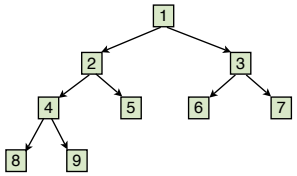
Kupac adatszerkezet

A bináris kupac egy majdnem teljes bináris fa, amely minden szintjén teljesen kitöltött kivéve a legalacsonyabb szintet, ahol balról jobbra haladva egy adott csúcsig vannak elemek. A fát egy tömbben reprezentáljuk, minden elem a szint szerinti bejárás szerinti sorszámának megfelelő eleme a tömbnek. A kupacot reprezentáló A tömbhöz két értéket rendelünk, hossz(A) a tömb mérete, kupacmeret(A) a kupac elemeinek a száma.

A kupac gyökere A[1], a szerkezeti kapcsolatok egyszerűen számolhatóak:

- $A[i]$ bal fia $A[2i]$
- $A[i]$ jobb fia $A[2i+1]$
- $A[i]$ apja $A[\lfloor i/2 \rfloor]$

A kupac minden gyökértől különböző elemére teljesül, hogy az értéke nem lehet nagyobb, mint az apjáé. Ennek következménye, hogy a kupac minden részfájára teljesül, hogy a gyökéreleme maximális.



1. ábra.

A=[1,2,3,4,5,6,7,8,9]

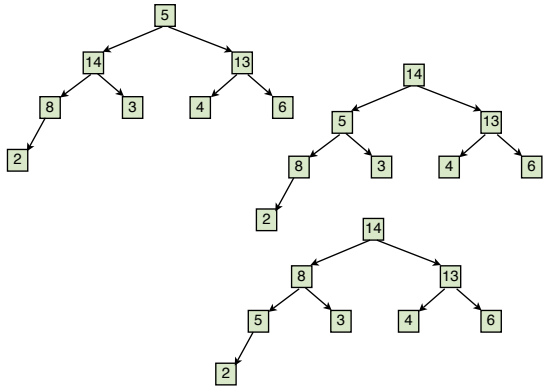
MAXIMUM-KUPACOL Eljárás

A MAXIMUM-KUPACOL eljárás helyreállítja az A[i] elemre a kupactulajdonságot. Az elemet süllyeszti cserékkel mindaddig, amíg a tulajdonság sérül.

```
MAXIMUM-KUPACOL(A,i)
l:=2i              //az A[i] elem bal fiának indexe
r:=2i+1            //az A[i] elem jobb fiának indexe
if l<=kupacmeret(A) and A[l]>A[i]
     then max:=l
     else max:=i
if r<=kupacmeret(A) and A[r]>A[max]
     then max:=r      // A[max] a három elem közül a legnagyobb
if max!=i
   then {Csere(A[i],A[max])
         MAXIMUM-KUPACOL(A,max)}
```

Példa

```
A=[5,14,13,8,3,4,6,2]
MAXIMUM-KUPACOL(A,1)
A=[14,5,13,8,3,4,6,2]
A=[14,8,13,5,3,4,6,2]
```



2. ábra.

Kupacrendezés

Elsőként a rendezendő elemek tömbjét egy kupaccá kell alakítani. Erre szolgál a // KUPACEPIT eljárás:

```
KUPACEPIT(A)
kupacmeret(A):=hossz(A)
for i=[hossz(A)/2] downto 1
    MAXIMUM-KUPACOL(A,i)
```

A kupacrendezés algoritmusa ekkor a következő

```
KUPACREND(A)
KUPACEPIT(A)
for i=hossz(A) downto 2
     {csere(A[1],A[i])
     kupacmeret(A):=kupacmeret(A)-1
     MAXIMUM-KUPACOL(A,1)}
```

Példa

```
A=[5,13,2,25,7,17,20,8,4]
KUPACEPIT(A):
  MAXIMUMKUPACOL(A,4):
    A=[5,13,2,25,7,17,20,8,4]
  MAXIMUMKUPACOL(A,3):
    A=[5,13,20,25,7,17,2,8,4]
  MAXIMUMKUPACOL(A,2):
```

```
    A=[5,25,20,13,7,17,2,8,4]
  MAXIMUMKUPACOL(A,1):
    A=[25,5,20,13,7,17,2,8,4]
    A=[25,13,20,5,7,17,2,8,4]
    A=[25,13,20,8,7,17,2,5,4]
A=[4,13,20,8,7,17,2,5,|25]
MAXIMUMKUPACOL(A,1):
  A=[20,13,4,8,7,17,2,5,|25]
  A=[20,13,17,8,7,4,2,5,|25]
A=[5,13,17,8,7,4,2,|20,25]
MAXIMUMKUPACOL(A,1):
  A=[17,13,5,8,7,4,2,|20,25]
A=[2,13,5,8,7,4,|17,20,25]
MAXIMUMKUPACOL(A,1):
   A=[13,2,5,8,7,4,|17,20,25]
   A=[13,8,5,2,7,4,|17,20,25]
A=[4,8,5,2,7,|13,17,20,25]
MAXIMUMKUPACOL(A,1):
   A=[8,4,5,2,7,|13,17,20,25]
   A=[8,7,5,2,4,|13,17,20,25]
A=[4,7,5,2,|8,13,17,20,25]
MAXIMUMKUPACOL(A,1):
   A=[7,4,5,2,|8,13,17,20,25]
A=[2,4,5,|7,8,13,17,20,25]
MAXIMUMKUPACOL(A,1):
   A=[5,4,2,|7,8,13,17,20,25]
A=[2,4,|5,7,8,13,17,20,25]
MAXIMUMKUPACOL(A,1):
   A=[4,2,|5,7,8,13,17,20,25]
A=[2,|4,5,7,8,13,17,20,25]
```

Műveletigény elemzése

A kupacrendezés során O(n) darab MAXIMUM-KUPACOL eljárást hajtunk végre.

Minden MAXIMUM-KUPACOL eljárás műveletigénye legfeljebb az aktuális fa magasságával arányos, azaz $O(\log n)$

Következésképpen a kupacrendezés műveletigénye $O(n \log n)$.

Priorítási sor megvalósítása kupaccal

- A maximális elemet a kupac gyökere tartalmazza
- A maximális elem törlését követően, a kupacot tároló tömb utolsó elemét a helyére rakva, a MAXIMUM-KUPACOL eljárást végrehajtva a kupactulajdonság helyreáll.
- A sorba művelet esetén az aktuális elemet a tömb végére helyezzük el, majd ha nagyobb az apjánál, akkor kicseréljük az apjával és az új helyén ugyanezt rekurzívan végrehajtjuk.

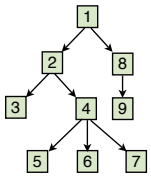
Megjegyzés: A kupacrendezés tulajdonképpen egy priorítási sorba rakja az elemeket, majd rendre kiveszi a maximális elemet.

Preorder bejárás

Fa bejárásán olyan algoritmust értünk, amelynek bemenete egy F fa és egy M művelet, és az algoritmus adott sorrendben pontosan egyszer végrehajtja az M műveletet a fa pontjaiban lévő adatokra.

A preorder bejárási sorrend azt jelenti, hogy F minden p és q pontjára ha q fia p-nek, vagy p bal-testvére q-nak, akkor p megelőzi q-t a bejárási sorrendben.

A bejárás első eleme a gyökér, utána az első fiú gyökerű részfát járjuk be rekurzívan, majd a második fiú részfáját, és így folytatva a gyökér utolsó fiának a részfáját.



3. ábra.

Egy rekurzív preorder bejárás

Feltesszük, hogy a fa a g gyökérpontja által van megadva elsőfiú testvér reprezentációval, és az M műveletet akarjuk minden ponton végrehajtani.

```
Preorder1(g,M)
if g=Nil then return //Üres fa
else
 {M(g)    //gyökéren végrehajtott művelet
  p:=g.Elsofiu
  while (p!=Nil)    // a gyerekeken sorbamenve rekurzív hívás
       {Preorder1(p,M)
        p:=p.Testver}}
```

Egy másik rekurzív preorder bejárás

Ismét feltesszük, hogy a fa a g gyökérpontja által van megadva elsőfiú testvér reprezentációval, és az M műveletet akarjuk minden ponton végrehajtani.

```
Preorder2(g,M)
if g=Nil then return //Üres fa
else
  {M(g)    //gyökéren végrehajtott művelet
  if g.Elsofiu!=Nil then Preorder2(g.Elsofiu,M)
  if g.Testver!=Nil then Preorder2(g.Tesver,M)}
```

Az első rekurzív hívás az g.Elsofiu pontból az elsőfiu és testvér kapcsolatok szerint elérhető pontokra végez helyes preorder bejárást, majd a második rekurzív hívás pedig a g.Testver pontból az elsőfiu és testvér kapcsolatok szerint elérhető pontokra végez helyes preorder bejárást, tehát a g gyökerű fa preorder bejárását kapjuk.

Kiskérdések

- Maximum kupacol algoritmus
- kupacrendezés algoritmusa (kupacepit kell, Maximum-kupacol nem)
- Kupacrendezés végrehajtása adott tömbön a közbenső kupacok megadásával
- rekurzív preorder bejárás algoritmus