előző órákon:

rendezési algoritmusok

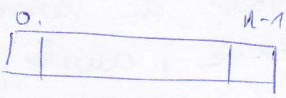
Minden összehasonlítás alapú algoritmus legalább $n \log(n)$ összehasonlítást használ.

összefésüléses $n \log(n)$
halmaz
kiválasztásos
⋮

Ebben az értelemben az összefésüléses rendezés

Leszámlálásos rendezés

Speciális input: T



különböző egész számok $0 \dots m-1$

$T[i] \in \{0, \dots, m-1\}$ $i = 0, \dots, n-1$

Algo: 1) m méretű B tömb létrehozása $\emptyset$ kell feltölteni

2) for $i = 0$ to $n-1$:

$B[T[i]] := 1$

3.) for $j = 0$ to $m-1$:

if $B[j] == 1$:

print(j)

LSz : 1.) $O(1)$ esetleg $O(m)$ } $O(n+m)$

2.) $O(n)$

3.) $O(m)$

$LSz_1 \leq c_1 \cdot m$

$LSz_2 \leq c_2 \cdot m$

$LSz = LSz_1 + LSz_2 \leq c_1 \cdot n + c_2 \cdot m \leq \underbrace{\max(c_1, c_2)}_{c_3} \cdot (n+m)$

$LSz \leq c_3 \cdot (n+m)$

$O(n+m)$

Tétel:

Kiíráskor a sorrendet I indexeinek sorrendjében történik, amik pedig rendezettek.

megjegyzés: 1.) ez nem összehasonlításos alapú rendezés

2.) ha $m = O(n)$ pl.: $m = 1000n$

vagy m = konstans

akkor $O(n+m) = O(n)$

3.) csak akkor jó, ha m ismert és nem nagy

pl.: nem jó : IPv6 címek 128 bit

$m = 2^{128}$

$n = 20000$

Eddig: számok rendezése szerepelt előadáson

megj.: valójában szavakat, kódokat, kulcsokat, indexeket rendezünk, de ez ugyanaz mint a számok rendezése csak más az összehasonlító reláció

cél: objektumok tárolása során az egyik komponens ~~(kulcs)~~ ("kulcs") alapján keresni

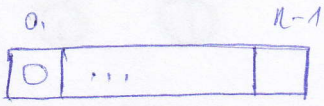
pl.: gyors keresés Neptun kód alapján:

1. ÖTL + bináris keresés

$O(n \log(n))$ $O(\log(n))$

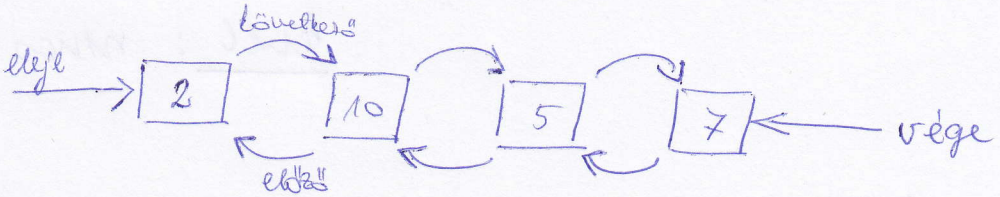
Kitérő: tárolási módok:

tömb: T



- 1 lépésben elérhető ($T[i]$) a tömb bármelyik indexű eleme
- beszúrás nehéz $O(n)$ lépés
- törlés: hasonlóan $O(n)$ lépés a mozgatás

2. Láncolt lista (kétirányú) struktúra

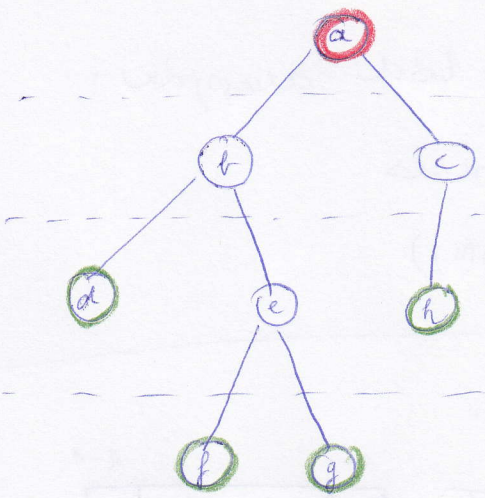


- beszúrás: gyors: 1 lépés
- keresés: lassú $O(n)$
- törlés: lassú, mert először keresni kell

Végső cél: keresés, beszúrás, törlés is gyors legyen!

⇝ bináris keresőfa ⇝ kiegyensúlyozott bináris keresőfa

Bináris fa:



GYÖKÉR 1.

2.

LEVELEK 3.

4.

fogalmak: gyökér

szintek

minden csúcsnak legfeljebb 2 gyereke van

bal gyerek

jobb gyerek

levél: nincs gyereke

Az adatszerkezetes megvalósítás:

tárolás mutatókkal (pointer $\S$, referencia)

- van 1 db gyökér mutató
- minden csúcsnál: - bal_gyerek
  - jobb_gyerek
  - Apa

  } mutató

pl.: Apa(b) = a

b.gy(f) = Null

j.gy(c) = g

MEGOLDANDÓ FELADATOK:

1.) Bejárás a fa összes csúcsának egyszeri végiglátogatása.

Lehetőségek:

a) Preorder bejárás (rekurzív)

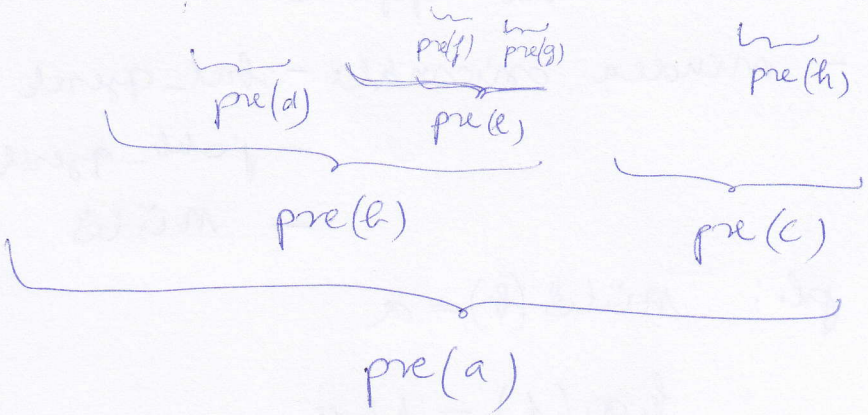
Input: bináris fa x csúcsa

Cél: x-nél gyökerező fa rész bejárása

```
pre(x):
    látogat(x)
    pre(bal_gyerek(x))   ha van b.gy.
    pre(jobb_gyerek(x))  ha van j.gy.
```

pl.: pre(a):

a, b, d, e, f, g, c, h



b) Inorder bejárás (rekurzív)

Input: x csúcs

Cél: x gyökerű részfa bejárása

in(x):

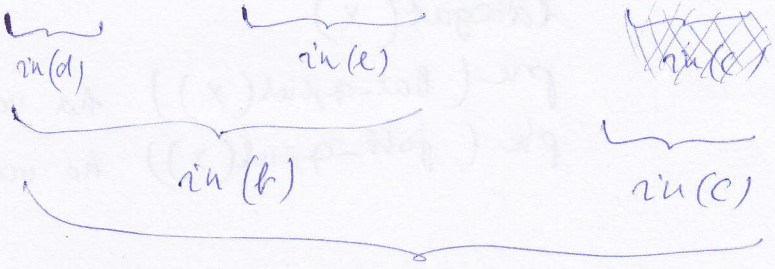
in(bal_ágyerek(x)) ha van b. gy.

feldolgozat(x)

in(jobb_ágyerek(x)) ha van j. gy.

pl.: in(a):

d b f e g a h c



C) Postorder bejárás (rekurzív)

Input: x csúcs

Cél: x gyökerű részfa bejárása

```
post(x):
    post(bal-gyerek(x))
    post(jobb-gyerek(x))
    látogat(x)
```

pl.: post(a):

a f g e b h c a

T_2: pre(x)

egy élnél: 3 lépés → látogat(x)
→ van-e bal gyerek
→ van-e jobb gyerek
} konstans lépés csúcsonként

⇝ O(1) x-enként

ha a csúcsok száma: n

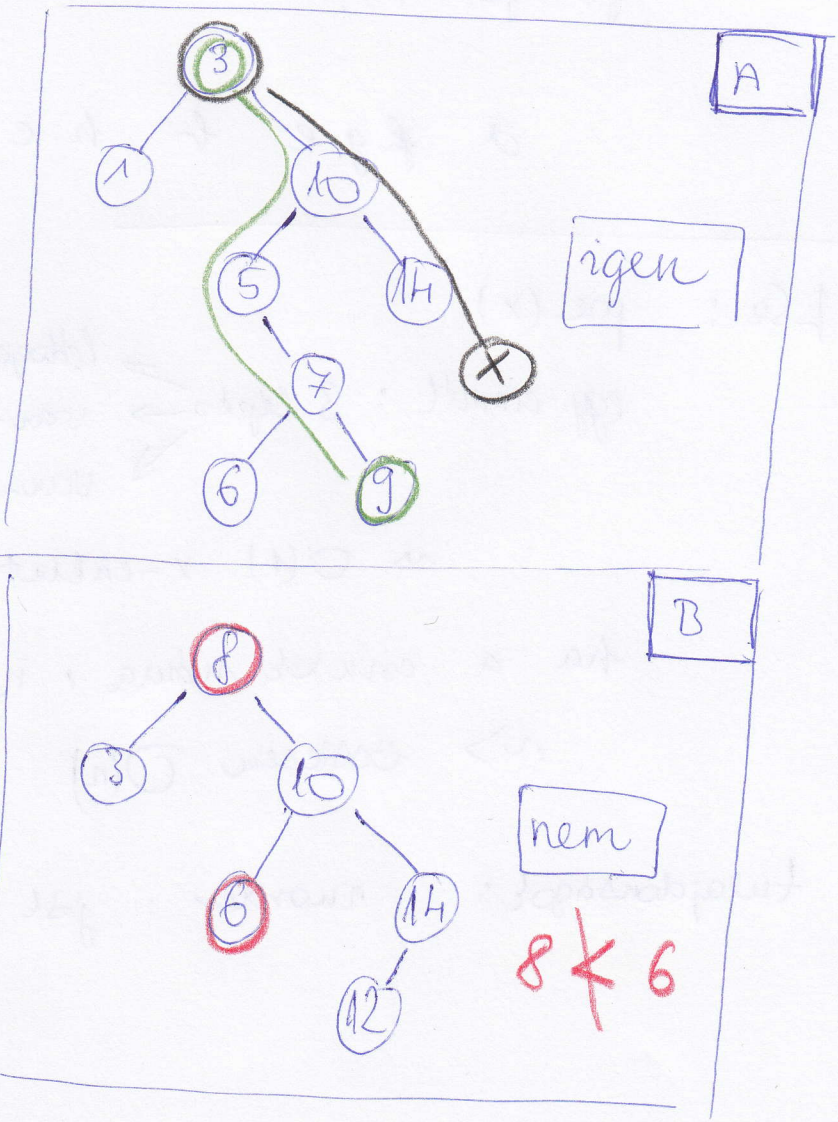
⇝ összesen O(n)

tulajdonságok: inorder: jól rendez

Bináris keresőfa:

Bináris fa melynek csúcsaiban számok vannak úgy, hogy ha egy csúcsnak van b. gy.-e akkor a b. gy. gyökerű részfában az eredeti csúcsnál ~~nagy~~ kisebb értékű csúcsok vannak és ~~hasonlóan~~ ha van az adott csúcsnak j. gy.-e akkor az onnan keresztül elérhető csúcsok nagyobb számot tartalmaznak

pl.:



pl.: „A": bináris keresőfában

keresés:

Keres(9): keresési út:

3, 10, 5, 7, 9

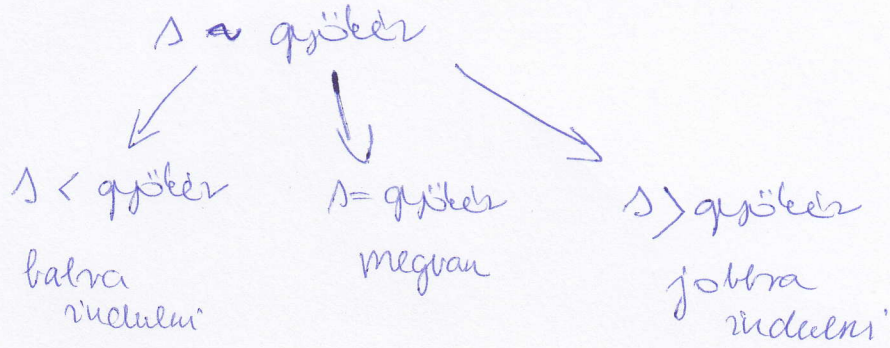
LSz: O(mintek száma)

Keres(17): keresési út:

3, 10, 14 → nincs!

LSz: O(mintek száma)

Keres(s) bináris keresőfában: O(mintek száma)



Beszúrás(s) bináris keresőfában: O(mintek száma)

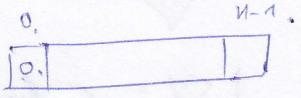
~~Keresés~~

Keres(s): nincs ott akit kerni'e kellene
és ott beszúrás

Beszúrásos rendezés:

Input: T tömb

Output: T elemei rendezve



ZH:
feladatmegoldás
kb. a gyakorlat feladatainak nehézsége
pl.: algoritmus leírása + időszag + LSz

for i=0 to n-1:

Ⓐ bináris keresés T[i]-re

Ⓑ mozgatás arra a helyre:
(jobbra csúszik a többi elem)

Lépésszám: Ⓐ $O(\log(i))$ lépés

Ⓑ $\leq i$ lépés

Összes lépés: Ⓐ $\underbrace{\log 1 + \log 2 + \dots + \log n}_{O(n \cdot \log(n))}$

Ⓑ $\underbrace{1 + 2 + \dots + n-1 = \frac{n(n-1)}{2}}_{O(n^2)}$

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

$O(n \log n)$: $LSz_1 \leq c_1 \cdot n \log n$

$O(n^2)$: $LSz_2 \leq c_2 n^2$ } +

$$LSz \leq c_1 \cdot n \cdot \log n + c_2 \cdot n^2$$

$$LSz \leq c_1 \cdot n \cdot \underbrace{\log_2(n)}_{\log n \leq n} + c_2 \cdot n^2 \leq c_1 n^2 + c_2 n^2 = (c_1 + c_2) n^2$$

$$LSz \leq c_3 \cdot n^2$$

$$\Downarrow$$

$$LSz : O(n^2)$$

3. feladatsor / 4. feladat:

[redacted] dtrendezés $O(n \log n)$

A: $a_0, a_1, a_2, \ldots, a_{n-1} \longrightarrow a_{i_0} < a_{i_1} > a_{i_2} < a_{i_3}$

1. lépés: eredeti T rendezése összefésüléses rendezéssel

$B = \text{ÖFR}(A)$

$b_1, b_2, b_3, b_4, \ldots, b_{n-1}$

if $b_1 < b_2$:
    pass
else:
    $b_1 \rightleftarrows b_2$

$\Big\} \; b_1 < b_2$

if $b_2 < b_3$
    $b_2 \rightleftarrows b_3$
else
    pass

```
for i=0 to n-2:
    if i % 2 = 0
        if b_i < b_{i+1}: pass
        else: tmp = b_i; b_i = b_{i+1}; b_{i+1} = tmp;
    else:
        if b_i > b_{i+1}: pass
        else: tmp = b_i; b_i = b_{i+1}; b_{i+1} = tmp;
```