Algoritmuselmélet

A kupac adatstruktúra, bináris keresőfák

Katona Gyula Y.

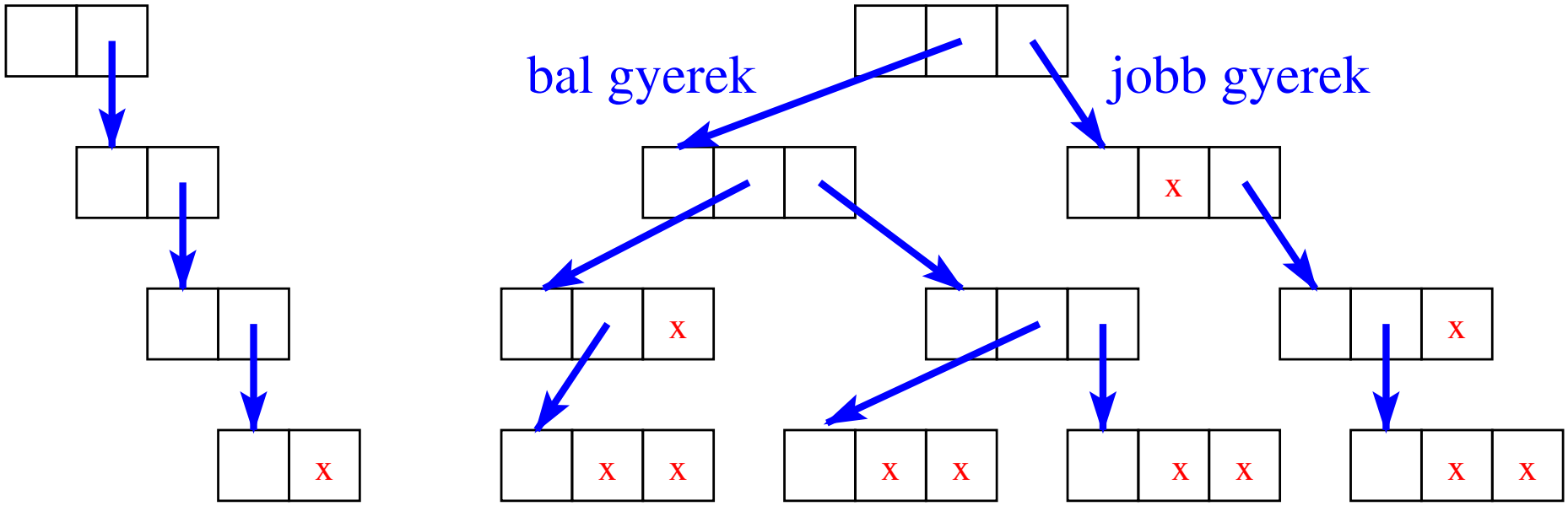
Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Adatstruktúrák

Műveletek lépésszáma különböző adatsrtuktúrákban

	beszúr	keres	min	max	mintör	maxtör	töröl	módosít	épít
tömb	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(n)$
rendezett tömb	$O(n)$	$O(\log n)$	$O(1)$	$O(1)$	$O(n)$	$O(1)$	$O(n)$	$O(n)$	$O(n \log n)$
láncolt lista	$O(1)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(1)$	$O(1)$	$O(n)$
kupac	$O(\log n)$	$O(n)$	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$	$O(\log n)$	$O(\log n)$	$O(n)$
piros-fekete fa	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(n \log n)$

Bináris fa adatszerkezet



Láncolt lista Bináris fa

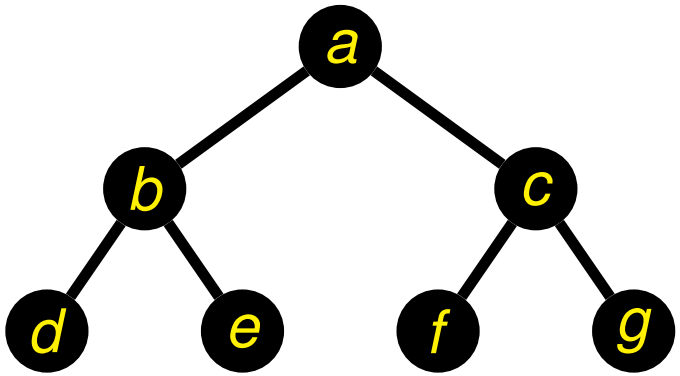
x: nincs további mutató

Alap műveletek: gyökér, bal-gyerek, jobb-gyerek
További műveletek: keres, beszúr, töröl

Bináris fa [Binary tree]

Volt Prog1-ből

Fa csúcsai $\rightarrow$ $elem(x), bal(x), jobb(x)$ esetleg
$szülő(x)$ és $reszfa(x)$ (=az x gyökerű részfa csúcsainak száma)



ha x a gyökér, y pedig a 9-es csúcs, akkor

$$\begin{aligned} bal(jobb(a)) &= f, \\ szülő(szülő(d)) &= a, \\ elem(bal(a)) &= b, \\ reszfa(a) &= 7, \\ reszfa(c) &= 3. \end{aligned}$$

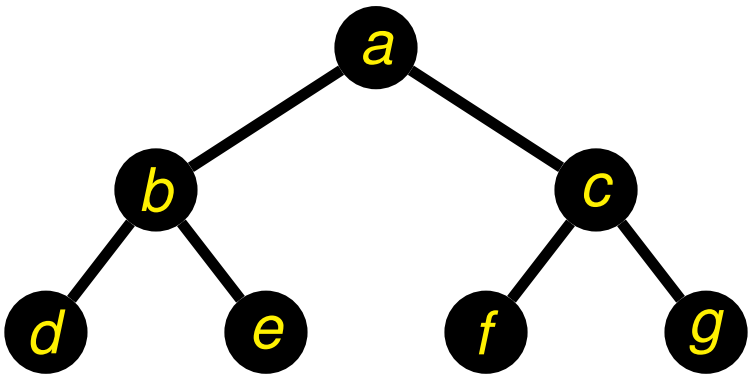
Bináris fa bejárásai

Tömbnél, láncolt listánál egyszerű a tárolt adatok felsorolása, a bináris fánál nem nyilvánvaló.

pre(x)
begin
látogat(x);
pre(*bal*(x));
pre(*jobb*(x))
end

in(x)
begin
in(*bal*(x));
látogat(x);
in(*jobb*(x))
end

post(x)
begin
post(*bal*(x));
post(*jobb*(x));
látogat(x)
end



PREORDER: *abdecfg*

INORDER: *dbeafcg*

POSTORDER: *debfgca*

Bináris fa bejárásainak lépésszáma

Tétel

A Bináris fa bejárásainak lépésszáma $O(n)$.

Bizonyítás.

Minden csúcsra 3 lépést végzek (valamilyen sorrendben):

1. Kiírom a csúcsát,
2. megnézem mi a bal gyereke,
3. megnézem mi a jobb gyereke.

Amikor nincs gyereke, vagy csak 1 gyereke van, akkor is 1 lépés ennek megnézése.
Tehát a lépések száma $3n \in O(n)$. □

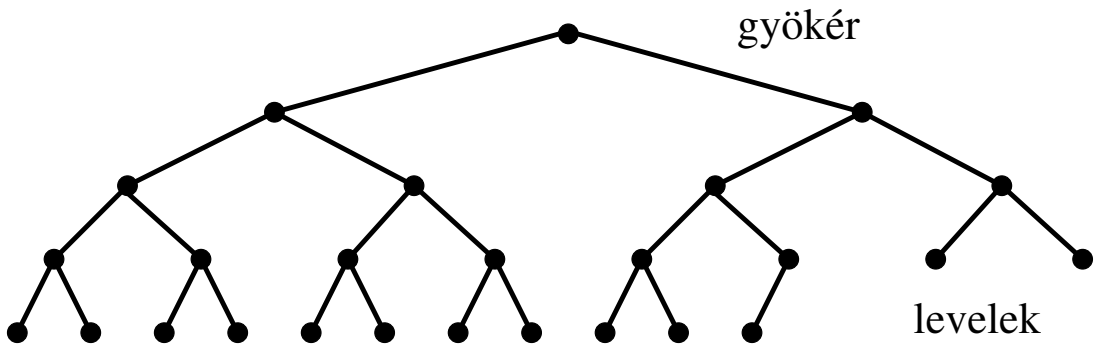
Kupac (Heap, Priority queue) adatszerkezet

Összehasonlítható elemek egy S véges részhalmazát szeretnénk tárolni, hogy a *beszúrás* és a minimális elem törlése (*mintör*) hatékony legyen. Feltesszük, hogy az elemek különbözőek.

Alkalmazások:

- Jobok indítása
- Gyors rendezési algoritmus
- Dijkstra gyorsabb megvalósítása

(majdnem) Teljes bináris fa:



A (majdnem) teljes fák levelei legfeljebb 2 szinten, a legalsón és esetleg a felette levőn helyezkednek el, és legfeljebb egy kivétellel minden belső csúcsuknak 2 fia van.

(A legalsó szint balra van „tömörítve”.)

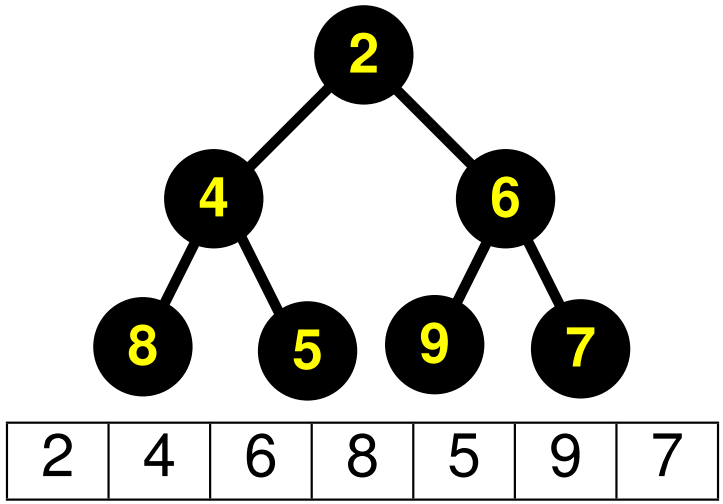
Bináris fa ábrázolása tömbbel, kupac definíciója

A fa csúcsai az $A[1 : n]$ tömb elemei.

(Itt most fontos, hogy az indexelés 1-től kezdődik.)

Az $A[i]$ csúcs bal fia $A[2i]$, a jobb fia pedig $A[2i + 1]$.

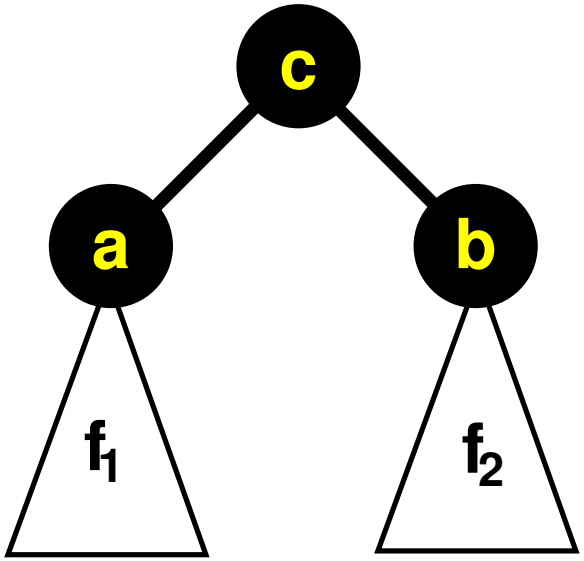
$\Longrightarrow$ $A[j]$ csúcs apja $A[\lfloor j/2 \rfloor]$



Definíció (Kupac)

A (majdnem) teljes bináris fában tárolt elemek kupacot alkotnak, ha minden csúcsra teljesül a kupac tulajdonság:
a szülőben tárolt elem $<$ gyerekekben tárolt elemek.

Kupacépítés



f_1 és f_2 teljesíti a kupac tulajdonságot

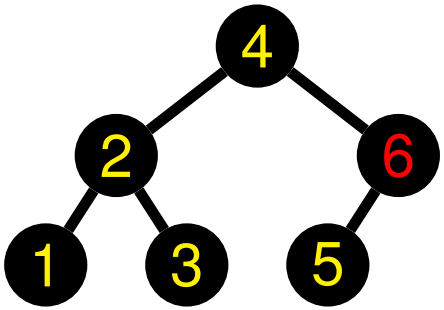
kupacol(f)
{ Ha $\min\{a, b\} < c$, akkor $\min\{a, b\}$ és c helyet cserél
Ha a c elem a-val cserélt helyet, akkor *kupacol*(f_1), ha b-vel, akkor *kupacol*(f_2) }

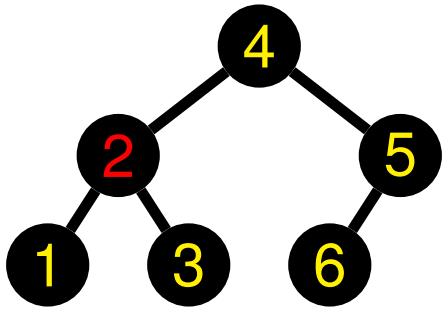
c addig megy lefelé, amíg sérti a kupac tulajdonságot.
Lépésszám: Ha ℓ a fa szintjeinek száma, akkor $\leq \ell - 1$ csere és $\leq 2(\ell - 1)$ összehasonlítás

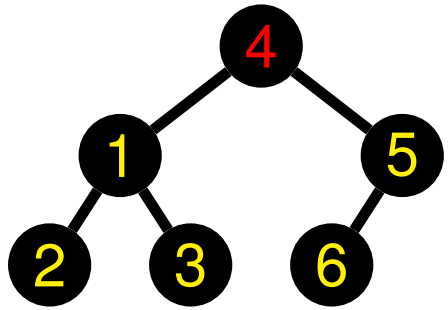
kupacépítés(f)
{ Az f fa v csúcsaira lentről felfelé, jobbról balra *kupacol*(v). }

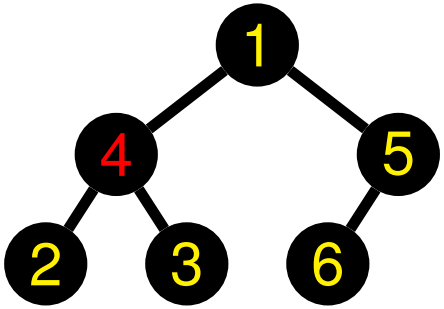
Az előbbi sorrend a kupac tömb reprezentációjában épp a jobbról balra sorrendet jelenti.

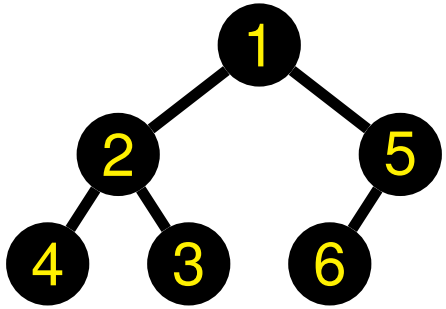
Kupacépítés példa











Kupac mélysége

(majdnem) teljes bináris fában:

1. szint: 1 pont
2. szint: 2 pont
3. szint: 2^2 pont

$\vdots$

$\ell - 1$-edik szint: $2^{\ell-2}$ pont

ℓ-edik szint: ≥ 1 és $\leq 2^{\ell-1}$ pont

$\implies n \geq 1 + \sum_{i=0}^{\ell-2} 2^i = 2^{\ell-1} \implies \ell \leq 1 + \log_2 n$

Tétel

Kupac mélysége: $O(\log n)$

Kupacépítés költsége

Lemma

$$\sum_{j=1}^{\ell-1} \frac{j}{2^j} < 2$$

$$\begin{array}{cccccc}
\frac{1}{2} & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & \\
\frac{1}{8} & \frac{1}{8} & \frac{1}{8} & & & \\
\vdots & & & & & \\
\frac{1}{2^{l-1}} & \frac{1}{2^{\ell-1}} & \frac{1}{2^{\ell-1}} & \cdots & \frac{1}{2^{\ell-1}} & \\
\hline
< 1 & < \frac{1}{2} & < \frac{1}{4} & \cdots & \leq \frac{1}{2^{\ell-1}} & \Leftarrow < 2
\end{array}$$

A kupacépítés költsége

Az i-edik szinten levő v csúcsra *kupacol*(v) költsége legfeljebb $\ell - i$ csere és legfeljebb $2(\ell - i)$ összehasonlítás.
A cserék száma ezért összesen legfeljebb

$$\ell - 1 + 2^1(\ell - 2) + 2^2(\ell - 3) + \ldots + 2^{\ell-2} \cdot 1 = \sum_{i=1}^{\ell-1} (\ell - i)2^{i-1}.$$

$\ell - i = j$ (azaz $i = \ell - j$) helyettesítéssel

$$\sum_{j=1}^{\ell-1} j2^{\ell-j-1} = \sum_{j=1}^{\ell-1} j\frac{2^{\ell-1}}{2^j} = 2^{\ell-1} \sum_{j=1}^{\ell-1} \frac{j}{2^j} < 2^{\ell} \leq 2n.$$

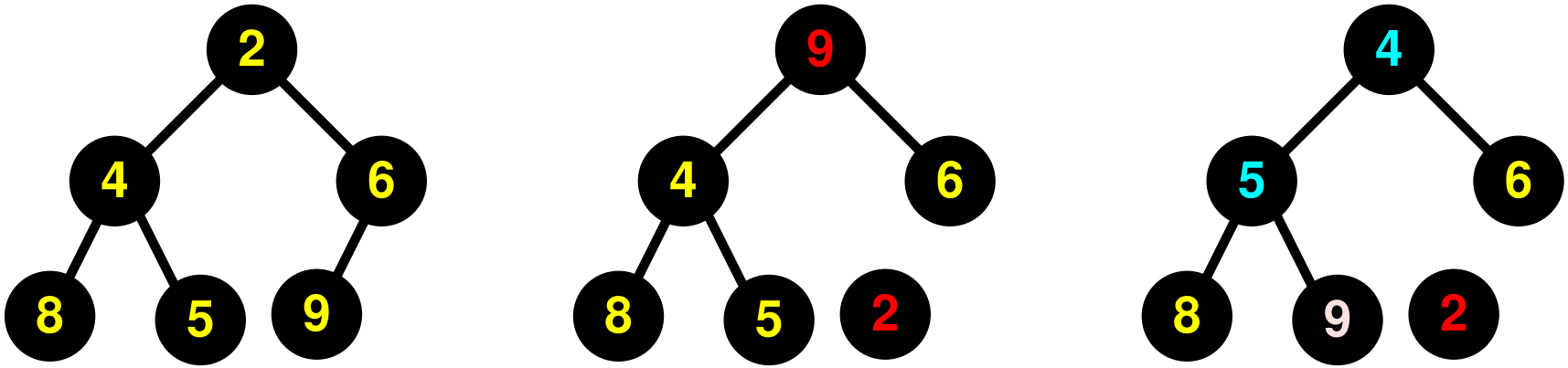
Összes művelet: $\leq 6n$

Tétel

A kupacépítés költsége: $O(n)$

MINTÖR

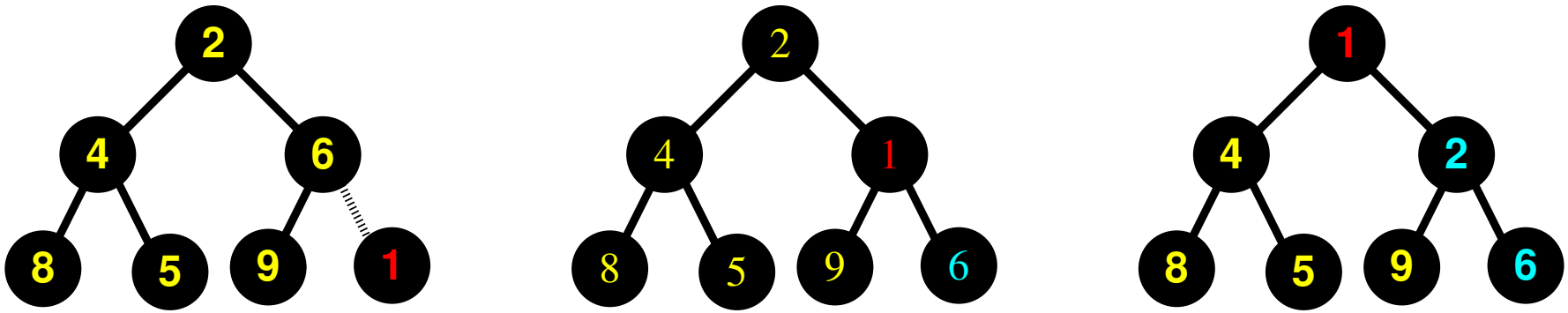
A minimális elem az f gyökerében van, ezt töröljük.
Helyére tesszük a fa utolsó szintjének jobb szélső elemét, majd *kupacol*(f).



Költség: $O(\ell) = O(\log_2 n)$

BESZÚR

Új levelet adunk a fához (ügyelve a teljességre), ide tesszük az s elemet. Ezután s-et mozgatjuk felfelé, mindig összehasonlítjuk az apjával.



Egy cserénél a szülőt kisebbre cseréljük, így a másik gyermeknél továbbra is kisebb marad. $\Longrightarrow$
A kupac tulajdonság csak felfelé sérülhet.

Költség: $O(\ell) = O(\log_2 n)$

A kupacos rendezés (heap sort)

Először kupacot építünk, utána n darab MINTÖR adja nem csökkenő sorrendben az elemeket.
[J. W. J. Williams és R. W. Floyd, 1964]

Költség: $O(n) + O(n \log_2 n) = O(n \log_2 n)$

Megjegyzés: Legjobb ismert rendező algoritmus.
Pontos implementációval:
$2n\lfloor \log_2 n \rfloor + 3n$ (összehasonlítások száma) és $n\lfloor \log_2 n \rfloor + 2,5n$ (cserék száma).