1. Absztrakt adattípusok

Az adatkezelés szintjei:

1. Probléma szintje.
2. Modell szintje.
3. Absztrakt adattípus szintje.
4. Absztrakt adatszerkezet szintje.
5. Adatszerkezet szintje.
6. Gépi szint.

1.1. definíció. *Absztrakt adattípus:* $A = (E, M)$

1. E*: értékhalmaz*
2. M*: műveletek halmaza*

"Absztrakt" jelző jelentése:

i. *Nem ismert az adatokat tároló adatszerkezet.*

ii. *Nem ismertek a műveleteket megvalósító algoritmusok, a műveletek specifikációjukkal definiáltak.*

2. Absztrakt adatszerkezetek

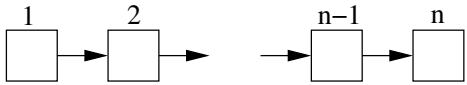
Absztrakt adatszerkezet egy $\mathbb{A} = (M, R, Adat)$ *rendezett hármas, ahol*
1. M *az absztrakt memóriahelyek, cellák halmaza.*
2. $R = \{r_1, \ldots, r_k\}$ *a cellák közötti szerkezeti kapcsolatok.*
3. $Adat$ *a cellák adattartalmát megadó (parciális) függvény,* $c \in M$ *esetén* $Adat(c)$ *a* c *cellában lévő adat.*
***Jelölések**:*
$f : A \to B$ *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.*
$f : E \to E$ *függvény és* $k \geq 0$ *esetén* f^k *az* f *függvény* k*-adik iteráltja:*
$f^0(x) = x$,
$f^{k+1}(x) = f(f^k(x))$.
Legyen $f : H \to H$ *tetszőleges (parciális) függvény. Az* f *függvény gráfja az a* $G_f = (H, E_f)$ *gráf, ahol*
$E_f = \{(x, f(x)) : x \in H \wedge f(x) \neq \perp\}$.
H *(véges) halmaz elemeiből képzett (véges) sorozatok halmazát* H^**-al jelöljük, azaz*
$H^* = \{\langle x_1, \ldots, x_n \rangle : x_i \in H, i = 1, \cdots, n, 0 \leq n\}$, *az üres sorozatra a* λ *jelölést is használjuk.*
$\langle x_1, \ldots, x_n \rangle \in H^*$ *esetén a sorozatot* $x_1.x_2 \ldots x_n$ *alakban is írhatjuk, tehát elhagyjuk a zárójeleket és az elemeket* . *(pont) választja el.*

2.1. Lánc

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

$$(\exists fej \in M)(\forall x \in M)(\exists! k \geq 0)(x = követ^k(fej)) \tag{1}$$

Nyilvánvalóan pontosan egy olyan $c \in M$ *cella van, amelyre* $követ(c) = \perp$, *ezt láncvégnek nevezzük.*
A lánc hosszán a cellák n *számát értjük. Ha* $n > 0$, *akkor* $követ^{(n-1)}(fej) = láncvég$



1. ábra. Lánc absztrakt adatszerkezet szemléltetése.

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

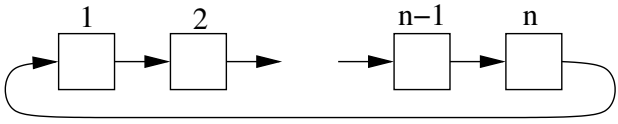
$$(\forall x \in M)(Adat(x) \leq Adat(követ(x)))$$

2.2. Körlánc

$\mathbb{A} = (M, R, Adat)$ *körlánc, ha* $R = \{követ\}$,
követ : $M \to M$ *(totális) függvény, amelyre teljesül:*

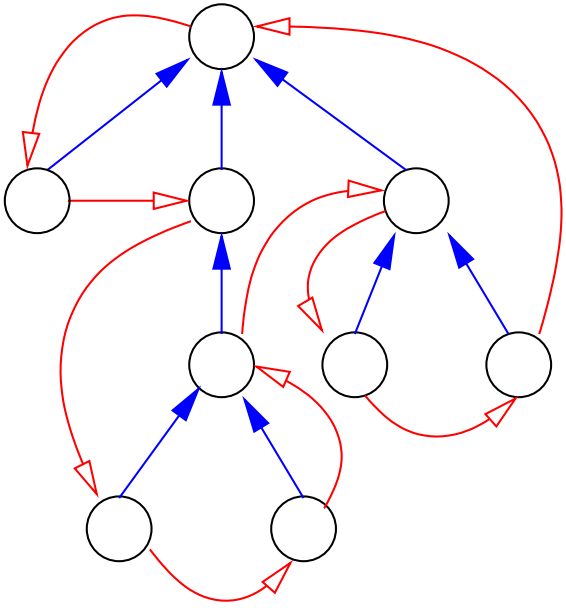
$$(\forall x, y \in M)(\exists k \geq 0)(y = követ^k(x)) \tag{2}$$

A (2) feltétel nyilvánvalóan egyenértékű azzal, hogy bármely $x \in M$*-re az* $\langle x = x_0, x_1, \ldots, x_{n-1} \rangle$



2. ábra. Körlánc absztrakt adatszerkezet szemléltetése.

sorozat, ahol $x_i = követ(x_{i-1}), i = 1, \ldots, n-1$*, az* M *halmaz elemeinek egy ciklikus permutációja.*



3. ábra. Absztrakt adatszerkezet két szerkezeti kapcsolattal

2.3. Általános absztrakt adatszerkezet

Olyan $\mathbb{A} = (M, R, Adat)$ *rendezett hármas, ahol*

1. M *az absztrakt memóriahelyek, cellák halmaza.*

2. $R = \{r_1, \ldots, r_k\}$ *a cellák közötti szerkezeti kapcsolatok,* $r_i : M \to (M \cup \{\perp\})^*$

3. $Adat : M \to E$ *parciális függvény, a cellák adattartalma.*

$x \in M$, $r \in R$ *és* $r(x) = \langle y_1, \ldots, y_i, \ldots, y_k \rangle$ *esetén az* x *cella* r *kapcsolat szerinti szomszédai* $\{y_1, \ldots, y_k\}$, y_i *pedig az* x *cella* i*-edik szomszédja.*

Ha $y_i = \perp$*, akkor azt mondjuk, hogy* x*-nek hiányzik az* r *szerinti* i*-edik szomszédja.*

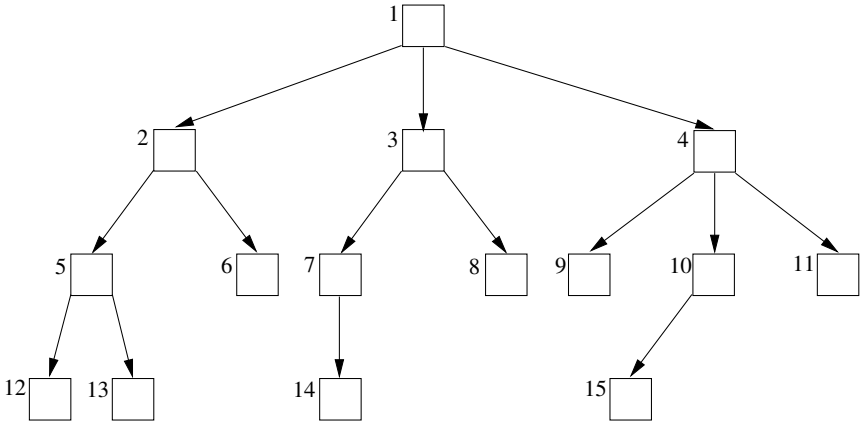
Különbségek a gráf és általános absztrakt adatszerkezetek között.

1. A gráf csak egy szerkezeti kapcsolatot ad meg.

2. Gráf esetén a szomszédok halmaza nem rendezett.

3. Gráfban nem tudunk hiányzó kapcsolatot megadni.

2.4. Fa

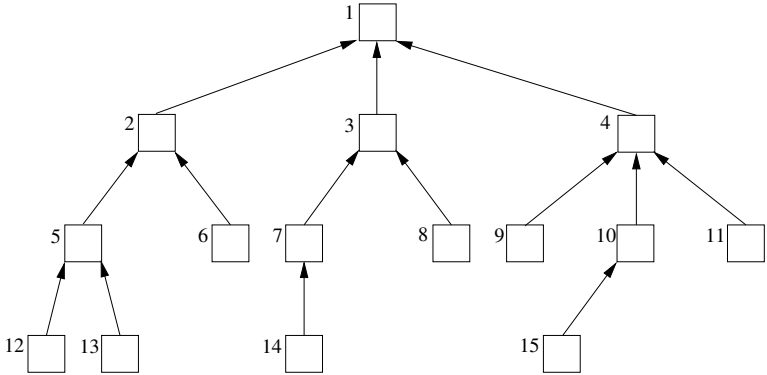


4. ábra. Fa absztrakt adatszerkezet szemléltetése.

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

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

Fordítva is igaz, minden olyan $f : M \to M$ *parciális függvény, amely pontosan egy* $g \in M$*-re*



5. ábra. Fa adatszerkezet megadása fiú-apa kapcsolattal.

nem definiált, a függvény gráfjának transzponáltja,

$$\{(f(x),x) : x \in M \wedge f(x) \neq \perp\} \subseteq M \times M \tag{3}$$

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

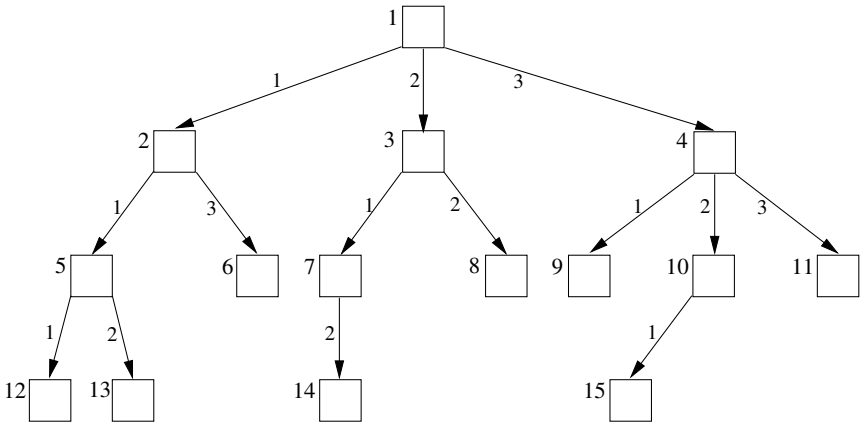
$$(\exists g \in M)((Apa(g) = \perp) \wedge (\forall x \in M)(\exists! k \geq 0)(Apa^k(x) = g)) \tag{4}$$

Ha $y = Apa(x)$, akkor azt mondjuk, hogy x apja y és y-nak fia x.

Az így megadott szerkezeti kapcsolat nem definiál rendezettséget adott x cella $\{y : Apa(y) = x\}$ fiainak halmazán.

Gyakran szükség van olyan szerkezeti kapcsolatra, ahol a fiúk sorrendje is lényeges. Továbbá, előfordulhat, hogy egy cellának lehetne i-edik fia, de ez a kapcsolat hiányzik.

Például, a következő ábrán olyan fa látható, amelynél a fiúk sorrendjét is megadtuk. A 2. cellának két fia van, de ezek az 1. és 3. sorszámúak, és a 2. fiú hiányzik.



6. ábra. Fa absztrakt adatszerkezet hiányzó kapcsolattal.

Legyen $\mathtt{A} = (M, R, Adat)$ *olyan absztrakt adatszerkezet, hogy* $R = \{f\}$, $f : M \to (M \cup \{\bot\})^*$.
$x \in M$, $f(x) = \langle y_1, \ldots, y_k \rangle$, $y_i \in M \cup \{\bot\}$, $i = 1, \ldots, k$.
Minden $i > 0$ *természetes számra definiáljuk az* $f_i : M \to M \cup \{\bot\}$ *függvényt a következőképpen:*

$$f_i(x) = \begin{cases} y_i & ha\ f(x) = \langle y_1, \ldots, y_k \rangle\ \acute{e}s\ i \leq k \\ \bot & ha\ i > k \end{cases} \tag{5}$$

Tehát $f_i(x)$ *az* x i*-edik fiát adja. Ha* $f_i(x) = \bot$*, akkor hiányzik az* i*-edik fiú. Az* $\mathtt{A}$ *adatszerkezetet fának nevezzük, ha van olyan* $g \in M$*, hogy teljesül az alábbi négy feltétel:*

$$(\forall x \in M)(\forall i)(g \neq f_i(x)) \tag{6}$$

$$(\forall y \neq g \in M)(\exists x \in M)(\exists i)(f_i(x) = y) \tag{7}$$

$$(\forall x, y \in M)(\forall i, j)(f_i(x) = f_j(y) \Rightarrow (x = y \wedge i = j)) \tag{8}$$

$$(\forall x \neq g \in M)(\exists i_1, \ldots, i_k)(x = f_{i_k} \ldots f_{i_1}(g)) \tag{9}$$

A (6) feltétel azt fejezi ki, hogy a gyökér nem fia egyetlen pontnak sem.
A (7) feltétel szerint minden, gyökértől különböző pont fia valamely pontnak.
A (8) feltétel azt mondja, hogy minden pontnak legfeljebb egy apja lehet.
A (9) feltétel azt mondja, hogy minden pont elérhető a gyökérből.

Mivel M *véges halmaz, ezért a 2. és 3. feltételből következik, hogy bármely ponthoz pontosan egy út vezet a gyökértől.*

Mivel minden cellához pontosan egy út vezet a gyökértől, ezért a cellák azonosíthatók a hozzájuk vezető (egyetlen) úttal. Például, a 13. cella az 1.1.2 *a 15. a* 3.2.1 *sorozattal azonosítható. Tehát egy fát megadhatunk úgy is, hogy megadjuk minden pontjához vezető utat.*

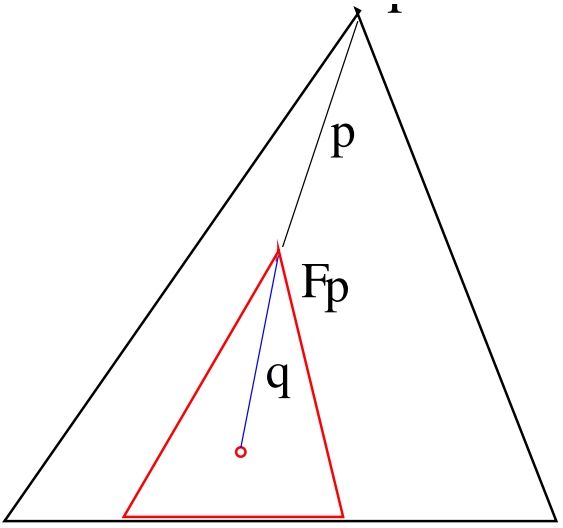
Legyen $F \subseteq \mathbb{N}^*$. Azt mondjuk, hogy F fatartomány, ha F zárt a prefix képzésre, azaz

$$(\forall p \in \mathbb{N}^*)(\forall i \in \mathbb{N})(p.i \in F \Rightarrow p \in F) \tag{10}$$

A definícióból következik, hogy a λ üres sorozat eleme F-nek, ami az F fa gyökere.
F a szerkezeti kapcsolatot is definiálja a következőképpen:
$\{(p, p.i) : p, p.i \in F \wedge i \in \mathbb{N}\}$

Fákkal kapcsolatos definíciók és jelölések

Legyen $F \subseteq \mathbb{N}^*$ fatartomány (fa), $p, q \in F$ és $i, j \in \mathbb{N}$.
F pontjainak száma $|F|$, azaz az F halmaz számossága.
Ha $q = p.i \in F$, akkor a q pont i-edik fia p-nek és p apja q-nak.
A p pont foka a fiainak száma, azaz $fok(p) = |\{k : p.k \in F\}|$.
Ha $p.i \in F \wedge p.j \in F$ akkor $p.i$ és $p.j$ testvérek.
Pontosabban, ha $i < j$ akkor $p.i$ bal-testvére $p.j$-nek és $p.j$ jobb-testvére $p.i$-nek.
Ha $j = i + 1$ akkor $p.j$ közvetlen jobb-testvére $p.i$-nek és $p.i$ közvetlen bal-testvére $p.j$-nek.
A q pont leszármazottja p-nek, ha van olyan $u \in \mathbb{N}^*$, hogy $q = p.u$.
A q pont őse p-nek, ha van olyan $u \in \mathbb{N}^*$, hogy $p = q.u$.
A p pont levél pont, ha nincs fia, azaz $(\forall k \in \mathbb{N})(p.i \notin F)$.
Ha $p.i \notin F$ de van olyan $j \in \mathbb{N}$, hogy $i < j$ és $p.j \in F$, akkor azt mondjuk, hogy p-nek hiányzik az i-edik fia.
F p-gyökerű részfája: $F_p = \{q \in \mathbb{N}^* : p.q \in F\}$.
p mélysége az F fában a gyökértől p-ig vezető út pontjainak száma; $d(p) = |p| + 1$.
Az F fa magassága; $h(F) = 0$, ha $F = \emptyset$, egyébként $h(F) = max(d(p) : p \in F)$.



7. ábra. Az F fa p gyökerű részfa.

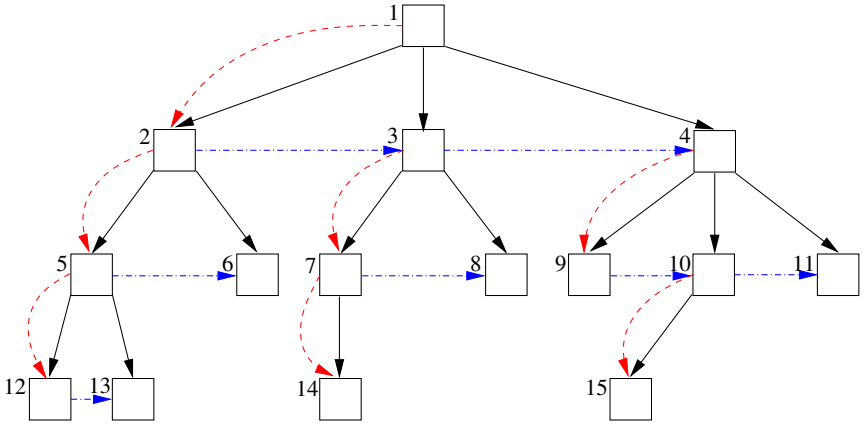
Rendezett fa megadható úgy is, hogy minden gyökértől különböző p pontjára megadjuk a q apját, és azt, hogy p az apjának hányadik fia. Tehát ekkor a szerkezeti kapcsolat egy $f : M \to M \times \mathbb{N}$ parciális függvény lesz, amely pontosan egy $g \in M$-re definiálatlan.
Ha $f(p) = (q,i)$, akkor p apja q és p az apjának i-edik fia.
(Gyakorlat: milyen feltételek mellett lesz f fa?)

Rendezett fa megadható úgy is, hogy minden pontra megadjuk az első fiát, ha van, és minden pontra megadjuk a jobb testvérét, ha van. Ezt elsőfiú-testvér kapcsolati ábrázolásnak nevezzük.
A példánkban szereplő fa ekkor a következőképpen ábrázolható:

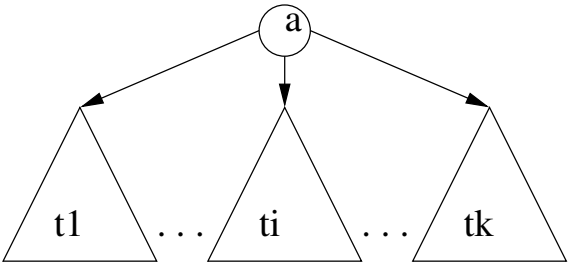
Fa megadása elsőfiú-testvér kapcsolattal.
Legyen $\mathtt{A} = (M, R, Adat)$ olyan absztrakt adatszerkezet, hogy $R = \{e,t\}$, $e,t : M \to (M \cup \{\bot\})$.
Legyen $f_i(x) = t^{i-1}(e(x))$ $i > 0$.
Az $\mathtt{A}$ adatszerkezet fa, ha az f_i függvények teljesítik a (6) - (9). feltételeket.

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

1. $\bot \in Fa(E)$
2. $(\forall a \in E)\ a \in Fa(E)$
3. $(\forall a \in E)(\forall t_1, \ldots, t_k \in Fa(E))(a(t_1, \ldots, t_k) \in Fa(E))$



8. ábra. Fa megadása elsőfiú-testvér kapcsolattal.



9. ábra. Fa algebrai megadásának szemléltetése.

Alapvető absztrakt adattípusok

Verem
Sor
Prioritási sor
Módosítható prioritási sor
Lista
Kétirányú lista
Tömb
Sorozat
Halmaz
Rendezett halmaz
Függvény
Reláció
UnioHolvan
File
Gráf
Súlyozott gráf

2.5. Verem

Értékhalmaz: $Verem = \{\langle a_1, \ldots, a_n \rangle : a_i \in E, i = 1, \ldots, n, n \geq 0\}$

Műveletek:

$$V : Verem, x : E$$

$$
\begin{array}{rcl}
\{Igaz\} & Letesit(V) & \{V = \langle\rangle\} \\
\{V = V\} & Megszuntet(V) & \{Igaz\} \\
\{V = V\} & Uresit(V) & \{V = \langle\rangle\} \\
\{V = \langle a_1, \ldots, a_n \rangle\} & VeremBe(V,x) & \{V = \langle a_1, \ldots, a_n, x \rangle\} \\
\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\} & VeremBol(V,x) & \{V = \langle a_1, \ldots, a_{n-1} \rangle \wedge x = a_n\} \\
\{V = V\} & NemUres(V) & \{NemUres = (Pre(V) \neq \langle\rangle)\} \\
\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\} & Teteje(V,x) & \{x = a_n \wedge V = Pre(V)\} \\
\{V = \langle a_1, \ldots, a_n \rangle \wedge n > 0\} & Torol(V) & \{V = \langle a_1, \ldots, a_{n-1} \rangle\}
\end{array}
$$

Még az egyszerű absztrakt adattípusoknak is különböző megvalósítása lehet. Mindig azt a megvalósítást kell kiválasztani, amelyik az adott feladat megoldásához a legmegfelelőbb. A megvalósítást alapvetően befolyásolja a megvalósításra használt programozási nyelv. Objektum orientált nyelvek esetén (java, C++, C#) interfész (vagy absztrakt) osztály használata biztosítja azt, hogy a megvalósítástól független algoritmust készíthessünk. Pontosabban, csak a LETESIT *művelet függ a megvalósítástól. Kívánatos lenne, hogy ne kelljen minden E elemtípusra külön megírni a megvalósítást, tehát egyetlen kódot kelljen csak megírni.*
Erre egy megoldást ad a java nyelv azzal, hogy minden típus (osztály) az $Object$ típus leszármazottja. Tehát a Verem adattípus interfész osztálya a következő lehet:

```
public interface Verem{
    public void Uresit();
    public boolean NemUres();
    public void VeremBe(Object x);
    public Object VeremBol();
    public Object Teteje();
    public void Torol();
}
```

Ekkor azonban bármilyen (nem csak azonos) típusú érték rakható ugyanabba a verembe. Továbbá, a VEREMBOL() *metódus $Object$ értéket ad, tehát a kivett értékkel nem végezhetek semmilyen műveletet (csak ami az Object-en definiált). Még akkor sem, ha tudom, hogy a kivett érték milyen (futási) típusú. Ez a korlátozás csak típuskényszerítéssel oldható fel.*

```
T x = (T) V.VeremBol();
```

A típuskényszerítés azonban olyan hibák forrása lehet, amelyek csak futáskor derülnek ki. Továbbá a hatékonyságot is ronthatják.
A megoldást a típus-paraméterezés mechanizmusa adja. Ahogy a műveletek (metódusok) argumentumainak van (fordítási időben rögzített) típusa, úgy legyen az elemtípus paramétere az osztálynak, és létesítéskor adhassuk meg a létesítendő objektum (összetet típus) elemtípusát.
Megjegyzés. *Java JDK 1.5 verziótól létezik a típus-paraméterezés.*

```
public interface Verem<E>{
    public void Uresit();
    public boolean NemUres();
    public void VeremBe(E x);
    public E VeremBol();
    public E Teteje();
    public void Torol();
}
```

Ha VeremL osztály egy megvalósítása a Verem adattípusnak (interfésznek), és egész számokat tartalmazó V vermet akarunk létesíteni, akkor ezt a VeremL osztály konstruktorának hívásával végezhetjük, ahol típus-paraméterként kell megadni az Integer típust (osztályt). ***Megjegyzés.***

Típus-paraméter csak referencia típus lehet, elemi típus (int, long, float, double, char, boolean, ...) nem. Azonban az $int \longleftrightarrow Integer$,... típuskonverzió automatikus. Tehát pl. Verem<Integer> V típus esetén alkalmazható a int x=V.VeremBol();, V.VeremBe(x); utasítás.

```
Verem<Integer> V = new VeremL<Integer>();
```

Verem megvalósítások:
VeremT: tömb adatszerkezettel,
VeremL: lánc adatszerkezettel,
VeremK: kombinált adatszerkezettel.

2.6. Sor

Értékhalmaz: $Sor = \{\langle a_1, \ldots, a_n \rangle : a_i \in E, i = 1, \ldots, n, n \geq 0\}$

Műveletek:

$$S : Sor, x : E$$

$\{Igaz\}$	$Letesit(S)$	$\{S = \langle\rangle\}$
$\{S = S\}$	$Megszuntet(S)$	$\{Igaz\}$
$\{S = S\}$	$Uresit(S)$	$\{S = \langle\rangle\}$
$\{S = \langle a_1, \ldots, a_n \rangle\}$	$SorBa(S,x)$	$\{S = \langle a_1, \ldots, a_n, x \rangle\}$
$\{S = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$SorBol(S,x)$	$\{x = a_1 \wedge S = \langle a_2, \ldots, a_n \rangle\}$
$\{S = \langle a_1, \ldots, a_n \rangle\}$	$Elemszam(S)$	$\{Elemszam = n\}$
$\{S = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$Elso(S,x)$	$\{x = a_1 \wedge S = Pre(S)\}$
$\{S = \langle a_1, \ldots, a_n \rangle \wedge n > 0\}$	$Torol(S)$	$\{S = \langle a_2, \ldots, a_n \rangle\}$

```
public interface Sor<E>{
   public void Uresit();
   public int Elemszam();
   public void SorBa(E x);
   public E SorBol();
   public E Elso();
   public void Torol();
}
```

Sor megvalósítások:

SorT: tömb adatszerkezettel,

SorL: lánc adatszerkezettel,

SorK: kombinált adatszerkezettel.

2.7. Prioritási Sor

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

$$S : PriSor, x : E$$

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

```
public interface PriSor<E extends Comparable<E>>
      extends Sor<E>{
}
```

Prioritási sor megvalósítások:

PriSorT: kupac-tömb adatszerkezettel,

PriSorR: kupac-tömb adatszerkezettel, a rendezési reláció konstruktor paraméter.

Példa prioritási sor alkalmazására.

Probléma: *Halmaz k-adik legkisebb elemének kiválasztása.*
Bement: $H = \{a_1, \ldots, a_n\}$, *különböző egész szám,* $1 \leq k \leq n$.
Kimenet: $a_i \in H$, *amelyre* $|\{x : x \in H, x \leq a_i\}| = k$,

```
public class PriSorPelda {
    public static int Kivalaszt(int[] A, int k){
       int n=A.length;        int x=0;
        PriSor<Integer> S = new PriSorT<Integer>(n);
       for (int i=0; i<k; i++)
            S.SorBa(A[i]);
        for (int i=k; i<n; i++)
          if (S.Elso()<A[i]){
             S.SorBol();
             S.SorBa(A[i]);
          }
        return S.Elso();
    }
    public static void main (String[] args){
      Scanner stdin = new Scanner(System.in);
      System.out.println("Elemek száma?");
      int n=stdin.nextInt(); stdin.nextLine();
       System.out.println("Hanyadik?");
       int k=stdin.nextInt();
       int[] A=new int[n];
       for (int i=0; i<n; i++) A[i]=stdin.nextInt();
       System.out.println("A keresett szám: "+Kivalaszt(A, k));
    }
 }
```

2.8. Lista

Értékhalmaz: $Lista = \{\langle a_1, \ldots, a_{i-1}\rangle \langle a_i, \ldots, a_n\rangle : a_j \in E, j = 1, \ldots, n, 0 \leq i-1 \leq n, n \geq 0\}$

Műveletek:

$$L, L1, L2 : Lista, x : E$$

$\{Igaz\}$	$Letesit(L)$	$\{L = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Megszuntet(L)$	$\{Igaz\}$
$\{L = L\}$	$Uresit(L)$	$\{L = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Urese(L)$	$\{Urese = (Pre(L) = \langle\rangle\langle\rangle\}$
$\{L = L\}$	$Elejen(L)$	$\{= Pre(L) = \langle\rangle\langle a_1,\ldots,a_n\rangle\}$
$\{L = L\}$	$Vegen(L)$	$\{Vegen = L = \langle a_1,\ldots,a_n\rangle\langle\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Elejere(L)$	$\{L = \langle\rangle\langle a_1,\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Vegere(L)$	$\{L = \langle a_1,\ldots,a_n\rangle\langle\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i \leq n\}$	$Tovabb(L)$	$\{L = \langle a_1,\ldots,a_{i-1},a_i\rangle \langle a_{i+1},\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i \leq n\}$	$Kiolvas(L,x)$	$\{x = a_i \wedge L = Pre(L)\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,a_{i+1}\ldots,a_n\rangle \wedge i \leq n\}$	$Modosit(L,x)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle x,a_{i+1},\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle\}$	$Bovit(L,x)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle x,a_i,\ldots,a_n\rangle\}$
$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,a_{i+1},\ldots,a_n\rangle \wedge i \leq n\}$	$Torol(L)$	$\{L = \langle a_1,\ldots,a_{i-1}\rangle \langle a_{i+1},\ldots,a_n\rangle\}$
$\{L1 = \langle\alpha_1\rangle\langle\beta_1\rangle, L2 = \langle\alpha_2\rangle\langle\beta_2\rangle\}$	$Kapcsol(L1,L2)$	$\{L1 = \langle\alpha_1\rangle\langle\alpha_2\beta_1\rangle, L2 = \langle\rangle\langle\beta_2\rangle$

```
public interface Lista<E>{
   public void Uresit();
   public boolean Urese();
   public boolean Elejen();
   public boolean Vegen();
   public void Elejere();
   public void Vegere();
   public void Tovabb();
   public E Kiolvas();
   public void Modosit(E x);
   public void Bovit(E x);
   public void Torol();
   public void Kapcsol(Lista<E> l2);
}
```

Lista megvalósítások:

ListaL: lánc adatszerkezettel,

ListaT: tömb adatszerkezettel.

2.9. Kétirányú lista

Értékhalmaz: $Lista2 = \{\langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle : a_i \in E, i = 1,\ldots,n, n \geq 0\}$
Műveletek: Lista műveletei + $Vissza$

$$L : Lista2$$

$$\{L = \langle a_1,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_n\rangle \wedge i > 0\} \quad Vissza(L) \quad \{L = \langle a_1,\ldots,a_{i-2}\rangle \langle a_{i-1},a_i,a_{i+1},\ldots,a_n\rangle\}$$

```
public class Lista2<E> extends Lista<E>{
   public void Vissza();
}
```

Kétirányú lista (Lista2) megvalósítások

Lista2L: kétirányú lánc adatszerkezettel.

2.10. Tömb

Értékhalmaz: $Tomb = \{\langle a_1,\ldots,a_n\rangle : a_i \in E \cup \{\bot\}, i = 1,\ldots,n, n \geq 1\}$

Műveletek:

$$T : Tomb, x : E, i : Integer$$

$$\begin{array}{rcl} \{Igaz\} & Letesit(T,n) & \{T = \langle \bot,\ldots,\bot\rangle\} \\ \{T = \langle a_1,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Kiolvas(T,i,x) & \{T = Pre(T) \wedge x = a_i\} \\ \{T = \langle a_1,\ldots,a_i,\ldots,a_n\rangle \wedge 1 \leq i \leq n\} & Modosit(T,i,x) & \{T = \langle a_1,\ldots,x,\ldots,a_n\rangle\} \end{array}$$

Megvalósítás

```
//Letesit(T,n):
E[] T=new E[n];
//Kiolvas(T,i,x):
x=T[i];
//Modosit(T,i,x):
T[i]=x;
```

A tömb típus rendelkezik iterátorral, tehát a

```
for (int i=0; i<T.length; i++) M(T[i])
```

ismétlés helyett írhatjuk az egyszerű

```
for (E x:T) M(x)
```

utasítást.

2.11. Sorozat

Értékhalmaz: $Sorozat = \{\langle a_1, \ldots, a_n \rangle : a_i \in E, i = 1, \ldots, n, n \geq 0\}$

Műveletek:

$$S : Sorozat, x : E, i : Integer$$

$$\begin{array}{rcl}
\{Igaz\} & Letesit(S) & \{S = \langle\rangle\} \\
\{S = S\} & Megszuntet(S) & \{Igaz\} \\
\{S = S\} & Uresit(S) & \{S = \langle\rangle\} \\
\{S = \langle a_1, \ldots, a_n \rangle\} & Elemszam(S) & \{Elemszam = n\rangle\} \\
\{S = \langle a_1, \ldots, a_i, \ldots, a_n \rangle \wedge 1 \leq i \leq n\} & Kiolvas(S,i,x) & \{x = a_i \wedge S = Pre(S)\} \\
\{S = \langle a_1, \ldots, a_i, \ldots, a_n \rangle \wedge 1 \leq i \leq n\} & Modosit(S,i,x) & \{S = \langle a_1, \ldots, x, \ldots, a_n \rangle\} \\
\{S = \langle a_1, \ldots, a_i, a_{i+1}, \ldots, a_n \rangle \wedge 0 \leq i \leq n\} & Bovit(S,i,x) & \{S = \langle a_1, \ldots, a_i, x, a_{i+1}, \ldots, a_n \rangle\} \\
\{S = \langle a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n \rangle \wedge 1 \leq i \leq n\} & Torol(S,i) & \{S = \langle a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \rangle\} \\
\{S = S\} & IterKezd(S,I) & \{\} \\
\{I = I\} & IterAd(I,x) & \{\} \\
\{I = I\} & IterVege(I) & \{\}
\end{array}$$

```
public interface Sorozat<E> extends Iterable<E>{
   public void Uresit();
   public int Elemszam();
   public E Kiolvas(int i);
   public void Modosit(int i, E x);
   public void Bovit(int i, E x);
   public void Torol(int i);
   public Iterator<E> iterator();
}
```

Sorozat megvalósítások

SorozatBKF: bináris keresőfa adatszerkezettel,

SorozatAVL: AVL-kiegyensúlyozott bináris keresőfa adatszerkezettel.

2.12. Halmaz

Értékhalmaz: $Halmaz = \{ H : H \subseteq E \}$

Műveletek:

$$H : Halmaz, x : E, I : Iterator$$

$$
\begin{array}{rcl}
\{Igaz\} & Letesit(H) & \{H = \emptyset\} \\
\{H = H\} & Megszuntet(H) & \{Igaz\} \\
\{H = H\} & Uresit(H) & \{H = \emptyset\} \\
\{H = \{a_1, \ldots, a_n\}\} & Elemszam(H) & \{Elemszam = n\} \\
\{H = H\} & Eleme(H,x) & \{Eleme = x \in Pre(H)\} \\
\{H = H\} & Bovit(H,x) & \{H = Pre(H) \cup \{x\}\} \\
\{H = H\} & Torol(H,x) & \{H = Pre(H) - \{x\}\} \\
\{H = H\} & IterKezd(H,I) & \{\} \\
\{I = I\} & IterAd(I,x) & \{\} \\
\{I = I\} & IterVege(I) & \{\}
\end{array}
$$

```
public interface Halmaz<E> extends Iterable<E>{
   public void Uresit();
   public int Elemszam();
   public boolean Eleme(E x);
   public void Bovit(E x);
   public boolean Torol(E x);
   public E Keres(E x);
}
```

2.13. RHalmaz

Értékhalmaz: $Halmaz = \{ H : H \subseteq E,$ *E-n értelmezett a* $\leq$ *lineáris rendezési reláció.* $\}$
Műveletek: Halmaz műveletek +

$$H : Halmaz, x : E, I : Iterator$$

$$\begin{array}{lll} \{H \neq \emptyset\} & Min(H) & \{= min\{x : x \in H\}\} \\ \{H \neq \emptyset\} & Max(H) & \{= max\{x : x \in H\}\} \\ \{H \neq \emptyset\} & Elozo(H,x) & \{= max\{y : y \in H - x \wedge y \leq x\}\} \\ \{H \neq \emptyset\} & Koveto(H,x) & \{= min\{y : y \in H - x \wedge x \leq y\}\} \end{array}$$

```
public interface RHalmaz<E extends Comparable<E>>
   extends Halmaz<E>, Iterable<E>{
      public E Min();
      public E Max();
      public E Elozo(E x);
      public E Koveto(E x);
}
```

A Halmaz adattípus megvalósításai:

HalmazB: bitvektor adatszerkezettel; E=1..n

HalmazT: tömb adatszerkezettel,

HalmazL: lánc adatszerkezettel,

HalmazHL: hasítótábla (ütközésfeloldás láncolással) adatszerkezettel,

HalmazHN: hasítótábla (ütközésfeloldás nyílt címzéssel) adatszerkezettel.

Az RHalmaz adattípus megvalósításai:

RHalmazT: rendezett tömb adatszerkezettel,

RHalmazL: rendezett lánc adatszerkezettel,

RHalmazBKF: bináris keresőfa adatszerkezettel,

RHalmazAVL: AVL-kiegyensúlyozott bináris keresőfa adatszerkezettel,

RHalmazPFFa: piros-fekete fa adatszerkezettel.

Önszervező bináris keresőfával

Ugrólistával (Skiplist)

```
Forall x in H Do
  M(x);
```

Diszkrét ismétléses vezérlés megvalósításai iterátorral.

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);
```

Példa Halmaz absztrakt adattípus alkalmazására.

Probléma: *Adott sorozat különböző elemének kiválasztása.*

Bement: $S = \langle a_1, \ldots, a_n \rangle$ *egész számok.*

Kimenet: $H = \{x : x \in S\}$.

```
import java.io.*; import java.util.*;
public class HalmazPelda{
    public static void main (String[] args){
        Scanner stdin = new Scanner(System.in);
        Halmaz<Integer> H=new HalmazL<Integer>();
        int x;
        System.out.println("Elemek száma?");
        int n=stdin.nextInt(); stdin.nextLine();
        for (int i=1; i<=n; i++){
            x=stdin.nextInt();
            if (!H.Eleme(x))
                H.Bovit(x);
        }
        stdin.close();
        Iterator<Integer> it=H.iterator();  //IterKezd(H,it)
        while (it.hasNext()){               //!IterVege(it)
            x=it.next();                    //IterAd(it,x)
            System.out.print(x+" ");
        }
        System.out.println();
        for (int e:H)               //H elemeinek kiíratása for ciklussal
            System.out.print(e+" ");
   }
}
```

2.14. Függvény (parciális függvény)

Értékhalmaz:

$$Fuggveny = \{\, F : F \subseteq E = K \times A, (\forall k \in K)(\forall x, y \in A)((k,x) \in F \wedge (k,y) \in F \Rightarrow x = y)\}$$

Műveletek:

$$F : Fuggveny, k : K, x : A$$

$$\{Igaz\} \quad Letesit(F) \quad \{F = \emptyset\} \tag{11}$$

$$\{F = F\} \quad Megszuntet(F) \quad \{Igaz\} \tag{12}$$

$$\{F = F\} \quad Uresit(F) \quad \{F = \emptyset\} \tag{13}$$

$$\{F = F\} \quad Eleme(F,k) \quad \{Eleme = (\exists x \in A)((k,x) \in F)\} \tag{14}$$

$$\{F = F\} \quad Elemszam(F) \quad \{Elemszam = |F|\} \tag{15}$$

$$\{(\exists a \in A)((k,a) \in F)\} \quad Kiolvas(F,k,x) \quad \{x = a \wedge F = Pre(F)\} \tag{16}$$

$$\{(\forall a \in A)((k,a) \notin F)\} \quad Kiolvas(F,k,x) \quad \{F = Pre(F) \wedge x = Pre(x)\} \tag{17}$$

$$\{(\exists a \in A)((k,a) \in F)\} \quad Modosit(F,k,x) \quad \{F = Pre(F) - \{(k,a)\} \cup \{(k,x)\}\} \tag{18}$$

$$\{(\forall a \in A)((k,a) \notin F)\} \quad Modosit(F,k,x) \quad \{F = Pre(F) \wedge x = Pre(x)\} \tag{19}$$

$$\{(\forall a \in A)((k,a) \notin F)\} \quad Bovit(F,k,x) \quad \{F = Pre(F) \cup \{(k,x)\}\} \tag{20}$$

$$\{(\exists a \in A)((k,a) \in F)\} \quad Bovit(F,k,x) \quad \{F = Pre(F) \wedge x = Pre(x)\} \tag{21}$$

$$\{(\exists a \in A)((k,a) \in F)\} \quad Torol(F,k) \quad \{F = Pre(F) - \{(k,a)\}\} \tag{22}$$

$$\{(\forall a \in A)((k,a) \notin F)\} \quad Torol(F,k) \quad \{F = Pre(F)\} \tag{23}$$

$$\{F = F\} \quad IterKezd(F,I) \quad \{\} \tag{24}$$

$$\{I = I\} \quad IterAd(I,k,x) \quad \{\} \tag{25}$$

$$\{I = I\} \quad IterVege(I) \quad \{\} \tag{26}$$

```
public interface Fuggveny<K, A>
    extends Iterable<KulcsPar<K, A>>{
    public int Elemszam();
    public void Uresit();
    public boolean Eleme(K k);
    public void Bovit(K k, A a);
    public boolean Torol(K k);
    public A Kiolvas(K k);
    public void Modosit(K k, A a);
}

public class KulcsPar<K, A > implements Cloneable{
   public K kulcs;
   public A adat;
   public KulcsPar(K k, A a){
      kulcs=k; adat=a;
   }
   public boolean equals(Object x){
      return this.kulcs.equals( ((KulcsPar<K,A>)x).kulcs );
   }
   public int hashCode(){
      return this.kulcs.hashCode();
   }
}
```

```
Forall (k,x) in F Do
  M(k,x);
```

≡

```
KulcsPar<K,A> par;
Iterator<KulcsPar<K,A>> I=F.iterator(); //IterKezd(F,I)
while (I.hasNext()){                    //!IterVege(I)
   par=I.next();                        //IterAd(I,par)
   M(par.kulcs, par.adat);
}

for (KulcsPar<K,A> par : F)
  M(par.kulcs, par.adat);
```

A Függvény adattípus megvalósításai:

FuggvenyT: tömb adatszerkezettel, az értelmezési tartomány (K) típusa Integer;
public class FuggvenyT<A> implements Fuggveny<Integer, A>.
FuggvenyH: <k,a> párok halmazaként, ekkor konstruktor paraméterként kell megadni, hogy milyen halmaz ábrázolást akarunk:

- *"Lanc" lánc adatszerkezet*
- *"Tomb" tömb adatszerkezet*
- *"HasitL" hasítótábla láncolással adatszerkezet*
- *"HasitN" hasítótábla nyílt címzéssel adatszerkezet*

FuggvenyR: az értelmezési tartományon értelmezett lin. rendezés, ekkor konstruktor paraméterké kell megadni, hogy milyen halmaz ábrázolást akarunk:

- *"BKF" bináris keresőfa*
- *"AVLFa" AVL-keresőfa*
- *"PFFa" piros-fekete keresőfa*

2.15. Reláció

Értékhalmaz:

$Relacio = \{ R : R \subseteq E = K \times A \}$

Műveletek:

$$R : Relacio, k : K, a : A$$

$\{Igaz\}$	$Letesit(R)$	$\{F = \emptyset\}$
$\{F = F\}$	$Megszuntet(R)$	$\{Igaz\}$
$\{F = F\}$	$Uresit(R)$	$\{F = \emptyset\}$
$\{F = F\}$	$Eleme(R,k,a)$	$\{Eleme = ((k,a) \in R)\}$
$\{F = F\}$	$Elemszam(R)$	$\{Elemszam = \lvert R \rvert\}$
$\{(k,a) \notin R)\}$	$Bovit(R,k,a)$	$\{R = Pre(R) \cup \{(k,a)\}\}$
$\{(k,a) \in R)\}$	$Torol(R,k,a)$	$\{R = Pre(R) - \{(k,a)\}\}$
$\{R = R)\}$	$KTorol(R,k)$	$\{R = Pre(R) - \{(k,a) : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$ATorol(R,a)$	$\{R = Pre(R) - \{(k,a) : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$KPar(R,k)$	$\{\{a : (k,a) \in Pre(R)\}\}$
$\{R = R)\}$	$APar(R,a)$	$\{\{k : (k,a) \in Pre(R)\}\}$
$\{R = R\}$	$IterKezd(R,I)$	$\{\}$
$\{I = I\}$	$IterAd(I,k,a)$	$\{\}$
$\{I = I\}$	$IterVege(I)$	$\{\}$

```
public interface Relacio<K, A>
    extends Iterable<Par<K, A>>{
    public int Elemszam();
    public void Uresit();
    public boolean Eleme(K k, A a);
    public void Bovit(K k, A a);
    public boolean Torol(K k, A a);
    public void KTorol(K k);
    public void ATorol(A a);
    public Halmaz<A> KPar(K k);
    public Halmaz<K> APar(A a);
}
```

A Relacio adattípus megvalósításai:

RelacioL: lánc adatszerkezettel,

RelacioBKF: bináris keresőfa adatszerkezettel, mind K*, mind* A *rendezett típus kell legyen.*

RelacioAVL: AVL-fa bináris keresőfa adatszerkezettel, mind K*, mind* A *rendezett típus kell legyen.*

2.16. Prioritási Sor

Értékhalmaz: $PriSor = \{\ S = \{a_1, \ldots, a_n\} : S \subseteq E\}$, E*-n értelmezett a* $\leq$ *lineáris rendezési reláció.*

Műveletek:

$$S : PriSor, x : E$$

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

Megvalósítás kupaccal.

```
public class PriSorT<E extends Comparable<E>>
      implements PriSor<E>{
   private int eszam=0;
   private int meret=100;
   private E[] tar;
   public PriSorT(){
      this.tar= (E[])new Comparable[meret+1];
   }
   public PriSorT(int kezdmeret){
      meret=kezdmeret;
      this.tar= (E[])new Comparable[meret+1];
   }
   public void Uresit(){
      eszam=0;
   }
   public int Elemszam(){
      return eszam;
   }
   public void SorBa(E x){
      if (eszam==meret){ throw new RuntimeException("A Sor megtelt");
      }
      tar[++eszam]=x;
      emel(eszam);
   }
```

```
public E SorBol(){
   if (eszam==0){
      throw new NoSuchElementException("A Sor üres");
   }
   E x=tar[1];
   tar[1]=tar[eszam--];
   if (eszam>0) sullyeszt(1);
   return x;
}
public E Elso(){
   if (eszam==0){
      throw new NoSuchElementException("A Sor üres");
   }
   return tar[1];
}
public void Torol(){
   tar[1]=tar[eszam--];
   if (eszam>0) sullyeszt(1);
}
```

```
private void sullyeszt(int k){
   int apa=k;
   int fiu;
   E e=tar[k];
   while ((fiu = apa << 1) <= eszam ) {
      if (fiu<eszam &&
          tar[fiu+1].compareTo(tar[fiu]) < 0)
            fiu++; // a kisebbik fiu
      if (e.compareTo(tar[fiu]) <= 0) break;
      tar[apa]=tar[fiu];
      apa = fiu;
   }
   tar[apa]=e;
}
```

```
   private void emel(int k){
      E e=tar[k];
      int fiu=k;
      int apa;
      while (fiu>1){
         apa= fiu >> 1;
         if (e.compareTo(tar[apa])<0){
            tar[fiu]=tar[apa];
            fiu=apa;
         } else
            break;
      }
      tar[fiu]=e;
   }
}
```

A műveletek futási ideje.

SORBA *és* SORBOL *futási ideje:* $T_{lr}(n) = O(\lg n)$

2.17. Módosítható prioritási Sor

Műveletek: Prioritási sor műveletek + MODOSIT

Feltesszük, hogy az adatelemeket az 1..n számokkal azonosítjuk, továbbá az elemek típusa:

```
public class KulcsPar<K, A > implements Cloneable{
   public K kulcs;
   public A adat;
   public KulcsPar(K k, A a){
      kulcs=k; adat=a;
   }
   public KulcsPar(){
   }
   public boolean equals(Object x){
      return this.kulcs.equals( ((KulcsPar<K,A>)x).kulcs );
   }
   public int hashCode(){
      return this.kulcs.hashCode();
   }
   public KulcsPar<K, A> clone() {
      KulcsPar<K, A> result = this;
      try {  result = (KulcsPar<K, A>)super.clone();
      } catch (CloneNotSupportedException e) {
         throw new InternalError();
      }
      return result;
   }
}
```

Az adatelemeket a kulcs-mező alapján azonosítjuk és a rendezés az adat-mező alapján történik.
MODOSIT(S,X) *művelet az* x *elemet a megváltozott adat-mezőjéneknek megfelelően módisítja az értékhalmazt (az* x *elemet az adatszerkezetben a megfelelő helyre teszi).*
Szükség van olyan Hol *függvényre, amely minden* i*-re megmondja, hogy hol van az* F *kupacban az* i*-edik elem:*

$$F[Hol[i]] = i \wedge Hol[F[j]] = j$$

```
public class ModPriSor<S extends Comparable<S>>
      implements Sor<KulcsPar<Integer,S>>{

   private int eszam=0;
   private int meret=100;
   private KulcsPar<Integer,S>[] tar;
   private int[] hol;

   public ModPriSor(){
      this.tar= (KulcsPar<Integer,S>[])new KulcsPar[meret+1];
   }
   public ModPriSor(int kezdmeret){
      meret=kezdmeret;
      hol=new int[meret+1];
      this.tar= (KulcsPar<Integer,S>[])new KulcsPar[meret+1];
   }
   public void Uresit(){
      eszam=0;
   }
```

```
public int Elemszam(){
   return eszam;
}
public void SorBa(KulcsPar<Integer,S> x){
   if (eszam==meret){
      throw new RuntimeException("A Sor megtelt");
   }
   tar[++eszam]=x;
   hol[x.kulcs]=eszam;
   emel(eszam);
}
public KulcsPar<Integer,S> SorBol(){
   if (eszam==0){
      throw new NoSuchElementException("A Sor üres");
   }
   KulcsPar<Integer,S> x=tar[1];
   tar[1]=tar[eszam--];
   if (eszam>0) sullyeszt(1);
   return x;
}
```

```
public void Modosit(KulcsPar<Integer,S> x){
   int k=hol[x.kulcs];
   int ken=x.adat.compareTo(tar[k].adat);
   tar[k]=x;
   if (ken<0){
      emel(k);
   }else
      sullyeszt(k);
}
public KulcsPar<Integer,S> Elso(){
   if (eszam==0){
      throw new NoSuchElementException("A Sor üres");
   }
   return tar[1];
}
public void Torol(){
   tar[1]=tar[eszam--];
   if (eszam>0) sullyeszt(1);
}
```

```
private void sullyeszt(int k){
   int apa=k;
   int fiu;
   KulcsPar<Integer,S> e=tar[k];
   while ((fiu = apa << 1) <= eszam ) {
      if (fiu<eszam &&
          tar[fiu+1].adat.compareTo(tar[fiu].adat) < 0)
            fiu++; // a kisebbik fiu
      if (e.adat.compareTo(tar[fiu].adat) <= 0) break;
      tar[apa]=tar[fiu];
      hol[tar[fiu].kulcs]=apa;
      apa = fiu;
   }
   tar[apa]=e;
   hol[e.kulcs]=apa;
}
```

```
   private void emel(int k){
      KulcsPar<Integer,S> e=tar[k];
      int fiu=k;
      int apa;
      while (fiu>1){
         apa= fiu >>> 1;
         if (e.adat.compareTo(tar[apa].adat)<0){
            tar[fiu]=tar[apa];
            hol[tar[apa].kulcs]=fiu;
            fiu=apa;
         } else
            break;
      }
      tar[fiu]=e;
      hol[e.kulcs]=fiu;
   }
}
```

MODOSIT *futási ideje:* $T_{lr}(n) = O(\lg n)$.

2.18. Az UnioHolvan adattípus megvalósítása

Az UnioHolvan absztrakt adattípus.

Értékhalmaz: $UnioHolvan = \{\{H_1,\ldots,H_k\} : H_i \subseteq E, i = 1,\ldots,k, i \neq j \Rightarrow H_i \cap H_j = \emptyset\}$

Műveletek:

$$S : UnioHolvan, x, y : Elemtip, n1, n2 : NevTip$$

$\{Igaz\}$	$Letesit(S,n)$	$\{S = \{\}\}$
$\{Igaz\}$	$Letesit(S)$	$\{S = \emptyset\}$
$\{S = S\}$	$Megszuntet(S)$	$\{Igaz\}$
$\{S = S\}$	$Uresit(S)$	$\{S = \emptyset\}$
$\{S = \{H_1,\ldots,H_k\} \wedge x \in \cup H_i\}$	$Holvan(S,x)$	$\}\{= y \wedge (x \in H_i \wedge y \in H_i)\}$
$\{S = \{H_1,\ldots,H_k\} \wedge x \notin \cup H_i\}$	$Holvan(S,x)$	$\{= x \wedge S = Pre(S) \cup \{\{x\}\}$
$\{S = \{H_1,\ldots,H_k\} \wedge n1 \in H_i \wedge n2 \in H_j\}$	$Unio(S,n1,n2)$	$\{S = Pre(S) - H_i - H_j \cup \{H_i \cup H_j\}\}$
$\{S = \{H_1,\ldots,H_k\}\}$	$Elemszam(S)$	$\{= k = \lvert S\rvert \wedge S = Pre(S)\}$
$\{S = \{H_1,\ldots,H_k\}\}$	$ReszElemszam(S,x)$	$\{= \lvert H_i\rvert \wedge x \in H_i \wedge S = Pre(S)\}$
$\{S = S\}$	$Iterator(S)$	$\{\}$
$\{S = \{H_1,\ldots,H_k\} \wedge (\exists i)(x \in H_i)\}$	$ReszIterator(S,x)$	$\{\}$

2.19. File absztrakt adattípus

Típusolt file: File of E

Értékhalmaz: $FileE = \{\langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle : a_i \in E, i = 1,\ldots,n\}$

Műveletek:

$$F : FileE, FN : String, x : E, j : Longint$$

$\{Igaz\}$	$Assign(F,FN)$	$\{F =$ az FN állomány tartalma$\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Reset(F)$	$\{F = \langle\rangle\langle a_0,\ldots,a_i,\ldots,a_n\rangle\}$
$\{F = F\}$	$Rewrite(F)$	$\{F = \langle\rangle\langle\rangle\}$
$\{F = F\}$	$EoF(F)$	$\{EoF = (Pre(F) = \langle a_0,\ldots,a_{n-1}\rangle\langle\rangle)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle \wedge i < n\}$	$Read(F,x)$	$\{x = a_i \wedge F = \langle a_0,\ldots,a_i\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Write(F,x)$	$\{F = \langle a_0,\ldots,a_i,x\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FilePos(F)$	$\{FilePos = i \wedge F = Pre(F)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FileSize(F)$	$\{FileSize = n \wedge F = Pre(F)\}$
$\{F = F \wedge 0 \le j \le FileSize(F)\}$	$Seek(F,j)$	$\{F = \langle a_0,\ldots,a_{j-1}\rangle\langle a_j,\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Truncate(F)$	$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle\rangle\}$

Típustalan (bináris) fájl: File

Értékhalmaz: $File = \{\langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle : a_i \in Byte, i = 1,\ldots,n\}$

Műveletek:

$$F : File, FN : String, x : E, k, j, r : Longint, V : Array[1..] \, of \, E$$

$\{Igaz\}$	$Assign(F,FN)$	$\{F =$ *az FN állomány tartalma*$\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Reset(F,Rh)$	$\{F = \langle\rangle\langle a_0,\ldots,a_i,\ldots,a_n\rangle, \lvert a_i\rvert = Rh\}$
$\{F = F\}$	$Rewrite(F,Rh)$	$\{L = \langle\rangle\langle\rangle\}$
$\{F = F\}$	$EoF(F)$	$\{EoF = (Pre(F) = \langle a_0,\ldots,a_{n-1}\rangle\langle\rangle)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle \wedge i < n\}$	$Read(F,x)$	$\{x = a_i \wedge F = \langle a_0,\ldots,a_i\rangle\langle a_{i+1},\ldots,a_n$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Write(F,x)$	$\{F = \langle a_0,\ldots,a_i,x\rangle\langle a_{i+1},\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FilePos(F)$	$\{FilePos = i \wedge F = Pre(F)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$FileSize(F)$	$\{FileSize = n \wedge F = Pre(F)\}$
$\{F = F \wedge 0 \le j \le FileSize(F)\}$	$Seek(F,j)$	$\{F = \langle a_0,\ldots,a_{j-1}\rangle\langle a_j,\ldots,a_{n-1}\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$Truncate(F)$	$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle\rangle\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$BlockRead(F,V,k,[r])$	$\{V[j] = a_{i+j-1}, j = 1,\ldots,r \wedge$ $F = \langle a_0,\ldots,a_{i+r-1}\rangle\langle a_{i+r},\ldots,a_{n-1}\rangle \wedge$ $r = \min(k, n-i)\}$
$\{F = \langle a_0,\ldots,a_{i-1}\rangle\langle a_i,\ldots,a_{n-1}\rangle\}$	$BlockWrite(F,V,k)$	$\{F = \langle a_0,\ldots,a_{i-1},V[1],\ldots V[k]\rangle$ $\langle a_{i+k},\ldots,a_{n-1}\rangle\}$

2.20. Definíciók

1. ***Irányítatlan gráf***:$G = (V, E)$
E *rendezetlen* $\{a, b\}, a, b \in V$ *párok halmaza.*
2. ***Irányított gráf***:$G = (V, E)$
E *rendezett* (a, b) *párok halmaza;* $E \subseteq V \times V$.
3. ***Multigráf***:
$G = (V, E, \mathit{Ind}, \mathit{Érk})$, $\mathit{Ind}, \mathit{Érk} : E \to V$
$\mathit{Ind}(e)$ *az* e *él induló,* $\mathit{Érk}(e)$ *az érkező pontja.*

Címkézett (súlyozott) gráf:$G = (V, E, C)$

$C : E \to Címke$

Minden irányítatlan $G = (V,E)$ gráf olyan irányított gráfnak tekinthető, amelyre teljesül, hogy $(\forall p,q \in V)((p,q) \in E \Rightarrow (q,p) \in E)$.

Minden $G = (V,E)$ gráf megadható olyan függvénnyel, amely a gráf minden $p \in V$ pontjához azon q pontok halmazát rendeli, amelyekre $(p,q) \in E$.

$$r_E : V \to 2^V; \quad r_E(p) = \{q : (p,q) \in E\}$$

Címkézett gráf esetén pedig

$$r_E : V \to 2^{V \times Cimke}; \quad r_E(p) = \{(q,s) : (p,q) \in E \wedge C(p,q) = s\}$$

Jelölések

$Ki(G,p) = \{q \in V : (p,q) \in E\}$

$Be(G,p) = \{q \in V : (q,p) \in E\}$

$KiFok(G,p) = |Ki(G,p)|$

$BeFok(G,p) = |Be(G,p)|$

2.21. Gráf absztrakt adattípus

Milyen műveleteket akarunk gráfokon végezni?

Értékhalmaz: $Graf = \{G = (V,E) : V \subseteq PontTip, E \subseteq V \times V\}$

Műveletek:

$$G : Graf,\ p, p1, p2 : PontTip,\ ir : boolean,\ I : Iterator,$$

$\{Igaz\}$	$Letesit(G,ir)$	$\{G=(\emptyset,\emptyset)\}$
$\{G=G\}$	$Megszuntet(G)$	$\{Igaz\}$
$\{G=G\}$	$Uresit(G)$	$\{G=(\emptyset,\emptyset)\}$
$\{G=G\}$	$Iranyitott(G)$	$\{\neg Iranyitott(G) \Rightarrow (p,q)\in E \Rightarrow (q,p)\in E)\}$
$\{G=G\}$	$Pontokszama(G)$	$\{=\lvert V\rvert\}$
$\{G=G\}$	$Elekszama(G)$	$\{=\lvert E\rvert\}$
$\{G=G\}$	$KiFok(G,p)$	$\{=\lvert Ki(G,p)\rvert\}$
$\{G=G\}$	$BeFok(G,p)$	$\{=\lvert Be(G,p)\rvert\}$
$\{G=(V,E)\}$	$PontBovit(G,p)$	$\{V=Pre(V)\cup\{p\}\wedge E=Pre(E)\}$
$\{G=(V,E)\wedge p\in V\}$	$PontTorol(G,p)$	$\{V=Pre(V)-\{p\}\wedge$
		$E=Pre(E)-\{(p,q):q\in Ki(G,p)\}-$
		$\{(q,p):q\in Be(G,p)\}\}$
$\{G=(V,E),p1,p2\in V\}$	$ElBovit(G,p1,p2)$	$\{E=Pre(E)\cup\{(p1,p2)\}\wedge V=Pre(V)\}$
$\{G=(V,E),p1,p2\in V\}$	$ElTorol(G,p1,p2)$	$\{E=Pre(E)-\{(p1,p2)\}\wedge V=Pre(V)\}$
$\{G=(V,E),p1,p2\in V\}$	$Vanel(G,p1,p2)$	$\{=(p1,p2)\in E\wedge E=Pre(E)\wedge V=Pre(V)\}$
$\{G=G\}$	$PIterator(G,I)$	$\{\}$
$\{G=G\}$	$KiEl(G,p)$	$\{=\{q:VanEl(p,q)\}\}$
$\{G=G\}$	$KiIterator(G,p)$	$\{\}$
$\{G=G\}$	$ElIterator(G,I)$	$\{\}$

A továbbiakban feltételezzük, hogy a gráf pontjait természetes számokkal azonosítjuk, pontosabban $V \subseteq \{1, \ldots, n\}$
Java nyelven az alábbi interface-t használhatjuk.

```
public interface Graf extends Iterable<Integer>{
   public boolean Iranyitott();
   public int Pontokszama();
   public int Maxpont();
   public int Elekszama();
   public int KiFok(int p);
   public int BeFok(int p);
   public void Uresit();
   public void PontBovit(int p);
   public void PontTorol(int p);
   public void ElBovit(int p, int q);
   public void ElTorol(int p, int q);
   public boolean VanEl(int p, int q);
   public Halmaz<Integer> KiEl(int p);
   public Iterator<Integer> KiIterator(int p);
   public Iterator<GrafEl> ElIterator();
}
```

Ahol a GrafEl osztály

```
public class GrafEl{
   public int ki;
   public int be;
   public GrafEl(int p, int q){
      ki=p;
      be=q;
   }
   public GrafEl(){
   }
   public String toString(){
      return Integer.toString(ki)+"->"+Integer.toString(be);
   }
}
```

Ekkor a gráf pontjainak bejárása:

```
for (int p:G)
  M(p);
```

vagy

```
Iterator<Integer> piter=G.iterator();
while (piter.hasNext()){
  int p=piter.next();
  M(p);
}
```

Adott p pont szomszédjainak (a p-ből induló élek) bejárása:

```
for (int q:G.KiEl(p))
   M(p,q);
```

vagy

```
Iterator<Integer> kiiter=G.KiEl(p).iterator();
 while (kiiter.hasNext()){
   int q=kiiter.next();
   M(p, q);
 }
```

A gráf minden $(p,q) \in E$ élének bejárása:

```
for (Iterator<GrafEl> eliter=G.ElIterator(); eliter.hasNext();){
   GrafEl el=eliter.next();
   M(el.ki, el.be);
}
```

vagy

```
Iterator<GrafEl> eliter=G.ElIterator();
   while (eliter.hasNext()){
     GrafEl el=eliter.next();
     M(el.ki, el.be);
  }
```

Az él-iteráció nyilvánvalóan megvalósítható a pont-iteráció és ki-iteráció műveletekkel, de fordítva nem.

```
for (int p:G){
   for (int q:G.Ki(p))
      M(p,q);;
   }
}
```

Címkézett (súlyozott) gráf absztrakt adattípus

Értékhalmaz:

$Graf = \{G = (V,E,C) : V \subseteq PontTip, E \subseteq V \times V, C : E \rightarrow CimkeTip\}$

Műveletek: Graf-műveletek +

$$G : Graf, P, P1, P2 : PontTip, S : CimkeTip, I : PIterator,$$

$\{G = (V,E), p1, p2 \in V\}$	$ElBovit(G, p1, p2, s)$	$\{E = Pre(E) \cup \{(p1, p2)\} \wedge C(p1, p2) = s\}$
$\{G = (V,E), (p1, p2) \in E\}$	$ElCimke(G, p1, p2, s)$	$\{s = Pre(C)(p1, p2) \wedge E = Pre(E)\}$
$\{G = (V,E), (p1, p2) \in E\}$	$ElCimkez(G, p1, p2, s)$	$\{s = C(p1, p2) \wedge E = Pre(E)\}$
$\{G = G\}$	$KiCEl(G, p)$	$\{= \{(q, s) : VanEl(p, q) \wedge C(p, q) = s\}\}$
$\{G = G\}$	$ElIterator(G, I)$	$\{\}$

Ha a CimkeTip típuson alapértelmezett lineáris rendezési reláció, akkor a címkézett gráfot súlyozott gráfnak nevezzük. Ekkor az élek halmazán is alapértelmezett az a rendezés, ami az él súlya szerinti rendezés. Tehát $(p_1, q_1) \leq (p_2, q_2) \Leftrightarrow$ ha $C(p_1, q_1) \leq C(p_2, q_2)$.

```
public interface CGraf<Cimke> extends Graf, Iterable<Integer>{
   public void ElBovit(int p, int q, Cimke s);
   public Cimke ElCimke(int p, int q);
   public void ElCimkez(int p, int q, Cimke s);
   public Fuggveny<Integer,Cimke> KiCEl(int p);
   public Iterator<CGrafEl<Cimke>> CElIterator();
}
```

```
public interface SGraf<Suly extends Comparable<Suly>>
 extends Graf, Iterable<Integer>{
   public void ElBovit(int p, int q, Suly s);
   public Suly ElSuly(int p, int q);
   public Fuggveny<Integer,Suly> KiSEl(int p);
   public void ElSulyoz(int p, int q, Suly s);
   public Iterator<SGrafEl<Suly>> SEliterator();
}
```

2.22. Példa Graf adattípus használatára: Erősen összefüggő komponensek kiszámítása

```
import java.lang.Exception;
import java.util.StringTokenizer;
import java.io.*;
//Gráf erősen összefüggő komponenseinek kiszámítása
public class GrafPelda{
   static Graf G;

   private static void BeOlvas() throws IOException{
      BufferedReader bef = new BufferedReader(new FileReader("be.txt"));
      StringTokenizer sor;
      sor = new StringTokenizer(bef.readLine());
      int n = Integer.parseInt(sor.nextToken());
      int m = Integer.parseInt(sor.nextToken());
      G=new GrafA(n,"Lanc");//irányítatlan gráf létesítése
      int p,q;
      for (int i=0; i<m; i++){
         sor = new StringTokenizer(bef.readLine());
         p=Integer.parseInt(sor.nextToken());
         q=Integer.parseInt(sor.nextToken());
         G.ElBovit(p,q);
      }
      bef.close();
   }
```

```
   public static void main (String[] args)throws Exception{
      BeOlvas();
      Halmaz<Halmaz<Integer>>S=GEOK.GEOK(G);
      System.out.println(S.Elemszam());
      for (Halmaz<Integer> H : S){
         for (int x:H)
            System.out.print(x+" ");
         System.out.println();
      }
   }
}
```

```
public class GEOK{
/**Irányított gráf erősen összefüggő komponenseinek kiszámítása
 */
   private enum Paletta {Feher, Szurke, Fekete};
   private static Paletta[] Szin;
   private static Verem<Integer> V;
   private static Graf GT;

   private static void MelyBejar(Graf G, int p){
      Szin[p]=Paletta.Szurke;
      for (int q:G.KiEl(p))      //p->q
         if (Szin[q]==Paletta.Feher)
            MelyBejar(G,q);
      Szin[p]=Paletta.Fekete;
      V.VeremBe(p);
   }
   private static Graf Transzponal(Graf G){
      GT=new GrafA(G.Maxpont(),"Lanc");
      Iterator<GrafEl> elek=G.ElIterator();
      while (elek.hasNext()){
         GrafEl tel=elek.next();
         GT.ElBovit(tel.be, tel.ki);
      }
      return GT;
   }
```

```
private static void MelyKeres(Graf G){
   Szin =new Paletta[G.Maxpont()+1];
   for (int p:G)
      Szin[p]=Paletta.Feher;
   V=new VeremL<Integer>();
   for (int p:G)
      if (Szin[p]==Paletta.Feher)
         MelyBejar(G,p);
}
private static void MelyBejarT(Graf G, int p, Halmaz<Integer> H){
   H.Bovit(p);
   Szin[p]=Paletta.Szurke;
   for (int q:G.KiEl(p))      //p->q
      if (Szin[q]==Paletta.Feher)
         MelyBejarT(G,q, H);
   Szin[p]=Paletta.Fekete;
}

public static Halmaz<Halmaz<Integer>> GEOK(Graf G){
   Halmaz<Integer> H;
   Halmaz<Halmaz<Integer>> S;
   MelyKeres(G);
   GT=Transzponal(G);
         for (int p:G)
      Szin[p]=Paletta.Feher;
```

```
      S=new HalmazL<Halmaz<Integer>>();
      while (V.NemUres()){
         int p=V.VeremBol();
         if (Szin[p]==Paletta.Feher){
            H=new HalmazL<Integer>();
            MelyBejarT(GT,p, H);
            S.Bovit(H);
         }
      }
      return S;
   }
}
```