

Kupac adattípus és műveletei. Kupacrendezés. Elsőbbségi sorok.

Kupac (heap) adattípus

- Bináris fa, amelyben minden csomópont értéke nagyobb mindegyik leszármazottjánál (=kupactulajdonság)
- Magasság: a csúcsból a levélig vezető leghosszabb út
- Kiegészített: csak az utolsó réteg lehet nem teljes
- Ábrázolás: vektorban, rétegenként
- Minden réteg mérete=korábbi rétegek+1. Minden réteg első elemindexe= 2^h

Adattípus definíció

- Kupac: csak a rendezésekre koncentrál
- Elsőbbségi sor: adatbeszúrás, törlés stb.
- Használatuk: az integer tömb csak az adatok „kulcsát” (a rendezés kulcsát) jelzik, ehhez a gyakorlatban még járulékos adatok is tartoznak
- Pl: operációs rendszerekben a párhuzamosan futó feladatok listájának kezelésére használják

Kupacolás működés közben

- Hatékonyság: $\Theta(\text{magasság}) \approx \Theta(\log(n))$
- Egyetlen elemet görget lefelé addig, amíg a saját helyét el nem éri

Kupacolás teljes fára

- Kupacolás: a rossz helyen levő elemeket lefelé görgeti, mindig a nagyobb gyerek felé
- Vagyis: levélelemekre nem lehet kupacolni
- Teljes fára: minden egyes elemére, a levelek feletti elemtől kezdve a gyökérig (bottom-up)
- ```
public void epit() {
 for (int i=(meret-1)/2; i>0; i--)
 kupacol(i); }
```

## Kupacépítés hatékonysága

- Hatékonyság(1):  $O(n \log n)$  ???
- Hatékonyság(2):
- kupacol futási ideje a csúcs magasságának lineáris függvénye  $O(h)$
- Bármely  $h$  magasságú (levéltől számítva) csúcsok száma legfeljebb  $\leq n/(2^{h+1})$  -- ha a fa teljes
- „Épít” futásideje „ $h$ ” szerint összegezve
- $\sum_{h=0}^{\lg(n)} \lg(n) \cdot (n/2^{h+1}) \cdot O(h) = O(n \cdot \sum_{h=0}^{\lg(n)} \lg(n) \cdot (h/2^h))$
- $\sum_{h=0}^{\infty} \dots = 0 + 1/2 + 2/4 + 3/8 \dots = \text{KONSTANS}$
- hatékonyság =  $O(n)$  → a kupaccá alakítás lineáris idő alatt fut le

## A kupacrendezés algoritmus

(Növekvő sorrendbe rendezünk...)

- a legnagyobb elem a gyökér, az legyen a legutolsó □ az utolsót és az elsőt cseréljük
- eggyel rövidebb vektort kupacoljuk. (csak az új első elem sértheti a kupactulajdonságot)
- ```
public void rendez() {
    epit();
    for (int i=hossz-1; i>0; i--) {
        int csere = tomb[i];
        tomb[i]=tomb[0];
        tomb[0] = csere;
        hossz--;
        kupacol(0); }
    }
```
- Hatékonysága: $O(n \log n)$

Elsőbbségi sorok

Minden elemhez tartozik egy kulcs (prioritás). Az elemek feldolgozása a kulcsok csökkenő/növekvő sorrendjében történik.

Felhasználják pl.: munkaütemezésnél az osztott működésű sz. gépeken, vagy eseményvezérelt szimulációnál.

- Maximális elem visszaadása:


```
public class elsobbSor
    extends kupac {
        public integer max() {
            return (tomb(0)); }
    }
```
- Maximális elem kivétele és törlése:


```
public integer kiveszMax() {
    int maxi = max();
    tomb[0] = tomb[meret--];
    kupacol(0);
    return (maxi); }
    }
```
- Elem beszúrása:


```
public void beszur(int x) {
    tomb[meret++] = x;
    int i = meret - 1;
    while (i > 0 && tomb[szulo(i)] < x) {
        tomb[i] = tomb[szulo(i)];
        i = szulo(i); }
    tomb[i] = x; }
    }
```
- 1. Utolsó (levél) elemként felvesszük
- 2. Buborékoztatjuk fel a gyökér felé, addig, amíg egy nála nagyobb szülőelemet nem találunk