IV. Keresések

Keresések

1. A kereső fa
2. Az AVL-fa
3. A 2-3 fa
4. A B-fa
5. Hasításos technikák (hash-elés) (második félévben)

Keresések

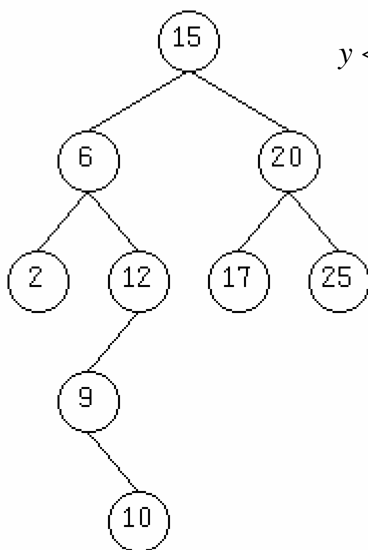
Sok adat esetén milyen adatszerkezetben lehet hatékonyan keresni, módosítani, beszúrni és törölni, stb.?
A gyakorlat szerint: fákban és táblázatokban.
Ha egy alkalmas adatszerkezetre értelmezve vannak a következők: Keres, Beszúr, Töröl, (Tól-ig), akkor ezt az adatszerkezetet szótárnak (dictionary) nevezik.
Ha ezen kívül még a Minimum, Maximum, Rákövetkező, Előző műveletek is értelmezettek az adott adatszerkezeten, akkor azt prioritási sornak nevezik. Ezzel lehet rendezni.

Adatok (a struktúrában): kulcs+mezők (rekordok)
Könnyítés: - csupa különböző kulcs van
-ahol nem lényeges, ott csak kulcs van

15. Keresőfák (=rendezőfák)

Definíció: t bármely x csúcsára és bal(x) bármely y csúcsára és jobb(x) bármely z csúcsára:ű
$y \leq x \leq z$, azaz $y < x < z$ (könnyítés).

Pl.:



$y < x < z$

A definíció ilyen erő, hogy egy elemet egyértelműen lehessen megkeresni. Minden csúcsban arról döntünk, hogy balra vagy jobbra menjünk.

(Fontos) tulajdonság: inorder bejárással rendezett sorozatot kapunk.

A felírt műveletek bevezetése
A műveleteknek összhangban kell lenni (arculatfilozófia…).
Egy elemnek az értékét vagy a címét használhatjuk-e? [cím gyakorlatban: pointer vagy index]

Pl.: Keres(t, 12) vagy Keres(t, p), ahol p→ 12

Mit használjuk: függvényt vagy eljárást?
Megtalált elem esetén az értéket adjuk vissza vagy a címet? Nagy rekordok esetén inkább a címet.
Ábrázolási kérdés: szülőre mutató pointer alkalmazása.

Lehetőségek:
I. Minden kulcs különböző
II. Lehetnek azonos kulcsok

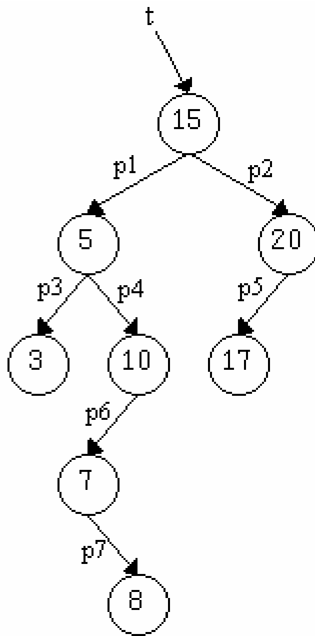
A. Rekordok: $(k, m_{1,\dots,} m_n)$
B. Skalárok: k
[I.-t és A.-t választottuk]

Műveletek:

Keresés p:= Keres(t, k)

A t keresőfájában keresi a k kulcsú elemet (csúcsot); ha ez létezik, akkor p. a címe, egyébként NIL adódik vissza.

Pl.:



p:= Keres(t, 10) $\Rightarrow$ $p = p_4$
p:= Keres(t, 9) $\Rightarrow$ $p = NIL$

Maximum p:= Max(t) Q: $t \neq NIL$

A t-ben maximális kulcsú elem címét adja vissza (Ha t=NIL, akkor p=NIL).

Pl.: p:= Max(t) $\Rightarrow$ $p = p_2$

Minimum p:= Min(t)

(a Maximumhoz hasonlóan)

Pl.: p:= Min(t) $\Rightarrow$ $p = p_3$

Következő r:= Következő(t, p)

A t-ben p címen kulcsérték rákövetkezőjének címét adja vissza, illetve NIL-t, ha ilyen nincs.

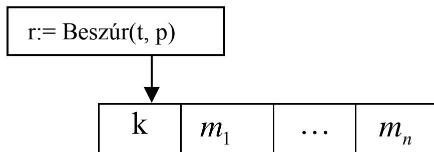
Pl.: r:= Következő(t, p_1) $\Rightarrow$ $r = p_6$ (5 $\Rightarrow$ 7)
r:= Következő(t, p_7) $\Rightarrow$ $r = p_4$ (8 $\Rightarrow$ 10)
r:= Következő(t, p_2) $\Rightarrow$ $r = NIL$

Előző r:= Előző(t, p)

(a Következőhöz hasonlóan)

Beszúrás

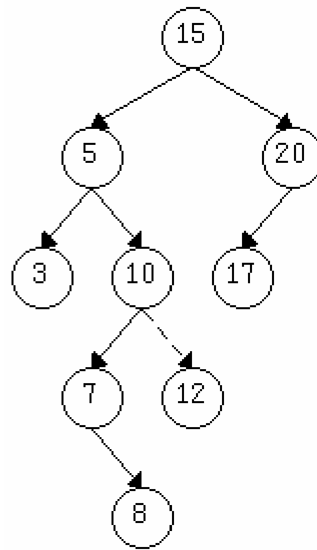
(1) Ha garantált, hogy a beszúrni kívánt kulcs még nem létezik t-ben, akkor



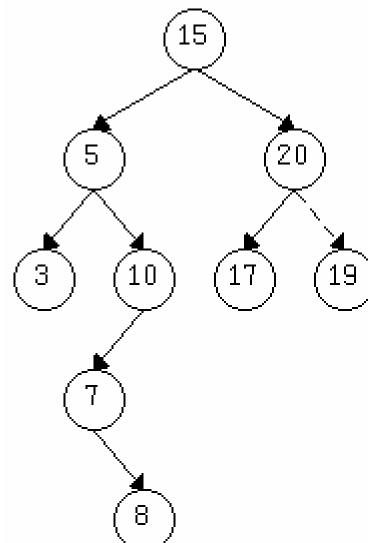
(2) Ha t-ben előfordulhat a k kulcs, akkor r:= Beszúr(t, p)

Sikeres beszúrás esetén: r = p
Sikertelen beszúrás esetén: r = NIL

Pl.: p → | 12 | … | r:= Beszúr(t, p) ⇒



p → | 19 | … | r:= Beszúr(t, p) ⇒



p→ | 7 | … | r:= Beszúr(t, p) ⇒ r = NIL

Törlés   r:= Töröl(t, p)

Fontos! Ez egy logikai szintű parancs: a p címen lévő elem kulcsa (és tartalma) az, amit törölni akarunk. Tehát nem a p című memóriadarabot akarjuk eltávolítani t-ből.

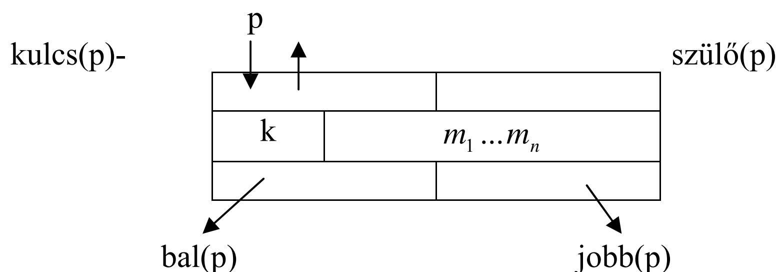
Pl.: (I.) r:= Töröl(t, p_3) ⇒ p_3^-nak nincs gyereke, ezért egyszerűen törölhető
(5 elem baloldali pointerét NIL-re cseréljük)
r = p_3

(II.) r:= Töröl(t, p_2) ⇒ p_2^-nak egy gyereke van, ezért úgy törölhető, hogy összekapcsoljuk a szülőjét és a gyerekét.
r = p_2

(III.) r:= Töröl(t, p_1), de tegyük fel, hogy a p_3^ elem (3) a fában van!
⇒ a p_1^-nak két gyereke van. Átszervezzük a fát, pl. a jobb részfát.
Kivágjuk a jobb részfából a minimumot, ennek legfeljebb jobb gyereke van, így 1-es vagy 2-es típusú törlés, majd a minimum (7) tartalmát a p_1^-ba írjuk.
r = p_6 !!!

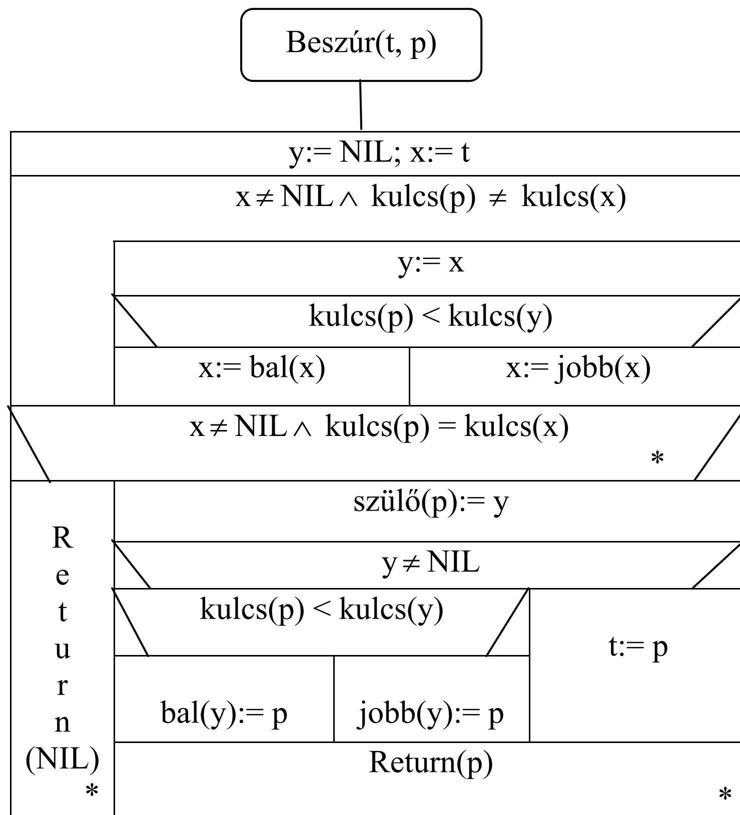
A műveleteket megírhatjuk:

(i) ADT (ne!)
(ii) ADS (lehetne, de úgyis pointerekre gondolunk)
(iii) Reprezentáció szintjén, pointerekkel



Az algoritmusokat megírhatjuk:

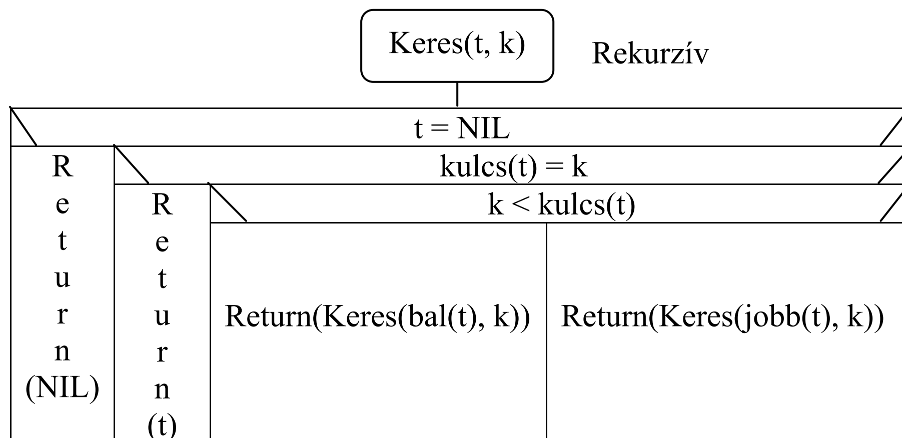
a) rekurzív módon
b) iteratív módon

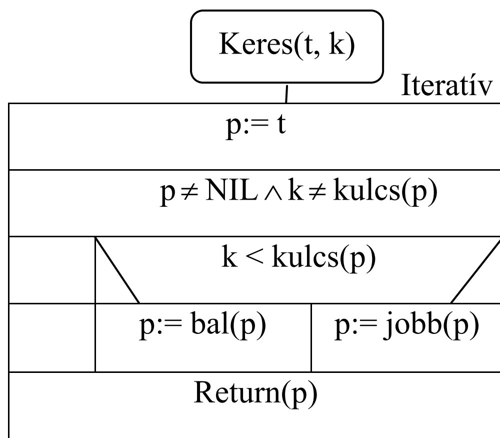


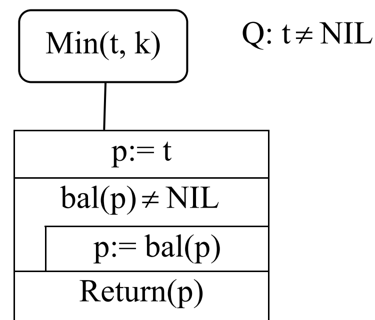
gyökérelem és szülő pointere

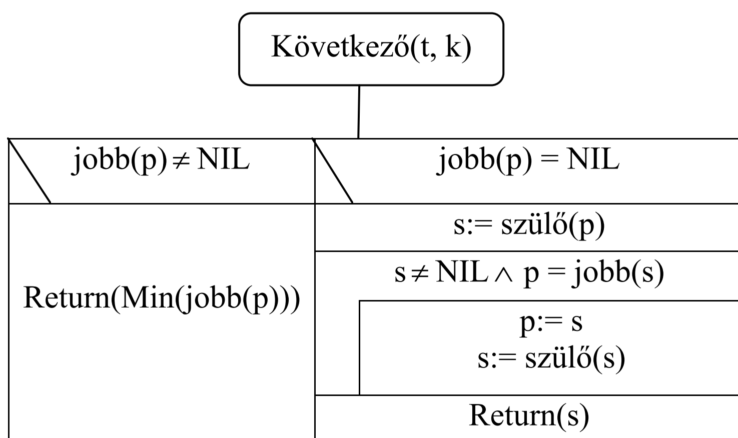
Ha nincs jobb oldali részfa, akkor fel kell menni a jobb gyerek pointereken, mert ezek kisebb értékek. Venni kell e legfelső ilyen elem szülőjét, amelynek az a bal gyereke; ebben már nagyobb érték van, mégpedig a következő legnagyobb.
← A jó esetben y mutat a beszúrás helyét jelző levélre, x pedig NIL.

A *-gal megjelölt részek csak akkor kellenek, ha számolunk a hibalehetőséggel.

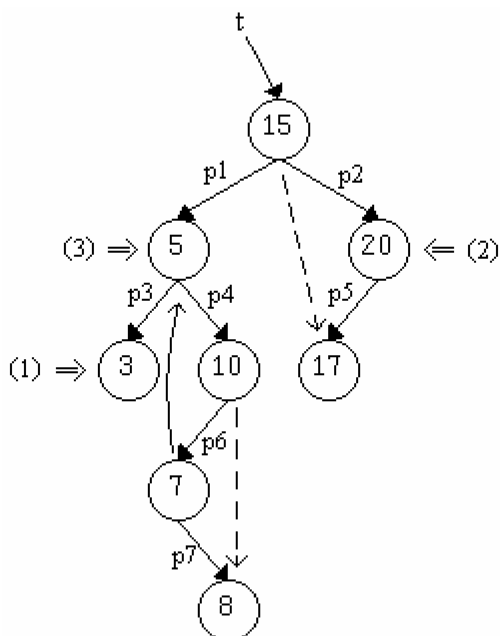






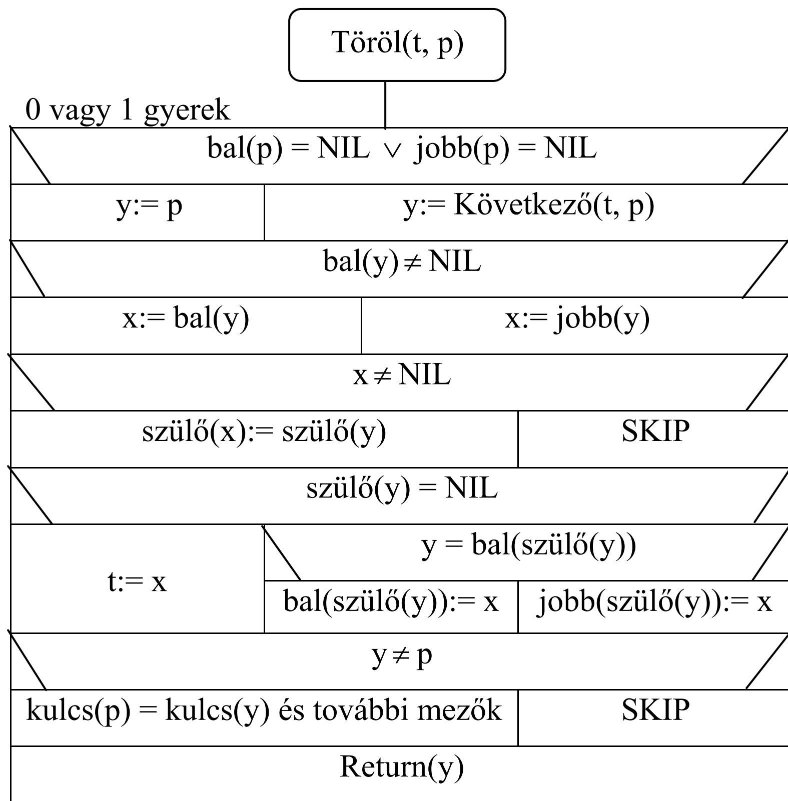


Törlés



Pl.: (1) r:= Töröl(t, p_3)
(2) r:= Töröl(t, p_2)
(3) r:= Töröl(t, p_1)

(tegyük fel, hogy (3) létezik)



2 gyerek
y fizikailag törlendő elemre mutat
y lehet levél; lehet csak jobb gyereke: eleve vagy a jobb részfa minimumaként; lehet csak bal gyereke

x az y 0 vagy 1 gyerekére mutat

bal(y)=NIL ∧ jobb(y) ≠ NIL	bal(y) ≠ NIL	jobb(y) ≠ NIL
x:= NIL	x:= bal(y)	x:= jobb(y)

Ha y-nak volt (egy) gyereke, akkor befűzzük azt a szülője alá

Az y szülőjének a megfelelő oldali mutatóját x-re állítjuk

Ha a logikailag törlendő elem ≠ a fizikailag törlendő elemmel, akkor az előbbit felírjuk
Az y^ elem újrafelhasználható

Elképzelhető, hogy a gyökeret töröljük, ekkor x^ lesz a gyökér.

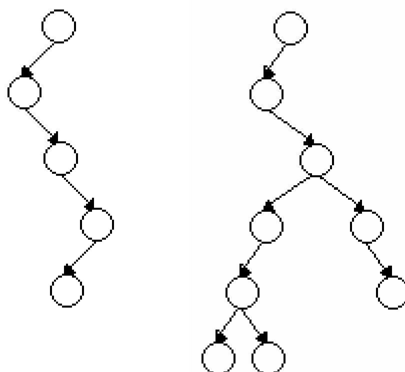
Mennyire hatékony adatszerkezet a bináris keresőfa?
Mindegyik művelet egy útvonal bejárását jelenti a fában.

$T(t) = O(h(t))$, ahol $h(t)$ a fa magassága

$T(n)=?$

a) $MT(n) = \Theta(n)$ pl.:

általában: „kevés adat-
-hosszú utak”, azaz:
nem „tömör” fák
⇓
nem hatékony (?)



b) mT(n)

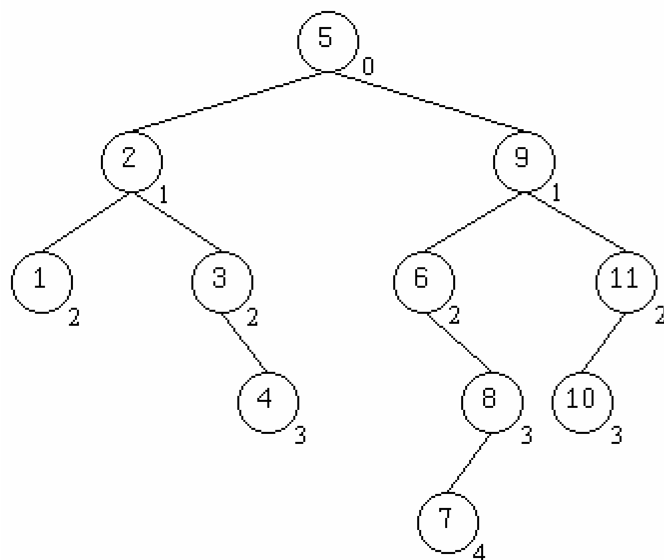
Ez nem az a triviális kérdés, hogy egy művelet a legjobb esetben hány összehasonlítást igényel (1-et), hanem: „tömör” fa esetén mi a T(n) ?
Szerencsés esetben t majdnem teljes, ekkor h(t) ~ $\log_2 n$

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

c) AT(n)

Tegyük fel, hogy a véletlen sorrendű 1, 2, …, n adatokból építjük fel a t keresőfát.
Várhatóan milyen magas lesz a fa?
Nehéz kérdés.
Válasz: $Ah(t(n)) = \Theta(\log n)$
Könnyebb kérdés: Mennyi az átlagos csúcsmagasság? = Hány összehasonlítással lehet felépíteni a t keresőfát átlagosan? (pontosan n-es szorzóval igaz az állítás)

Építsünk keresőfát: 5 2 9 3 11 1 6 10 8 4 7 =:p



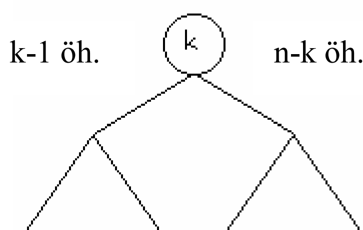
Ö(p) =
(1+1)+(2+2+2+2)+(3+3+3)+4=23

Az adatok más sorrendje esetén ettől eltérő érték adódhat.

Határozzuk meg ennek átlagát.
Jelölés: f(n), (n|k): először a k érték jön (k az input sorozat első eleme).
f(n) tehát azt adja, hogy n adatból hány összehasonítással lehet keresőfát építeni –átlagosan-, feltéve, hogy minden sorrend egyformán valószínű.

Tegyük fel, hogy minden sorozat $\frac{1}{10!}$ valószínűségű.

$f(n)=\frac{1}{n}\sum_{k=1}^{n} f(n \mid k)$



$$f(n) = \frac{1}{n}\sum_{k=1}^{n}(k-1+f(k-1)+(n-k)+f(n-k))$$

1) f(n-k) $\Downarrow$

n-k

$$\begin{cases} f(0)=0 \\ f(n)=(n-1)+\frac{2}{n}\sum_{k=1}^{n} f(k) \end{cases}$$

Ez ugyanaz, mint a QuickSort-nál.
f(n)< $2n\ln n \approx 1{,}39 n \log_2 n$

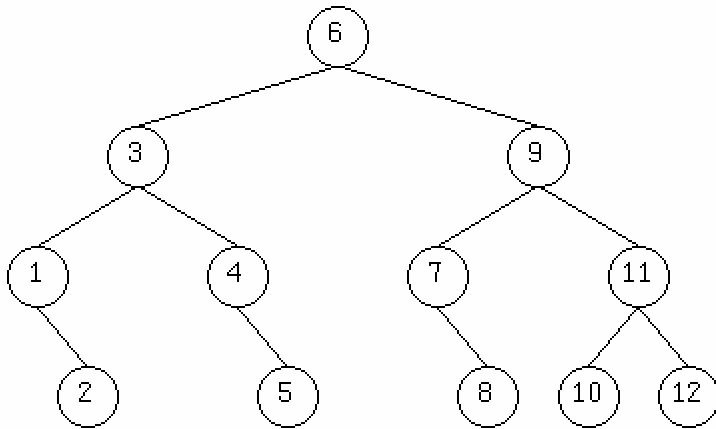
Tehát a t átlagos csúcsmagassága $1{,}39 n \log_2 n$.

Néhány megjegyzés:

1. Logaritmikus keresés $\left\lfloor \frac{i+j}{2} \right\rfloor$

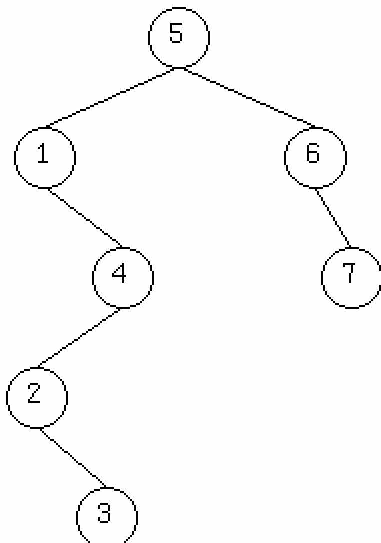
1	2	3	4	5	6	7	8	9	10	11	12

$\Downarrow$ speciális keresőfa



A keresés ezen a keresőfán algoritmikusan ugyanúgy működik, mint a logaritmikus keresés a fenti tömbön.

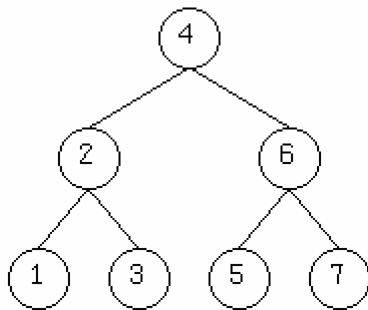
2. Adott egy t keresőfa. Egy elvi lehetőség a „tömörítésre”:



inorder bejárás $\Rightarrow$ A[1…7]

logaritmikus keresés szerinti t’ bináris fa ⇐

1	2	3	4	5	6	7



Ötlet: A „csúnya” fákat esetleg érdemes átrendezni
Természetesen nem ilyen kevéssé hatékony módon.