Randomizált adatszerkezetek

Iván Szabolcs

2017 tavasz

Rendezési algoritmusok

Tömbök rendezésére pl. a következő két rendezőalgoritmust ismerjük:

- gyorsrendezés
- kupacrendezés

Ezeket most körüljárjuk kicsit.

Gyorsrendezés

- válasszunk egy pivot elemet
- érjük el, hogy a tömbben legyen egy pozíció, amire a pivot elem kerül, tőle balra a nála kisebbek, tőle jobbra a nála nagyobbak
- rekurzívan hívjuk a két félre a rendezést

6	8	0	3	7	2	9	5	4	1

0	3	2	5	4	1	6	8	7	9

0	3	2	5	4	1	6	7	8	9

0	2	1	3	5	4	6	7	8	9

0	1	2	3	4	5	6	7	8	9

(ha max kételemű, cseréljük ha kell)

Gyorsrendezés

A pivotálást lehet külön tömbben is elvégezni, de nem kell:

- legyen a pivot elem az első
- viszünk két mutatót, egyet elölről hátra, másikat szembe
- ,,rossz" elemen megállunk
- ha még nem értek össze a mutatók és rossz elemen állnak, cseréljük
- ha összeértek, becseréljük a pivotot a helyére (nem stabil)

6	8	0	3	7	2	9	5	4	1
6	8	0	3	7	2	9	5	4	1
6	1	0	3	7	2	9	5	4	8
6	1	0	3	7	2	9	5	4	8
6	1	0	3	4	2	9	5	7	8
6	1	0	3	4	2	5	9	7	8
5	1	0	3	4	2	6	9	7	8

Gyorsrendezés

Időigény?

- Egy N méretű tömbön $O(N)$ idő végigmenni, majd egy K és egy $N-K-1$ méretű részre rekurzívan
- $T(N) \leq N + T(K) + T(N-K-1)$
- Legrosszabb eset: $T(N) = N + \max_{K=0}^{N-1}(T(K) + T(N-K-1))$
- Legjobb eset: minimummal

Ennek a legjobb esete, ha a pivot mindig középre esik: $T(N) = N + 2T(N/2)$, ez $O(N \log N)$ időigény

A legrosszabb eset négyzetes!

Rendezett tömbre!!NÉGY!

Gyorsrendezés random pivottal

A randomizálás bevonásakor (most)

- a cél: ne legyen rossz input, csak rossz randomgenerálás
- így user nem tud direkt rossz inputot adni
- nagy számok törvénye szerint sok futtatás hozni fogja átlagban a várható értéket

Randomizált változat: pivot elem választásakor cseréljük be a pivot pozícióba a rendezendő tömb egy random elemét.

6	8	0	3	7	2	9	5	4	1

4	8	0	3	7	2	9	5	6	1

2	1	0	3	4	7	9	5	6	8

0	1	2	3	4	8	7	5	6	9

0	1	2	3	4	5	7	8	6	9

0	1	2	3	4	5	6	7	8	9

Gyorsrendezés random pivottal

- Várható értéket számolni nem mindig könnyű
- Most nem is nehéz
- N hosszú tömbből az $i = 1, \ldots, N.$ elem kiválasztásának $\frac{1}{N}$ az esélye
- Tehát ha $E(N)$ az N méretű tömb rendezése lépésszámának a várható értéke, akkor

$$E(N) = N + 1 + \frac{1}{N} \sum_{i=0}^{N-1} (E(i) + E(N - i - 1))$$

$$E(N) = N + 1 + \frac{2}{N} \sum_{i=0}^{N-1} E(i)$$

Gyorsrendezés random pivottal

$$E(N) = N + 1 + \frac{2}{N} \sum_{i=0}^{N-1} E(i)$$

$$NE(N) = N^2 + N + 2 \sum_{i=0}^{N-1} E(i)$$

$$(N-1)E(N-1) = (N-1)^2 + (N-1) + 2 \sum_{i=0}^{N-2} E(i)$$

$$NE(N) - (N-1)E(N-1) = 2N + 2E(N-1)$$

$$NE(N) = (N+1)E(N-1) + 2N$$

Gyorsrendezés random pivottal

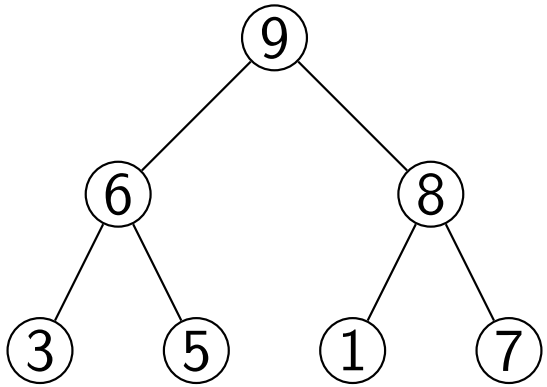
$$
\begin{aligned}
NE(N) &= (N+1)E(N-1) + 2N \\
\frac{E(N)}{N+1} &= \frac{E(N-1)}{N} + \frac{2}{N+1} \\
&= \frac{E(N-2)}{N-1} + \frac{2}{N} + \frac{2}{N+1} \\
&= \dots \\
&= \frac{E(2)}{3} + \sum_{i=3}^{N+1} \frac{2}{k}
\end{aligned}
$$

az ott jobb oldalon meg kisebb, mint $2 \ln N$, mert tudunk integrálni

$$\Rightarrow \text{átlagos eset: } \Theta(N \log N)$$

(Maximumos) kupac

- Bináris fa
- Minden csúcsban az adat nagyobb-egyenlő, mint a gyerekeiben

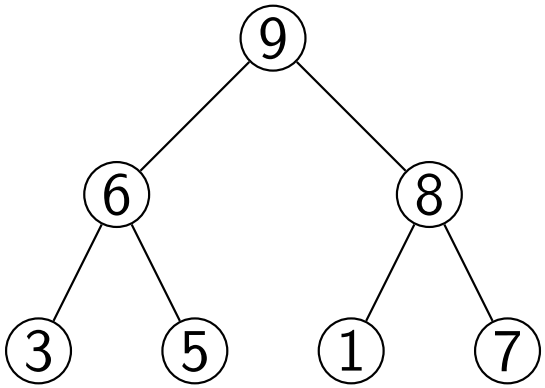


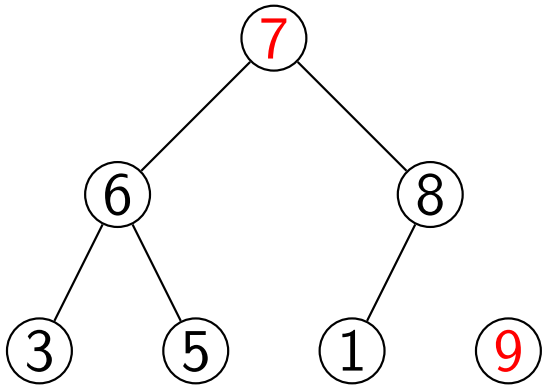
Legnagyobb elem mindig a gyökér.
Algoritmus:

- Kicseréljük a legnagyobb elemet az utolsóval, levágjuk
- Megjavítjuk a kupac tulajdonságot

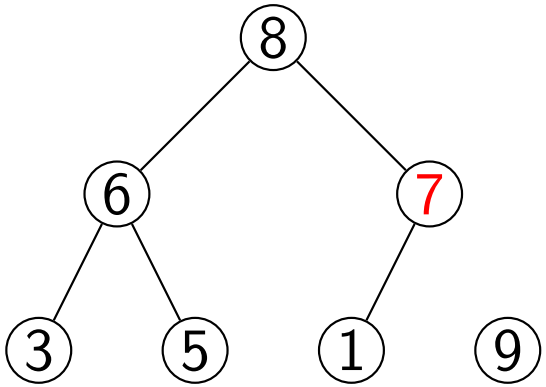
(A kupacrendezésben ez tömbben van a ,,szokásos" módon.)

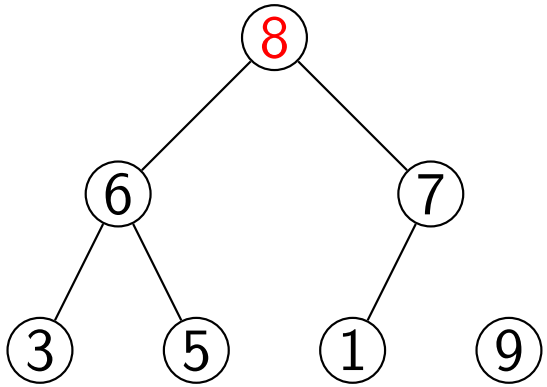
Kupac

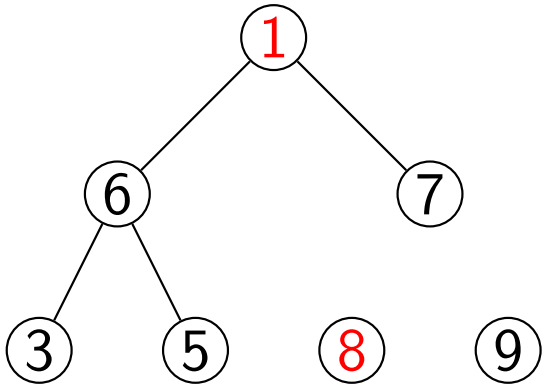




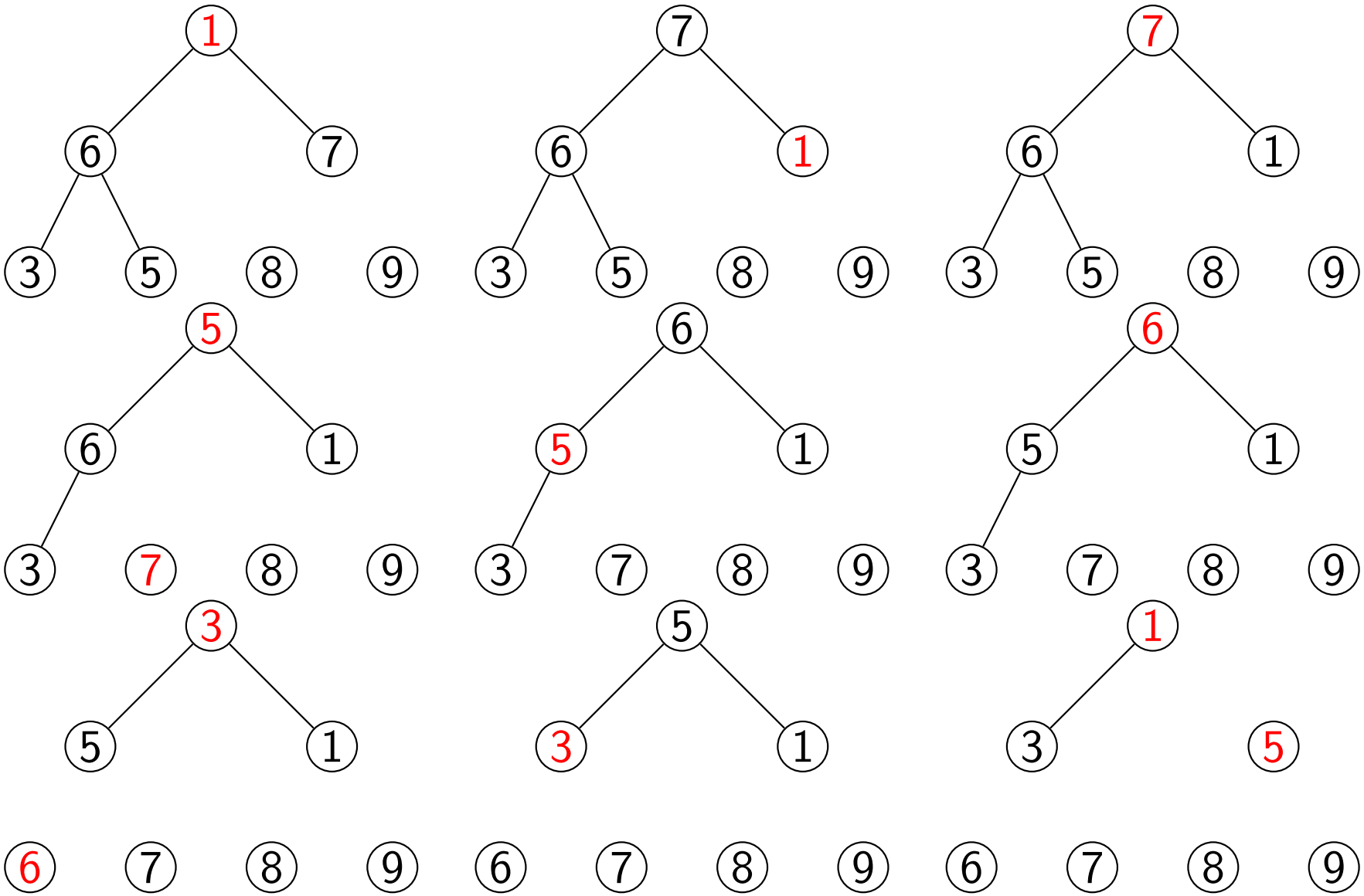
Helyreállítás: amíg van a csúcsnak nagyobb gyereke, a nagyobb gyerekével kicseréljük és megyünk le vele







Kupac

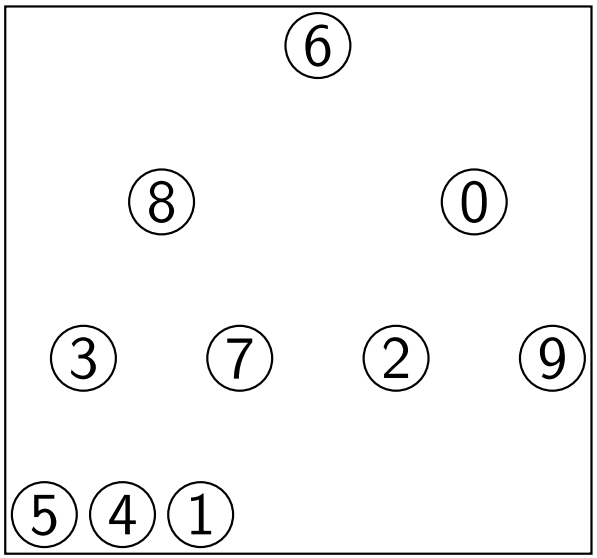

1 6 7 3 5 8 9
5 6 1 3 7 8 9
3 5 1 6 7 8 9
7 6 1 3 5 8 9
6 5 1 3 7 8 9
5 3 1 6 7 8 9
7 6 1 3 5 8 9
6 5 1 3 7 8 9
1 3 5 6 7 8 9

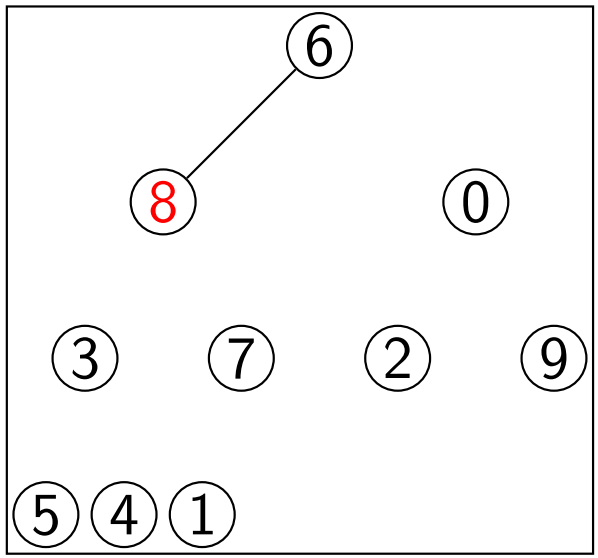
Kupac időigény

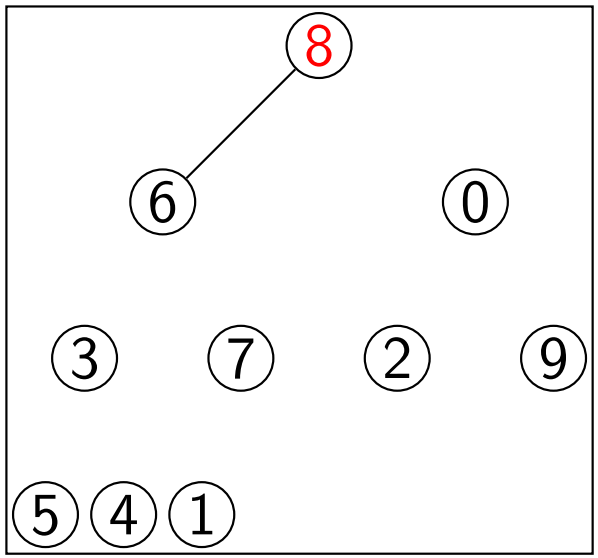
- Így a kupac egy majdnem teljes bináris fa
- N elemnél mélysége $\log N$
- Egy csere plusz legfeljebb $\log N$ a helyreállításnál
- Összesen $N \log N$ lépés, miután felépült a kupac (legrosszabb eset)

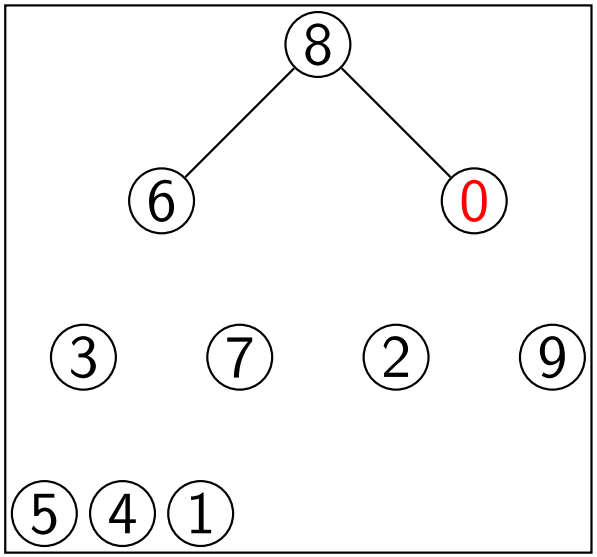
Kupac felépítése: a kupacba az alján szúrjuk be az új elemet és amíg az apja kisebb, cseréljük őket

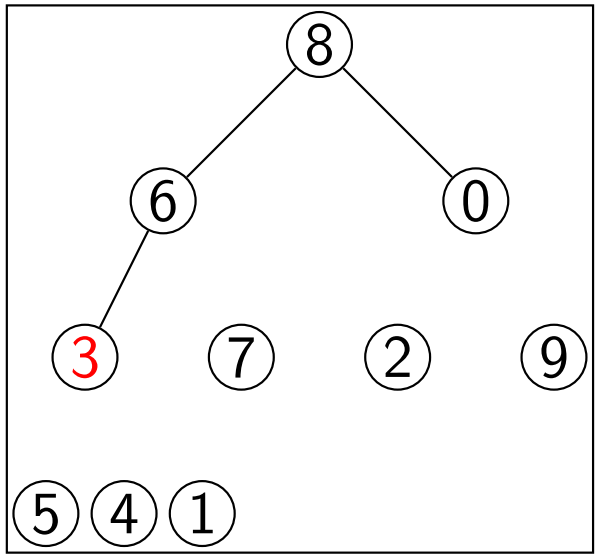
Kupac építése

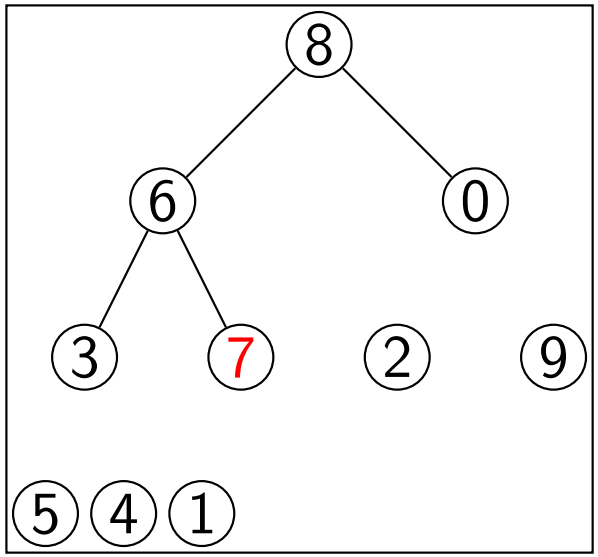




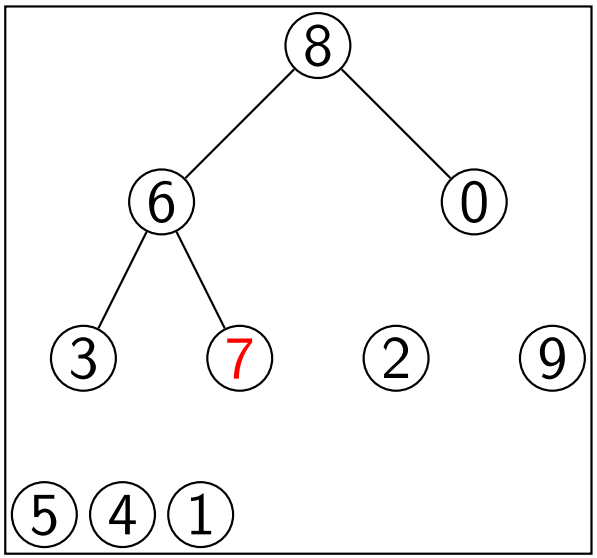


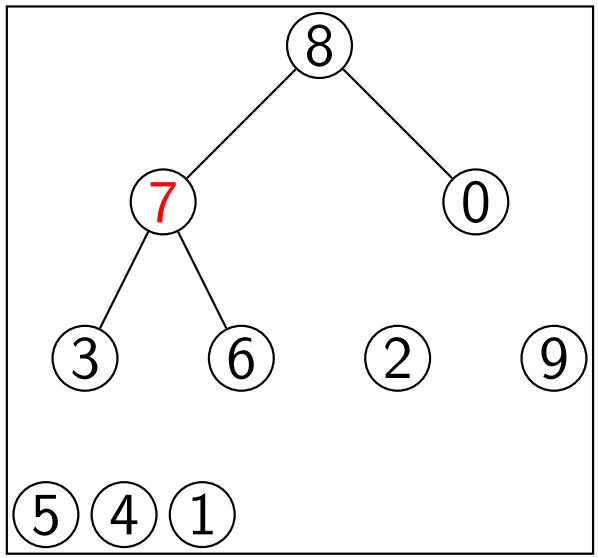


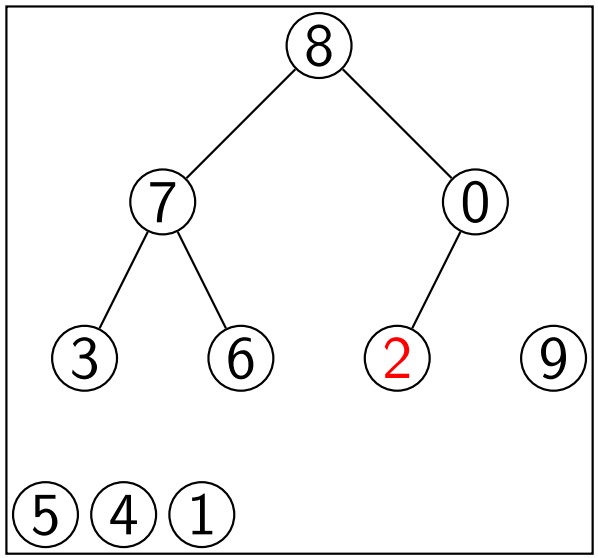


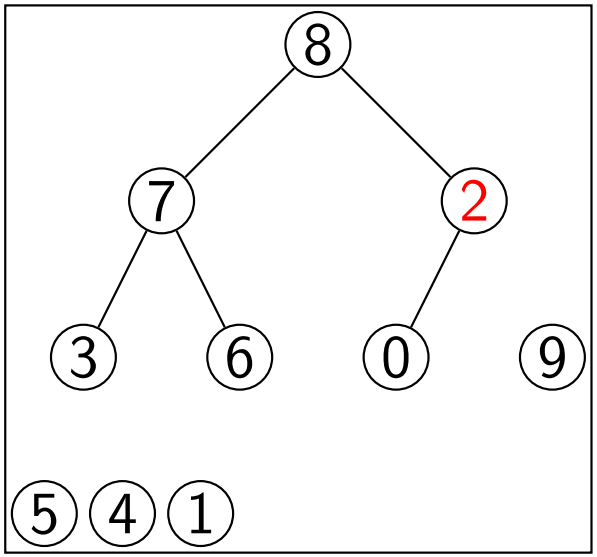


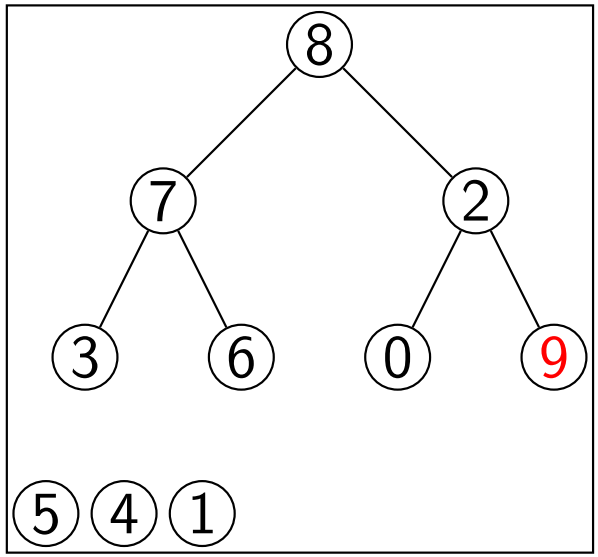
Kupac építése

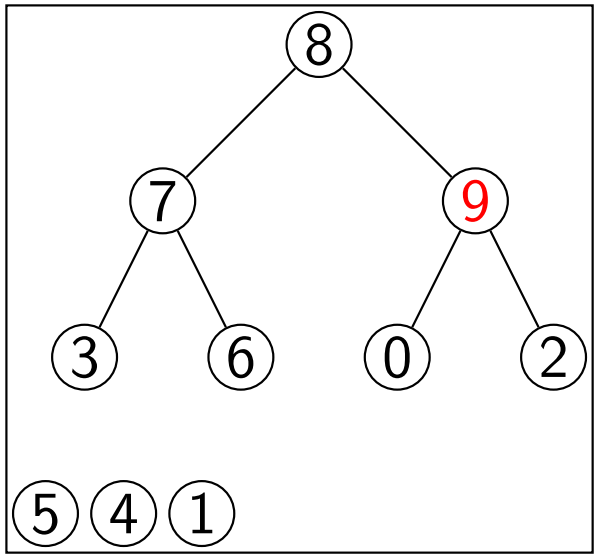




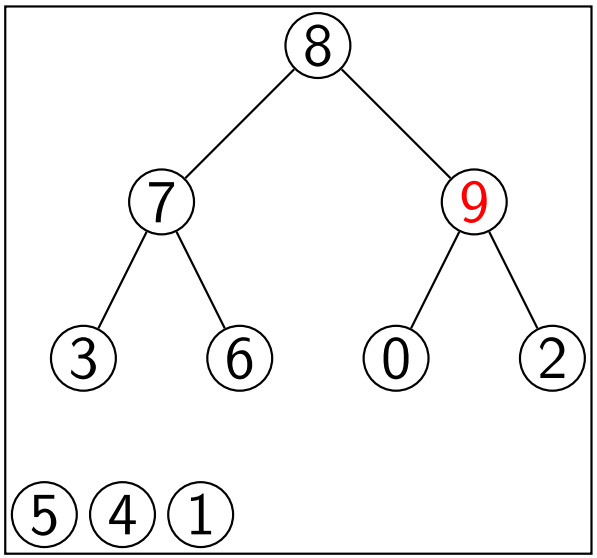


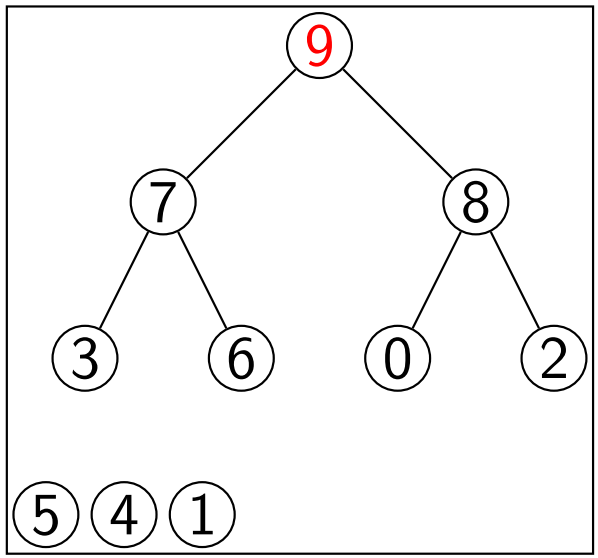


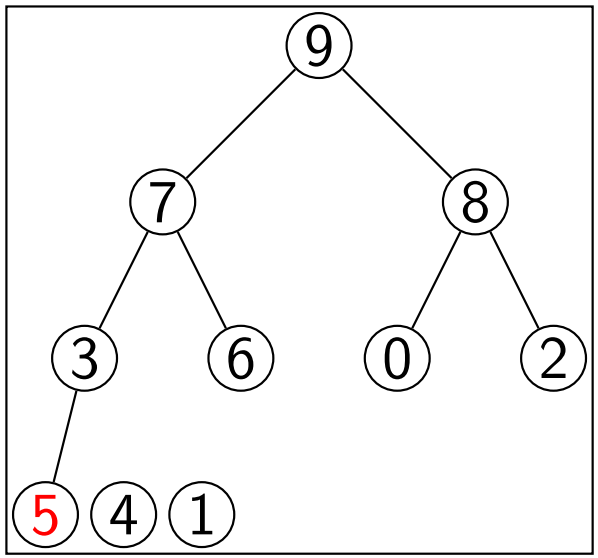


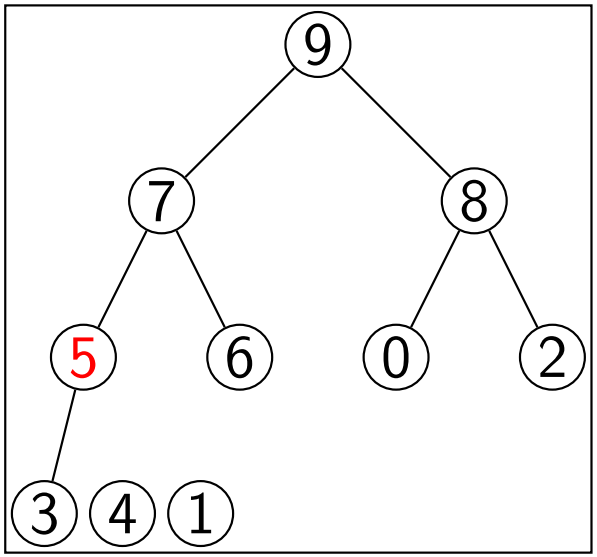


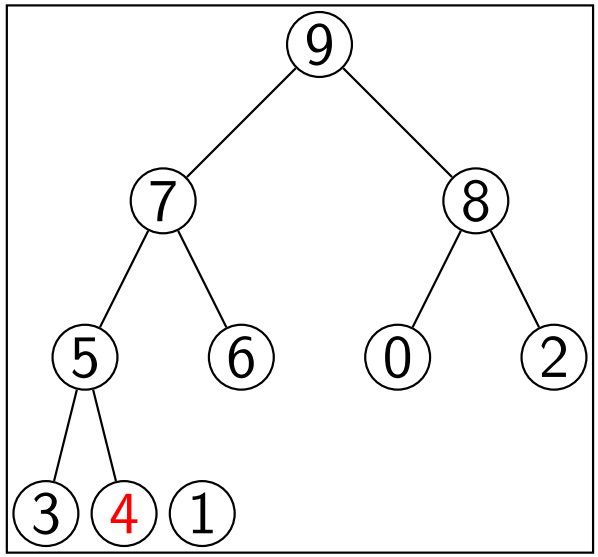
Kupac építése

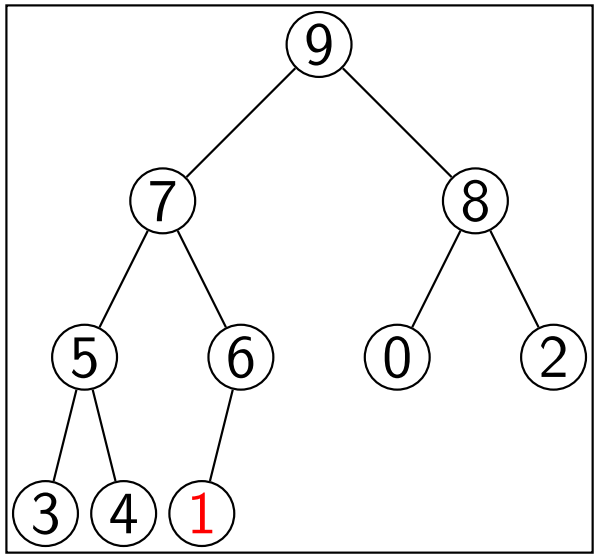












Kupac építése – időigény

- Egy elemet legfeljebb $\log N$ szinten át emelhetünk a helyére
- Összesen ez is $O(N \log N)$.

Tehát a kupacrendezés $O(N \log N)$ legrosszabbeset, helyben rendező algoritmus.

Heap sort.

(Nem stabil.)

- SGI SRL-ben, GNU STL-ben, dotnetben: introsort – quicksort, majd ha már $\log N$ mély a rekurzió, heapsort.
- Quicksort quickselect – sorozatos pivottal keresünk középső elemet, és az lesz a végső pivot elem
- Quicksort median-of-3 – első, középső, utolsó elem közepe lesz a pivot
- Quicksort introselect: quickselect, median-of-medians fallbackkel.

Median-of-medians

- A tömbben minden hármas középső elemét becseréljük a hármas elejére
- Ezt rekurzívan tesszük a hárommal osztható indexű elemek (a mediánok) ,,tömbjére”

6	8	0	3	7	2	9	5	4	1

6	8	0	3	7	2	5	9	4	1

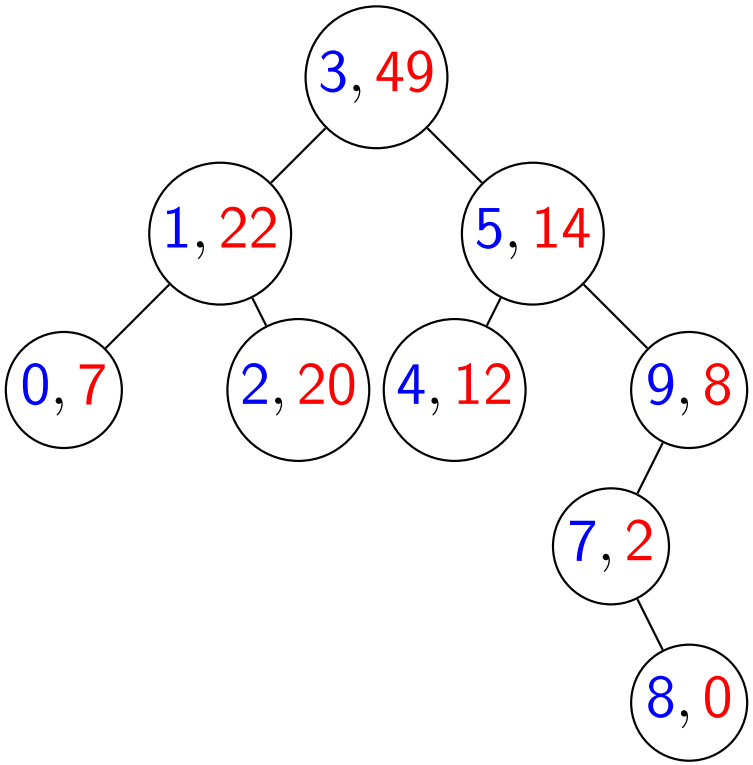
5	8	0	3	7	2	6	9	4	1

- Median-of-medians:
  - ötösével csoportosítunk hármasok helyett
  - az ötösök középső elemeinek a valódi mediánját keressük
  - quickselecttel
  - ami rekurzívan választ pivot elemet

Treap – farakás :P

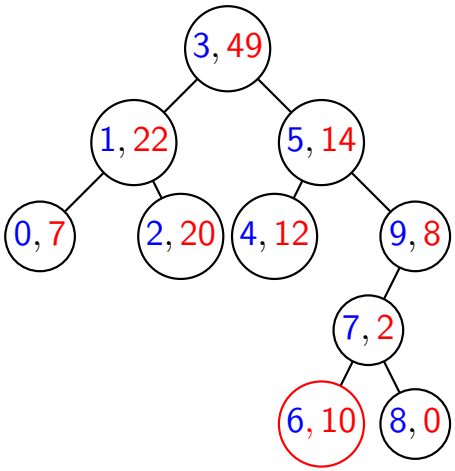
Tree + Heap

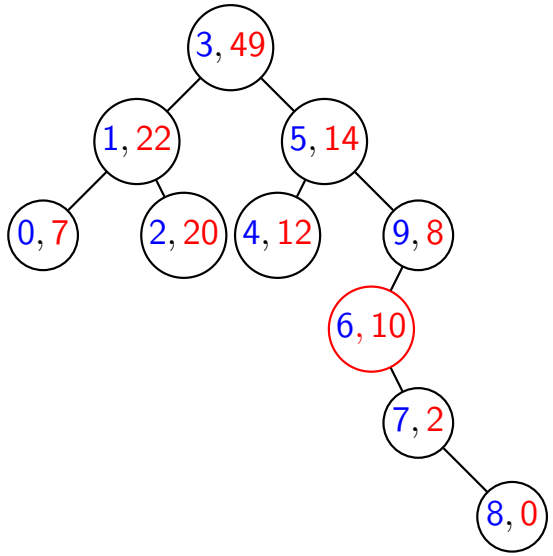
- Bináris keresőfa
- A csúcsokban az adat mellett egy random prioritás is van
- Az adat szerint keresőfa
- A prioritás szerint kupac
- ,,Cartesian tree" ez is...

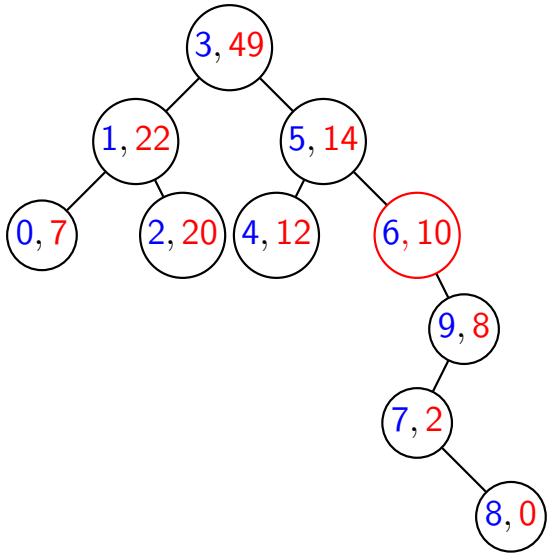


Treap beszúrás

- Beszúrjuk mint keresőfába random prioritással
- Kupactulajdonságot forgatással javítunk

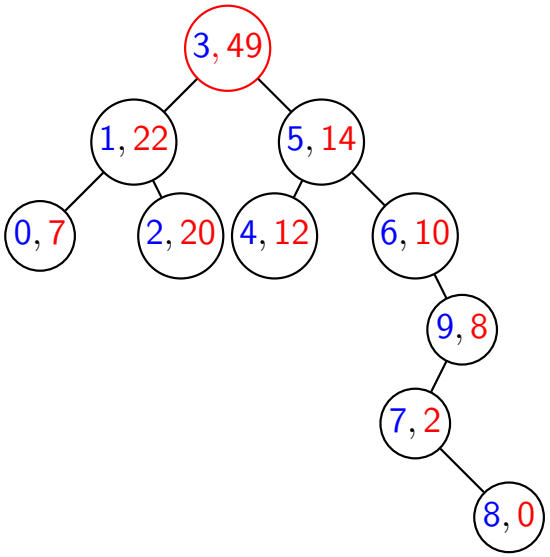


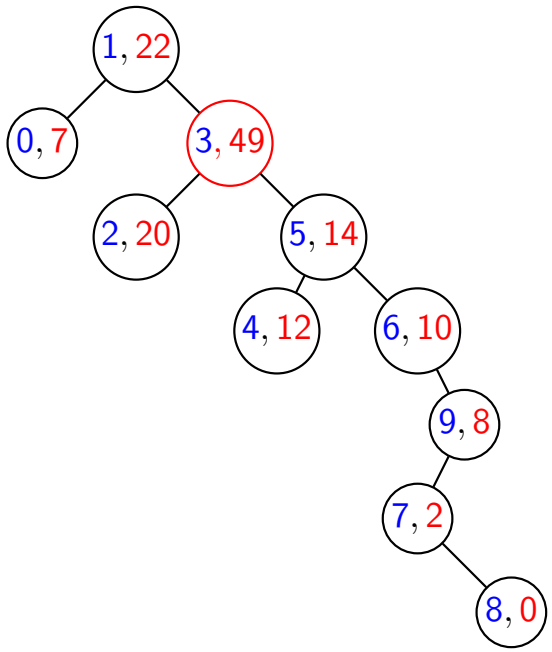


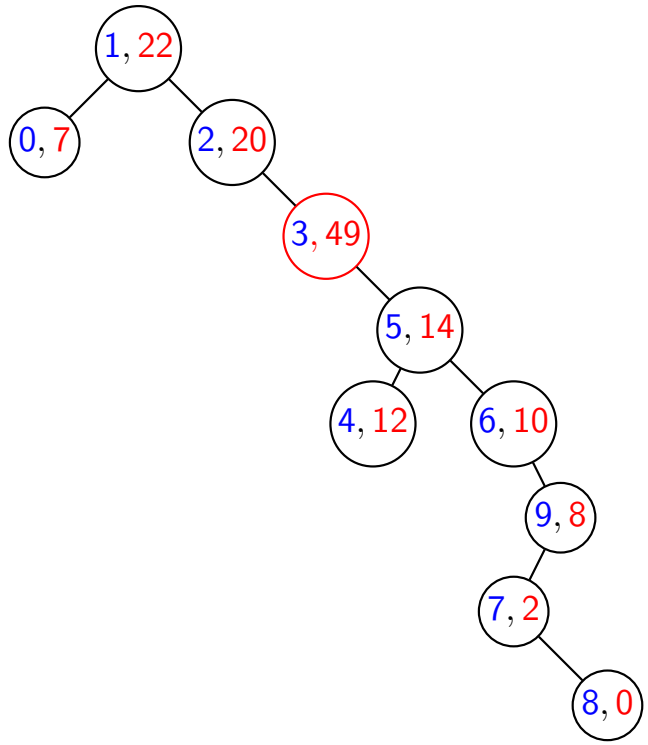


Treap törlés

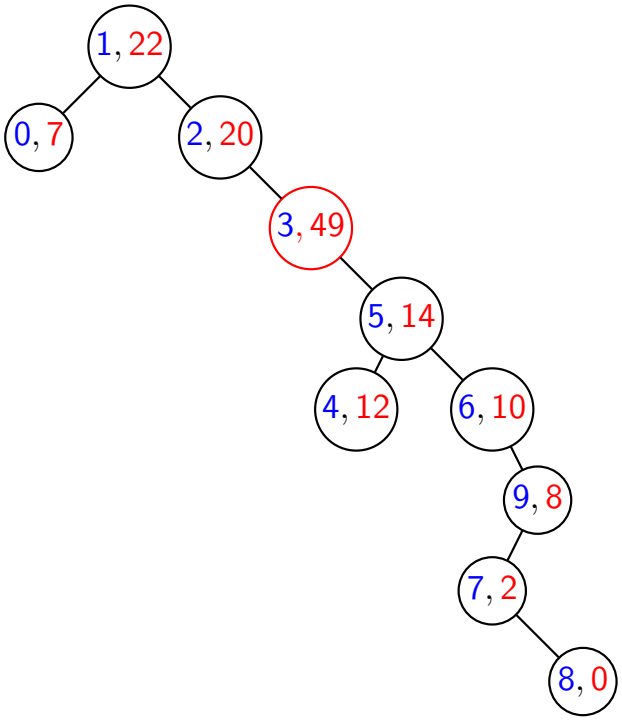
- Leforgatjuk a törlendő elemet a kupactulajdonság fenntartásával
- (ha két gyerek van, a nagyobbat forgatjuk fel)

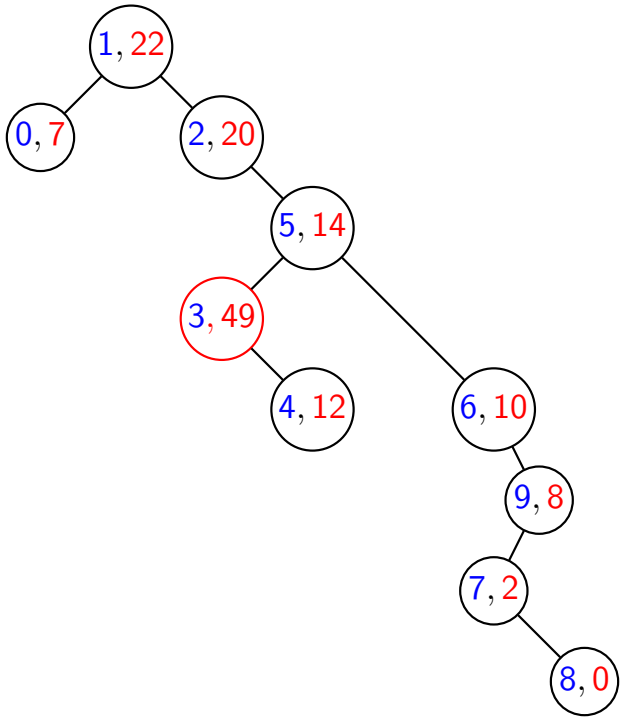


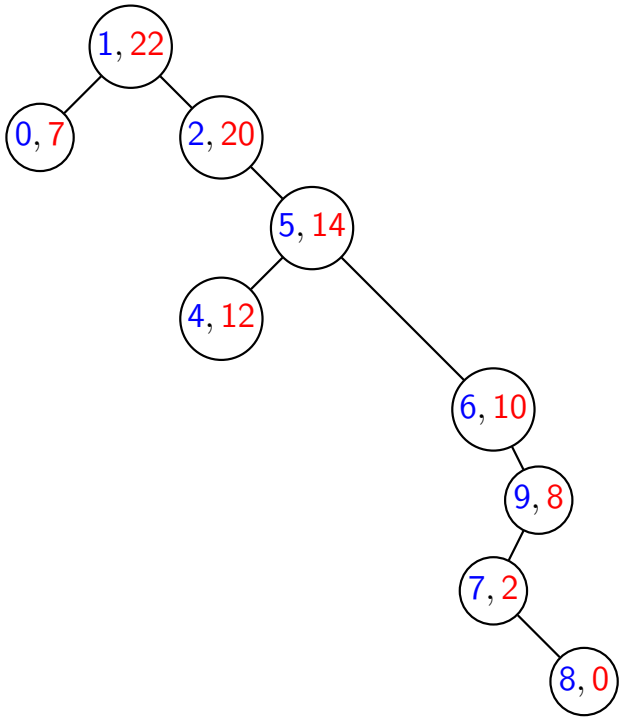




Treap törlés





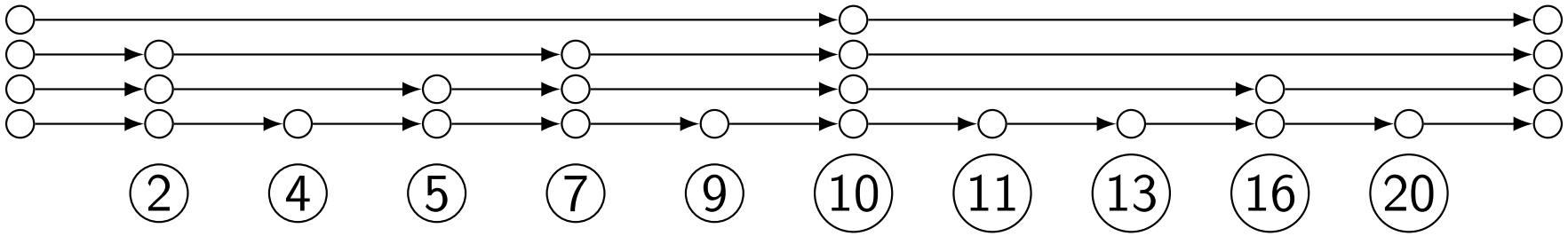


Minden művelet átlagos költsége $O(\log N)$.

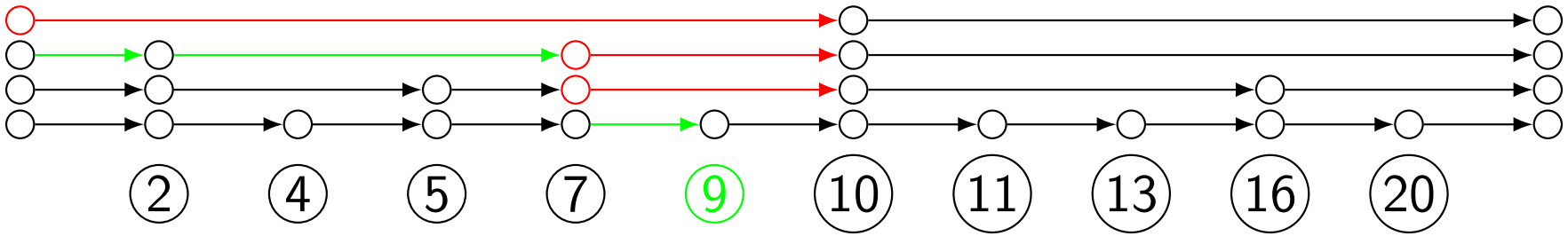
Skiplist

- Rendezett láncolt lista gyorsabb kereséssel

,



- Keresés: a felső szintről indulok
- Ha kisebb-egyenlőre ugrik, ugrok
- Ha nem, egy szinttel lejjebb lépek



Skiplist

- Törlés: az elem tornyába ütköző láncokból kell kivenni
- Beszúrás: építünk egy szintet a toronynak és mindig $\frac{1}{2}$ valószínűséggel felhúzunk még egy szintet
- (ha túlmegy a magasság az eddigi maxon, megállunk és a kezdő-záró tornyokat is megemeljük)
- Várható magasság: $\frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{16} \cdot 4 + \ldots = 2$
- $\Rightarrow$ várhatóan csak $O(n)$ memória
- átlagosan $O(\log n)$ keresési, beszúrási, törlési idő
- jobban párhuzamosítható, mint egy önkiegyensúlyozó keresőfa
- plusz mezővel tud tömbcímzést is $O(\log n)$ átlagos időben