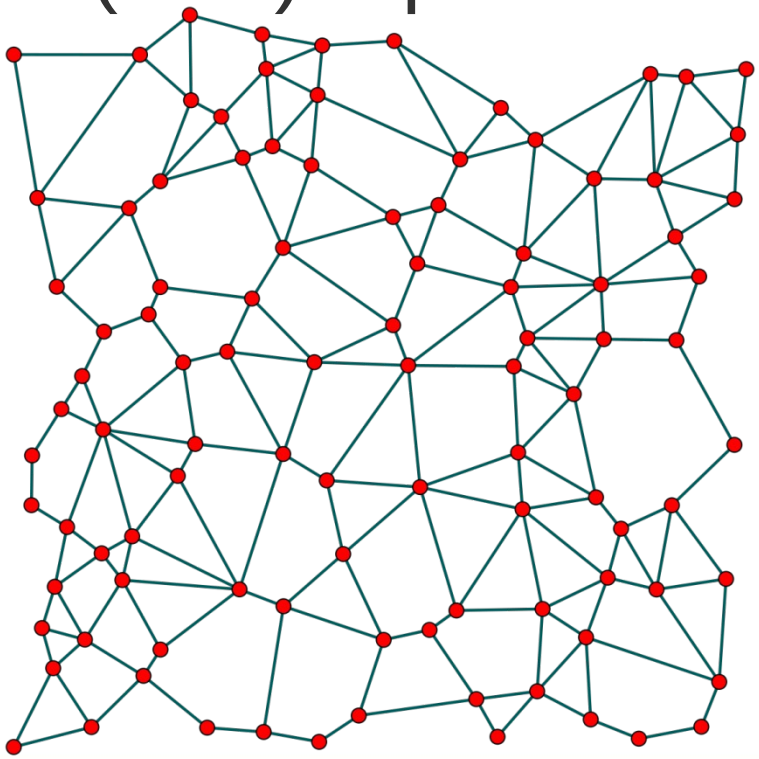
Gráfok számítógépes reprezentációja

2020. november

9. hét – 1. videó
S09E01

Gráf

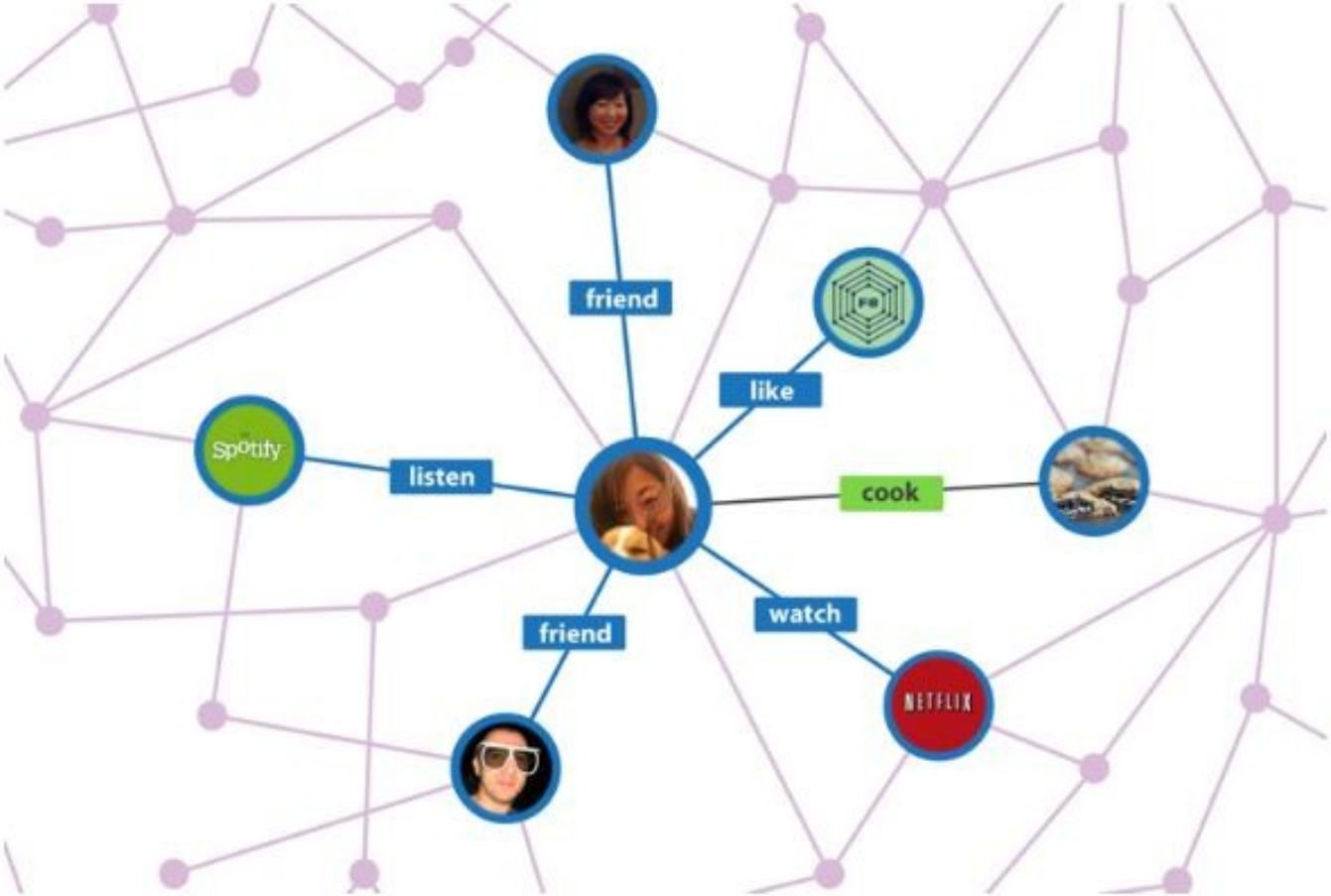
Dolgok (csúcsok) és köztük fennálló kapcsolatok (élek) reprezentációja



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

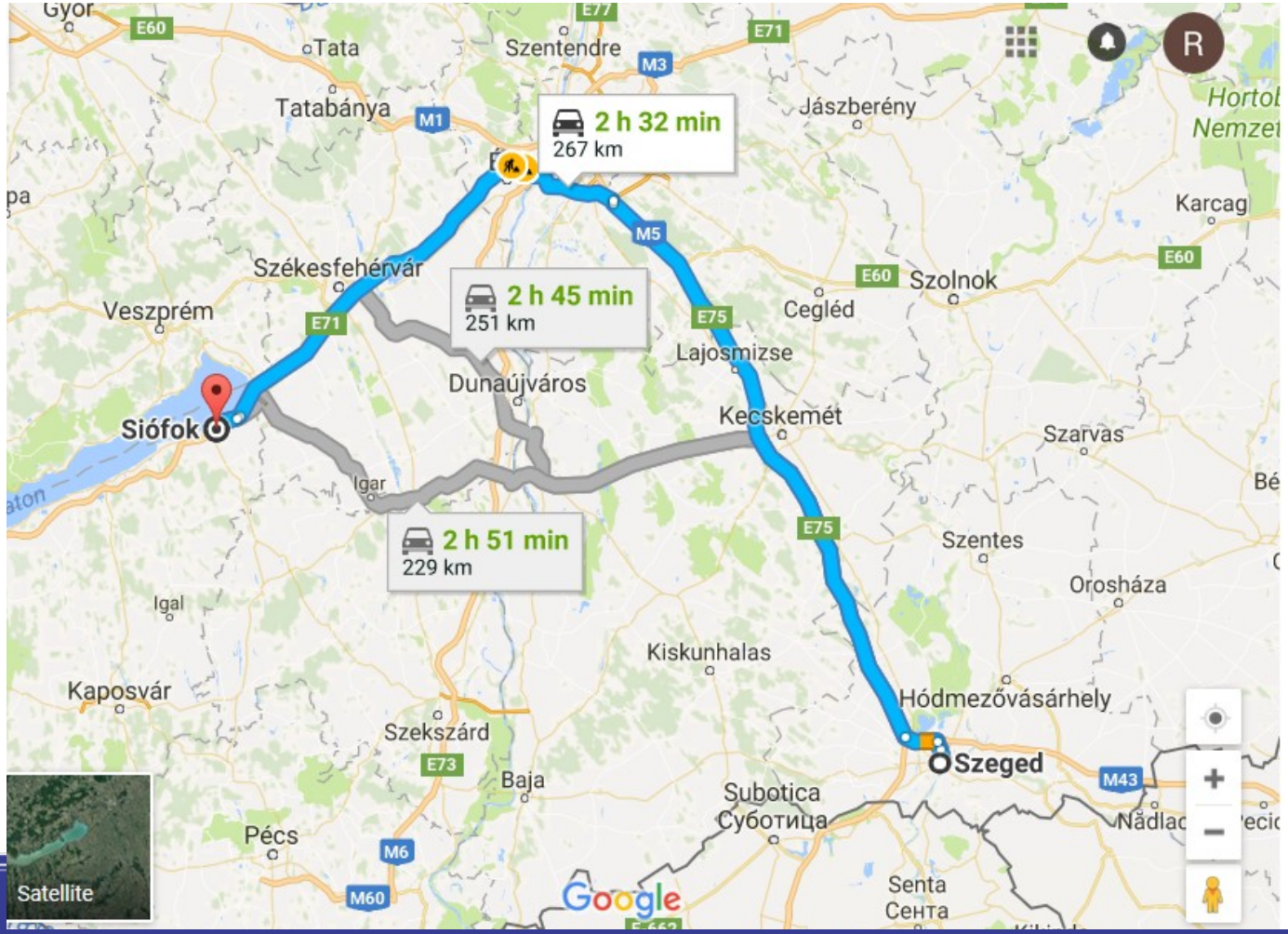
Valós életbeli gráfok (hálózatok)



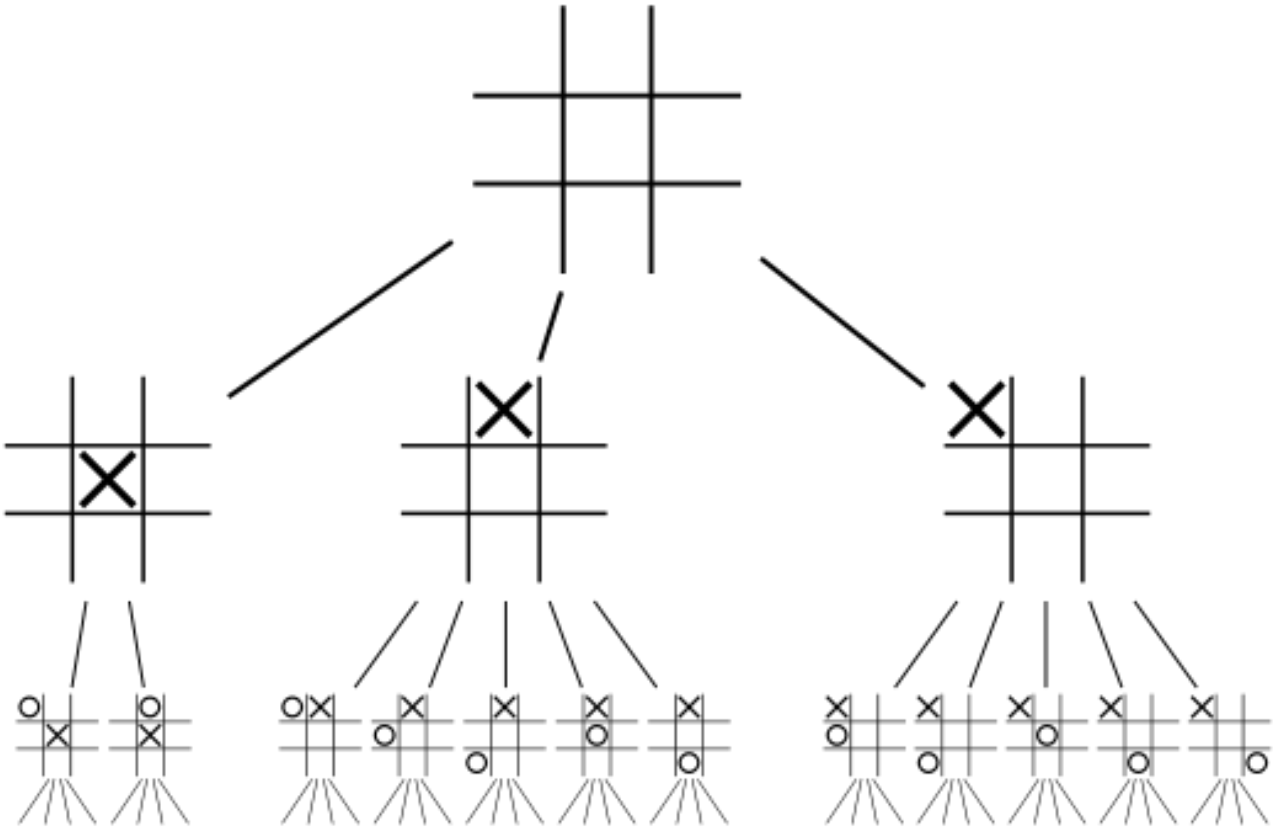
Valós életbeli gráfok (hálózatok)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Valós életbeli gráfok (hálózatok)



Gráfok absztrakt reprezentációra



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Gráfok

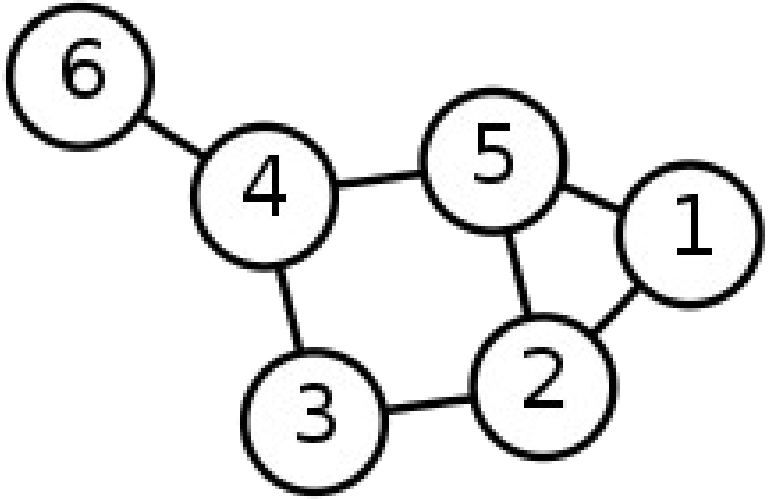
G=(V,E)

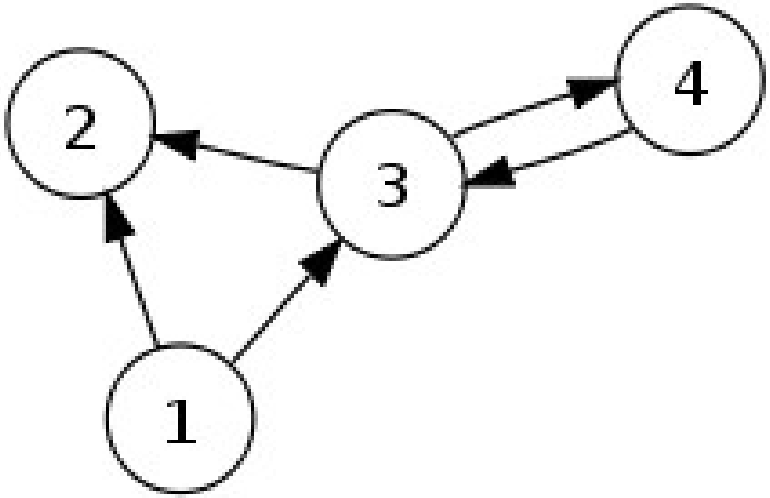
G a gráf

V: csúcsok halmaza (általában azonosítóval ellátva, gyakorlatban egyéb tulajdonságokkal is)

E: élek halmaza. Egy él két csúcs közti kapcsolatot ír le. A kapcsolat lehet irányított vagy irányítatlan ill. lehet súlyozott (gyakorlatban egyéb tulajdonságok is)

Irányítatlan és irányított gráfok

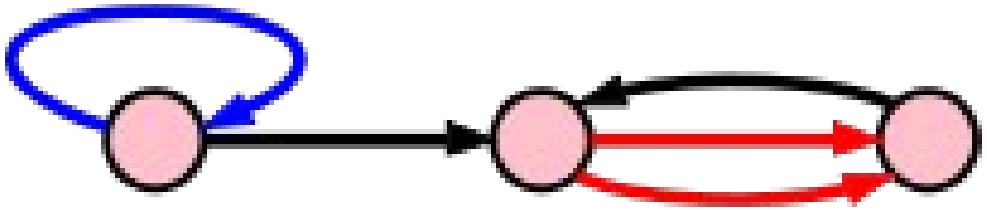




UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű gráfok

Egy gráf **egyszerű** ha nincsenek benne sem hurokélek sem többszörös élek.



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

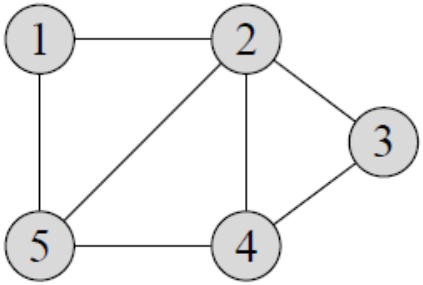
Gráfok számítógépes reprezentációja

Gráfok számítógépes reprezentációja

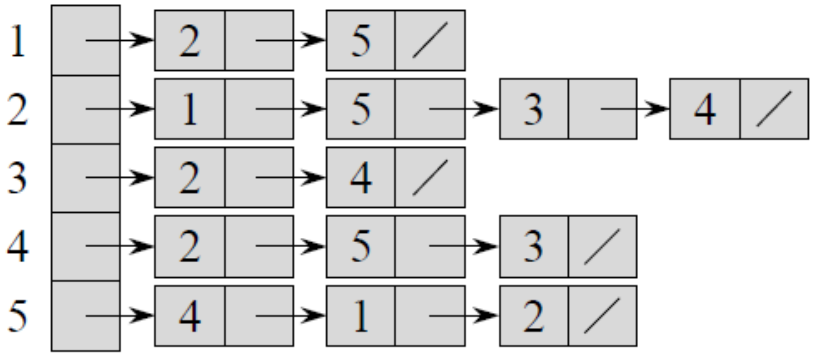
1. Csúcsok halmaza és élek halmaza
   - matematikai definíció ez, de nagyon előnytelen a gyakorlatban, nem használjuk
2. Szomszédsági listás ábrázolás
3. Szomszédsági mátrix ábrázolás

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Gráfok számítógépes reprezentációja



(a)



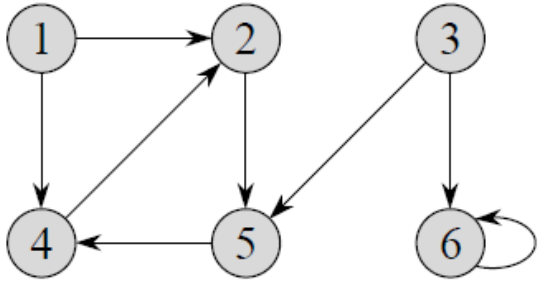
(b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

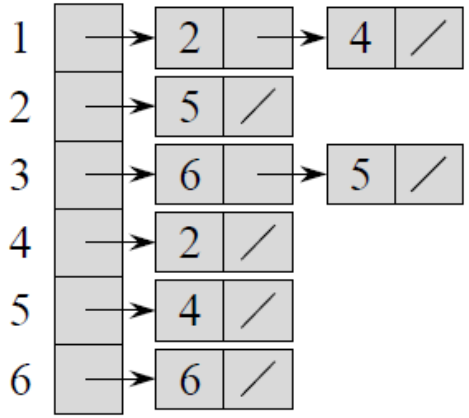
(c)

22.1. ábra. Irányítatlan gráf kétféle ábrázolása. **(a)** Egy *G* irányítatlan gráf 5 csúccsal és 7 éllel. **(b)** *G* ábrázolása szomszédsági listákkal. **(c)** *G* ábrázolása csúcsmátrixszal.

Gráfok számítógépes reprezentációja



(a)



(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

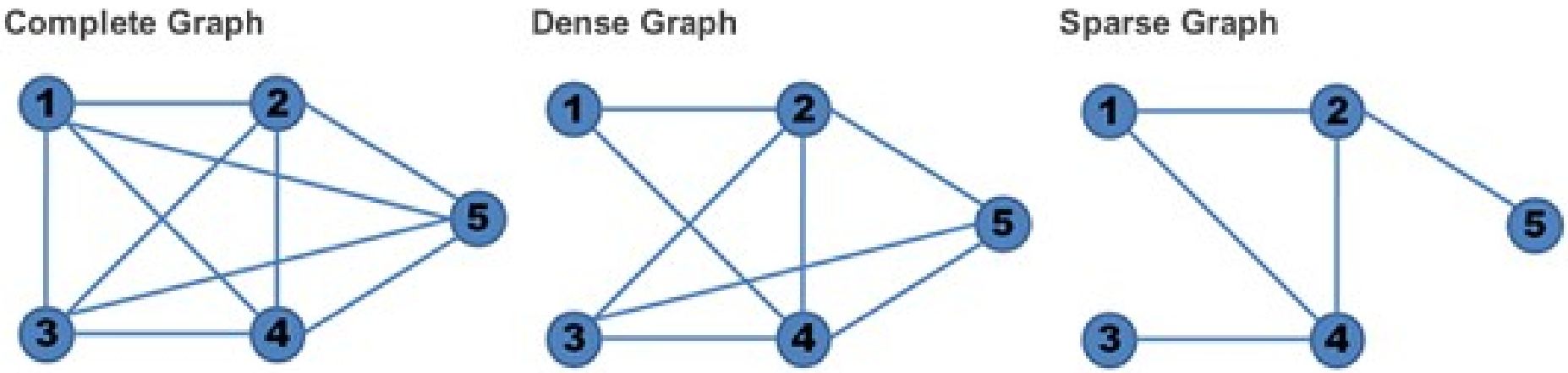
22.2. ábra. Irányított gráf kétféle ábrázolása. **(a)** Egy G irányított gráf 6 csúccsal és 8 éllel. **(b)** G ábrázolása szomszédsági listákkal. **(c)** G ábrázolása csúcsmátrixszal.

Gráfok sűrűsége

Teljes gráf ha $|E|=|V|(|V|-1)/2$

Ritka gráf ha $|E|\sim|V|$

Sűrű gráf ha $|E|\sim|V|^2$



Adatszerkezetet a feladathoz!

Ritka, hogy nem szomszédsági listát használunk.

Gráfalgoritmusok hatékonysága

- Futásidő és tárigény szempontjából mind |V| és |E| is számít(hat)
- Különböző karakterisztikájú gráfokon (pl. ritka vs sűrű) más és más algoritmusok lehetnek előnyösek

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

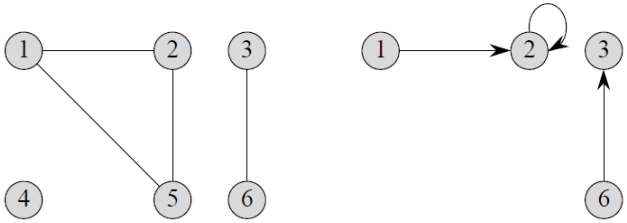
Szélességi keresés

2020. november

9. hét – 2. videó
S09E02

Szélességi keresés (SzK)
Breadth First Search (BFS)

Feladat: Járjuk be az összes csúcsot ami egy *s* kezdő csúcsból elérhető.



Mindeközben kiszámoljuk az elérhető csúcsok távolságát *s*-től

- távolság: legkevesebb élt tartalmazó út élszáma

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Szélességi keresés (SzK)
Breadth First Search (BFS)

Bemenet: Irányítatlan vagy irányított G gráf és annak egy *s* csúcsa

Kimenet: Egy szótár, ami tartalmazza az *s*-ből elérhető csúcsokat és azok távolságát

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SzK alkalmazások

- **social network:** k távolságban lévő ismerőseim
- **Navigáció:** közelben lévő helyek listázása
- **Broadcasting:** üzenet elküldése (pl. peer-to-peer hálózatokban)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SzK(G, s)

```
for minden u ∈ V[G] − {s} csúcsra
    do szín[u] ← FEHÉR
       d[u] ← ∞
       π[u] ← NIL
szín[s] ← SZÜRKE
d[s] ← 0
π[s] ← NIL
Q ← ∅
SORBA(Q, s)
while Q ≠ ∅
    do u ← SORBÓL(Q)
       for minden v ∈ Adj[u]-re
           do if szín[v] = FEHÉR
                 then szín[v] ← SZÜRKE
                      d[v] ← d[u] + 1
                      π[v] ← u
                      SORBA(Q, v)
       szín[u] ← FEKETE
```

felfedezetlen

futásidő: Θ (V+E)

elért

bejárt

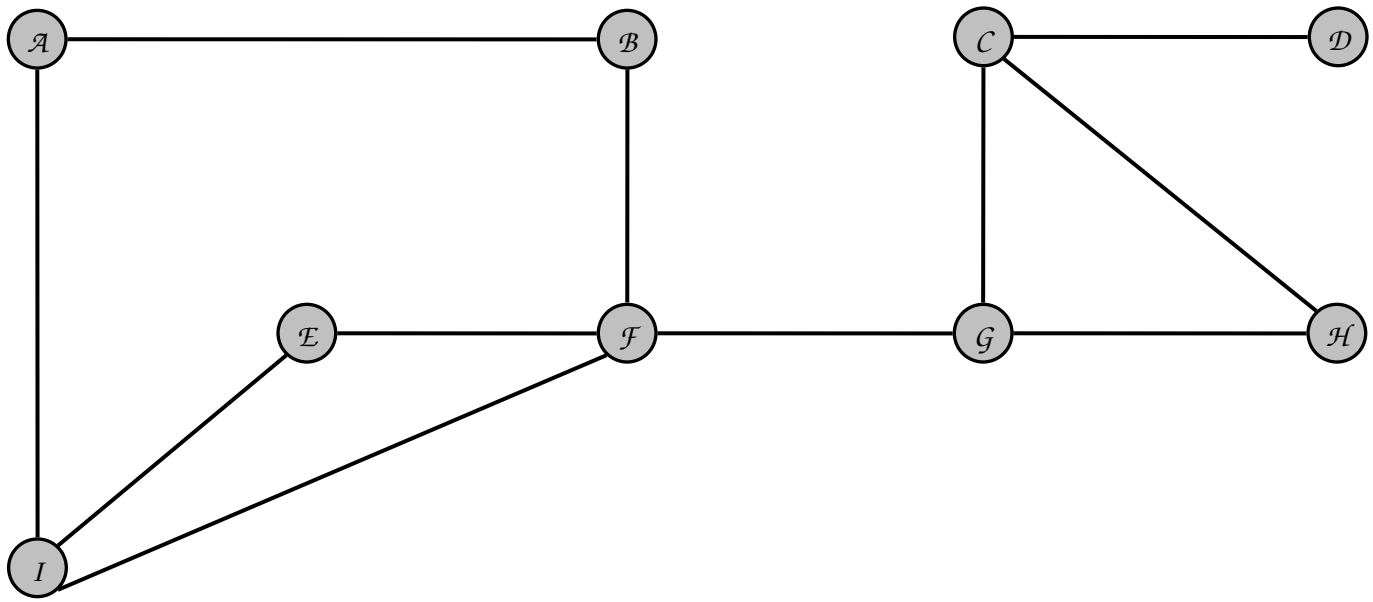
SzK helyessége

δ(s,v) *v* legrövidebb út távolsága *s*-től

22.5. tétel (a szélességi keresés helyessége). *Legyen $G = (V, E)$ irányított vagy irányítatlan gráf, és tegyük fel, hogy* SzK *algoritmust a G gráfra alkalmazzuk egy adott $s \in V$ kezdő csúcsból kiindulva. Ekkor* SzK *működése során minden s-ből elérhető $v \in V$ csúcsot elér, és a befejezéskor $d[v] = \delta(s, v)$ minden $v \in V$ csúcsra. Továbbá, bármely s-ből elérhető $v \neq s$ csúcsra, az s-ből v-be vezető legrövidebb utak egyikét megkapjuk, ha az s-ből $\pi[v]$-be vezető legrövidebb utat kiegészítjük a $(\pi[v], v)$ éllel.*

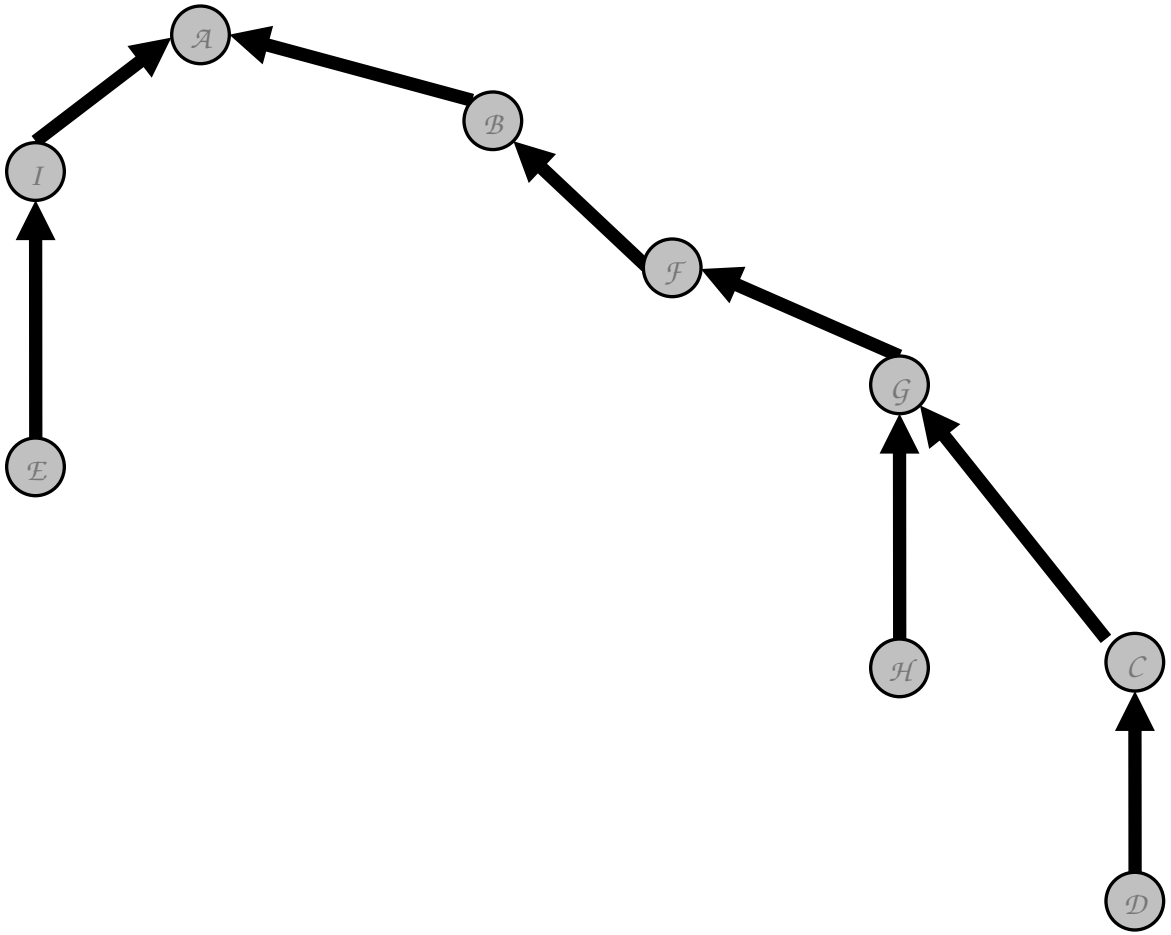
Breadth First Search

-



FIFO Queue

Szélességi fa



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Szélességi fa

Egy G gráf szélességi fája függ a kimenő élek bejárási sorrendjétől, de távolságok egyértelműek

s-ből *v*-be vezető egy legrövidebb út:

```
ÚT-KIÍR(G, s, v)

if v = s
   then print s
   else  if π[v] = NIL
            then print „Nincs út” s „és” v „között”
            else  ÚT-KIÍR(G, s, π[v])
                  print v
```

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Mélységi keresés

2020. november

9. hét – 3. videó
S09E03

Mélységi keresés (MK)
Depth First Search (DFS)

```
MK(G)

for minden u ∈ V[G] csúcsra
    do szín[u] ← FEHÉR
       π[u] ← NIL
idő ← 0
for minden u ∈ V[G] csúcsra
    do if szín[u] = FEHÉR
          then MK-BEJÁR(u)
```

futásidő: Θ (V+E)

elérési idő

```
MK-BEJÁR(u)

szín[u] ← SZÜRKE                    ▷ Most érjük el a fehér u csúcsot.
idő ← idő +1
d[u] ← idő
for minden v ∈ Adj[u] csúcsra       ▷ (u, v) él vizsgálata.
    do if szín[v] = FEHÉR
          then π[v] ← u
               MK-BEJÁR(v)
szín[u] ← FEKETE                    ▷ u fekete, mert elhagytuk.
f[u] ← idő ← idő +1
```

elhagyási idő

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

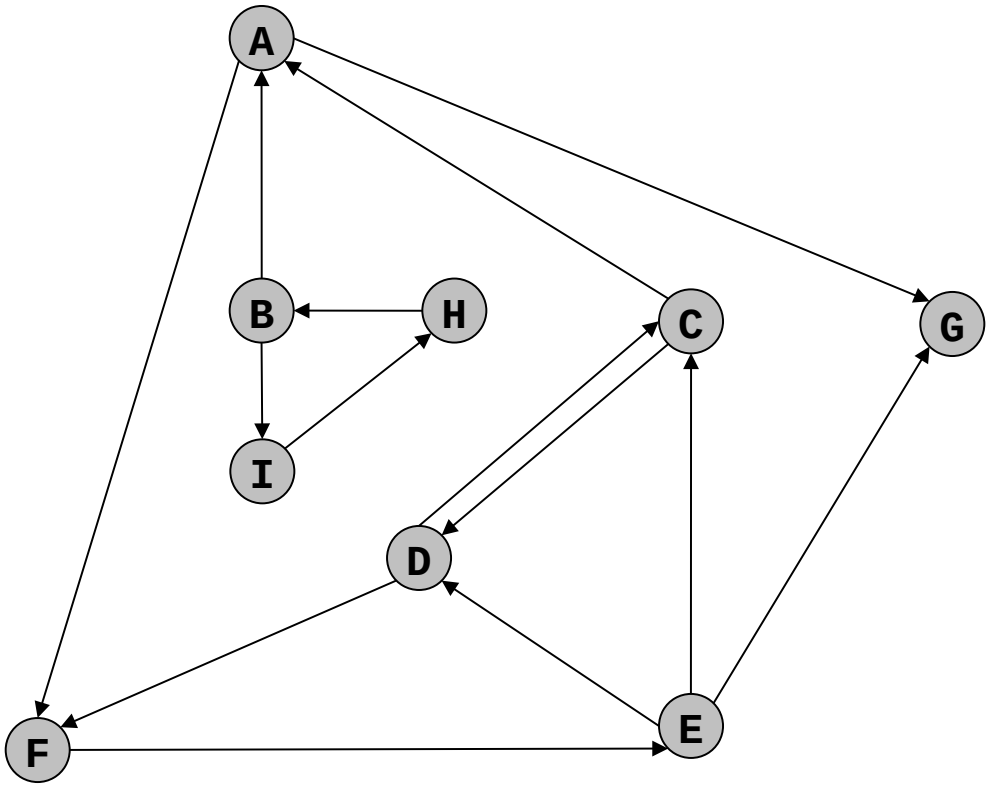
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

MK alkalmazásai

Amikor egy megoldást megtalálni elégséges, nincs szükség mindre/ optimálisra, pl. (ki)útkeresés

- Topológikus rendezés
- Erősen összefüggő komponensek

Directed Depth First Search

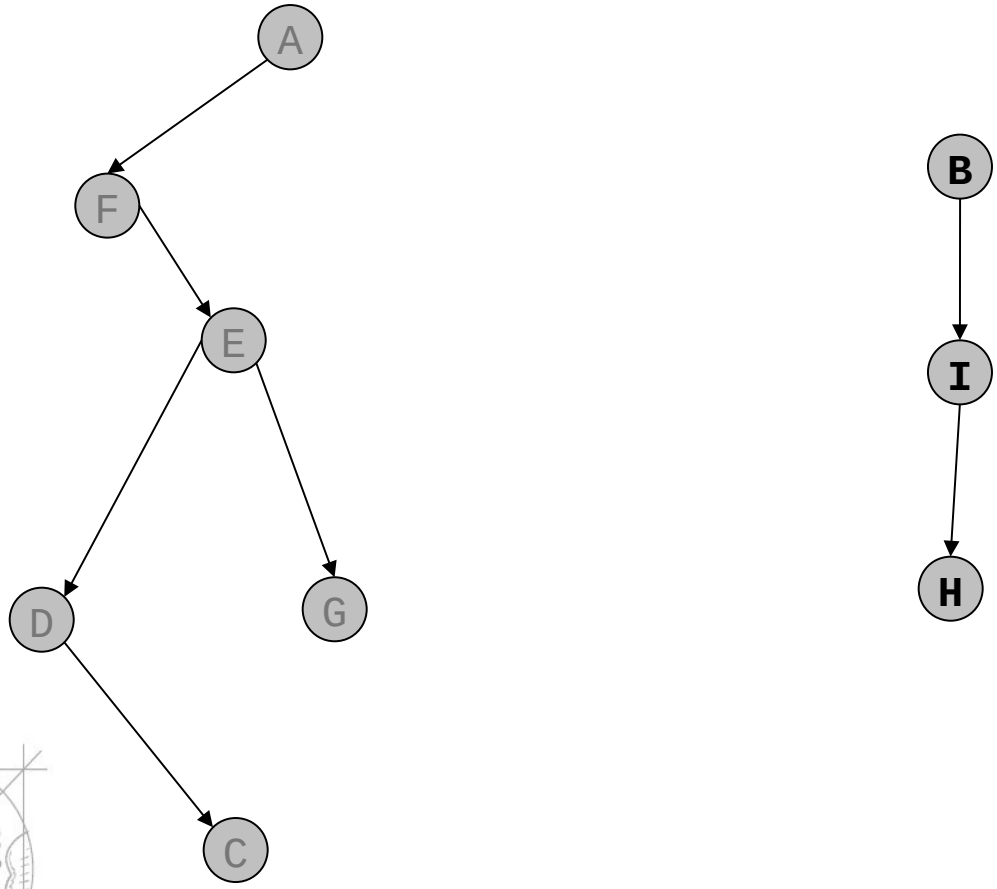


Adjacency Lists

A:	F	G	
B:	A	H	
C:	A	D	
D:	C	F	
E:	C	D	G
F:	E		
G:			
H:	B		
I:	H		

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Mélységi erdő



Elérési és elhagyási idők

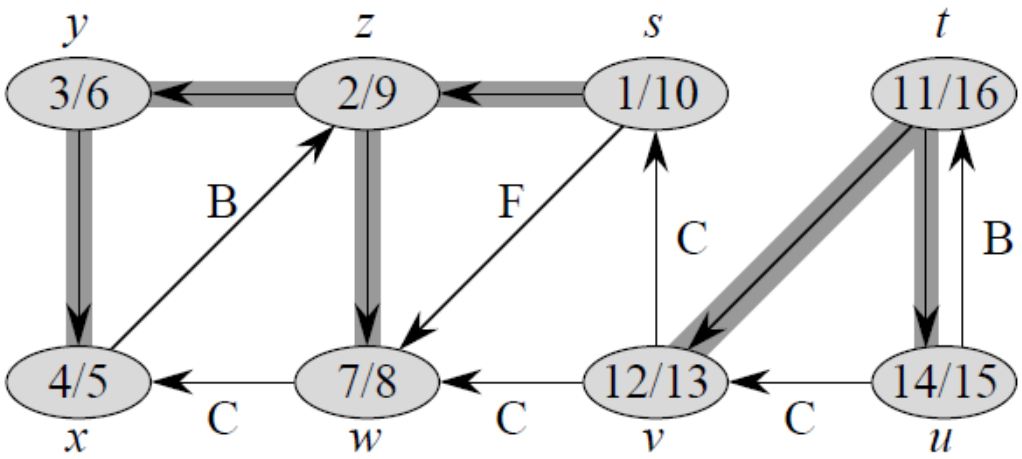
22.7. tétel (zárójelezés tétele). *Mélységi keresést alkalmazva egy (irányított vagy irányítatlan) $G = (V, E)$ gráfra, a következő három feltétel közül pontosan egy teljesül bármely u és v csúcsra:*

- *a $[d[u], f[u]]$ és $[d[v], f[v]]$ intervallumok diszjunktak, és u és v csúcsok közül egyik sem leszármazottja a másiknak a mélységi erdőben,*
- *a $[d[v], f[v]]$ intervallum tartalmazza a $[d[u], f[u]]$ intervallumot, és u csúcs v leszármazottja a mélységi fában, vagy*
- *$[d[u], f[u]]$ intervallum tartalmazza a $[d[v], f[v]]$ intervallumot, és v csúcs u leszármazottja a mélységi fában.*

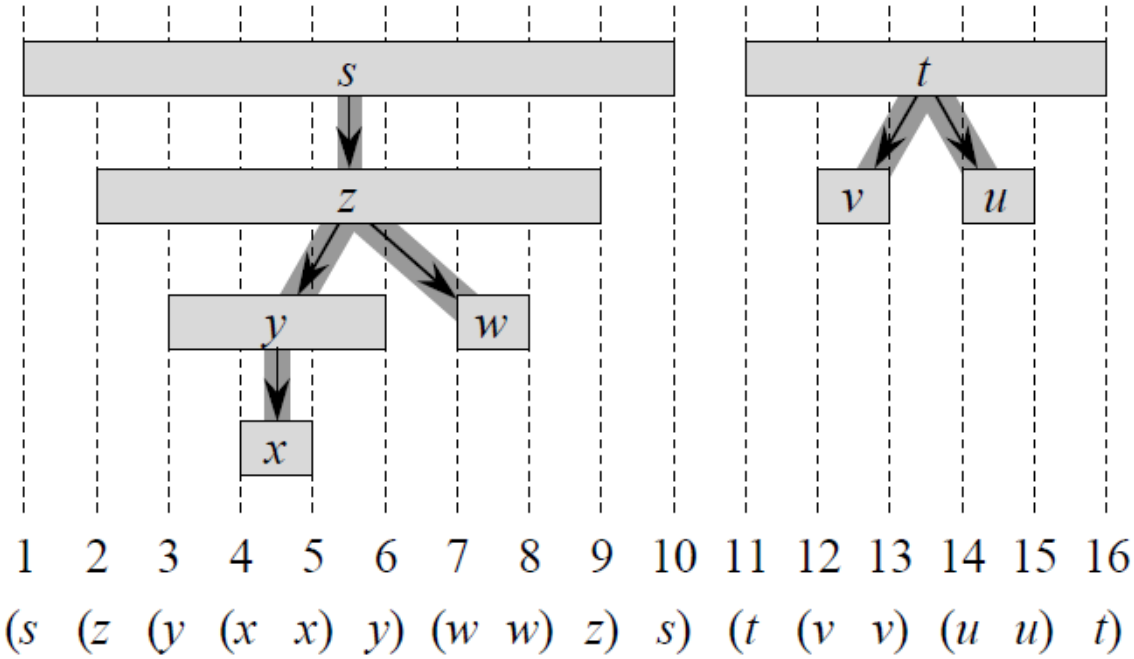
22.8. következmény (leszármazottak intervallumainak beágyazása). *v csúcs akkor és csak akkor leszármazottja az u csúcsnak az (irányított vagy irányítatlan) G gráf mélységi erdejében, ha $d[u] < d[v] < f[v] < f[u]$.*

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

(a)



(b)



Szélességi vs mélységi keresés

- Mindkettő bejárja a teljes gráfot (gráfbejárások)
- Bejárási erdők mást fejeznek ki

			✓	✓
			✓	✗

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

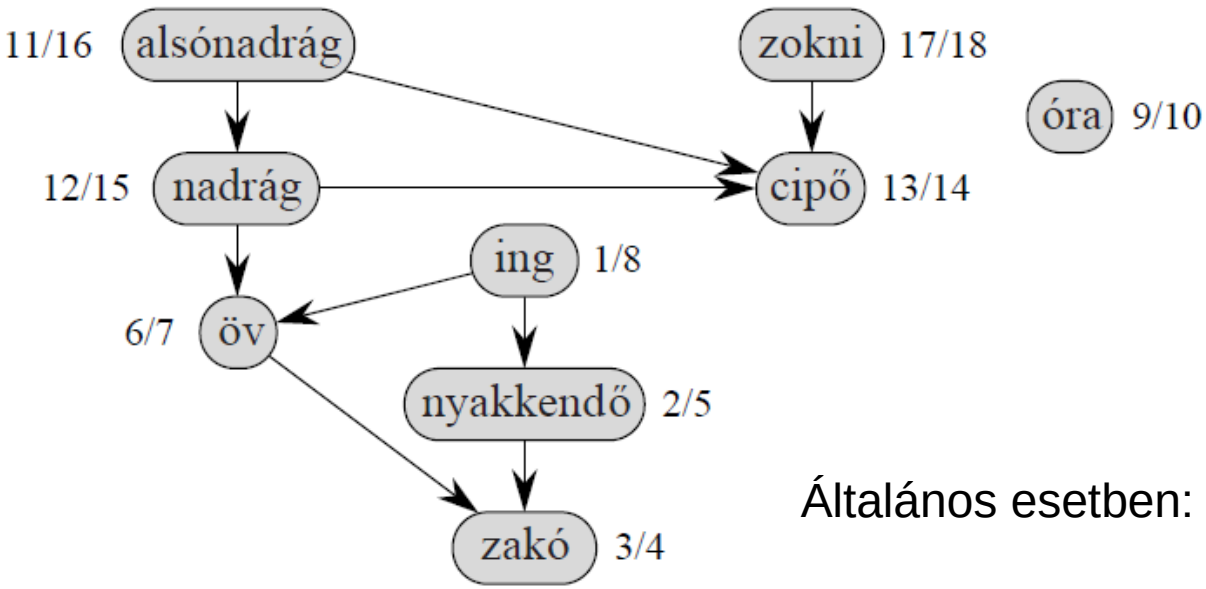
Algoritmusok és Adatszerkezetek I.

Topológikus rendezés

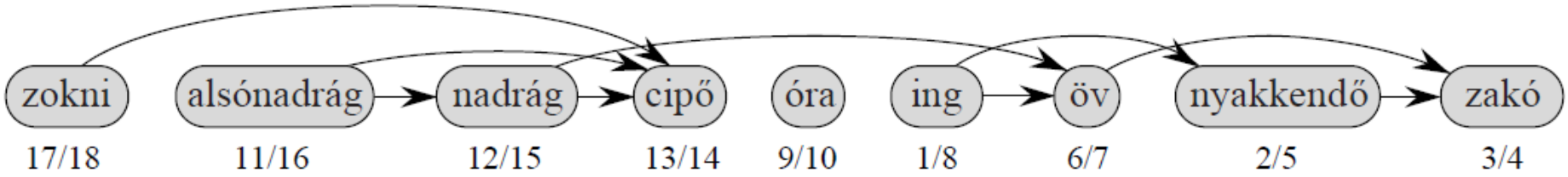
2020. november

9. hét – 4. videó
S09E04

Topológikus rendezés



Általános esetben: végrehajtási sorrend tervezése



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

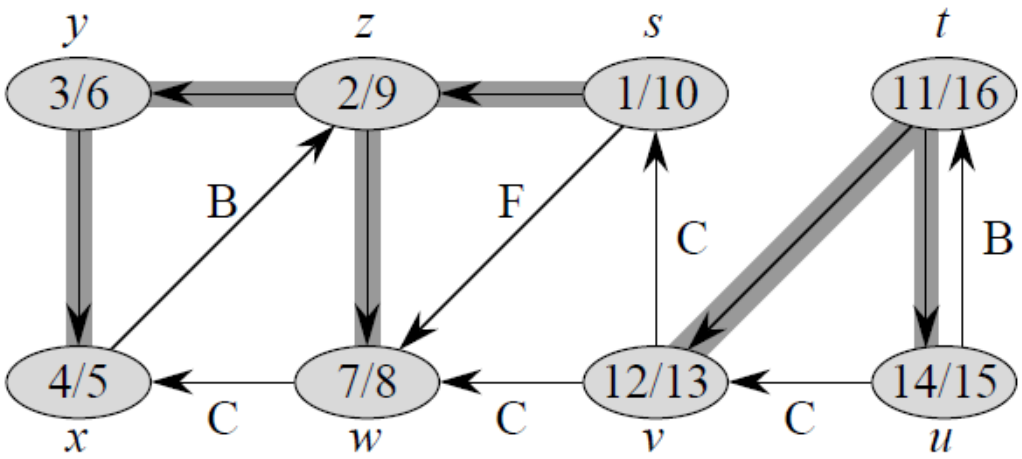
Topológikus rendezés

Egy $G = (V, E)$ irányított gráf ***topologikus rendezése*** a csúcsainak sorba rendezése úgy, hogy ha G-ben szerepel az (u, v) él, akkor u előzze meg v-t a sorban.

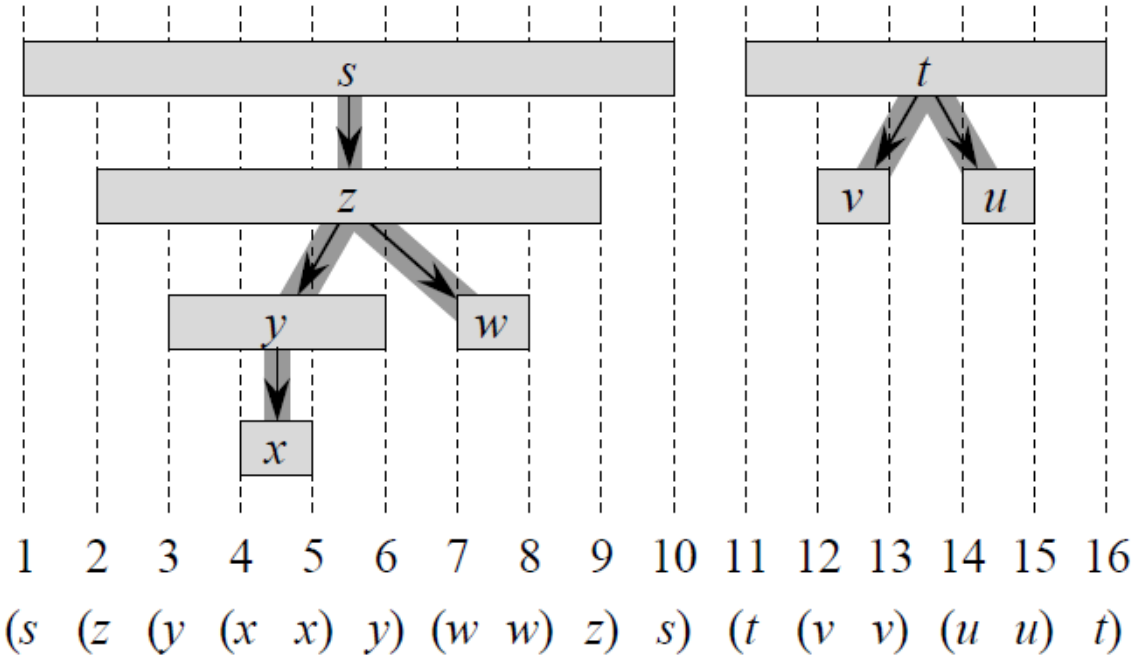
(Ha a gráf tartalmaz irányított kört, akkor nincs ilyen sorba rendezés.)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

(a)



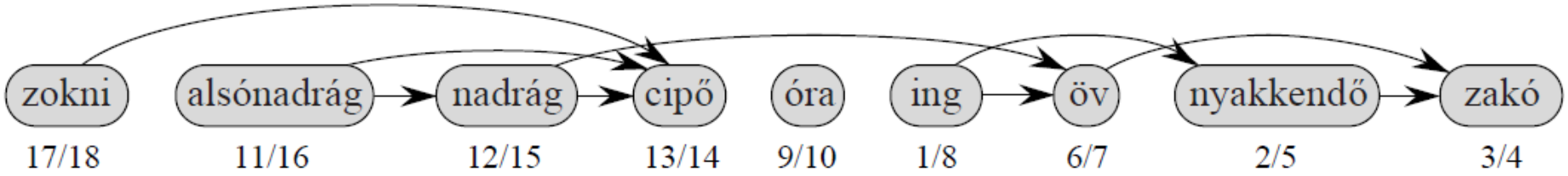
(b)



Topológikus rendezési algoritmus

TOPOLOGIKUS-RENDEZÉS(G)

1 MK(G) hívása; minden v csúcsra meghatározzuk az $f[v]$ elhagyási időt
2 az egyes csúcsok elhagyásakor szúrjuk be azokat egy láncolt lista elejére
3 **return** a csúcsok láncolt listája



SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Feszítőfák és legrövidebb utak

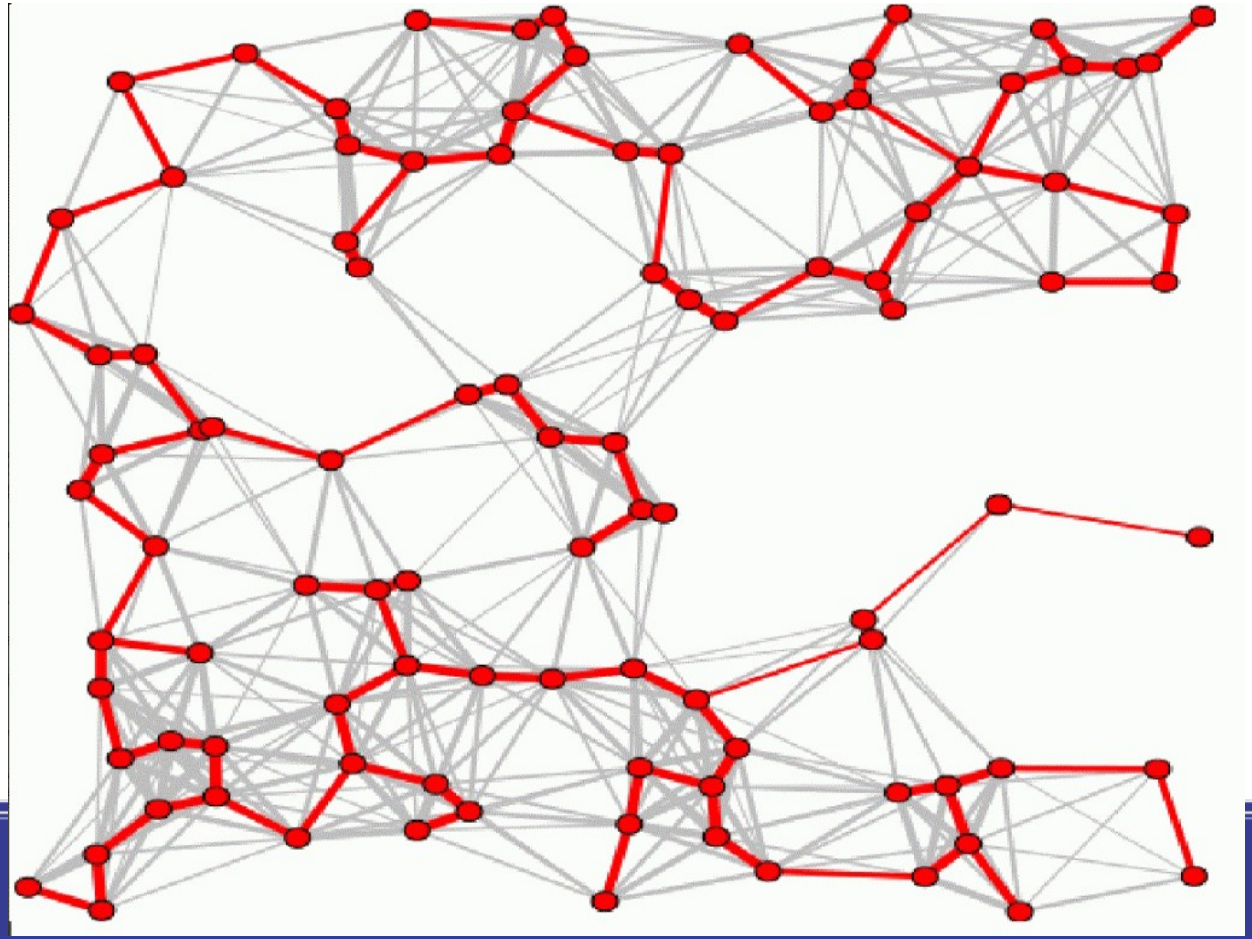
2019. november 20.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Minimális feszítőfák

Feszítőfa

Telefon, út stb. hálózat vagy elektromos áramkör tervezése



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Minimális feszítőfa probléma (Minimum Spanning Tree, MST)

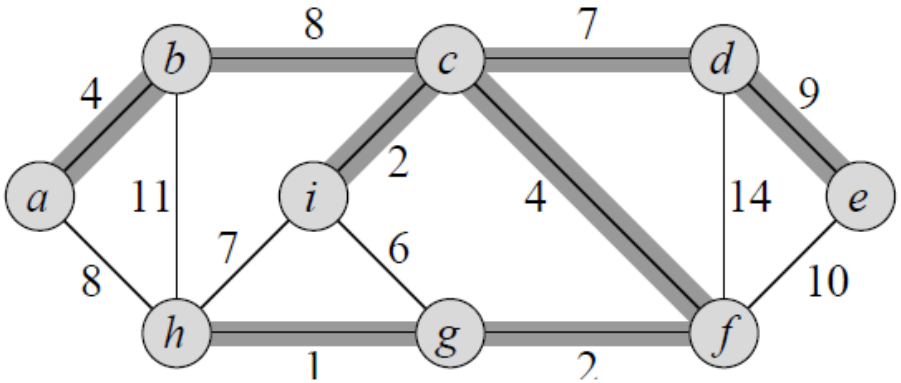
Bemenet: összefüggő, irányítatlan, súlyozott G=(V,E) gráf. A w(u,v) súly az (u,v) él költségét fejezi ki.

Kimenet: Egy T feszítőfa, amire minimális

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

Feszítőfa: minden csúcsot érintő, összefüggő, körmentes élhalmaz (fa)

Minimális feszítőfa



23.1. ábra. Egy összefüggő gráf minimális feszítőfája. Az élek súlyait az ábrán feltüntettük. A minimális feszítőfa éleit vastag vonalak jelölik. A fa teljes súlya 37. Nem ez az egyetlen minimális feszítőfa: cseréljük ki a (b, c) élt az (a, h) élre, és egy másik 37 súlyú feszítőfát kapunk.

Kruskal algoritmus

- Minimális fák erdejét tároljuk
- Kezdetben minden pont egy külön fa
- Minden lépésben a legkisebb, két fát összekötő élt húzzuk be (egyesítjük egyetlen fává a két fát)
- Mohó algoritmus!

https://visualgo.net/en/mst

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁ

Kruskal algoritmus

MFF-KRUSKAL(G, w)

```
A ← ∅
for minden v ∈ V[G]-re
    do HALMAZT-KÉSZÍT(v)
rendezzük az E éleit a w súly szerint növekvő sorrendben
for minden (u, v) ∈ E élre, az élek súly szerint növekvő sorrendjében
    do if HALMAZT-KERES(u) ≠ HALMAZT-KERES(v)
          then A ← A ∪ {(u, v)}
               EGYESÍT(u, v)
return A
```

$O(E\log E)$

$O(\log E)$

össz futásidő: $O(E\log E)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Kruskal helyessége

- véges lépésben véget ér
- eredmény élhalmaz:
  - minden V csúcsot érint
  - körmentes
  - összefüggő
  - minimális költségű

(bizonyítások)

Prim algoritmus

- egyetlen fát növesztünk
- tetszőleges gyökérpontból indulva
- minden lépésben új csúcsot kötünk be a fába
- legolcsóbb éllel elérhető csúcsot választjuk
- Mohó algoritmus!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁ

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Prim algoritmus

gyökérpont

MFF-PRIM(G, w, r)

```
for minden v ∈ V[G]re
    do kulcs[v] ← ∞
       π[u] ← NIL
kulcs[r] ← 0
Q ← V[G]
while Q ≠ ∅
    do u ← KIVESZ-MIN(Q)
       for minden v ∈ Szomszéd[u]-ra
           do if v ∈ Q and w(u, v) <kulcs[v]
                 then π[v] ← u
                      kulcs[v] ← w(u, v)
```

össz futásidő: $O(V\log V+E\log V)$ = $O(E\log V)$ ha V<E

fában nem szereplő csúcsok prioritási sora a legközelebbi fapont szerint

$O(\log V)$

$O(\log V)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Kruskal vs Prim

- **Kruskal** $O(E\log E)$
- **Prim** $O(E\log V)$
  (tovább gyorsítható $O(E+V\log V)$)

Sűrű gráfok esetén (E nagy) Prim előnyösebb, egyébként Kruskal egyszerűbb adatszerkezetet használ

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Adott csúcsból induló legrövidebb utak

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Legrövidebb utak probléma

Bemenet: irányított, súlyozott G=(V,E) gráf és egy *s* kezdőcsúcs. A w(u,v) súly az (u,v) él költségét fejezi ki.

Kimenet: Minden V csúcshoz a legrövidebb út s-ből indulva

$$p = \langle v_0, v_1, \ldots, v_k \rangle \text{ út } \textbf{\textit{súlya}}$$

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\rightsquigarrow} v\}, & \text{ha vezet út } u\text{-ból } v\text{-be,} \\ \infty, & \text{különben.} \end{cases}$$

Dijsktra algoritmusa

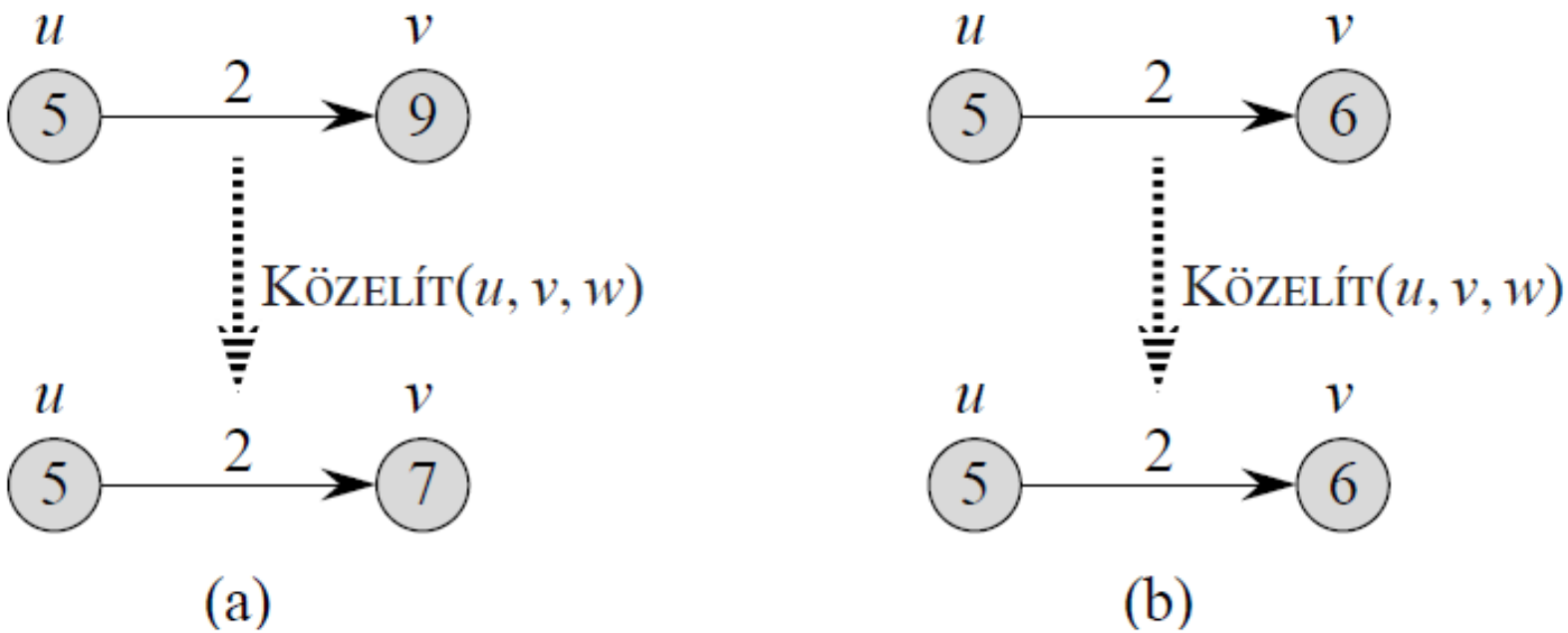
- azokat a csúcsokat tárolja amihez már megtalálta a legrövidebb utat
- minden lépésben egyel bővíti az elért csúcsok halmazát
- legkisebb legrövidebb úttal bíró csúcsot választja
- Mohó algoritmus!

https://visualgo.net/en/sssp

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁ

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Dijsktra algoritmusa



KÖZELÍT(u, v, w)

```
if d[v] > d[u] + w(u, v)
    then d[v] ← d[u] + w(u, v)
         π[v] ← u
```

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Dijsktra algoritmusa

össz futásidő: $O(E \log V)$

```
DIJKSTRA(G, s)
EGY-FORRÁS-KEZDŐÉRTÉK(G, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
        do u ← KIVESZ-MIN(Q)
            S ← S ∪ {u}
            for minden v ∈ Szomszéd[u]-ra
                do KÖZELÍT(u, v, w)
```

még nem lezárt csúcsok prioritási sora a legrövidebb összúthosszak szerint

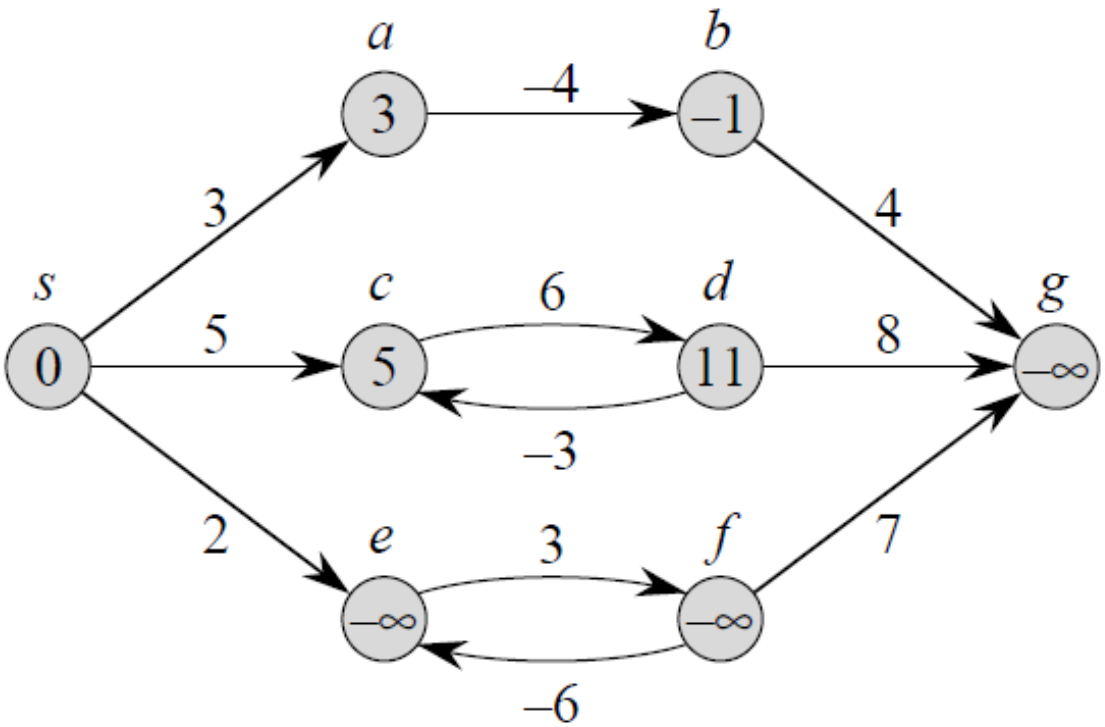
$O(\log V)$

Dijkstra helyessége

24.1. lemma (legrövidebb út részútja is legrövidebb út). *Adott egy* $w : E \to \mathbf{R}$ *súlyfüggvénnyel élsúlyozott, irányított* $G = (V, E)$ *gráf, legyen egy* v_1*-ből* v_k*-ba vezető legrövidebb út a* $p = \langle v_1, v_2, \ldots, v_k \rangle$*, és bármely olyan* i *és* j*-re, amelyre* $1 \leq i \leq j \leq k$*, legyen* $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ *egy* v_i*-ből* v_j*-be vezető részútja* p*-nek. Ekkor* p_{ij} *egy legrövidebb* v_i*-ből* v_j*-be vezető út.*

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

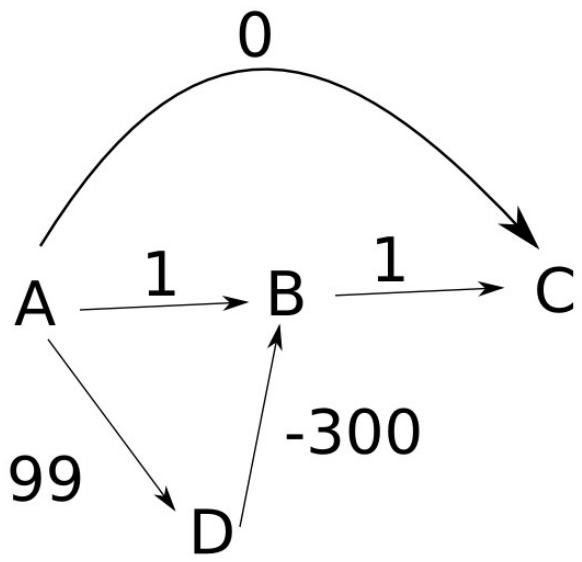
Negatív élsúlyok és körök



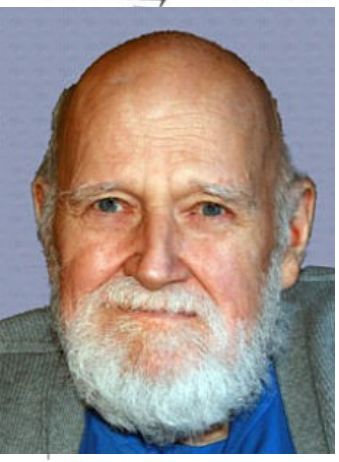
Körök nem okoznak problémát, de a negatív összsúlyú körök igen!

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Negatív élsúlyok és Dijkstra



Dijkstra nem ad helyes megoldást negatív élsúlyok esetén



Bellman-Ford algoritmus

- Negatív élsúlyok mellett is működik
- Ha elérhető negatív kör a kezdőcsúcsból akkor azt észreveszi
- Egy iterációban minden élen megpróbálunk javítani
- V-1 iterációra van szükség legrosszabb esetben

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Bellman-Ford algoritmus

$O(VE)$

```
BELLMAN–FORD(G, w, s)
EGY-FORRÁS-KEZDŐÉRTÉK(G, s)
for i ← 1 to |V[G]| – 1
    do for minden (u, v) ∈ E[G]
            KÖZELÍT(u, v, w)
for minden (u, v) ∈ E[G]
    do if d[v]⟩d[u] + w(u, v)
            then return HAMIS
return IGAZ
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Legrövidebb utak minden csúcspárra

Legrövidebb utak minden pontpárra

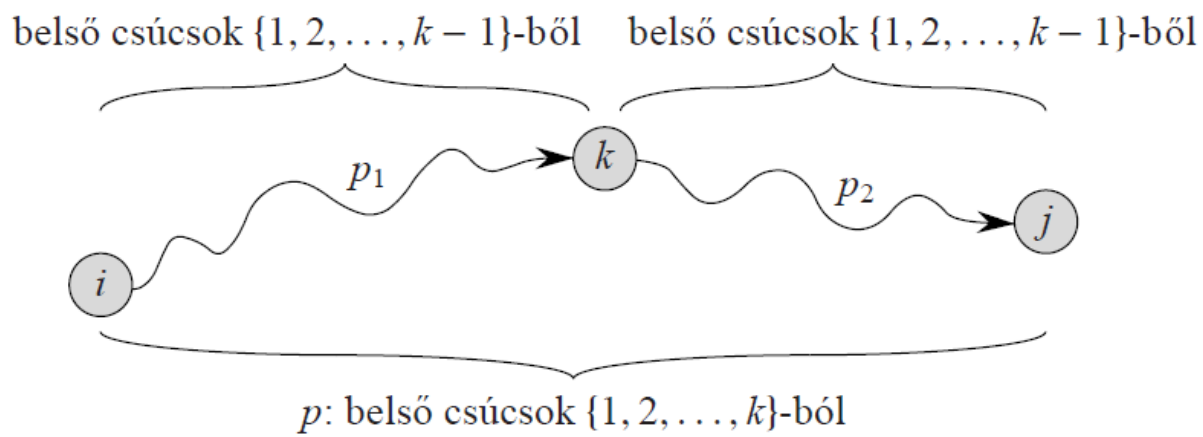
Dijsktra minden kezdőpontból:
$O(VE\log V)$

Bellman-Ford minden kezdőpontból:
$O(V^2E)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Floyd-Warshall algoritmus

- Dinamikus programozás
- d mátrix d_{ij} eleme az ismert legrövidebb út i-ből j-be
- dinamikus programozással a belső csúcsként használható csúcsokat bővítjük



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

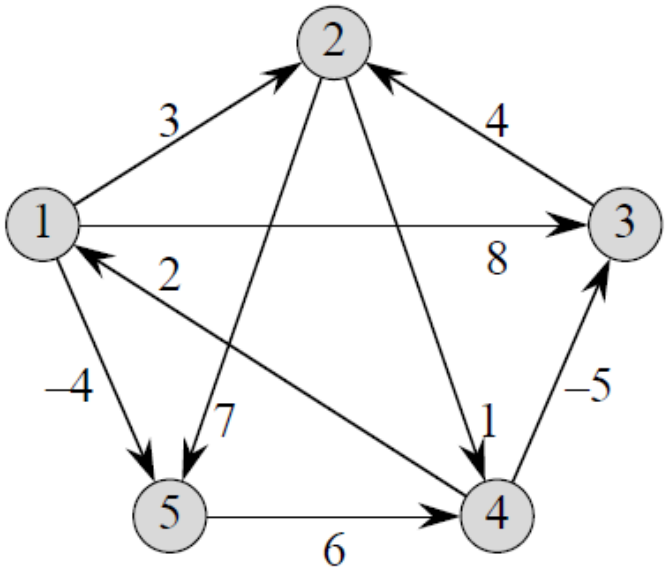
Floyd-Warshall algoritmus

$$w_{ij} = \begin{cases} 0, & \text{ha } i = j, \\ \text{az irányított } (i, j) \text{ él hossza}, & \text{ha } i \neq j \text{ és } (i, j) \in E, \\ \infty, & \text{ha } i \neq j \text{ és } (i, j) \notin E. \end{cases}$$

FLOYD–WARSHALL(W)

```
n ← sorok-száma[W]
D^(0) ← W
for k ← 1 to n
    do for i ← 1 to n
            do for j ← 1 to n
                    d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

$O(V^3)$



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Legrövidebb utak minden pontpárra

- Ha nincsenek negatív élsúlyok és ritka a gráf akkor Dijsktra minden kezdőpontból $O(VE\log V)$
- Ha vannak negatív élsúlyok, de nincsenek negatív összköltségű körök vagy sűrű a gráf akkor Floyd-Warshall $O(V^3)$
- Ha negatív összköltésgű körök is lehetnek akkor Ford-Bellman minden kezdőcsúcsra $O(V^2E)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Gráfalgoritmusok

- Minimális feszítőfák
  - Kruskal
  - Prim
- Legrövidebb utak egy csúcsból
  - Dijsktra
  - Bellman-Ford
- Legrövidebb utak összes pontpárra
  - Floyd-Warshall

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

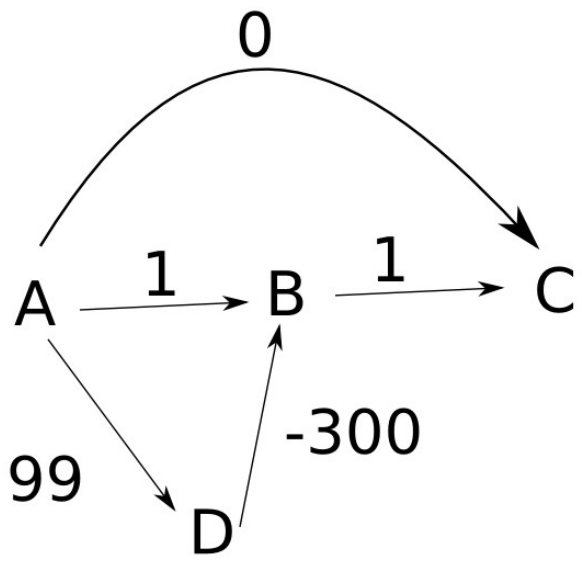
Bellman-Ford algoritmus

2020. november

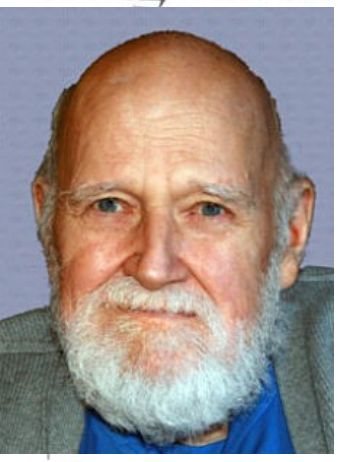
10. hét – 3. videó
S10E03

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Negatív élsúlyok és Dijkstra



Dijkstra nem ad helyes megoldást negatív élsúlyok esetén



Bellman-Ford algoritmus

- Negatív élsúlyok mellett is működik
- Ha elérhető negatív kör a kezdőcsúcsból akkor azt észreveszi
- Egy iterációban minden élen megpróbálunk javítani
- V-1 iterációra van szükség legrosszabb esetben

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Bellman-Ford algoritmus

$O(VE)$

```
BELLMAN–FORD(G, w, s)
    EGY-FORRÁS-KEZDŐÉRTÉK(G, s)
    for i ← 1 to |V[G]| – 1
        do for minden (u, v) ∈ E[G]
            KÖZELÍT(u, v, w)
    for minden (u, v) ∈ E[G]
        do if d[v] > d[u] + w(u, v)
            then return HAMIS
    return IGAZ
```

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Legrövidebb utak minden csúcspárra

2020. november

10. hét – 4. videó
S10E04

Legrövidebb utak minden pontpárra

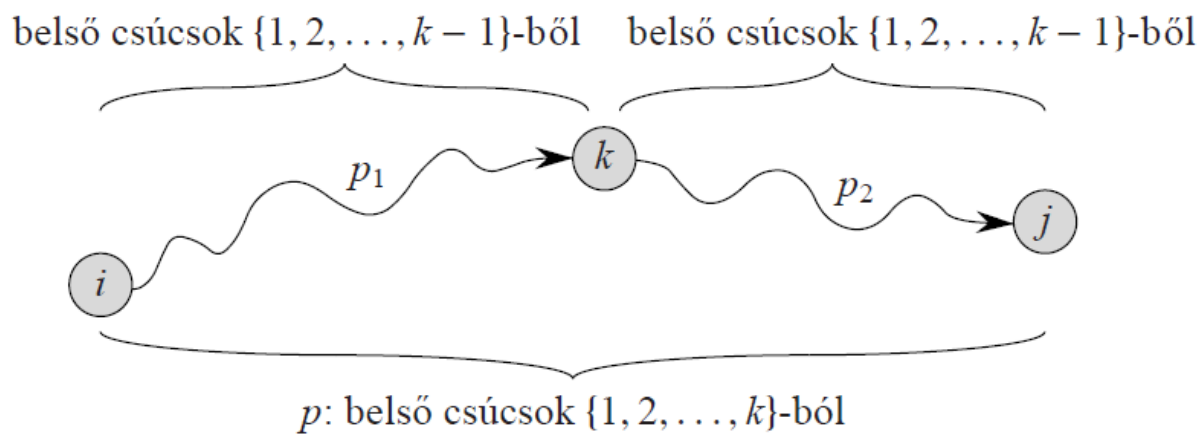
Dijsktra minden kezdőpontból:
$O(VE\log V)$

Bellman-Ford minden kezdőpontból:
$O(V^2E)$

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Floyd-Warshall algoritmus

- Dinamikus programozás
- d mátrix d_{ij} eleme az ismert legrövidebb út i-ből j-be
- dinamikus programozással a belső csúcsként használható csúcsokat bővítjük



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

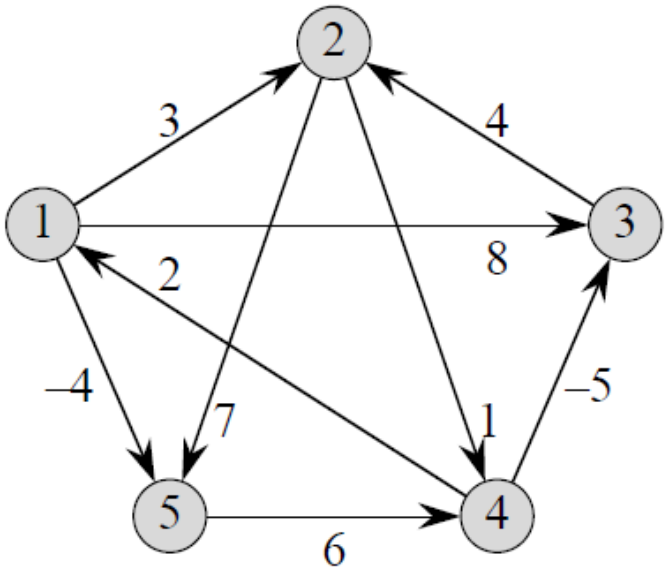
Floyd-Warshall algoritmus

$$w_{ij} = \begin{cases} 0, & \text{ha } i = j, \\ \text{az irányított } (i, j) \text{ él hossza}, & \text{ha } i \neq j \text{ és } (i, j) \in E, \\ \infty, & \text{ha } i \neq j \text{ és } (i, j) \notin E. \end{cases}$$

FLOYD–WARSHALL(W)

```
n ← sorok-száma[W]
D^(0) ← W
for k ← 1 to n
    do for i ← 1 to n
            do for j ← 1 to n
                    d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

$O(V^3)$



$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

Legrövidebb utak minden pontpárra

- Ha nincsenek negatív élsúlyok és ritka a gráf akkor Dijsktra minden kezdőpontból $O(VE\log V)$
- Ha vannak negatív élsúlyok, de nincsenek negatív összköltségű körök vagy sűrű a gráf akkor Floyd-Warshall $O(V^3)$
- Ha negatív összköltésgű körök is lehetnek akkor Ford-Bellman minden kezdőcsúcsra $O(V^2E)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Rendező algoritmusok

2019. november 27.

Rendezési feladat (sort)

- **Bemenet:** egy n számból álló a_1, a_2, . . . , a_n tömb.
- **Kimenet:** a bemenő tömb elemeinek olyan a_1, a_2, . . . , a_n permutációja (átrendezése), amelyre $a_1 \leq a_2 \leq \ldots \leq a_n$

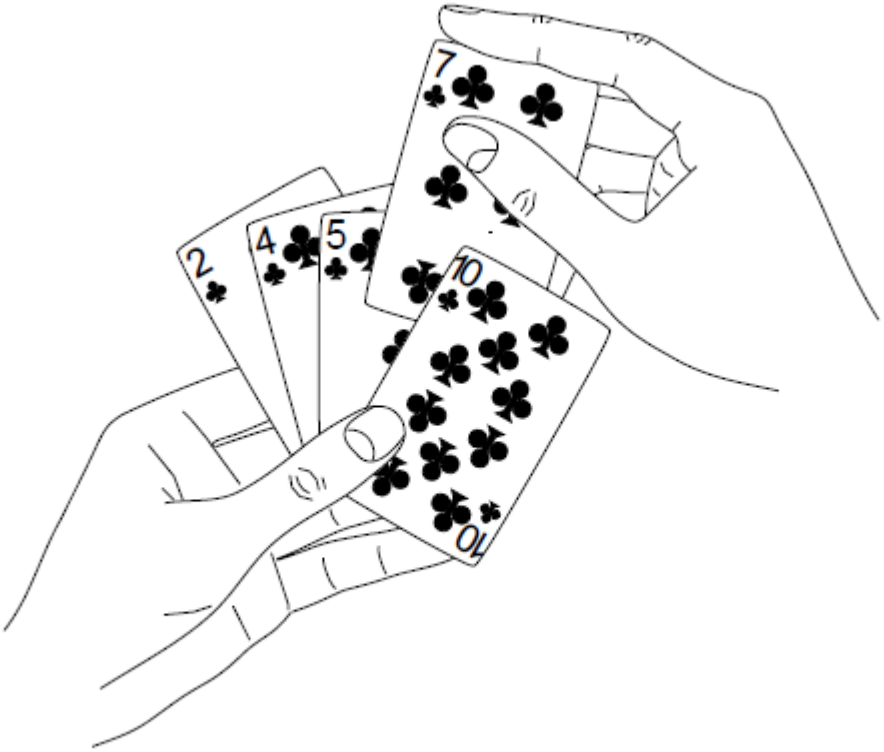
SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Miért fontosak a rendező algoritmusok?

- Gyakran használt részfeladat
- Sok probléma trivilás vagy gyors ha rendezett a bemenet → 0. lépésben rendezzünk
  - medián megtalálása
  - bináris keresés
- Számtalan megoldás → különböző szituációkban más és más (saját?)

Beszúró rendezés (insertion sort)



2.1. ábra. Kézben tartott lapok rendezése beszúró rendezéssel.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Beszúró rendezés (insertion sort)

```
BESZÚRÓ-RENDEZÉS(A)

for j ← 2 to hossz[A]
    do kulcs ← A[j]
        ▷ A[j] beszúrása az A[1 . . j − 1] rendezett sorozatba.
        i ← j − 1
        while i > 0 és A[i] > kulcs
            do A[i + 1] ← A[i]
                i ← i − 1
        A[i + 1] ← kulcs
```

$O(n^2)$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összefésülő rendezés (merge sort)

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.

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Összefésülő rendezés

ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, r)

```
if p < r
    then q ← ⌊(p + r)/2⌋
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, p, q)
         ÖSSZEFÉSÜLŐ-RENDEZÉS(A, q + 1, r)
         ÖSSZEFÉSÜL(A, p, q, r)
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

ÖSSZEFÉSÜL(A, p, q, r)

```
n1 ← q − p + 1
n2 ← r − q
az L[1 .. n1 + 1] és R[1 .. n2 + 1] tömbök létrehozása
for i ← 1 to n1
    do L[i] ← A[p + i − 1]
for j ← 1 to n2
    do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
    do if L[i] ≤ R[j]
          then A[k] ← L[i]
               i ← i + 1
          else A[k] ← R[j]
               j ← j + 1
```

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összefésülő rendezés futási ideje

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

1. Rekurziós fa módszerrel megsejtük
2. Helyettesítő módszerrel bizonyítjuk

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

https://visualgo.net/en/sorting

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Helyben rendezés

Rendező algoritmusok memóriaigénye

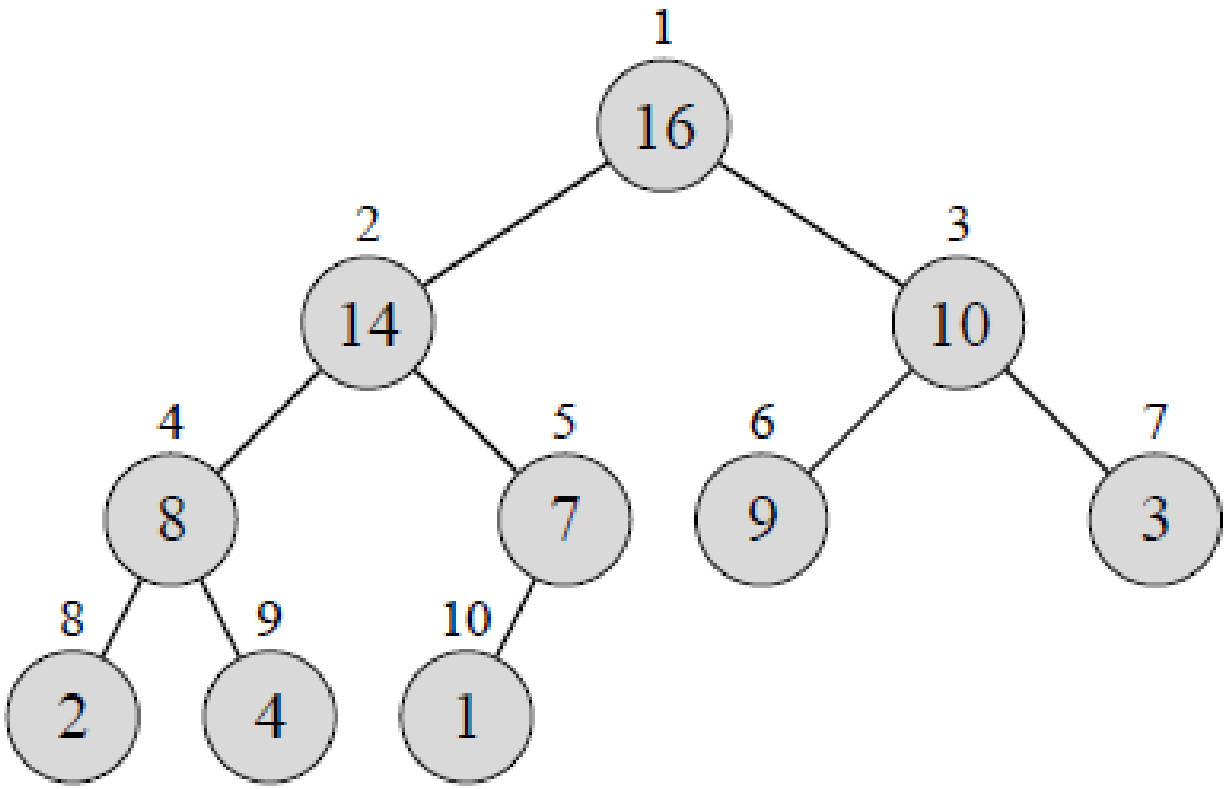
Beszúró rendezés $O(1)$

Összefésülő rendezés $O(n)$

Helyben rendezés: bemeneti tömbön felül legfeljebb konstans méretű tár

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Kupac



https://visualgo.net/en/heap

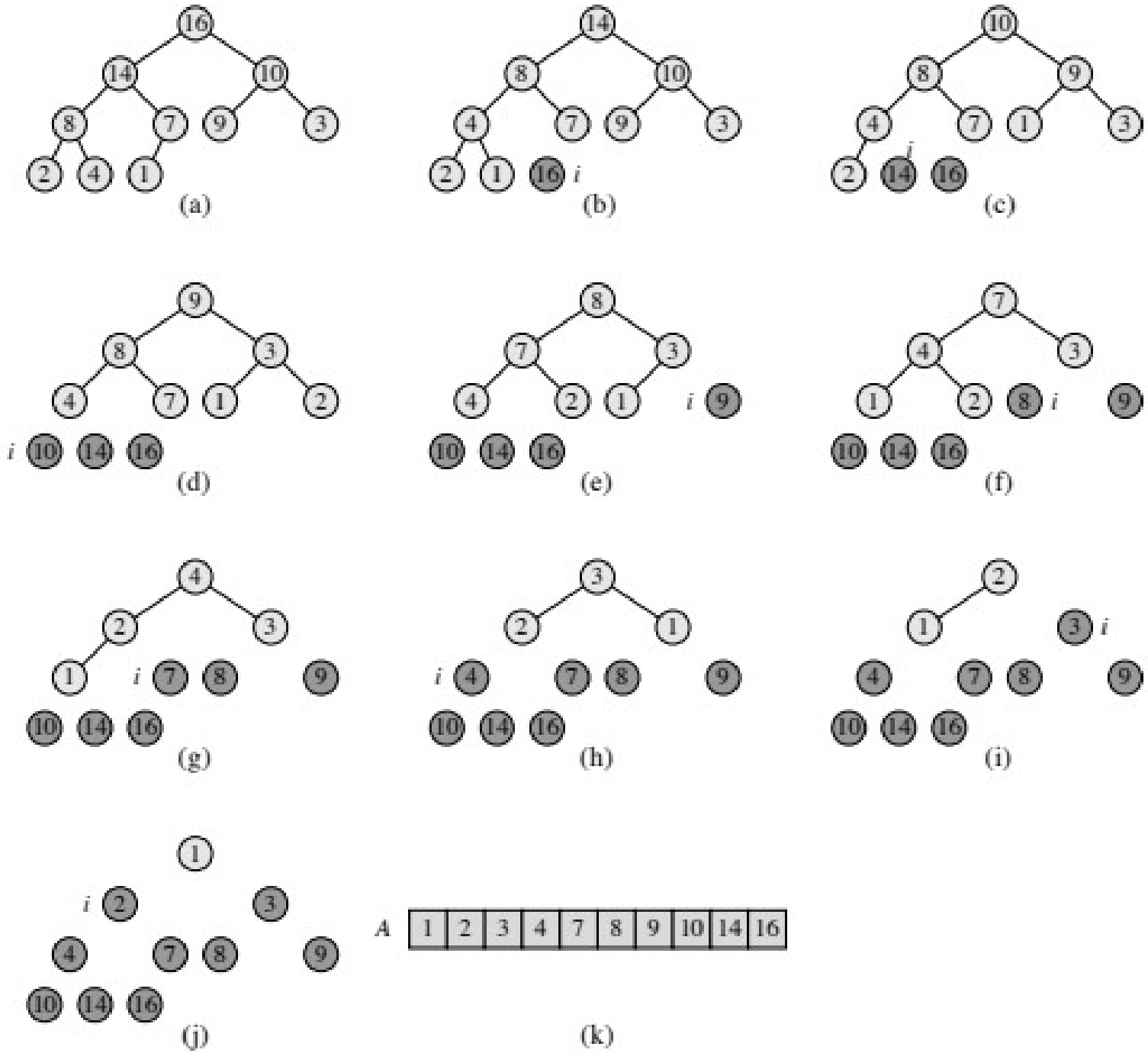
Kupacrendezés (heap sort)

```
KUPACRENDEZÉS(A)

MAXIMUM-KUPACOT-ÉPÍT(A)
for i ← hossz[A] downto 2
    do A[1] ↔ A[i] csere
        kupac-méret[A] ← kupac-méret[A] − 1
        MAXIMUM-KUPACOL(A, 1)
```

$O(n\log n)$

Helyben rendezés

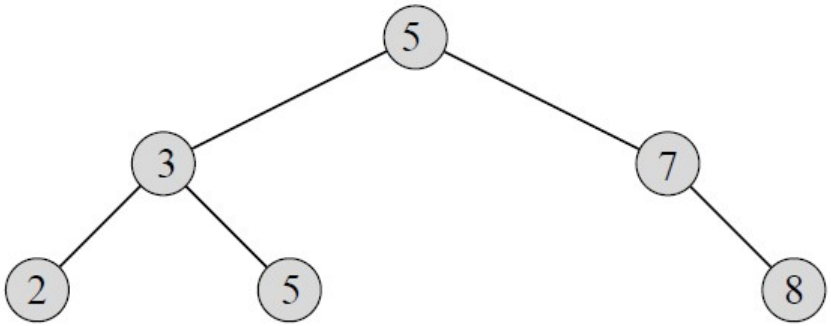
SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

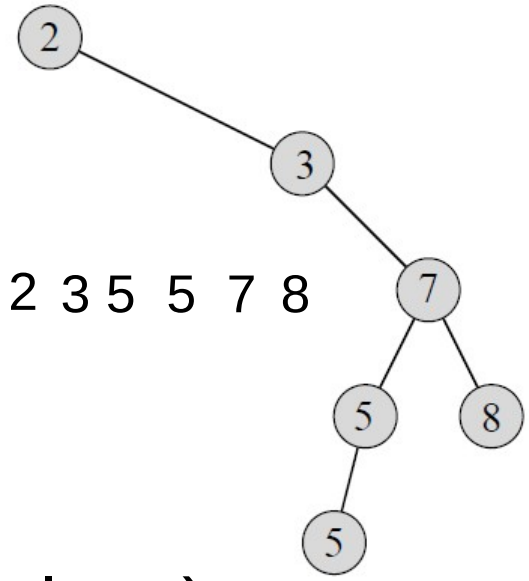

(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
A
1 2 3 4 7 8 9 10 14 16
(k)

Rendezés bináris keresőfával

(tree sort)

if $x \neq$NIL
 then FABEJÁRÁS(*bal*[x])
 muvelet(*kulcs*[x])
 FABEJÁRÁS(*jobb*[x])





2 3 5 5 7 8

2 3 5 5 7 8 piros-fekete fával: $O(n\log n)$

Helyben rendezés, de pointerek miatt nagyobb tárigény

Gyorsrendezés (quick sort)

Felosztás: Az $A[p\ldots r]$ tömböt két (esetleg üres) $A[p\ldots q\text{-}1]$ és $A[q+1\ldots r]$ résztömbre osztjuk úgy, hogy az

$A[p\ldots\ q-1]$ minden eleme kisebb vagy egyenlő $A[q]$-nál és $A[q+1\ldots r]$ minden eleme nagyobb $A[q]$-nál. A q index kiszámítása része a felosztó eljárásnak

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Gyorsrendezés (quick sort)

- **Uralkodás:** Az $A[p \ldots q-1]$ és $A[q+1 \ldots r]$ résztömböket a gyorsrendezés rekurzív hívásával rendezzük.
- **Összevonás:** helyben rendezés, nincs szükség egyesítésre

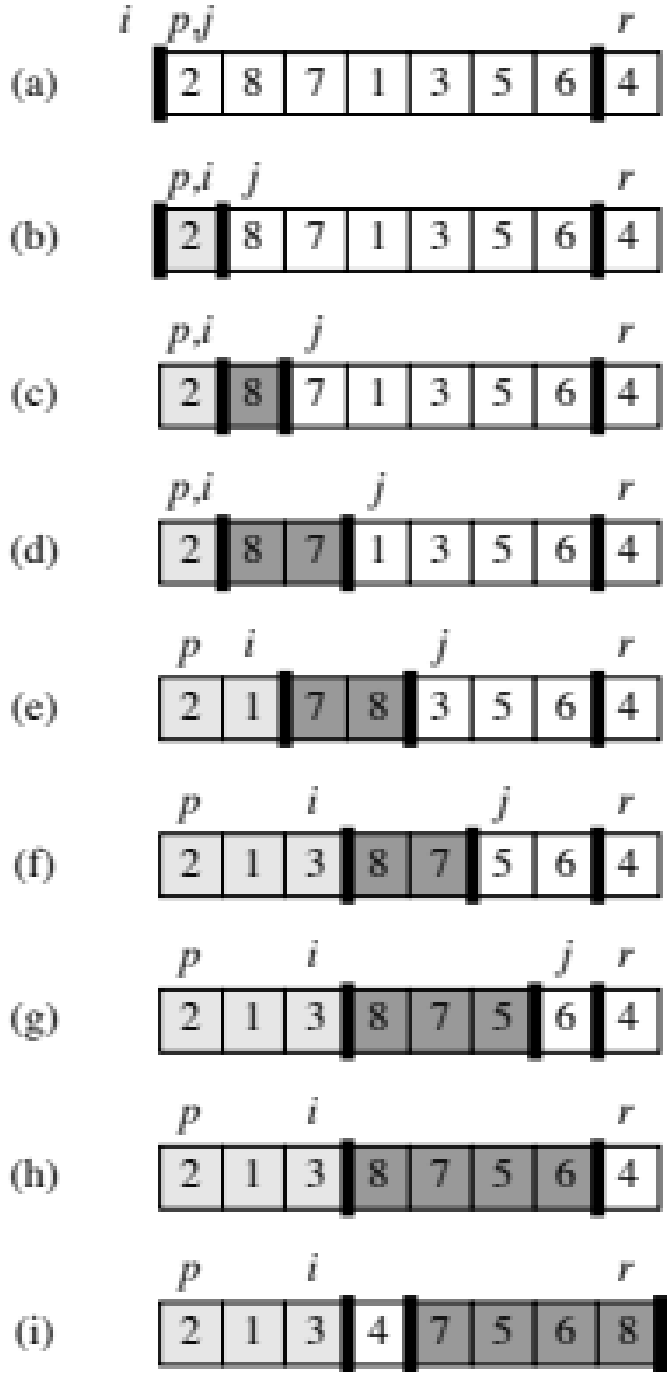
SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

```
GYORSRENDEZÉS(A, p, r)

if p < r
    then q ← FELOSZT(A, p, r)
         GYORSRENDEZÉS(A, p, q − 1)
         GYORSRENDEZÉS(A, q + 1, r)
```

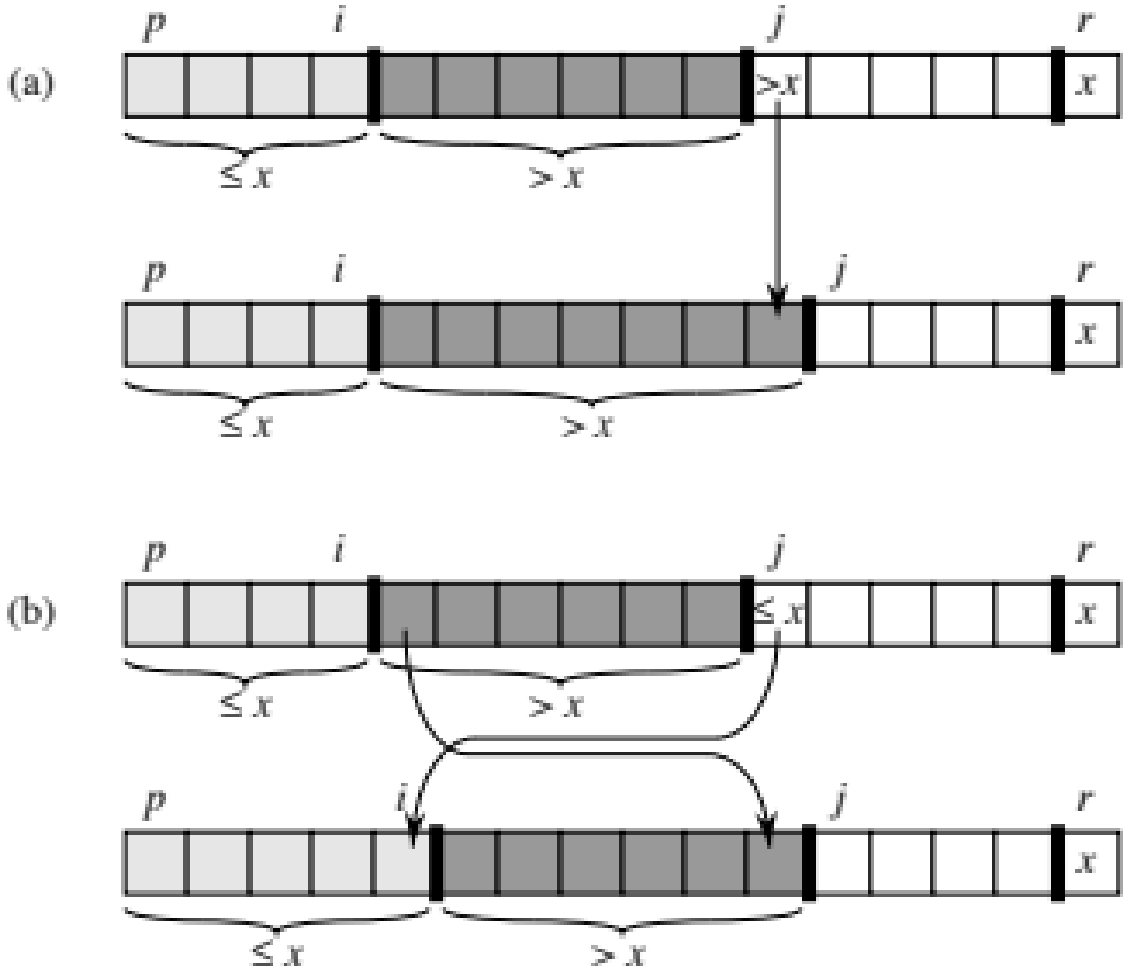
```
FELOSZT(A, p, r)

x ← A[r]
i ← p − 1
for j ← p to r − 1
    do if A[j] ≤ x
          then i ← i + 1
               A[i] ↔ A[j] csere
A[i + 1] ↔ A[r] csere
return i + 1
```

pivot (őrszem)
bárhogyan választhajuk,
pl. első elem vagy véletlen


(a) i p,j r
2 8 7 1 3 5 6 4
(b) p,i j r
2 8 7 1 3 5 6 4
(c) p,i j r
2 8 7 1 3 5 6 4
(d) p,i j r
2 8 7 1 3 5 6 4
(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 r
2 1 3 8 7 5 6 4
(i) p i r
2 1 3 4 7 5 6 8

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Gyorsrendezés helyessége



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Gyorsrendezés futásideje

Legrosszabb eset: minden FELOSZT egy 0 és egy n-1 méretű résztömbre oszt

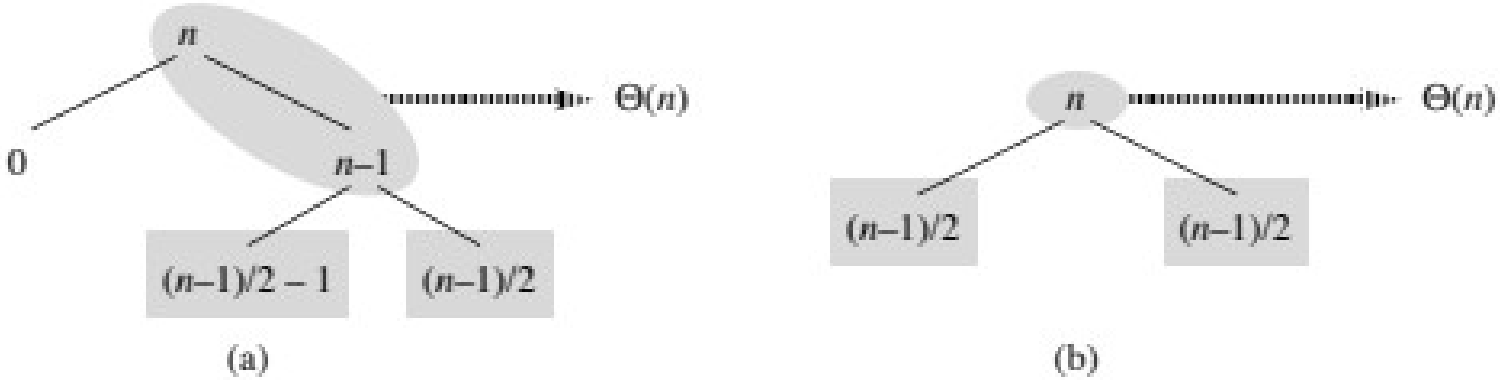
$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n). \end{aligned} \qquad T(n) = \Theta(n^2)$$

Legjobb eset: minden FELOSZT felezi a tömböt

$$T(n) \le 2T\left(\frac{n}{2}\right) + \Theta(n), \qquad T(n) = \Theta(n \lg n)$$

Gyorsrendezés futásideje

- **Átlagos eset:** közelebb a legjobb esethez
- bármilyen állandó a [illegible] s esetén $O(n\log n)$, pl $T(n) \le T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + cn$



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

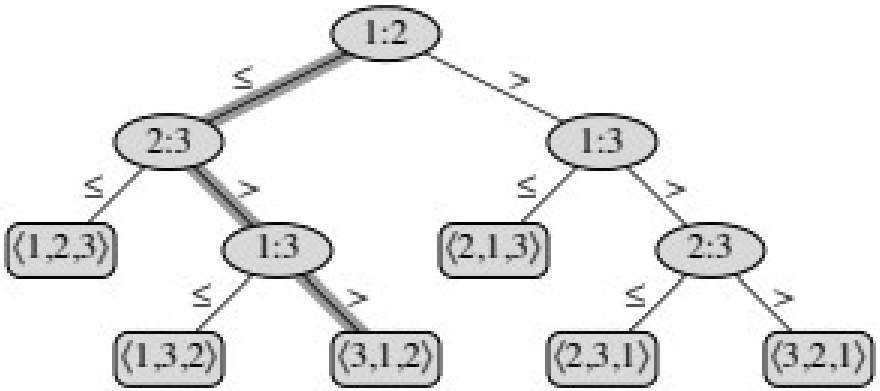
Összehasonlító rendezések

- ***Összehasonlító rendezés***: rendezéshez elemek párjainak < = > műveleteit használjuk

- Eddigi rendező algoritmusok mind összehasonlító rendezések voltak

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Összehasonlító rendezések futásideiének alsó korlátia



8.1. ábra. Döntési fa három elem beszúrásos rendezéséhez. Az $i:j$ párral jelölt belső csúcs az a_i és a_j elem összehasonlítását, a $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ permutációval jelölt levél pedig az $a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$ rendezést jelenti. Az árnyékolt útvonal az $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$ bemeneti sorozat rendezése során meghozott döntéseket jelzi, a levélen lévő $\langle 3, 1, 2 \rangle$ permutáció pedig azt, hogy a sorba rendezés $a_3 = 5 \leq a_1 = 6 \leq a_2 = 8$. A bemenő elemeknek $3! = 6$ permutációja lehetséges, így a döntési fának legalább 6 levelűnek kell lennie.

Bármely összehasonlító rendező algoritmus a legrosszabb esetben $\Omega(n \log n)$ összehasonlítást végez!

Leszámláló rendezés (counting sort)

<u>HA</u> az *n* bemeneti elem mindegyike 0 és *k* közötti egész szám, ahol *k* egy egész.

```
LESZÁMLÁLÓ-RENDEZÉS(A, B, k)

for i ← 0 to k
    do C[i] ← 0
for j ← 1 to hossz[A]
    do C[A[j]] ← C[A[j]] + 1
▷ C[i] most azoknak az elemeknek a számát tartalmazza, amelyek értéke i.
for i ← 1 to k
    do C[i] ← C[i] + C[i − 1]
▷ C[i] most azoknak az elemeknek a számát tartalmazza,
  amelyek értéke kisebb vagy egyenlő, mint i.
for j ← hossz[A] downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] − 1
```

$\Theta(n+k)$

tár: $n+k$

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3

	0	1	2	3	4	5
C	2	0	2	3	0	1

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

	1	2	3	4	5	6	7	8
B							3	

	0	1	2	3	4	5
C	2	2	4	6	7	8

(c)

	1	2	3	4	5	6	7	8
B		0					3	

	0	1	2	3	4	5
C	1	2	4	6	7	8

(d)

	1	2	3	4	5	6	7	8
B		0				3	3	

	0	1	2	3	4	5
C	1	2	4	5	7	8

(e)

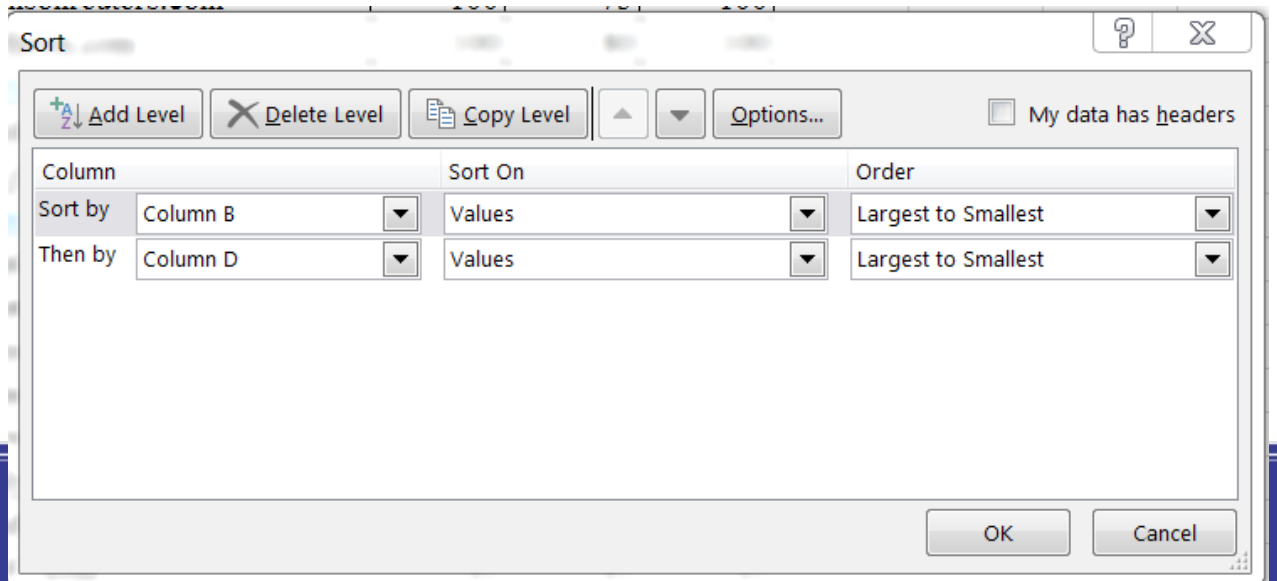
	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

(f)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Rendezések stabilitása

Egy rendezés ***stabil*** ha az azonos értékű elemek ugyanabban a sorrendben jelennek meg a kimeneti tömbben, mint ahogyan a bemeneti tömbben szerepeltek.



Számjegyes rendezés (radix sort)

Rendezzünk dátumokat év, hónap, nap szerint!

Megoldás1: összehasonlító rendezés

Megoldás2: 3 db stabil rendezés először napok, majd hónapok, végül évek szerint

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

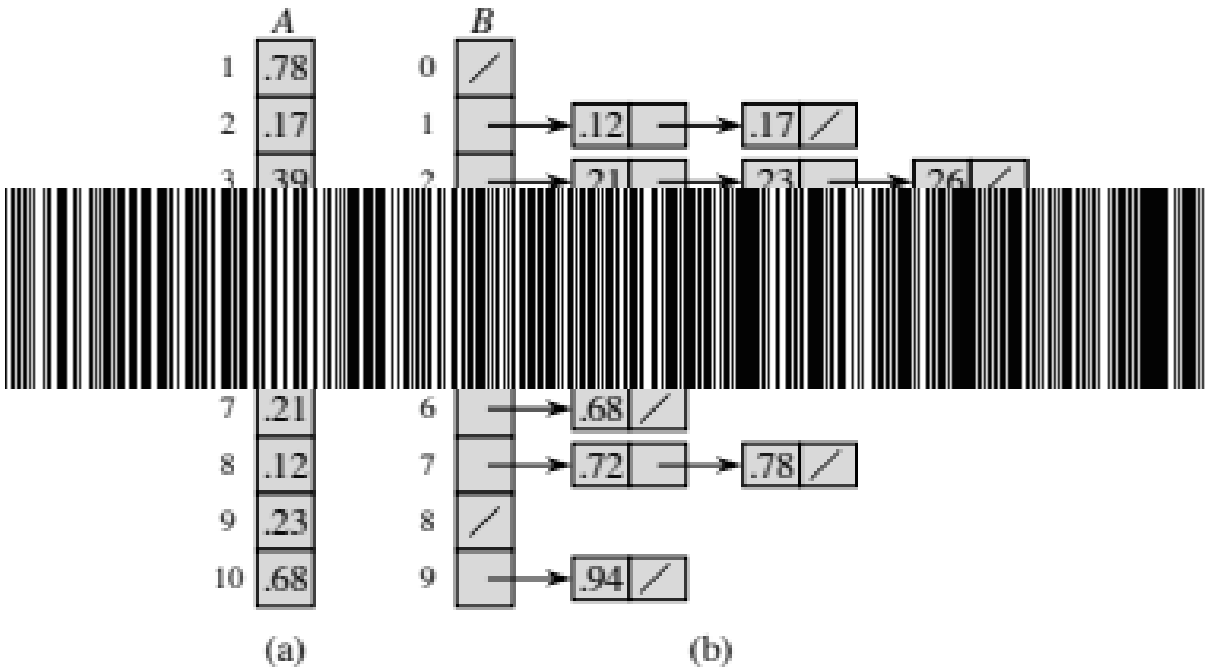
Számjegyes rendezés (radix sort)

Számjegyes rendezés: legalacsonyabb helyiértéktől legmagasabbig minden lépésben egy stabil renezés

8.3. lemma. *Legyen adott n darab d jegyből álló szám, ahol a számjegyek legfeljebb k értéket vehetnek fel. Ekkor a* SZÁMJEGYES-RENDEZÉS $\Theta(d(n+k))$ *időben rendezi helyesen ezeket a számokat.*

Edényrendezés (bucket sort)

HA a rendezendő tömb elemeinek eloszlása közel egyenletes



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Edényrendezés (bucket sort)

EDÉNY-RENDEZÉS(A)

$n \leftarrow hossz[A]$
for $i \leftarrow 1$ **to** n
    do szúrjuk be az $A[i]$ elemet a $B[\lfloor nA[i] \rfloor]$ listába.
for $i \leftarrow 0$ **to** $n-1$
    do rendezzük a $B[i]$ listát beszúrásos rendezéssel.
sorban összefűzzük a $B[0], B[1], \ldots, B[n-1]$ listákat.

n_i az i-edik edénybe eső elemek számának várható értéke

$$T(n) = \Theta(n) + \sum_{i=0}^{n-1} O(n_i^2).$$

Engem válassz ha közel rendezett a tömb vagy nagyon kicsi (n<10) bemenet esetén!

	Legjobb	Átlagos	Legrosszabb	Tár	Stabil
Beszúró	n	n^2	n^2	1	✓
Összefésülő	[illegible]	[illegible]	[illegible]	[illegible]	✓
Kupac	[illegible]	[illegible]	[illegible]	[illegible]	✗
Bináris Fa	[illegible]	[illegible]	[illegible]	[illegible]	✓
Gyors	[illegible]	[illegible]	[illegible]	[illegible]	✗
Leszámoló	[illegible]	[illegible]	[illegible]	[illegible]	✓
Számjegy	[illegible]	[illegible]	[illegible]	[illegible]	✓
Edény	[illegible]	[illegible]	[illegible]	[illegible]	✓

Ha stabilitásra vágysz engem válassz! Csak hozz magaddal memóriát is…

Elvi[illegible] leg[illegible] cs[illegible]

Engem senki sem szeret mert a gyakorlatban fákat memóriában tárolni drága ⏸

Ha nem tudsz semmit az adatról én vagyok általában a leggyorsabb (összehasonlító rendezés)! Viszont nem vagyok stabil…

[illegible] kell rendezned!

Ha az értékek eloszlása egyenletes én vagyok a legjobb!

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Sztringalgoritmusok

2020. december

12. hét – 1. videó
S12E01

Sztringalgoritmusok

Google

algoritmusok

All Images Videos News Maps More Settings Tools

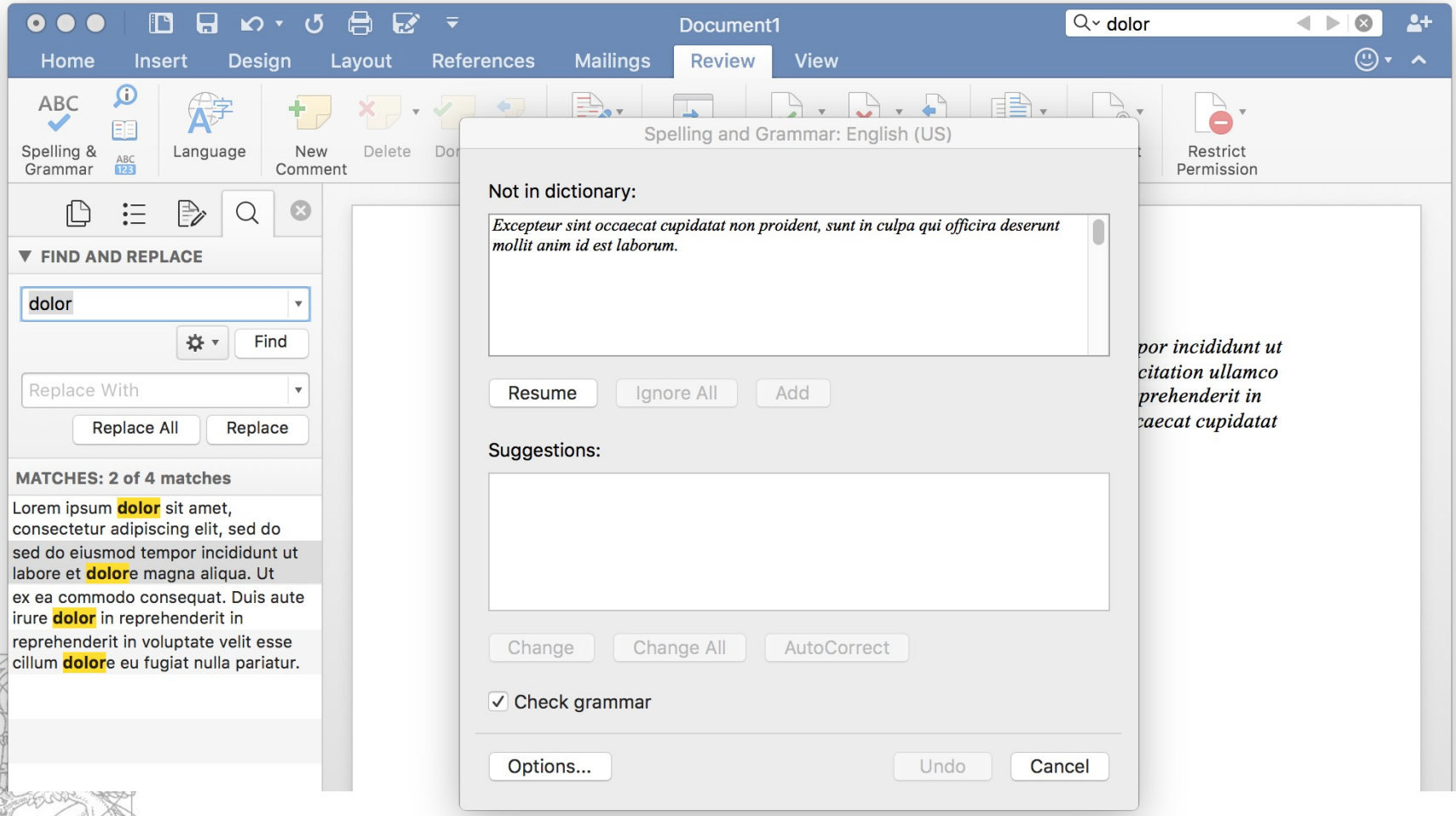
About 197,000 results (0.73 seconds)

Algoritmus – Wikipédia
https://hu.wikipedia.org/wiki/Algoritmus ▾ Translate this page
Valamivel később megjelent az első, **algoritmusok** hatékonyságát elemző matematikai cikk is; mely az euklideszi algoritmus időbonyolultságát vizsgálta.
Turing-gépek és ... · Tulajdonságok · Algoritmusok elemzése · Típusok és példák

Az algoritmusok típusai - Sulinet Tudásbázis
tudasbazis.sulinet.hu › Tudásbázis › Informatika › Informatika ▾ Translate this page
Egyszerű **algoritmusok**. Az algoritmusban megadjuk egy feladat véges számú lépésben végrehajtható megoldását. A lineáris, vagyis egyszerű algoritmus, elemi ...

Forrás: https://www.coursera.org/learn/algorithms-on-strings/lecture/avHa3/welcome

Sztringalgoritmusok



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Sztringalgoritmusok

Kompare

File Difference Settings Help

Source Folder	Destination Folder	Source File	Destination File
/home/robamler/Documents/	/home/robamler/Documents/	hitchhiker-040624	hitchhiker-current

Source Line	Destination Line	Difference
32	48	Changed 1 line
38	56	Changed 1 line
51	69	Changed 1 line
57	75	Inserted 2 lines
59	79	Deleted 1 line

hitchhiker-040624

```
52
53 [[Neil Gaiman]] has written ''[[Don't Panic: The Officia
54
55 A novel, ''[[Douglas Adams's Starship Titanic]]'', based
56
57 Two collaborative [[Internet]] projects were inspired by
58
59 ==Characters==
60
61 * [[Arthur Dent]]
62 * [[Ford Prefect (HHG)|Ford Prefect]]
63 * [[Trillian]]
64 * [[Zaphod Beeblebrox]]
65 * [[Marvin the Paranoid Android]]
66 * [[Vogon]]s
67 * [[Deep Thought]]
68 * [[Concepts and minor characters from The Hitchhiker's
69
70 ==See also==
71
72 * [[Places in The Hitchhikers Guide to the Galaxy|Places
73 * [[Infinite Improbability Drive]]
74 * [[Total Perspective Vortex]]
75 * [[The Answer to Life, the Universe, and Everything]]
76 * [[Dirk Gently]] -- another [[Douglas Adams]] creation
77 * [[SEP field]]
78 * [[Hitchhiking]]
79 * [[Sirius Cybernetics Corporation]]
80 * [[H2G2]]
81 * [[42 (number)|42]]
82
```

hitchhiker-current

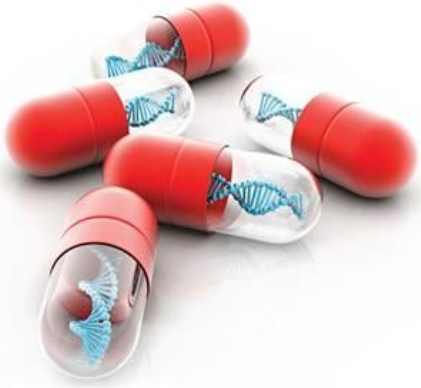
```
66 * ''[[So Long, and Thanks For All the Fish]]'' ([[1984]])
67 * ''[[Mostly Harmless]]'' ([[1992]])
68
69 A short story was also written, ''[[Young Zaphod Plays i
70
71 [[Neil Gaiman]] has written ''[[Don't Panic: The Officia
72
73 A novel, ''[[Douglas Adams's Starship Titanic]]'', based
74
75 Wowbagger the Infinitely Prolonged, a character from ''I
76
77 Two collaborative [[Internet]] projects were inspired by
78
79
80 {{HitchhikerBooks}}
81
82
83 ==See also==
84
85 * [[Dirk Gently]] -- another [[Douglas Adams]] creation
86 * [[Hitchhiking]]
87 * [[H2G2|h2g2]]
88
89 ==External links==
90
91 * [http://www.bbc.co.uk/cult/hitchhikers/ ''The Hitchhik
92 * [http://www.hitchhikersmovie.com ''The Hitchhiker's Gu
93 * The [http://www.cinescape.com/0/editorial.asp?obj_id=3
94 * [http://www.imdb.com/title/tt0371724/ The film's status
95 * [http://www.abovethetitle.com/content/programmes/hitchh
96
```

Comparing file file:/home/robamler/Documents/hitchhiker-...th file file:/home/robamler/Documents/hitchhiker-current | 1 of 23 differences, 0 applied | 1 of 1 file

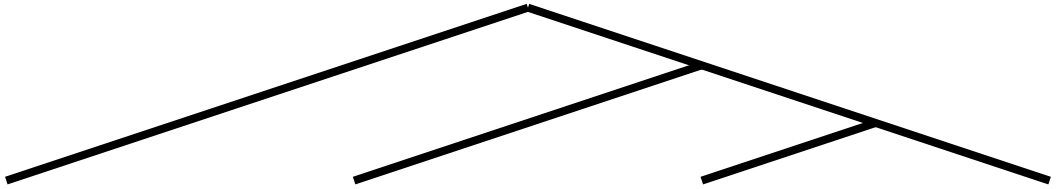
Bioinformatika

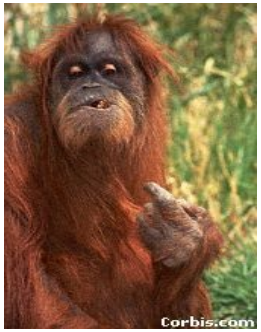
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Bioinformatika



- Személyre szabott orvoslás
- Génmódosított növények
- Filogenetika





Antivírusok és IDS

- Gyanús mintázatok keresése

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Leghosszabb közös részsorozat

2020. december

12. hét – 2. videó
S12E02

Astraptes fulgerator CELT
Astraptes fulgerator TRIGO
Bubo virginianus
Tyto alba

A leghosszabb közös részsorozat

- Két vagy több élőlény DNS-ének összehasonlítása
- bázisok: {A,C,G,T}
- pl ACCGGTCGAGTGCGCGGAAGCCGGCCGAA
- Két DNS tekinthető hasonlónak ha minél hosszabb közös részsorozattal rendelkeznek

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

A leghosszabb közös részsorozat

Adott két sorozat *X és Y*. Azt mondjuk, hogy egy *Z* sorozat ***közös részsorozatuk*** (nem feltétlen folytonos)**,** ha *Z* részsorozata *X*-nek is és *Y*-nak is.

pl:

X=(A,B,C,B,D,A)

Y=(B,D,C,A,B,A,A)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

A leghosszabb közös részsorozat

Adott két sorozat *X és Y*. Azt mondjuk, hogy egy *Z* sorozat ***közös részsorozatuk,*** ha *Z* részsorozata *X*-nek is és *Y*-nak is.

pl:

X=(A,**B**,**C**,B,D,**A**)

Y=(**B**,D,**C**,**A**,B,A,A)

egy közös részsorozat: (B,C,A)

A leghosszabb közös részsorozat

Adott két sorozat *X és Y*. Azt mondjuk, hogy egy *Z* sorozat ***közös részsorozatuk,*** ha *Z* részsorozata *X*-nek is és *Y*-nak is.

pl:

X=(A,**B**,**C**,**B**,D,**A**)

Y=(**B**,D,**C**,A,**B**,A,**A**)

egy közös részsorozat: (B,C,A)

egy leghosszabb közös részsorozat: (B,C,B,A)

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

A leghosszabb közös részsorozat

Bemenet: X = (x_1, x_2, . . . , x_m) és Y = (y_1, y_2, . . . , y_n) ahol minden x_i és y_i egy véges halmaz elemi

Feladat: Találjuk meg X és Y leghosszabb közös részsorozatát (LKR)

- Optimalizálási feladat
- Nyers erő módszere: $O(2^n)$
- Részfeladatokra bontás?

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Optimális részstruktúra:

15.1. tétel (az LKR optimális részstruktúrája). *Legyen* $X = (x_1, x_2, \ldots, x_m)$ *és* $Y = (y_1, y_2, \ldots, y_n)$ *két sorozat és* $Z = (z_1, z_2, \ldots, z_k)$ *ezek egy LKR-je. Ekkor igazak a következő állítások:*

1. *Ha* $x_m = y_n$, *akkor* $z_k = x_m = y_n$, *és* Z_{k-1} *az* X_{m-1} *és* Y_{n-1} *egy LKR-je.*
2. *Ha* $x_m \neq y_n$, *akkor* $z_k \neq x_m$ *esetén* Z *az* X_{m-1} *és* Y *egy LKR-je.*
3. *Ha* $x_m \neq y_n$, *akkor* $z_k \neq y_n$ *esetén* Z *az* X *és* Y_{n-1} *egy LKR-je.*

$$c[i,j] = \begin{cases} 0, & \text{ha } i = 0 \text{ vagy } j = 0, \\ c[i-1, j-1] + 1, & \text{ha } i, j > 0 \text{ és } x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\}, & \text{ha } i, j > 0 \text{ és } x_i \neq y_j. \end{cases}$$

Itt (ellentétben a pénzváltás ill. hátizsák problémával) nem kell minden részfeladatot megoldani!

Átfedő részfeladatok

$O(mn)$ részfeladat

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

LKR DP megoldása

```
LKR-HOSSZ(X, Y)
m ← hossz[X]
n ← hossz[Y]
for i ← 1 to m
    do c[i, 0] ← 0
for j ← 0 to n
    do c[0, j] ← 0
for i ← 1 to m
    do for j ← 1 to n
            do if x_i = y_j
                  then c[i, j] ← c[i − 1, j − 1] + 1
                       b[i, j] ← „↖”
                  else if c[i − 1, j] ≥ c[i, j − 1]
                          then c[i, j] ← c[i − 1, j]
                               b[i, j] ← „↑”
                          else c[i, j] ← c[i, j − 1]
                               b[i, j] ← „←”
return c és b
```

backpointerek

futásidő: $O(nm)$

i	j	0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

15.6. ábra. Az $X = (A, B, C, B, D, A, B)$ és $Y = (B, D, C, A, B, A)$ esetére az LKR-hossz által meghatározott c és b tömb. Az i sor és j oszlop kereszteződésében álló négyzet tartalmazza $c[i, j]$ értékét és a $b[i, j]$ értékének megfelelő nyilat. A tömb jobb alsó sarkában lévő $c[7, 6]$-ban található 4 érték X és Y (B, C, B, A) LKR-jének a hossza. Ha $i, j > 0$, akkor $c[i, j]$ értéke csak a már korábban kiszámított $c[i-1, j]$, $c[i, j-1]$ és $c[i-1, j-1]$ értékétől, valamint attól függ, hogy igaz-e az $x_i = y_j$ egyenlőség. Az LKR elemeinek előállításához a jobb alsó sarokból indulva kell követni a $b[i, j]$ nyilakat. A ténylegesen bejárt út satírozva látható. Az útban minden „↖” nyíl az LKR egy elemének felel meg, amire teljesül, hogy $x_i = y_j$, és ez az ábrán kiemelten látható.

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

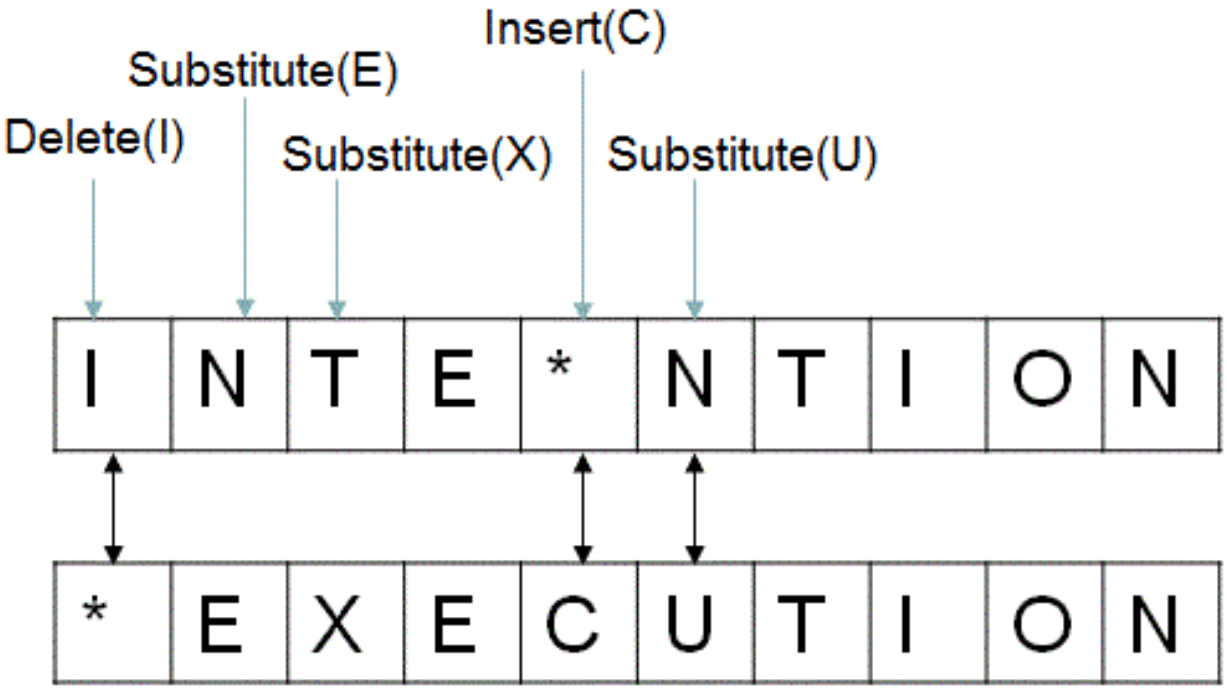
Szerkesztési távolság

2020. december

12. hét – 3. videó
S12E03

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Szerkesztési távolság



SZEGEDI TUDOMÁNYEGYETEM
UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Szerkesztési távolság

beszúrás költsége

törlés költsége

$$d(S_1, S_2) = D(n, m) = \begin{cases} 0 & ; \quad i = 0,\ j = 0 \\ i & ; \quad j = 0,\ i > 0 \\ j & ; \quad i = 0,\ j > 0 \\ \min(& \\ \quad D(i, j-1) + 1 & \\ \quad D(i-1, j) + 1 & ; \quad j > 0,\ i > 0 \\ \quad D(i-1, j-1) + m(S_1[i], S_2[j]) & \\ ) & \end{cases}$$

csere költsége

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Sztring hasonlósági feladat

Két módszer és algoritmus két sztring hasonlóságának kiszámítására:

- leghosszabb közös részsorozat
  elterjedt a bioinformatikában
- szerkesztési távolság
  elterjedt a természetes nyelvi szöveg feldolgozásban

SZEGEDI TUDOMÁNYEGYETEM

UNIVERSITAS SCIENTIARUM SZEGEDIENSIS

Algoritmusok és Adatszerkezetek I.

Mintaillesztési feladat

2020. december

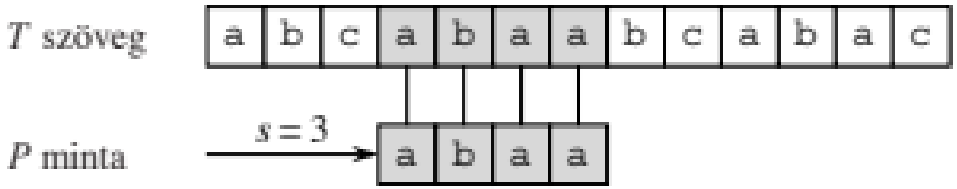
12. hét – 4. videó
S12E04

Mintaillesztés (pattern matching)

Bemenet: T[1…n] szöveg (text) és P[1...m] minta (pattern)

Kimenet: P összes illeszkedési helye T-ben (érvényes eltolása), 0≤s≤n-m

illeszkedés: T[s+1…s+m]=P[1…m]



UNIVERSITAS SCIENTIARUM SZEGEDIENSIS
SZEGEDI TUDOMÁNYEGYETEM

Egyszerű mintaillesztő (brute force)

össz futásidő: $O((n-m+1)m)$

EGYSZERŰ-MINTAILLESZTŐ(T, P)

$n \leftarrow hossz[T]$
$m \leftarrow hossz[P]$
for $s \leftarrow 0$ **to** $n - m$
  do if $P[1..m] = T[s+1..s+m]$
    then print „A minta illeszkedik az (”$s + 1$„)-edik pozícióra”

$O(m)$

http://whocouldthat.be/visualizing-string-matching/