3. Gyűjtemények I.

1. Zsák típus

Valósítsuk meg egy adott halmaz (E) elemeit tartalmazó zsák típusát úgy, hogy nincs felső korlát a zsákba bekerülő elemek számára. A szokásos (üres-e, betesz, kivesz, hányszor van benn egy szám) műveletek mellett szükségünk lesz a leggyakoribb elemet lekérdező műveletre is.

Bag

<table>
<tr>
<td>azon zsákok halmaza, amelyek elemei (E) rendezhetőek</td>
<td>l := Empty(b)   b : Bag, l : $\mathbb{L}$ // üres-e zsák
c:= Multipl(b, e)   b : Bag, e : E, c : $\mathbb{N}$ // multiplicitás
m:= Max(b)   b : Bag, m : E   // leggyakoribb
b := SetEmpty(b)   b : Bag   // kiüríti a zsákot
b := Insert(b, e)   b : Bag, e : E   // elemet tesz be
b:= Remove(b, e)   b : Bag, e : E   // elemet vesz ki</td>
</tr>
</table>

Egy zsák reprezentálására (elemeinek tárolására) több lehetőség is van:

- elemek sorozata (ugyanaz az elem többször is előfordulhat): rendezett vagy rendezetlen
- elem és előfordulása által alkotott párok sorozata: elem szerint rendezett vagy rendezetlen

Kitérő: Elem keresése egy elem és darabszám párokból álló rendezett sorozatban

Logaritmikus keresés

$A = (seq : Pair^*, e : E, l : \mathbb{L}, ind : \mathbb{N}) \qquad Pair = rec(data: E, count: \mathbb{N})$

$Ef = (seq=seq_0 \wedge e=e_0 \wedge \forall i\in[1 .. |seq|-1] : seq[i].data \leq seq[i+1].data)$

$Uf = (Ef \wedge l = \exists i\in[1 .. |seq|] : seq[i].data = e \wedge$
$(l \rightarrow ind\in[1 .. |seq|] \wedge seq[ind].data = e) \wedge$
$(\neg l \rightarrow \forall i\in [1..ind-1]: seq[i].data < e \wedge \forall i\in[ind.. |seq|] : seq[i].data > e))$

l, ind := LogSearch (seq, e)

<table>
<tr><td colspan="3">l, ah, fh := hamis, 1, |seq|</td></tr>
<tr><td colspan="3">$\neg l \wedge ah \leq fh$</td></tr>
<tr><td colspan="3">ind := ⌊(ah + fh) / 2⌋</td></tr>
<tr><td>seq[ind].data > e</td><td>seq[ind].data = e</td><td>seq[ind].data < e</td></tr>
<tr><td>fh := ind-1</td><td>l := igaz</td><td>ah := ind+1</td></tr>
<tr><td colspan="3">$\neg l$</td></tr>
<tr><td>ind := ah</td><td colspan="2">—</td></tr>
</table>

ah, fh : $\mathbb{N}$

seq : Pair*

maxind : ℕ

ahol

Pair = rec(data: E, count: ℕ)

Inv:

- a seq-ben az elemeket tartalmuk (data) szerint rendezetten tároljuk

seq↗$_{data}$

- a maxind a nem üres seq sorozat legnagyobb count értékű elemének indexe

|seq|>0 →

$(\_ , maxind) = \mathrm{MAX}_{i=1..|seq|}\ seq[i].data$

l := Empty(b) b : Bag, l : 𝕃

l := |seq|=0

c := Multipl(b, e) b : Bag, e : E, c : ℕ

l, ind := LogSearch(seq, e)
if l **then** c :=seq[ind].count **else** c := 0 **endif**

m := Max(b) b : Bag, m : E

|seq| > 0	
m := seq[maxind].data	hiba

b := SetEmpty(b) b : Bag

seq := <>

b := Insert(b,e) b : Bag, e : E

l, ind := LogSearch(seq, e)				
l				
++ seq[ind].count		seq:=seq.Insert(ind, (e,1))		
seq[ind].count > seq[maxind].count		|seq|=1	|seq|>1 ∧ maxind≥ind	else
maxind := ind	–	maxind := 1	++maxind	–

b := Remove(b,e) b : Bag, e : E

l, ind := LogSearch(seq, e)		
l		
seq[ind].count > 1	seq[ind].count = 1	–
seq[ind].count := seq[ind].count–1	seq.Remove(ind)	
|seq|>0		
max, maxind := $\mathrm{MAX}_{i=1..\|seq\|}$ (seq[i].count)	–	

Osztály:

```
Pair
+ data : E
+ count : nat
```

```
Bag
- seq : Pair[]
- maxind : nat
+ Empty() : bool {query}
+ Multipl(e : E) : int {query}
+ Max() : E {query}
+ SetEmpty()
+ Insert(e: E)
+ Remove(e: E)
- LogSearch(e:E) : (bool,int)  {query}
```

Empty():

```
return |seq|=0
```

Multipl(e : E):

```
l, ind := LogSearch(e)
if l then  return seq[ind].count
else return 0
endif
```

Max():

```
if |seq|>0 then return seq[maxind].data else error endif
```

SetEmpty():

```
seq := <>
```

Insert(e: E):

```
l, ind := LogSearch(e)
if l then
   ++seq[ind].count
   if seq[ind].count > seq[maxind].count then maxind := ind endif
else
   seq.Insert(ind, (e,1))
   if |seq|=1 then maxind := 1
   elsif maxind>ind then ++maxind
   endif
endif
```

Remove(e: E):

```
l, ind := LogSearch(e)
if l then
   if seq[ind].count > 1 then - - seq[ind].count
   elsif seq[ind].count  = 1 then seq.Remove(ind)
   endif
   if |seq|>0 then max, maxind := MAX_{i=1..|seq|} (seq[i].count) endif
endif
```

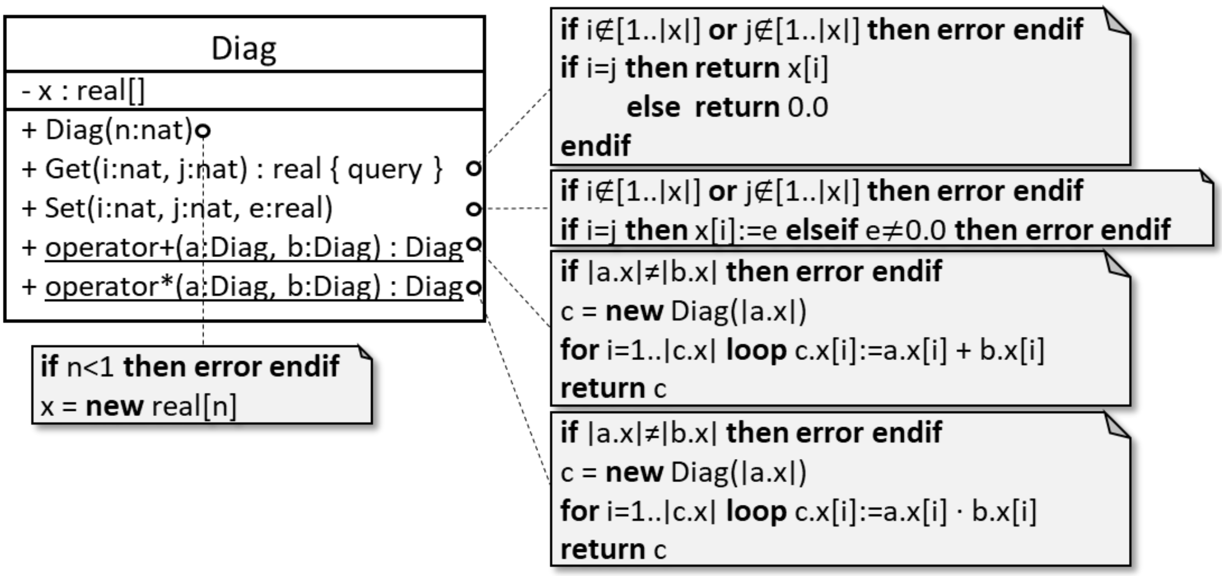
LogSearch(e:E):

```
l, ah, fh := false, 1, |seq|
while not l and ah ≤ fh loop
   ind := ⌊(ah + fh) / 2⌋
   if      seq[ind].data > e then  fh := ind-1
   elsif seq[ind].data = e  then     l := true
   elsif seq[ind].data < e then ah := ind+1
endloop
if not l then ind := ah endif
return (l, ind)
```

2. Diagonális mátrix
   Valósítsuk meg a diagonális mátrix típust (az ilyen mátrixoknak csak a főátlójukban lehetnek nullától eltérő elemek)! Ilyenkor elegendő csak a főátlóbeli elemeket tárolni egy sorozatban. Implementáljuk a mátrix i-edik sorának j-edik elemét lekérdező, illetve megváltoztató műveleteket, valamint két mátrix összegét és szorzatát kiszámoló műveleteket!

Diag // dim(a) ~ egy 'a' mátrix sorainak száma, dim(a)≥ 1

olyan négyzetes mátrixok, amelyek a főátlójukon kívül csak nullákat tárolnak	e := a[i,j] (a : Diag, i,j : [1.. dim(a)], e : $\mathbb{R}$)
	a[i,j] := e (a : Diag, i,j : [1.. dim(a)], e : $\mathbb{R}$) // i=j
	c := a + b (a, b, c : Diag) // dim(a)=dim(b)=dim(c)
	c := a · b (a, b, c : Diag) // dim(a)=dim(b)=dim(c)
x : $\mathbb{R}^n$ Invariáns: n ≥ 1	**if** i=j **then** e := a.x[i] **else** e := 0.0 **endif**
	if i=j **then** a.x[i] := e **elseif** e≠0.0 **then error endif**
	if not (a.n = b.n = c.n) **then error endif** **for** i=1 .. a.n **loop** c.x[i]:=a.x[i]+b.x[i] **endloop**
	if not (a.n = b.n = c.n) **then error endif** **for** i=1 .. a.n **loop** c.x[i]:=a.x[i]·b.x[i] **endloop**



A kódolásnál:

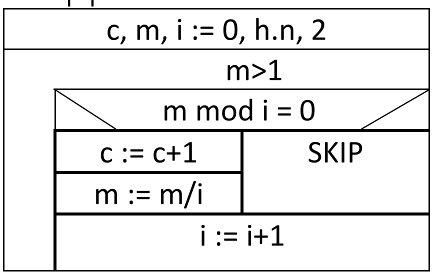
- bevezethetünk más konstruktorokat is
- a Get() és Set() helyett használjuk a C# „indexer”-ét

3. Prímek halmazai. Ábrázoljuk a pozitív prím számokból álló halmazokat a bennük levő számok szorzatával; az üres halmazt az 1-gyel.

PrimSet

prímszámok véges halmazai	l := p ∈ h (h : PrimSet, p : ℙ, l : 𝕃)
	h := h ∪ p (h : PrimSet, p : ℙ) // ha p ∈ h, akkor hatástalan
	h := h \ p (h : PrimSet, p : ℙ) // ha p ∉ h, akkor hatástalan
	l := h = ∅ (h : PrimSet, l : 𝕃)
	h := ∅ (h : PrimSet)
	c := |h| (h : PrimSet, c : ℕ)
n : ℕ ahol az n számnak a prímtényezős felbontásában a prímek egyszeresen fordulnak elő	l := (n mod p=0)
	if n mod p ≠ 0 **then** n := n·p **endif**
	if n mod p = 0 **then** n := n/p **endif**
	l := (n=1)
	n := 1
	lásd külön

c := |h|



m, i : ℤ