Az adatkezelés szintjei

- Probléma szintje.
- Modell szintje.
- Absztrakt adattípus szintje.
- Absztrakt adatszerkezet szintje.
- Adatszerkezet szintje.
- Gépi szint.

Absztrakt adattípus

Az absztrakt adattípus egy (E,M) párral adható meg, ahol E az értékhalmaz, M a műveletek halmaza.
Fő tulajdonságok

- Nem ismert az adatokat tároló adatszerkezet.
- Nem ismertek a műveleteket megvalósító algoritmusok, a műveletek specifikációjukkal definiáltak.

Verem

Értékhalmaz: E $Verem = [a_1, \ldots, a_n : a_i \in E, i = 1, \ldots, n,]$ Műveletek: V : Verem, x : E

- $\{Igaz\}$ Letesit(V) $\{V = []\}$
- $\{V = V\}$ Megszuntet(V) $\{Igaz\}$
- $\{V = V\}$ Uresit(V) $\{V = []\}$
- $\{V = [a_1, \ldots, a_n]\}$ VeremBe(V,x) $\{V = [a_1, \ldots, a_n, x]\}$
- $\{V = [a_1, \ldots, a_n], n > 0\}$ VeremBol(V,x) $\{V = [a_1, \ldots, a_{n-1}], x = a_n\}$
- $\{V = V\}$ NemUres(V) $\{NemUres = (Pre(V) \neq [])\}$
- $\{V = [a_1, \ldots, a_n], n > 0\}$ Teteje(V,x) $\{x = a_n, V = Pre(V)\}$
- $\{V = [a_1, \ldots, a_n], n > 0\}$ Torol(V) $\{V = [a_1, \ldots, a_{n-1}]\}$

Megvalósítás: tömb, lánc, kombinált.

Lényeg: Az az elem kerül ki, amely utolsónak került be.

Sor

Értékhalmaz: E $Sor = [a_1, \ldots, a_n : a_i \in E, i = 1, \ldots, n,]$ Műveletek: S: Sor, x : E

- $\{Igaz\}$ Letesit(S) $\{S = []\}$
- $\{S = S\}$ Megszuntet(S) $\{Igaz\}$
- $\{S = S\}$ Uresit(S) $\{S = []\}$

- $\{S = [a_1, \ldots, a_n]\}$ SorBa(S,x) $\{S = [a_1, \ldots, a_n, x]\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ SorBol(S,x) $\{S = [a_2, \ldots, a_n], x = a_1\}$
- $\{S = [a_1, \ldots, a_n]\}$ Elemszam(S) $\{Elemszam = n\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ Elso(S,x) $\{x = a_1, S = Pre(S)\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ Torol(S) $\{S = [a_2, \ldots, a_n]\}$

Megvalósítás: tömb, lánc kombinált.

Lényeg: Az az elem kerül ki, amely elsőként került be.

Prioritási Sor

Értékhalmaz: E, E-n értelmezett a $\leq$ lineáris rendezési reláció $PriSor = S \subseteq E$. Műveletek: S : PriSor, x : E.

- $\{Igaz\}$ Letesit(S,$\leq$) $\{S = \emptyset\}$
- $\{S = S\}$ Megszuntet(S) $\{Igaz\}$
- $\{S = S\}$ Uresit(S) $\{S = \emptyset\}$
- $\{S = S\}$ SorBa(S,x) $\{S = Pre(S) \cup \{x\}\}$
- $\{S \neq \emptyset\}$ SorBol(S,x) $\{x = \min(Pre(S)), Pre(S) = S \cup \{x\}\}$
- $\{S = \{a_1, \ldots, a_n\}\}$ Elemszam(S) $\{Elemszam = n\}$
- $\{S \neq \emptyset\}$ Elso(S,x) $\{x = \min(Pre(S)), Pre(S) = S\}$
- $\{S \neq \emptyset\}$ Torol(S) $\{S = Pre(S) \setminus \{\min(Pre(S))\}\}$

Prioritási sor megvalósítások:
PriSorT: kupac-tömb adatszerkezettel,
PriSorR: kupac-tömb adatszerkezettel, a rendezési reláció konstruktor paraméter.

Lényeg: Az az elem kerül ki, amely legkisebb.

Példa

Probléma: Halmaz k-adik legkisebb elemének kiválasztása.
Bemenet: $H = \{a_1, \ldots, a_n\}$, különböző egész számok, $1 \leq k \leq n$.
Kimenet: $a_i \in H$, amelyre $|\{x : x \in H, x \leq a_i\}| = k$,
Feltételezzük, hogy az input egy T tömbben van eltárolva. Ekkor a megoldás.

```
Valaszt
Letesit S:Prisor
for i:=1 to n
   Sorba(S,T[i])
for i=1 to k
   Sorbol(S,x)
Return x
```

Halmaz

Értékhalmaz: E, Halmaz= $H \subseteq E$ Műveletek: H : Halmaz, x : E, I : Iterator

- $\{Igaz\}$ Letesit(H) $\{H = \emptyset\}$
- $\{H = H\}$ Megszuntet(H) $\{Igaz\}$
- $\{H = H\}$ Uresit(H) $\{H = \emptyset\}$
- $\{H = \{a_1, ..., a_n\}\}$ Elemszam(H) $\{Elemszam = n\}$
- $\{H = H\}$ Eleme(H,x) $\{Eleme = (x \in Pre(H))\}$
- $\{H = H\}$ Bovit(H,x) $\{H = Pre(H) \cup \{x\}\}$
- $\{H = H\}$ Torol(H,x) $\{H = Pre(H) \setminus \{x\}\}$
- $\{H = H\}$ IterKezd(H, I) $\{\}$
- $\{I = I\}$ IterAd(I,x) $\{\}$
- $\{I = I\}$ IterVege(I) $\{\}$

Megvalósítás: bitvektor, tömb, lánc, hasítótábla.

Iterátor használata

Pascal megvalósítás:

```
IterKezd(H,I);
While Not IterVege(I) Do Begin
   IterAd(I,x);
   M(x);
End;
```

java megvalósítás:

```
Iterator<E> I=H.iterator(); //IterKezd(H,I)
while (I.hasNext()){ //!IterVege(I)
   x=I.next(); //IterAd(I,x)
   M(x);
}
```

vagy kiterjesztett for-ciklussal:

```
for (E x:H)
  M(x);
```

Példák további absztrakt adattípusokra

- Lista
- Kétirányú Lista

- Tömb
- Sorozat
- Függvény
- Reláció

Absztrakt adatszerkezetek

Absztrakt adatszerkezet egy A = (M,R,Adat) rendezett hármas, ahol

- M az absztrakt memóriahelyek, cellák halmaza.
- $R = \{r_1, \ldots, r_k\}$ a cellák közötti szerkezeti kapcsolatok.
- Adat a cellák adattartalmát megadó (parciális) függvény, $c \in M$ esetén Adat(c) a c cellában lévő adat.

$f : A \to B$ parciális függvény esetén, ha $x \in A$ elemre f nem értelmezett, akkor az $f(x) = \perp$ jelölést alkalmazzuk, tehát mint ha $f : A \to B \cup \{\perp\}$ mindenütt értelmezett (totális) függvény lenne.

Lánc

A = (M,R,Adat) lánc, ha $R = \{\text{kovet}\}$, ahol kovet $M \to M$ parciális függvény, amelyre teljesül:

$(\exists \text{fej} \in M)(\forall x \in M)(\exists! k \geq 0)(x = kovet^k(fej))$

Nyilvánvalóan pontosan egy olyan $c \in M$ cella van, amelyre $kovet(c) = \perp$, ezt láncvégnek nevezzük. A lánc hosszán a cellák n számát értjük.

Ha $n > 0$, akkor $kovet^{(n-1)}(fej) =$láncvég.

Az A = (M,R,Adat) lánc rendezett lánc a $\leq$ relációra nézve , ha

$(\forall x \in M)(Adat(x) \leq Adat(kovet(x)))$

Az A = (M,R,Adat) körlánc, ha $R = \{kovet\}$, ahol kovet $M \to M$ (totális) függvény, amelyre teljesül: $(\forall x, y \in M)(\exists k \geq 0)(y = kovet^k(x))$

Kétirányú lánc

Az A = (M,R,Adat) kétirányú lánc, ha $R = \{kovet, eloz\}$, ahol kovet, eloz $M \to M$ parciális függvények, hogy: $(M, \{kovet,\}, Adat)$ és $(M, \{eloz\}, Adat)$ mindegyike lánc, és

$(\forall x \in M)(x = eloz(kovet(x)) = kovet(eloz(x)))$

Az A = (M,R,Adat) kétirányú körlánc, ha $R = \{kovet, eloz\}$, ahol kovet, eloz $M \to M$ (teljesen értelmezett) függvények, hogy: $(M, \{kovet,\}, Adat)$ és $(M, \{eloz\}, Adat)$ mindegyike körlánc, és

$(\forall x \in M)(x = eloz(kovet(x)) = kovet(eloz(x)))$

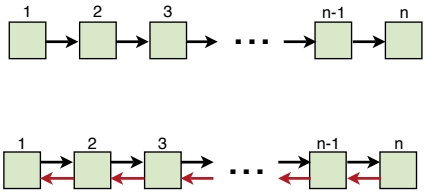
Fa

A = (M,R,Adat) nem-rendezett fa, ha $R = \{r\}$, $r \in M \times M$ bináris reláció, és van olyan $g \in M$, hogy a $G = (M, r)$ irányított gráfban bármely $x \in M$-re pontosan egy (g, x) út vezet. g-t a fa gyökerének nevezzük.

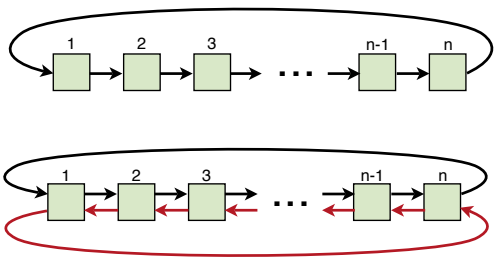
Vegyük észre, hogy ha minden él irányát megfordítjuk, akkor egy olyan $f : M \to M$ parciális függvény gráfját kapjuk, amely csak g-re nincs értelmezve, és f körmentes.

Tehát minden nem-rendezett fa egyértelműen megadható egy olyan $Apa : M \to M$ parciális függvénnyel, amelyre teljesül a következő feltétel.

$(\exists g \in M)((Apa(g) = \perp) \wedge (\forall x \in M)(\exists! k \geq 0)(Apa^k(x) = g))$



1. ábra.



2. ábra.

Rendezett fa

Legyen A=(M,R,Adat) olyan absztrakt adatszerkezet, hogy $R = \{f\}$, $f : M \to (M \cup \{\perp\})^*$.

Tehát $x \in M$, $f(x) = (y_1, ..., y_k)$, ahol $y_i \in (M \cup \{\perp\})$, $i = 1, \ldots, k$.

Minden $i > 0$ természetes számra és $x \in M$-re legyen $f_i(x)$ az $f(x)$ i-edik komponense. Tehát $f_i(x)$ az x i-edik fiát adja. Ha $f_i(x) = \perp$, akkor hiányzik az i-edik fiú. Az A adatszerkezetet fának nevezzük, ha van olyan $g \in M$, hogy teljesül az alábbi négy feltétel:

- $(\forall x \in M)(\forall i)(g \neq f_i(x))$ a gyökér nem fia senkinek,
- $(\forall y \neq g \in M)(\exists x \in M)(\exists i)(f_i(x) = y)$ minden pont, ami nem gyökér fia valakinek
- $(\forall x, y \in M)(\forall i, j)(f_i(x) = f_j(y)) \Rightarrow (x = y \wedge i = j))$ minden pontnak legfeljebb egy apja van
- $(\forall x \neq g \in M)(\exists i_1, \ldots, i_k)(x = f_{i_k} \ldots f_{i_1}(g))$ minden pontba vezet út a gyökérből.

Fa megadása elsőfiú-testvér kapcsolattal

Legyen A = (M,R,Adat) olyan absztrakt adatszerkezet, hogy $R = \{e, t\}$, $e, t : M \to M \cup \{\perp\})$. Legyen $f_i(x) = t^{i-1}(e(x))$ $i > 0$. Az A adatszerkezet fa, ha az f_i függvények teljesítik a megfelelő feltételeket.

Fák algebrai definíciója

Legyen E tetszőleges adathalmaz. Az E-feletti fák $Fa(E)$ halmaza az a legszűkebb halmaz, amelyre teljesül az alábbi három feltétel:

- $\perp \in Fa(E)$
- $(\forall a \in E) a \in Fa(E)$
- $(\forall a \in E)(\forall t_1, \ldots, t_k \in Fa(E))(a(t_1, ..., t_k) \in Fa(E))$

Adatszerkezetek

Egy A = (M,R,Adat) absztrakt adatszerkezet megvalósítása:

- Konkrét memória allokálás az M-beli absztrakt memória cellák számára.
- Az R szerkezeti kapcsolatok ábrázolása.
- Alapműveletek algoritmusainak megadása.

Belső adatszerkezet

A cellákat a főtárban lefoglalt memóriamezők tárolják. Minden cellát a számara lefoglalt memóriamező kezdőcíme azonosít.

Külső adatszerkezet

A cellákat külső tárolón (lemez) fájl tárolja. Minden cellát egy (F, p) pár azonosít, ahol F a tároló fájl azonosítója és p az F fájlon belüli rekordsorszám.

Elosztott adatszerkezet

Az egyes cellákat különböző számítógépeken tárolhatjuk. Egy cella azonosításához egy (G,F, p) hármast kell megadni, ahol G a hálózatba kapcsolt számítógép (adott hálózati protokoll szerinti) azonosítója, F a fájl azonosítója és p a fájlon belüli rekordsorszám.

Endogén ábrázolás

Az adatot és a szerkezeti kapcsolatot ugyanaz a cella tartalmazza.

Exogén ábrázolás

Külön cella tartalmazza az adatot és a szerkezeti kapcsolatokat. A szerkezeti kapcsolatot tartalmazó cellában a megfelelő adatra mutató hivatkozást tároljuk. Objektum orientált programozási nyelv esetén az adatszerkezetek ábrázolása alapvetően exogén. Például a java esetén csak akkor alkalmazható endogén ábrázolás, ha az adatok típusa elemi típus (int, long, float, double, char, boolean).

Heterogén ábrázolás

Egyes cellák csak szerkezeti kapcsolatot tartalmaznak, mások tartalmazhatnak adatot és a szerkezeti kapcsolatokat.

Statikus ábrázolás

A memória lefoglalás statikus tömbbel történik. Minden cellát tömbbeli indexe azonosít. Hiányzó szomszédot a 0 (vagy -1) index azonosítja.

Dinamikus ábrázolás

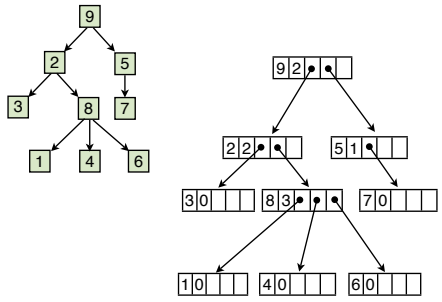
A cellák számára dinamikusan foglalunk memóriát, minden cellát pointer érték - memóriacím - azonosít. A hiányzó kapcsolatot a null (nil) pointer érték ábrázolja. A szerkezeti kapcsolatot, mint pointer értéket a cellában tároljuk.

Szerkezeti kapcsolat számítása

A szerkezeti kapcsolat esetenként megadható számítási eljárással is, nincs szükség a szerke- zeti kapcsolat tárolására. Minden x cellára és r szerkezeti kapcsolatra és adott i-re kiszámítható x-nek r-szerinti i-edik szomszédja. Például, teljes bináris fák esetén tömbben tárolva a bal és jobb fiú indexe számolható.

Kapcsolati tömb

Kapcsolati tömb esetén egy pont fiaira mutatókat tárolunk statikusan, és tároljuk a fiúk számát is.



3. ábra.

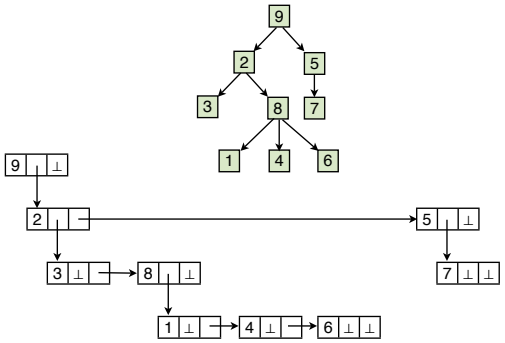
A kapcsolati tömb előnye:
Minden pont i-edik fia közvetlenül (konstans időben) elérhető.
A kapcsolati tömb hátránya:
Statikus, ezért nem lehet konstans időben bővíteni és törölni. Nagy memóriaigény.

Kapcsolati lánc esetén egy pont fiaira mutatókat dinamikusan tároljuk láncban.

Elsőfiú Apa testvér

Elsőfiú tesvér ábrázolás esetén a pont az adaton kívül két mutatót tartalmaz, egyet a közvetlen testvérére egyet az elsőfiára.

Elsőfiú Apa tesvér ábrázolás esetén a pont az adaton kívül három mutatót tartalmaz, egyet a közvetlen testvérére egyet az elsőfiára, egyet az apjára.



4. ábra.

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 $A[\lfloor i/2 \rfloor]$

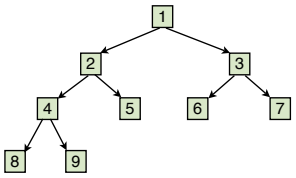
A kupac minden gyökértől különböző elemére teljesül, hogy az értéke nem lehet nagyobb, mint az apjáé. Ennek következménye, hogy a kupac minden részfájára teljesül, hogy a gyökéreleme maximális.

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]
```

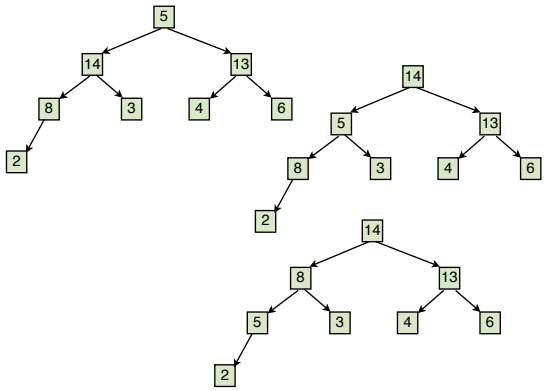


5. ábra.

```
     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]
```



6. 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.

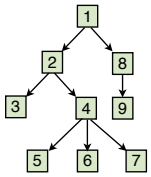
Megjegyzés: A kupacrendezés tulajdonképpen egy priorítási sorba rakja az elemeket, majd rendre kiveszi a maximális elemet.

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.



7. 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.

Nemrekurzív preorder bejárás veremmel

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

```
PreorderV(g,M)       // Preorder bejárás veremmel.
if (g=nil) then return  //Üres fa
Letesit(V:Verem)
VeremBe(V,g)
while (NemUres(V))
    {VeremBol(V,p)
    M(p)
    if p.Testver!=nil then Verembe(V,p.Testver)
    if p.Elsofiu!=nil then VeremBe(V,p.Elsofiu)}
```

Példa: V=[1],p=1,V=[],M(1),V=[2],p=2,V=[],M(2),V=[8],V=[8,3],p=3,V=[8],M(3),V=[8,4], p=4, V=[8], M(4),V=[8,5],p=5,V=[8],M(5),V=[8,6],p=6,V=[8], M(6),V=[8,7],p=7,V=[8],M(7),p=8,V=[], M(8),V=[9],p=9,V=[], M(9)

Helyesség

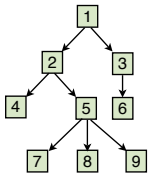
Tekintsük a while ciklus végrehajtásának egy adott pillanatát. Legyen $B = \{p1, \ldots, p_k\}$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a veremből kikerültek. Legyen $V = \{q_1, \ldots, q_m\}$ a V verem tartalma, ezek az aktív pontok. A következő négy állítás teljesül.

- Az B sorozatban a pontok helyes preorder sorrendben vannak.
- A preorder bejárásban p_k-t közvetlenül q_m követi.

- A preorder sorrendben q_i megelőzi q_{i-1}-et $(i = 2, \ldots, m)$.
- $B \cap V = \emptyset$ és a fa bármely $p \notin B$ pontjára, pontosan egy olyan $q \in V$ pont van, hogy p leszármazottja q-nak az elsőfiú-testvér fában.

Szintszerinti bejárás

Egy F fa bármely két p és q pontjára p akkor és csak akkor előzi meg q-t a szintszerinti bejárásban, ha $d(p) < d(q)$ vagy $d(p) = d(q)$ és p balrább esik, mint q.



8. ábra.

Az algoritmus

```
SzintBejar(g,M)
if (g=nil) then return  /Üres fa
Letesit(S:Sor)
SorBa(S,g)
while (Elemszam(S)!=0)
   {SorBol(S,p)
   M(p)
   p=p.Elsofiu;
   while (p!=nil)    //p gyerekeit berakjuk S-be
       {SorBa(S,p)
        p=p.Testver}}
```

Példa: S=[1], p=1,S=[],M(1),p=2,S=[2],p=3,S=[2,3],p=Nil,p=2,S=[3],M(2),p=4,S=[3,4], p=5, S=[3,4,5],p=Nil,p=3,S=[4,5],p=Nil,p=4,S=[5],M(4),p=5,S=[],M(5),p=Nil,p=6,S=[6],M(3),p=7, S=[6,7],p=8,s=[6,7,8],p=9,S=[6,7,8,9],p=Nil,p=6,S=[7,8,9],M(6),p=Nil,p=7,S=[8,9],M(7), p=Nil,p=8, S=[9],M(8),p=Nil,p=9,S=[],M(9),p=nil.

Helyesség

Tekintsük a külső while ciklus végrehajtásának egy adott pillanatát. Legyen $B = \{p_1, \ldots, p_k\}$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a sorból kikerültek. Legyen $S = \{q_1, \ldots, q_m\}$ a S sor aktuális tartalma, ezek az aktív pontok. A következő négy állítás teljesül.

- Az B sorozatban a pontok szintszerinti sorrendben vannak.
- Az S sorban a pontok szintszerinti sorrendben vannak.

- $d(q_m) \leq d(q_1) + 1$
- A szintszerinti bejárásban p_k megelőzi q_1-et.

Bejárás Adagolóval

Ha nem fontos a szintszerinti bejárási sorrend, a fenti algoritmusban az aktív pontok tárolására minden olyan absztrakt adattípus használható lenne, amely rendelkezik az alábbi műveletekkel.

Értékhalmaz: E $Adagol = A \subseteq E$ Műveletek: A: Adagoló, x : E

- $\{Igaz\}$ Letesit(A) $\{A = \emptyset\}$
- $\{A = A\}$ Megszuntet(A) $\{Igaz\}$
- $\{A = A\}$ Uresit(A) $\{A = \emptyset\}$
- $\{S = [a_1, \ldots, a_n\}$ Betesz(A,x) $\{A = Pre(A) \cup \{x\}$
- $\{A \neq \emptyset\}$ Kivesz(A,x) $\{x \in Pre(A) \wedge Pre(A) = A \cup \{x\}\}$
- $\{A = A\}$ Elemszam(A) $\{Elemszam = |A|\}$