6. Elsőbbségi (prioritásos) sor

Köznapi fogalma, megjelenése: pl. sürgősségi osztályon a páciensek nem a beérkezési időnek megfelelően, hanem a sürgősség mértéke szerint kerülnek ellátásra. Az operációs rendszerekben a prioritásos jobok feldolgozása is hasonló elv alapján történik.

Keressük azt az adatszerkezetet, amelyben az elsőbbségi sor típus tárolása hatékony módon lehetséges. Ez azt jelenti, hogy szeretnénk, ha az elsőbbségi sor műveleteit a lehető leggyorsabban tudnánk végrehajtani. Jó lenne olyan struktúrát találni, amelyben a beérkező beteg elhelyezése, illetve az ellátásra következő páciens kiválasztása (együttesen) minél rövidebb időt venne igénybe. (Milyen elrendezésben ültessük le őket a váróteremben? A válasz nem triviális, általában egy kezdő nem is ismeri, noha a veremről, sorról már tudja, hogy hogyan néznek ki.)

6.1. ADT

Az egyszerűség kedvéért a P prioritásos sor típusba csak az egyes elemek prioritását tesszük be, az elemet magát nem. Így a sorban csak $\mathbb{N}$-beli elemek vannak.

1. Műveletek

		Ezen műveletek elnevezése más könyvekből	
Empty:	$\to P$	Üres	Create
IsEmpty:	$P \to L$	Üres?	Empty?
Insert:	$P \times \mathbb{N} \to P$	Sorba	Enqueue
Max:	$P \to \mathbb{N}$	Első Prior / Első Elem	MaxPrior
DelMax:	$P \to P \times \mathbb{N}$	Sorból	Serve / Dequeue

2. Megszorítások:

$D_{Max} = D_{DelMax} = P \setminus \{Empty\}$

3. Axiómák: $\forall p \in$ P-re, $\forall n \in \mathbb{N}$

1. IsEmpty(Empty) vagy p = Empty → IsEmpty(p)
2. IsEmpty(p) → p = Empty
3. ¬IsEmpty(Insert(p,n))
4. Max(p) = DelMax(p)$_2$, ahol a 2-es index a pár második komponensét jelzi.
5. Insert(DelMax(p)) = p
6. ¬IsEmpty(DelMax(p)$_1$) → Max(p) ≥ Max(DelMax(p)$_1$)
7. $n \geq$ Max(p) → DelMax(Insert(p,n))$_1$ = $p \wedge$ Max(Insert(p,n)) = n
8. $n <$ Max(p) → Max(Insert(p,n)) = Max(p)
9. DelMax(Insert(Empty,n)) = (Empty,n)
10. Max(Insert(Empty,n)) = n

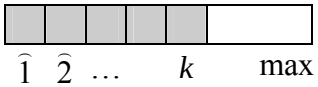
Megjegyzés: A 4.-8. axiómáknál feltettük, hogy ¬IsEmpty(p).

6.2. ADS és reprezentációs szint

A két pontot együtt tárgyaljuk, ugyanis meg kell vizsgálni, hogy milyen reprezentációval tudjuk leghatékonyabbá tenni a típus műveleteit. A különböző megvalósítási módok pedig különböző szerkezeti tulajdonságokat eredményeznek, melyekkel az ADS-szint foglalkozik.

A következőkben látni fogjuk, hogy bármely megvalósítást is választjuk, az Empty és IsEmpty műveletek $\Theta(1)$ hatékonyságúak, ezért mindig csak a többi művelet hatékonyságát vizsgáljuk.

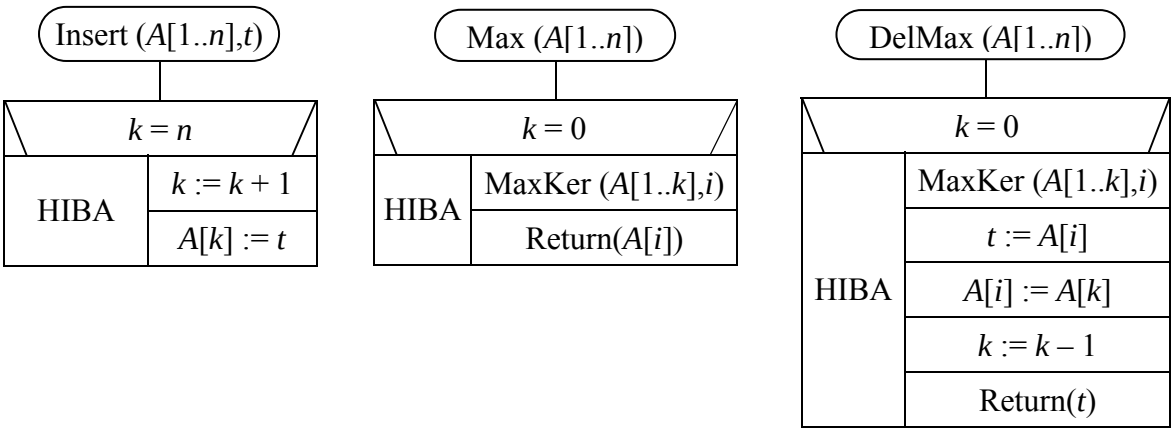
1. Megvalósítás: Rendezetlen tömbbe kerülnek az elemek, pl. a beírás sorrendjében.



k jelzi a mindenkori elemszámot.

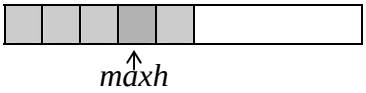
Az egyes típusműveletek hatékonysága (T(*n*) az összehasonlítások számát jelöli):

- Insert (*A*[1..*n*],*t*): Ha a tömb nincs tele, akkor *k* értékét növeljük 1-el, *A*[*k*] értékét *t*-re állítjuk. T(*n*) = Θ(1).
- Max (*A*[1..*n*]): Ha a sor nem üres, akkor egy MaxKer-rel megoldható a feladat. T(*n*) = *n* – 1 = Θ(*n*).
- DelMax (*A*[1..*n*]): A maximális és egyben törlendő elem megkeresése az előzőek alapján történik. Ha *i* a legnagyobb elem indexe, akkor a törléshez *A*[*i*] megkapja *A*[*k*] értékét, *k* értéke pedig 1-el csökken. T(*n*) = Θ(*n*).



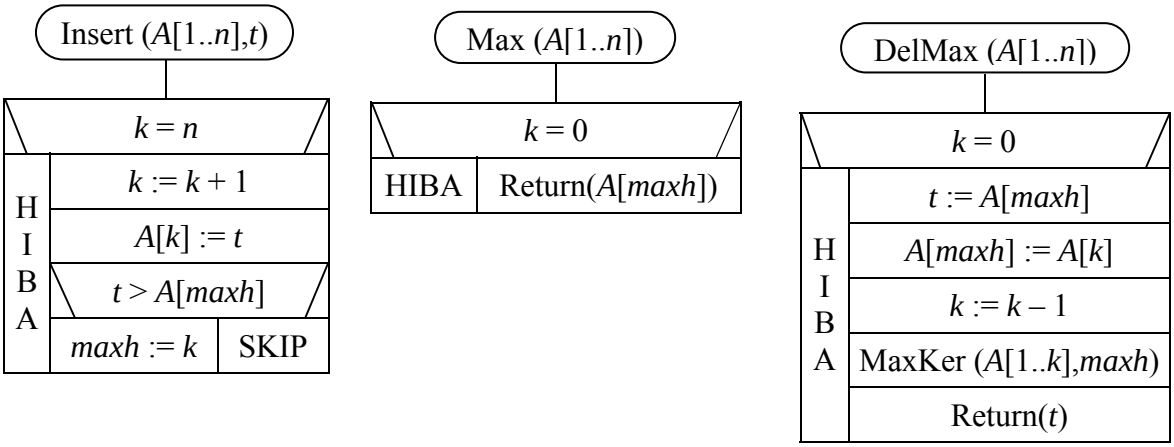
Megjegyzés: Ha T(*n*) nem korlátozódik az összehasonlítások számára, hanem általában végrehajtási időt jelöl, a fenti nagyságrendek akkor is jók.

Javítás: Nem nevezhetjük új megvalósítási módnak az alábbiakat, az előzőhez képest csak új változót vezetünk be:



maxh mindig a maximális elem indexe (helye)

Az így módosított stuktogramok:

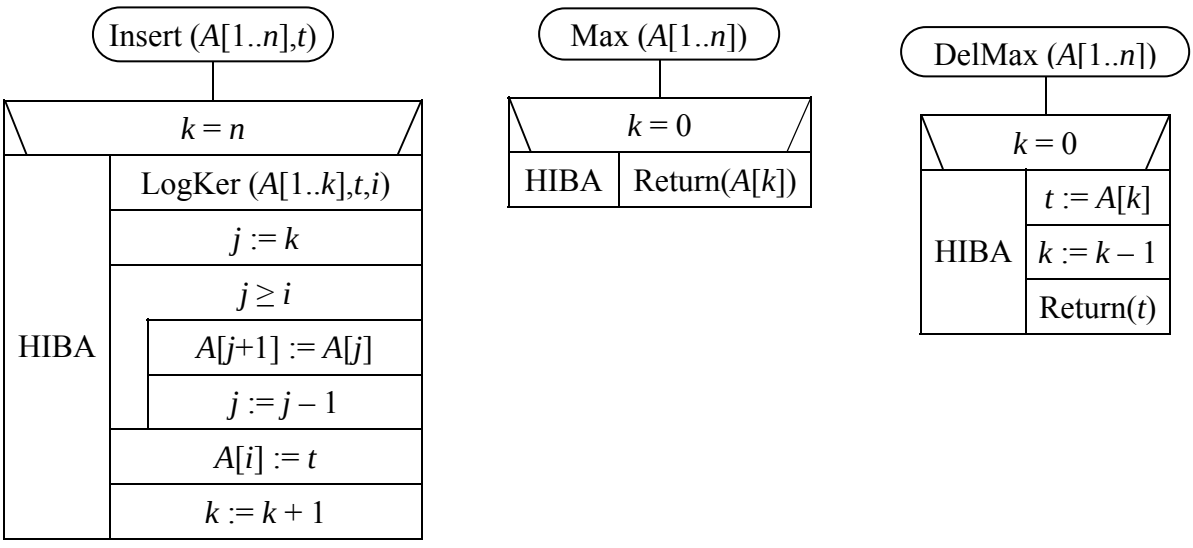


A műveletigények:

- Insert: változatlanul $T(n) = \Theta(1)$, mivel *maxh* karbantartása itt egyetlen (feltételes) értékadással megoldható.
- Max: $T(n) = \Theta(1)$, itt tehát javítottunk!
- DelMax: $T(n) = \Theta(n)$, mert a (eleve megjelölt) maximum törlése után a *maxh* karbantartásához újra végre kell hajtani egy MaxKer-t.

2. Megvalósítás: Rendezett tömbben helyezzük el az elemeket. A maximális elem mindig az utolsó.

Használjuk ki azt, hogy egy rendezett tömbben egy elemet logaritmikus (=bináris) kereséssel lehet megtalálni. Ha a keresett elem nincs a tömbben, akkor a logaritmikus keresés jelzi ezt és ebben az esetben a keresés indexeinek végső helyzete alapján meg lehet mondani, hogy hol lenne a keresett elem helye (HF!). Oda be is szúrhatjuk az elemet, ha a tőle jobbra lévőket eggyel jobbra léptetjük. Az alábbi LogKer ($A[1..k],t,i$) tehát úgy működik, hogy ha a t elem nincs $A[1..k]$-ban, akkor az i index értéke jelzi, hogy hol lenne a helye, azaz hová kell beszúrni.



Megjegyzés: Ez a speciális Insert már egy javítási kísérlet eredménye ahhoz a változathoz képest, amely lineárisan keresi meg a beszúrás helyét.

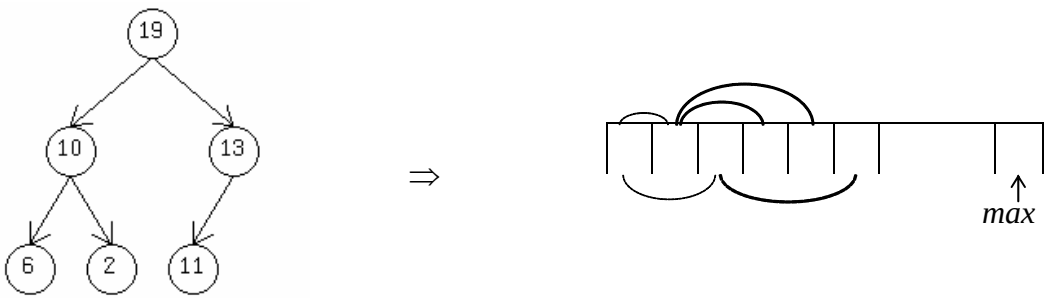
A műveletigényekre térve, itt az Insert miatt MT(n)-et, vagy AT(n)-et kell számolnunk. A logaritmikus keresés átlagos összehasonlítás száma kb. $log_2 n - 1$, tehát nagyságrendben $log\ n$, a léptetés pedig átlagosan $k/2$ elemre vonatkozik, ahol k a tömb aktuális elemszáma, ami pedig átlagos esetben $n/2$-nek vehető. A léptetések várható száma így kb. $n/4$, ami nagyságrendben lineáris.

- Insert: $AT(n) = \Theta(log\ n) + \Theta(n) = \Theta(n)$.
  $MT(n) = \Theta(n)$ (nyilvánvalóan)
- Max: $T(n) = \Theta(1)$.
- DelMax: $T(n) = \Theta(1)$.

Az Insert és a DelMax műveletek hatékonyságára pont fordított eredményt kaptunk, mint az előző reprezentációban. Ez a második ábrázolás érezhetően kevesebbet dolgozik, de nagyságrendben nincs köztük különbség.

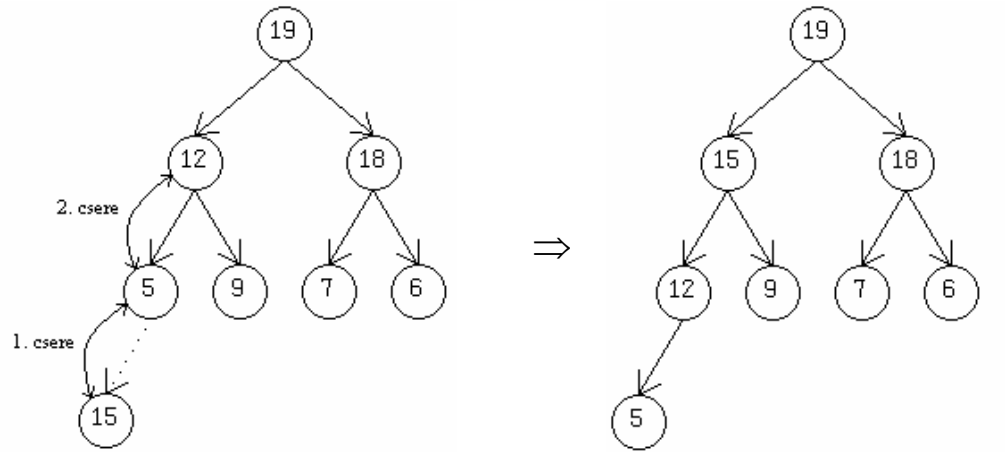
3. Megvalósítás: Egy kupac tulajdonságú fát töltünk fel az elemekkel. A kupac (heap) egy olyan majdnem teljes, balra tömörített bináris fa (tehát az utolsó szint jobb oldaláról esetleg hiányozhatnak elemek), amelyben minden belső elem nagyobb, vagy egyenlő, mint a gyermekei.

De a számítástechnikai megvalósítás ebben az esetben sem lesz pointeres, ajánlott a fát szintfolytonosan egy tömbbe rakni. (Jegyezzük meg: a heap-et mindig tömbben ábrázoljuk, nem pedig pointeresen!) Ha az eredetileg meglévő kapcsolatokat lánconként behúzzuk a vektor egyes elemei között, akkor láthatjuk, hogy a tömb sem nem rendezetlen, sem nem teljesen rendezett, hanem egy bizonyos kompromisszumos részben rendezettség áll fenn: az elemek az egyes láncok mentén rendezettek, mivel csökkenő sorozatokat alkotnak.



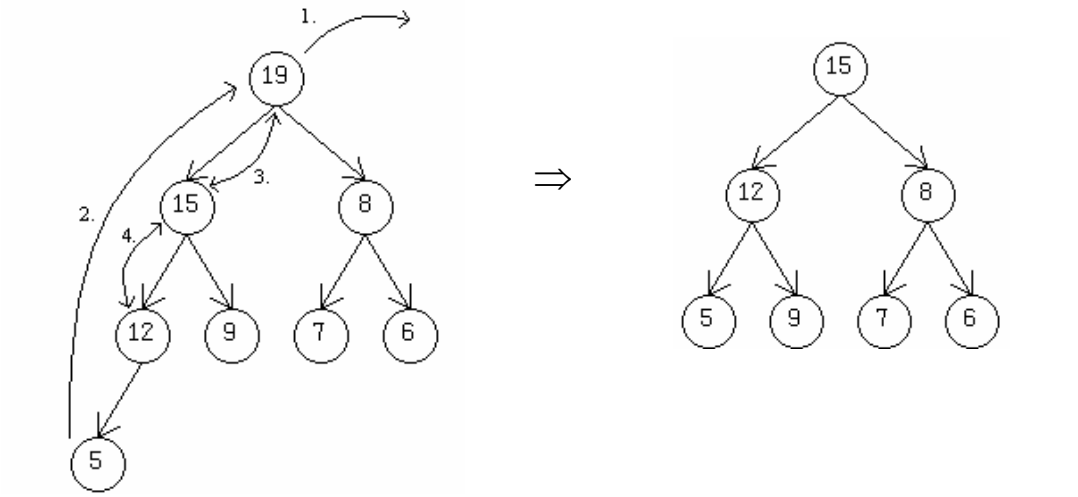
Az egyes típusműveleteket itt csak intuitív szinten adjuk meg, pontos megírásukra később kerül sor (lásd: kupacrendezés / heap sort).

- Insert: Az új csúcs beszúrásával meg kell tartanunk a kupac tulajdonságot, ezért az új csúcs az utolsó sorba, balról az első üres helyre, vagy ha az utolsó sor teljes, akkor egy új sor bal szélére kerül. De még rendbe kell hozni a csúcsok értékeinek viszonyát, tehát elemcseréket végzünk alulról a gyökér felé haladva a megfelelő élsorozaton (tömbnél a megfelelő láncon), ameddig szükséges. Nézzük ennek működését az alábbi kupacon, ha a 15-ös értéket szúrjuk be:



A műveletigány ekkor átlagos ill. legrosszabb esetben is a fa magasságával lesz egy nagyságrendben, ami n elemű fa esetében $\lfloor \log_2 n \rfloor$. Tehát $AT(n) = MT(n) = \Theta(\log n)$.

- Max: Ebben az esetben csak kiolvassuk a gyökérben lévő értéket.
- DelMax: A gyökérben lévő értéket kiolvassuk, majd a szintfolytonos bejárás során utolsóként bejárt (jobb alsó) értéket rakjuk a gyökérbe, az utolsó csúcs kitörlődik. A kupac tulajdonság megtartása érdekében a gyökérbe került értéket elemcserékkel lesüllyesztjük (mindig a nagyobb gyerekkel cserélve) a megfelelő helyre. Az alábbi kupacon láthatjuk ennek működését:



A műveletigény itt is a fa magasságával arányos: $AT(n) = MT(n) = \Theta(\log n)$.

Foglaljuk össze egy hatékonysági táblázatban a különböző megvalósítások jellemzőit:

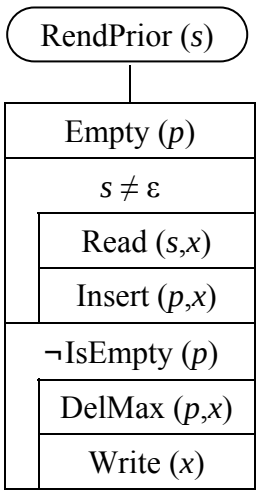
	Rendezetlen tömb 1.	Rendezetlen tömb 2.	Rendezett tömb	Kupac
Insert	$\Theta(1)$	$\Theta(1)$	$\Theta(\log n) + \Theta(n) = \Theta(n)$	$\Theta(\log_2 n)$
DelMax	$\Theta(n)$	$\Theta(n)$	$\Theta(1)$	$\Theta(\log_2 n)$
Max	$\Theta(n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$

A Max műveletre a javítás után már mindhárom reprezentációban konstans lépésszámot kaptunk.

A két fontosabb művelet az Insert és a DelMax a kupac reprezentációban kiegyensúlyozott módon hatékonyabb, mint az első kettőben. Ezt akkor látjuk igazán, ha egy olyan művelet sorozatot hajtunk végre, amelyben vegyesen szerepel beszúrás és a maximum törlése.

6.3. A prioritásos sor felhasználása rendezésre

Találtunk egy nagyon egyszerű és talán meglepő algoritmust, amely az említett két műveletet egyenlő számban hajtja végre működése során.



Az eljárás az *s* szekvenciális inputról beolvasott értékeket sorban beteszi egy (kezdetben üres) elsőbbségi sorba. Azután pedig sorban kiveszi az elemeket az elsőbbségi sorból. A kiírt sorozat nyílván rendezett lesz! Ez a rendezés minden bizonnyal meglepő, hiszen nincs benne egymásba ágyazott ciklus. Ennek az a magyarázata, hogy az algoritmus szintjén el van takarva előlünk a felhasznált adatszerkezet megvalósításának módja.

(Érdekes HF és vizsgakérdés annak a meggondolása, hogy milyen rendező algoritmust kapunk az egyes reprezentációk esetén!)

Elemezzük most ennek az algoritmusnak a műveletigényét mindhárom reprezentáció esetén! Ennek során a bevezetett műveletek egész sorát vesszük számba, ezáltal úgynevezett amortizációs elemzést hajtunk végre. Mivel csak nagyságrendekkel fogunk számolni, mindegy, hogy az összehasonlítások maximális vagy átlagos számát határozzuk meg, hiszen a két mennyiség mindig azonos nagyságrendű.

Az alábbi elemzés alapján joggal mondhatjuk, hogy az elsőbbségi sornak kupaccal megvalósítása hatékonyabb a másik két ábrázolási módnál!

Reprezentációtól függetlenül igaz, hogy

$$MT(n) = T_{Empty} + \sum_{k=0}^{n-1} MT_{Insert}(n) + \sum_{k=1}^{n} MT_{DelMax}(n)$$

és AT(n)-re is hasonló összefüggés igaz.

Számítsuk ki a fenti kifejezés nagyságrendjét a három reprezentációban.

1. Rendezetlen tömb

$$MT(n) = \Theta(1) + n\Theta(1) + n\Theta(n) = \Theta(n^2)$$

2. Rendezett tömb

$$MT(n) = \Theta(1) + n\Theta(n) + n\Theta(1) = \Theta(n^2)$$

3. Kupacos ábrázolás

$$MT(n) = \Theta(1) + 2\Theta(\log_2 1 + \log_2 2 + ... + \log_2 n)$$

Azt állítjuk, hogy ennek az összegzésnek az eredménye nagyságrendben $n \log n$-es, azaz:

$$MT(n) = \Theta(n \log n).$$

Bizonyítás:

A kiszámítandó összeget, mint a $\log_2 x$ függvény beírt téglalapjainak területösszegét, tekintsük integrál közelítő összegnek. Ekkor

$$\log_2 1 + \log_2 2 + ... + \log_2 n \approx \int_1^{n+1} \log_2 x \, dx = \frac{1}{\ln 2} \int_1^{n+1} \ln x \, dx =$$

$$= \frac{1}{\ln 2} \left[x \ln x - x \right]_1^{n+1} = \frac{1}{\ln 2} \left[(n+1)\ln(n+1) - (n+1) + 1 \right] \approx 1{,}44 \cdot n \ln n =$$

$$= n \log_2 n = \Theta(n \log n)$$