Algoritmuselmélet 2. előadás

Katona Gyula Y.
Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi Tsz.
I. B. 137/b
kiskat@cs.bme.hu

2002 Február 12.

Buborék-rendezés

Input: $A[1:n]$ (rendezetlen) tömb
Ha valamely i-re $A[i] > A[i+1]$, akkor a két cella tartalmát kicseréljük. A tömb elejéről indulva a cserélgetve eljutunk a tömb végéig. Ekkor a legnagyobb elem $A[n]$-ben van. Ismételjük ezt az $A[1:n-1]$ tömbre, majd az $A[1:n-2]$ tömbre, stb.

procedure buborék
(az $A[1:n]$ tömböt nem csökkenően rendezi *)*
for $(j = n-1, j > 0, j := j-1)$ **do**
  for $(i = 1, i \leq j, i := i+1)$ **do**
    { ha $A[i+1] < A[i]$, akkor cseréljük ki őket.}

összehasonlítások száma: $n-1+n-2+\ldots+1 = \frac{n(n-1)}{2}$

cserék száma: $\frac{n(n-1)}{2}$

Java animáció: Buborék rendezés

Beszúrásos rendezés

Ha az $A[1:k]$ résztömb már rendezett, akkor szúrjuk be a következő elemet $A[k+1]$-et lineáris vagy bináris kereséssel, majd a következőt ebbe, stb.

	lineáris	bináris
összehasonlítás	$\frac{n(n-1)}{2}$	$\sum_{k=1}^{n-1} \lceil \log_2(k+1) \rceil$
mozgatás	$\frac{(n+2)(n-1)}{2}$	$\frac{(n+2)(n-1)}{2}$
átlagos összehasonlítás	$\frac{n(n-1)}{4}$	$\sum_{k=1}^{n-1} \lceil \log_2(n+1) \rceil$
átlagos mozgatás	$\frac{n^2}{4}$	$\frac{n^2}{4}$

Bináris beszúrásos rendezés lépésszáma

$$K := \lceil \log_2 2 \rceil + \lceil \log_2 3 \rceil + \ldots + \lceil \log_2 n \rceil \leq n \lceil \log_2 n \rceil$$

Jobb becslés: használjuk fel, hogy $\lceil \log_2 k \rceil \leq 1 + \log_2 k$

$$K < n - 1 + \log_2 2 + \ldots + \log_2 n = n - 1 + \log_2(n!)$$

Felhasználva a Stirling formulát: $n! \sim (n/e)^n \sqrt{2\pi n}$ kapjuk, hogy

$$\log_2 n! \sim n(\log_2 n - \log_2 e) + \frac{1}{2} \log_2 n + \log_2 \sqrt{2\pi} \leq n(\log_2 n - 1,442)$$

Ezért $K \leq n(\log_2 n - 0,442)$ elég nagy n-re.
Java animáció: Beszúrásos rendezés

Alsó becslés összehasonlítás alapú rendezésre

Ugyanaz, mintha Bar Kochba-ban kellene kitalálni, hogy az elemek melyik sorrendje (permutációja) az igazi sorrend.
Kezdetben $n!$ lehetséges sorrend jön szóba.
Két elemet összehasonlítva, a válasz két részre osztja a sorrendeket. Ha pl. azt kapjuk, hogy $x < y$, akkor az olyan sorrendek, amikben x hátrébb van y-nál, már nem jönnek szóba.
Ha az ellenség megint úgy válaszol, hogy minél több sorrend maradjon meg, akkor k kérdés után még szóba jön $\frac{n!}{2^k}$ sorrend.
Ha $\frac{n!}{2^k} > 1$ nem tudjuk megadni a rendezést. $\Longrightarrow$

Tétel. *Minden összehasonlítás alapú rendező módszer n elem rendezésekor legalább $\log_2(n!)$ összehasonlítást használ.*

Összefésüléses rendezés

Összefésülés (MERGE):

Két már rendezett sorozat (tömb, lista, stb.) tartalmának egy sorozatba való rendezése:
$A[1:k]$ és $B[1:l]$ rendezett tömbök $\Longrightarrow C[1:k+l]$ rendezett tömb
Nyilván $C[1] = \min\{A[1], B[1]\}$, pl. $A[1]$,
ezt rakjuk át C-be és töröljük A-ból.
$C[2] = \min\{A[2], B[1]\}$, stb.

Példa

A	B	C
$\mathbf{12}, 15, 20, 31$	$13, 16, 18$	
$15, 20, 31$	$\mathbf{13}, 16, 18$	$12,$
$\mathbf{15}, 20, 31$	$16, 18$	$12, 13$
$20, 31$	$\mathbf{16}, 18$	$12, 13, 15$
$20, 31$	$\mathbf{18}$	$12, 13, 15, 16$
$\mathbf{20}, 31$		$12, 13, 15, 16, 18$
31		$12, 13, 15, 16, 18, 20$
		$12, 13, 15, 16, 18, 20, 31$

[trans='Replace']

összehasonlítások száma: $k + l - 1$, ahol k, l a két tömb hossza

Összefésüléses rendezés

Alapötlet: Rendezzük külön a tömb első felét, majd a második felét, végül fésüljük össze.
Ezt csináljuk rekurzívan.

$$\text{MSORT}(A[1:n]) :=$$
$$\text{MERGE}(\text{MSORT}(A[1:\lceil n/2 \rceil]), \text{MSORT}(A[\lceil n/2 \rceil + 1 : n])).$$

Hogy elvarrjuk a rekurzió alját, legyen $\text{MSORT}(A[i,i])$ az üres utasítás.

Összehasonlítások száma

Jelöljük $T(n)$-el a lépésszámot n hosszú tömb rendezésekor. Az egyszerűség kedvéért tegyük fel, hogy $n = 2^k$.

$$T(n) \leq n - 1 + 2T(n/2),$$

$T(n) \leq n - 1 + 2(n/2 - 1 + 2T(n/4)) = n - 1 + 2(n/2 - 1) + 4T(n/4).$
$T(n) \leq n-1+2(n/2-1)+4(n/4-1)+\cdots+2^{k-1}(n/2^{k-1}-1) \leq n\lceil \log_2 n \rceil.$
Felhasználva, hogy $T(1) = 0$.

Az összefésüléses rendezés konstans szorzó erejéig optimális.

Mozgatások száma: $2n\lceil \log_2 n \rceil$
Tárigény: $2n$ cella (bonyolultabban megcsinálva elég $n + konst.$)

Java animáció: Összefésüléses rendezés

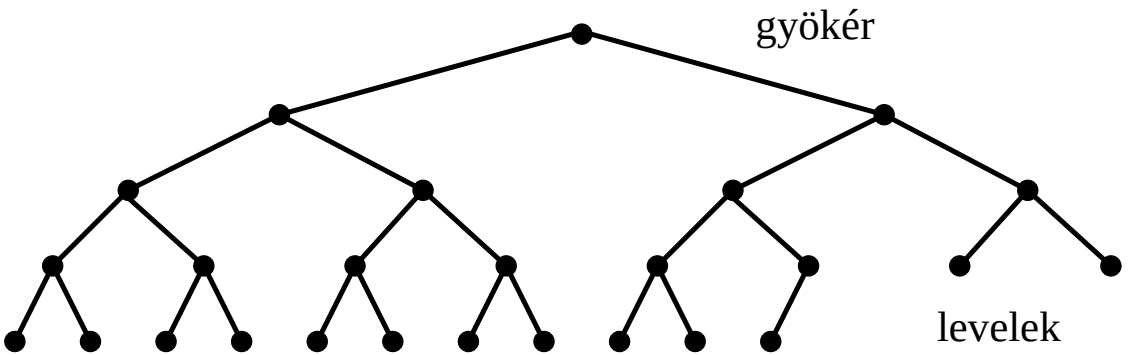
Kupac adatszerkezet

Egy $(U, <)$ rendezett halmaz egy S véges részhalmazát szeretnénk tárolni, hogy a *beszúrás* és a minimális elem törlése (*mintör*) hatékony legyen.

Alkalmazások:

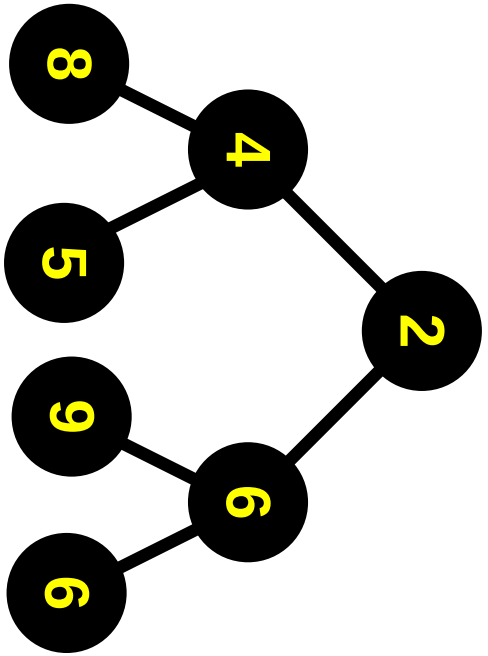
- Jobok indítása
- Több rendezett halmaz összefésülése
- Gyors rendezési algoritmus

[trans='Box,I']Teljes bináris fa:



Bináris fa ábrázolása tömbbel

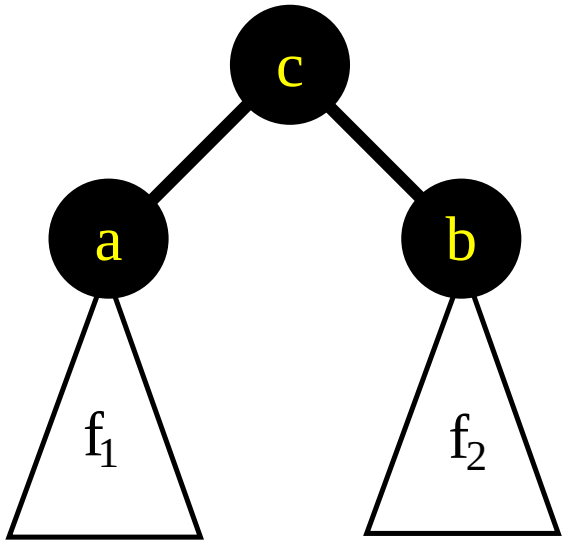
A fa csúcsai az $A[1:n]$ tömb elemei.
Az $A[i]$ csúcs bal fia $A[2i]$, a jobb fia pedig $A[2i+1]$. [trans='Replace']
$\Longrightarrow$ $A[j]$ csúcs apja $A[\lfloor j/2 \rfloor]$



2	4	6	8	5	9	6

Kupac tulajdonság: apa $<$ fia

Kupacépítés



f_1 és f_2 kupacok

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

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

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

Kupacépítés költsége

Bináris fában:

1. szint: 1 pont

2. szint: 2 pont

3. szint: 2^2 pont

⋮

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

$$\Longrightarrow n \geq 1 + \sum_{i=0}^{l-2} 2^i = 2^{l-1} \Longrightarrow l \geq 1 + \log_2 n$$

Az i-edik szinten levő v csúcsra *felszivárog*(v) költsége legfeljebb $l - i$ csere és legfeljebb $2(l - i)$ összehasonlítás.
A cserék száma ezért összesen legfeljebb $\sum_{i=1}^{l}(l - i)2^{i-1}$.

$j = l - i$ (azaz $i = l - j$) helyettesítéssel

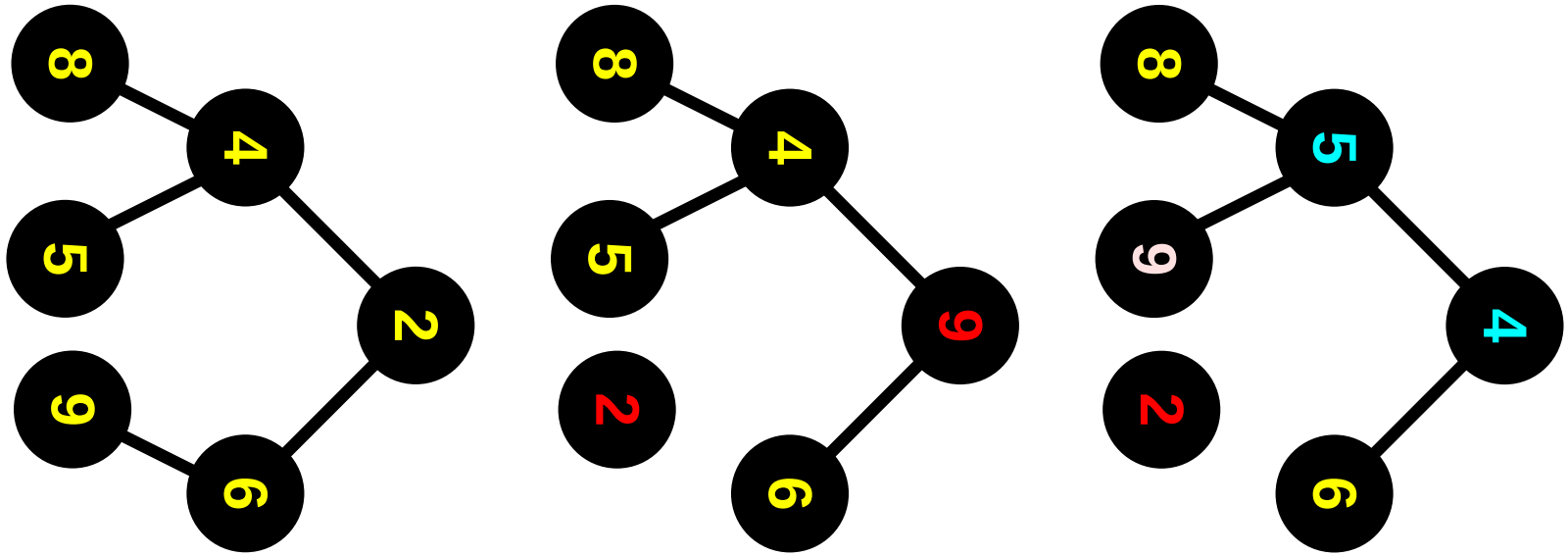
$$\sum_{j=0}^{l-1} j 2^{l-j-1} = 2^{l-1} \sum_{j=0}^{l-1} j/2^j < 2^l \leq 2n.$$

$$\begin{array}{ccccc} 1/2 & & & & \\ 1/4 & 1/4 & & & \\ 1/8 & 1/8 & 1/8 & & \\ \vdots & & & & \\ \frac{1}{2^{l-1}} & \frac{1}{2^{l-1}} & \frac{1}{2^{l-1}} & \cdots & \frac{1}{2^{l-1}} \\ \hline < 1 & < 1/2 & < 1/4 & \cdots & < \frac{1}{2^{l-1}} \end{array}$$

Tétel. *Kupacépítés költsége:* $O(n)$

MINTÖR

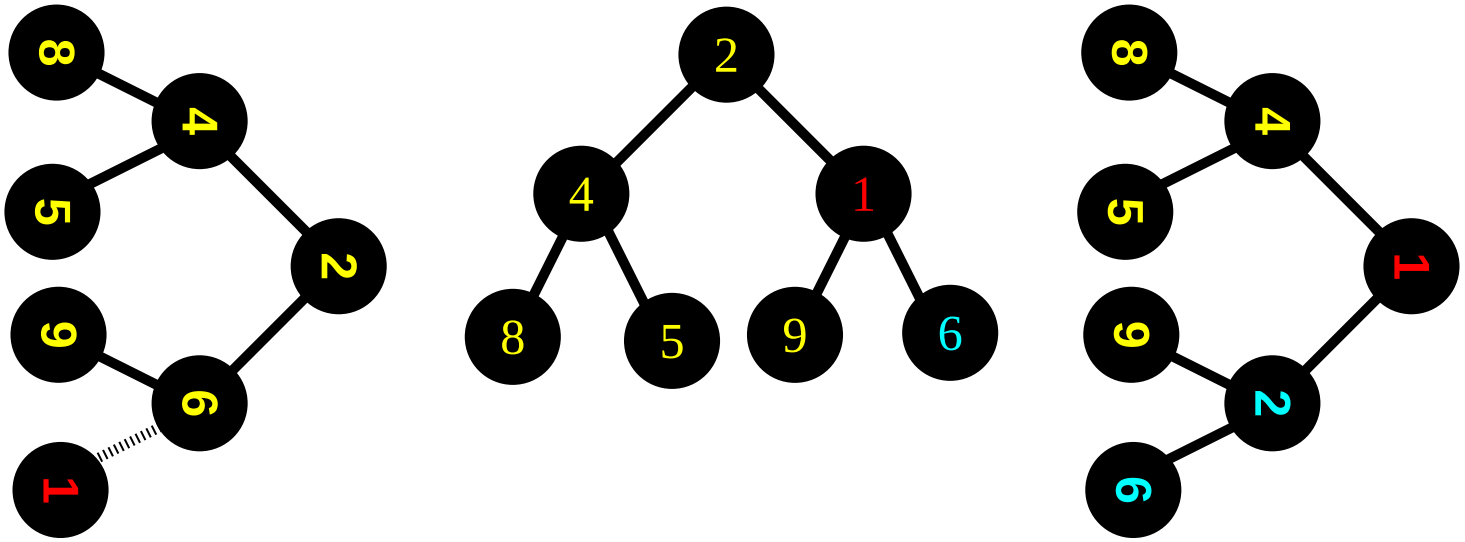
A minimális elem az f gyökérben van, ezt töröljük.
A f-be tesszük a fa utolsó szintjének jobb szélső elemét, majd *felszivárog*(f).



Költség: $O(l) = O(\log_2 n)$

BESZÚR

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



Költség: $O(l) = O(\log_2 n)$

A kupacos rendezés

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

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

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).

Java animáció: Kupacos rendezés