# OEP

## 4. táblás gyakorlat

adattípus definiálás

sorozattal reprezentált típusok II.

# Tartalom

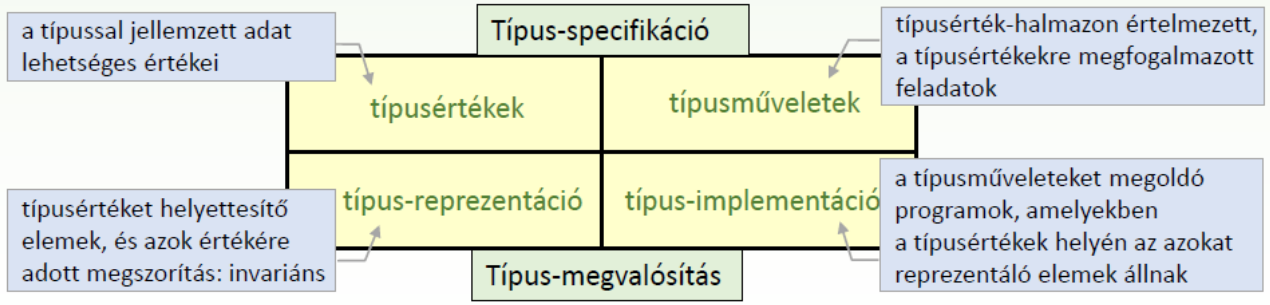

- Asszociatív tömb
- Logaritmikus keresés tétel
- Zsák rendezett tömbbel

# Adattípus fogalma (1. előadás)

- Típus-specifikáció
  - Típusértékek
  - Típusműveletek
- Típus megvalósítás
  - Típus-reprezentáció
  - Típus-implementáció

## Adattípus fogalma

- Egy adat (változó) típusának definiálásához szükség van a típus specifikációjára és annak megvalósítására.
- A típus-specifikáció megadja:
  - az adat által felvehető értékek halmazát
  - a típusértékekkel végezhető műveleteket
- A típus-megvalósítás megmutatja:
  - hogyan ábrázoljuk (reprezentáljuk) a típus értékeit
  - milyen programok helyettesítsék (implementálják) a műveleteket



# Asszociatív tömb

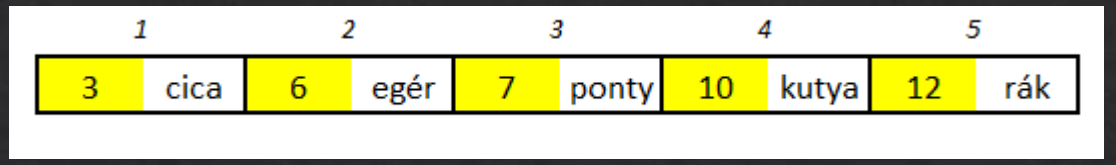
- Ez egy olyan kulcs-adat párokat tároló gyűjtemény, amelyben kulcs alapján lehet visszakeresni az értékeket.
- A kulcs típusa egész lesz, az adat típusa pedig szöveg.
- A tárolóban az elemeket kulcsuk alapján lehet megkeresni, elérni, így fontos, hogy a kulcs egyedi legyen.
- A típust Map-nek fogjuk nevezni.
- A tervezésnél fontos szempont, hogy a visszakeresés gyors legyen, akár a beszúrás, törlés rovására!

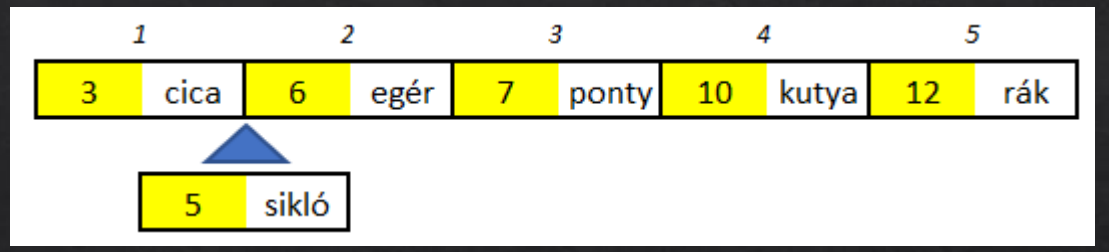
# Típus specifikáció

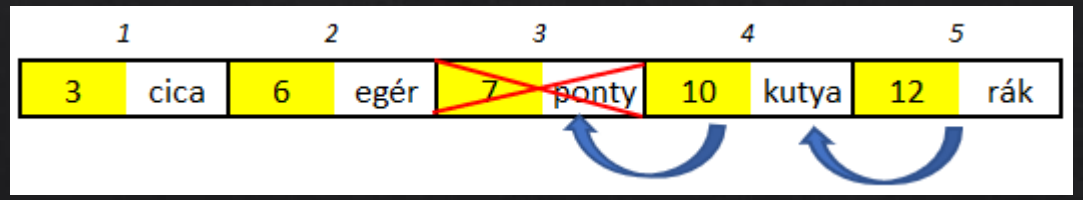
| Típus értékek: | Típus műveletek: |
| --- | --- |
| Map<br>azon asszociatív tömbök halmaza, amely tömböknek az elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok | setEmpty(map) map : Map<br>//kiüríti az asszociatív tömböt<br>c := count(map) map : Map, c : $\mathbb{N}$<br>//megadja az elemek számát<br>map := insert(map,e) map : Map, e : $\mathbb{Z} \times \mathbb{S}$<br>//új elemet tesz be, ha a kulcsa még nem létezik<br>map := erase(map, key) map : Map, key : $\mathbb{Z}$<br>// törli az adott kulcsú elemet, ha a kulcs létezik, különben hiba<br>l := in(map, key) map : Map, key : $\mathbb{Z}$, l : $\mathbb{L}$<br>//lekérdezi, van-e adott kulcsú elem<br>data := operator[](map, key) map : Map, key : $\mathbb{Z}$, data : $\mathbb{S}$<br>// lekérdezi az adott kulcsú elem adatát, ha a kulcs létezik, különben hiba |

# Kulcs szerint rendezett tárolás

- Kulcs szerint szigorúan monoton növekedően tároljuk az asszociatív tömb elmeit. A sorozatban kulcs szerint szigorúan monoton növekvő sorrendben tároljuk az elemeket.
- Ekkor a kulcsok kereséséhez használhatjuk a logaritmikus keresést.
- Ha tömbben tárolnánk az elemeket, akkor beszúrásnál „helyet kell csinálni” az új elemnek: hátrébb kell csúsztatni az új elemnél nagyobb kulcsúakat.
- törlésnél pedig a keletkezett „lyukat” el kell tüntetni: előrébb kell csúsztatni a törlés pozíciója utáni elemeket.



# Map reprezentáció

- Típus reprezentációja:
  - **Item** = rec(key: $\mathbb{Z}$, data: $\mathbb{S}$)
  - **seq: Item*** – az elemeket kulcsuk szerint rendezetten tároló sorozat
- Típus műveletek implementációja:
  - **map:=setEmpty(map) map:Map** *//kiüríti az asszociatív tömböt*

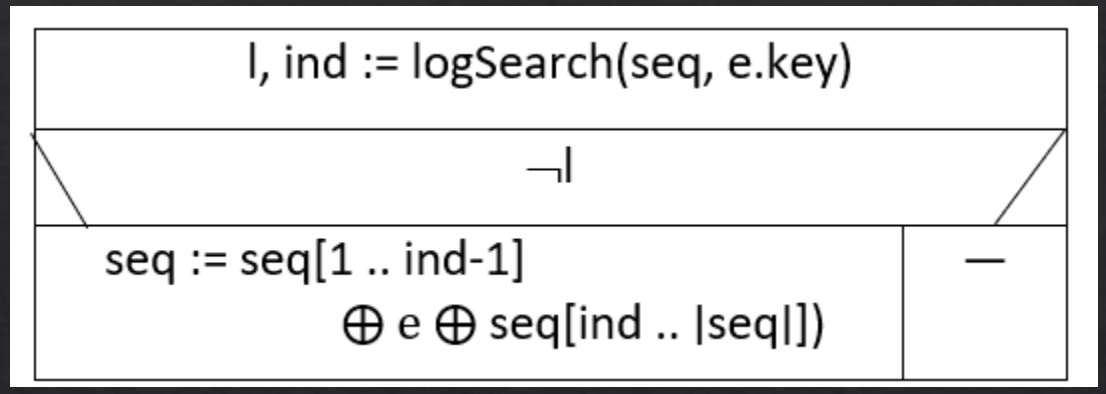
```
seq := <>
```

  - **c := count(map) map:Map, c:** $\mathbb{N}$ *//megadja az elemek számát*

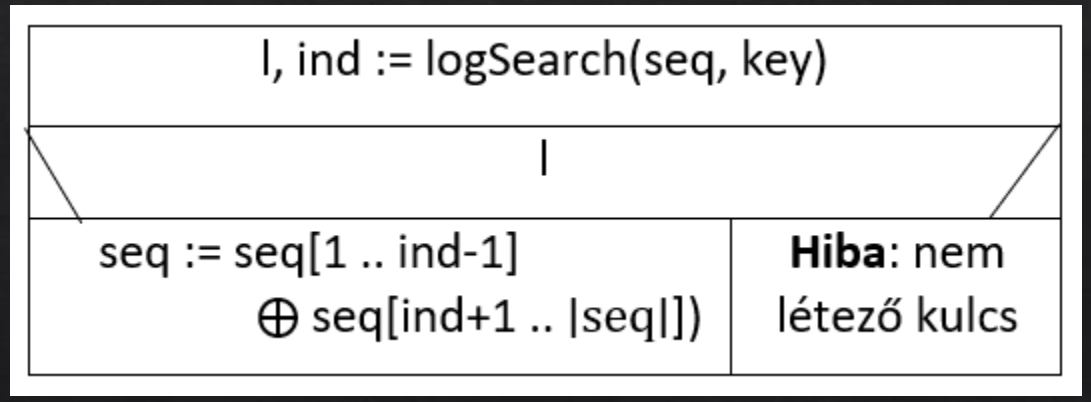
```
c := |seq|
```

# Map reprezentáció

- **map := insert(map,e)** map:Map, e: Item *//új elemet tesz be, ha a kulcsa még nem létezik*



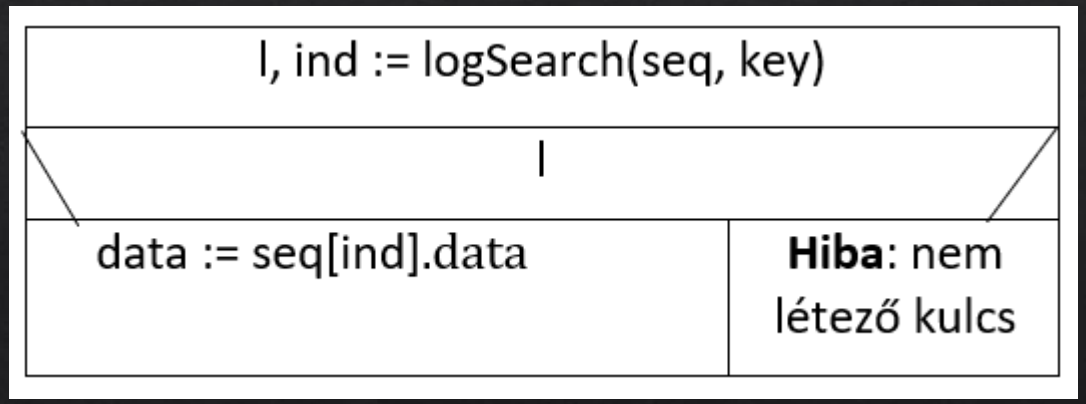

- **map := erase(map, key)** map:Map, key: ℤ
  *//törli az adott kulcsú elemet, ha a kulcs létezik, különben hiba*



# Map reprezentáció

- **l := in(map, key)** map:Map, key: $\mathbb{Z}$, l : $\mathbb{L}$ *//lekérdezi, van-e adott kulcsú elem*


- **data := operator[](map, key)** map:Map, key: $\mathbb{Z}$ , data: $\mathbb{S}$
  *// lekérdezi az adott kulcsú elem adatát, ha a kulcs létezik, különben hiba*

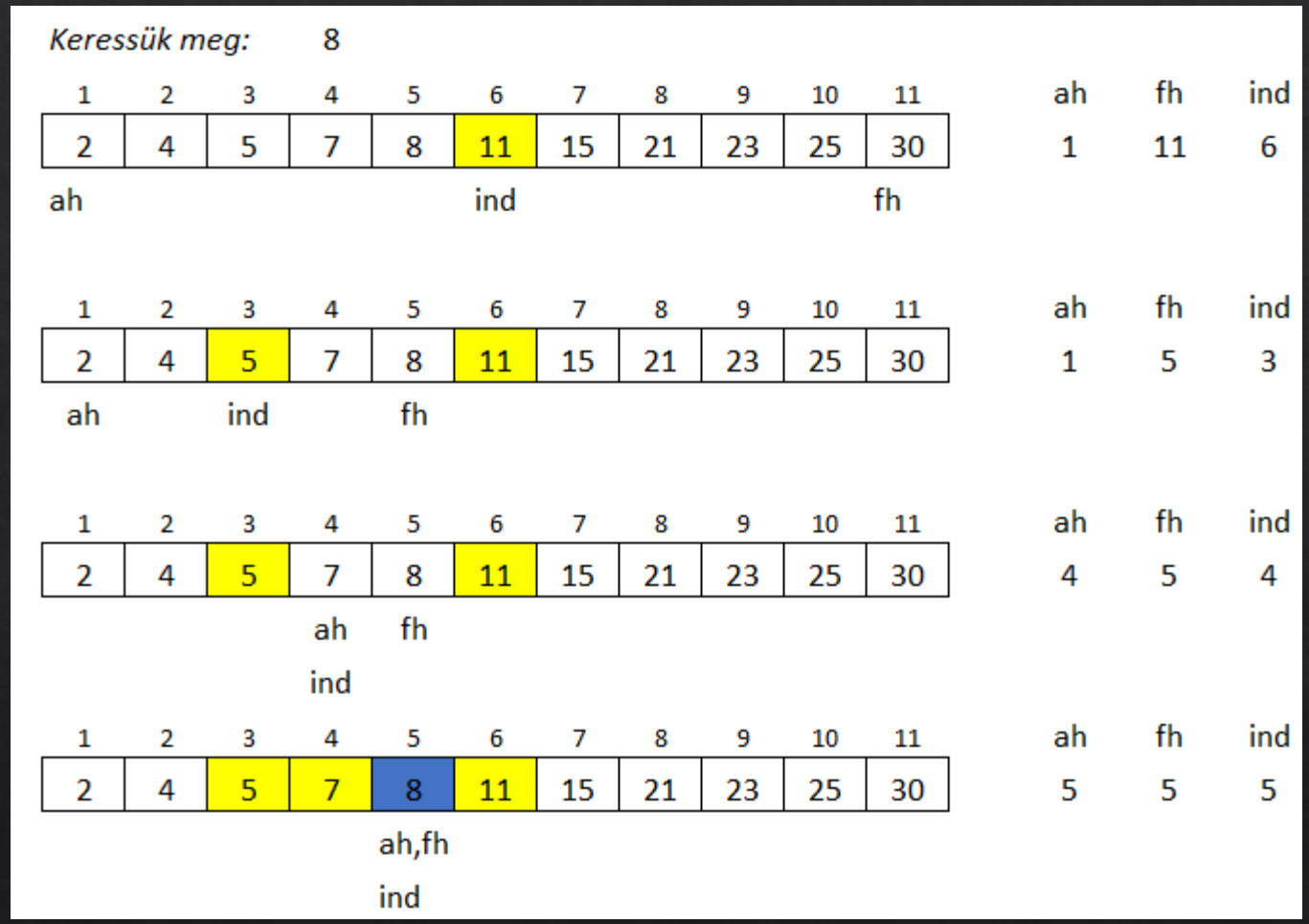


# Logaritmikus keresés

- Rendezett sorozatban nagyon hatékony keresést biztosít.
- Elfelezve a keresés intervallumát, a középső elemmel összehasonlítja a keresett kulcsot, három eset lehetséges: [ah…fh]  ind=(ah+fh) div 2    //egész osztás!
  - Keresett kulcs kisebb – bal oldali részben folytatja a keresést: [ah…ind-1]
  - Keresett kulcs nagyobb – jobb oldali részben folytatja a keresést [ind+1…fh]
  - Egyenlő – megtaláltuk, készen vagyunk: ind a keresett elem indexe
  - Ha leszűkítés után kapott intervallum üres, akkor a keresés véget ért, sikertelen volt

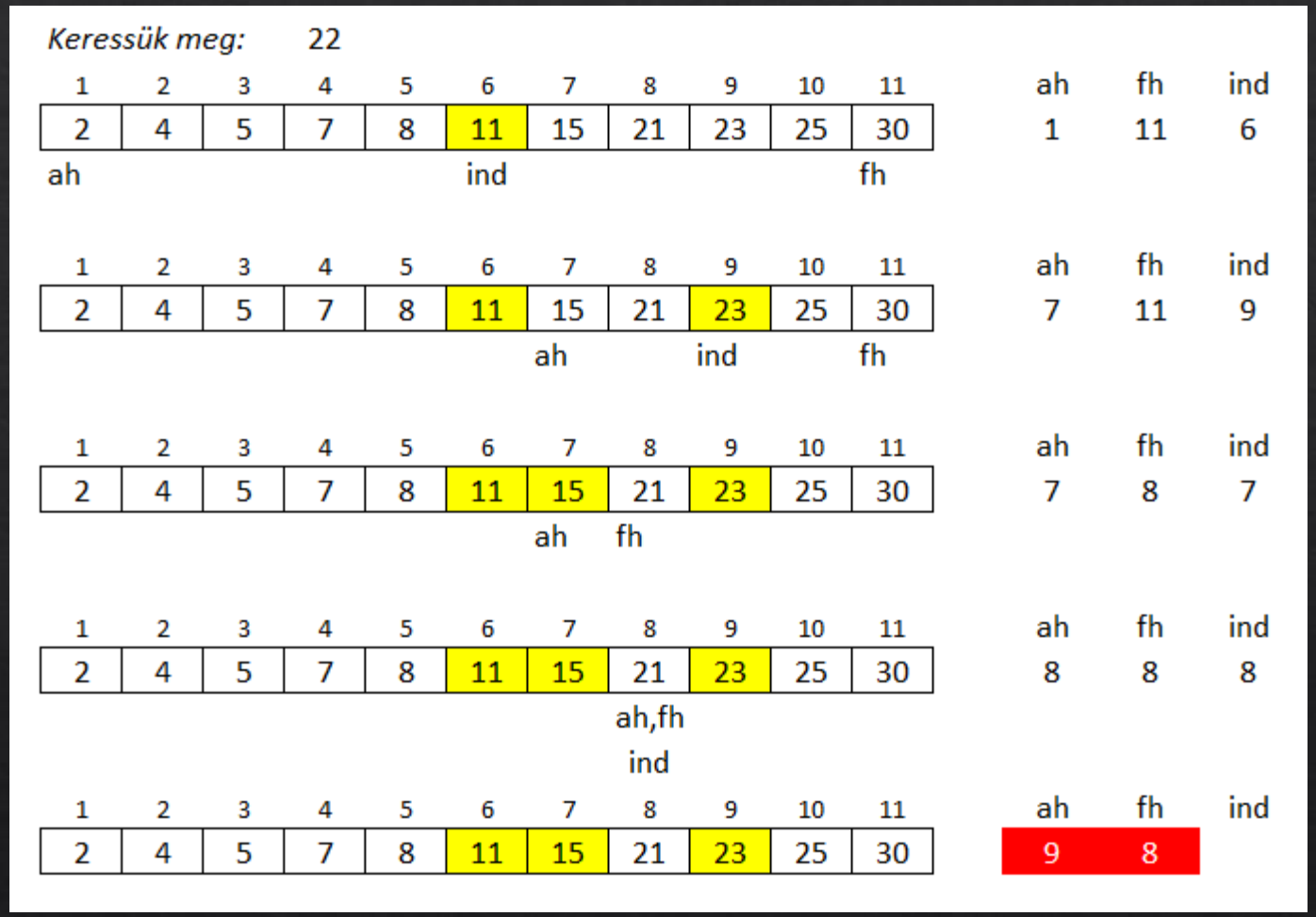
# Logaritmikus keresés

- Keresett kulcs megtalálható:



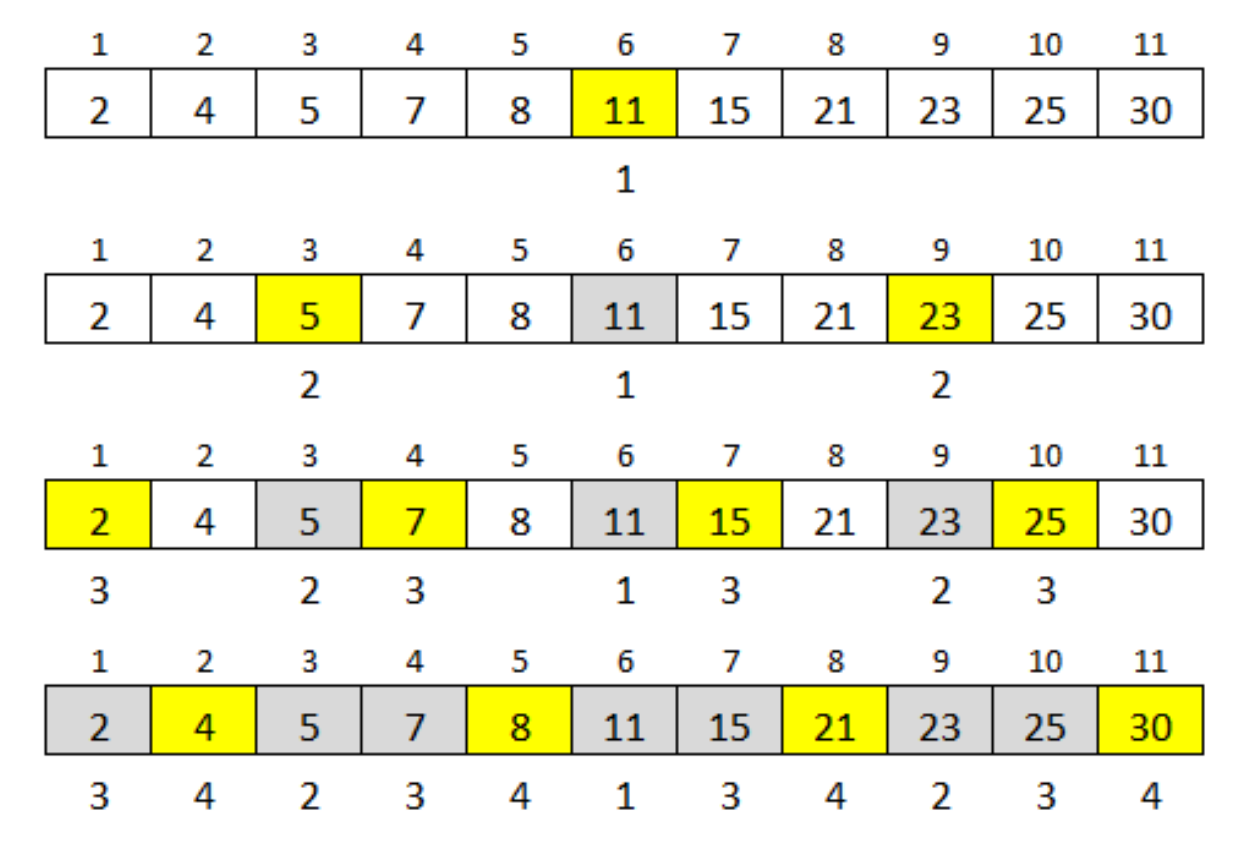
# Logaritmikus keresés

- Keresett kulcs nem található:



# Logaritmikus keresés

- Honnan van a neve?
- Legfeljebb $\lceil \log_2 n \rceil$ összehasonlítást hajt végre az algoritmus:



# Logaritmikus keresés programozási tétel

***7. Logaritmikus keresés***

*Feladat*: Adott az egész számok egy $[m..n]$ intervalluma és egy $f:\mathbb{Z}\to H$ függvény, amelyik az $[m..n]$ intervallumon monoton növekvő. (A $H$ halmaz elemei között értelmezett egy rendezési reláció.) Keressünk meg a függvény $[m..n]$ intervallumon felvett értékei között egy adott értéket!

*Specifikáció*:

$$A = (m:\mathbb{Z}, n:\mathbb{Z}, h:H, l:\mathbb{L}, ind:\mathbb{Z})$$
$$Ef = ( m=m' \wedge n=n' \wedge h=h' \wedge \forall j\in[m..n-1]: f(j) \leq f(j+1) )$$
$$Uf = ( Ef \wedge l=( \exists j\in[m..n]: f(j)=h ) \wedge l\to( ind\in[m..n] \wedge f(ind)=h ) )$$

# Logaritmikus keresés algoritmusa

*Specifikáció*:

$A = (m{:}\mathbb{Z}, n{:}\mathbb{Z}, h{:}H, l{:}\mathbb{L}, ind{:}\mathbb{Z})$

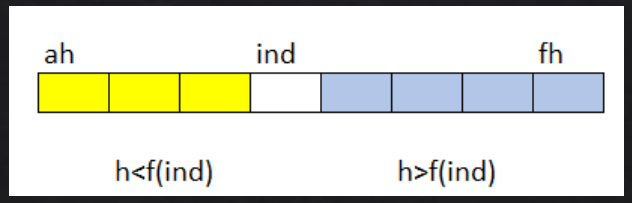
$Ef = ( m{=}m' \wedge n{=}n' \wedge h{=}h' \wedge \forall j{\in}[m..n\text{-}1]{:}\ f(j) \leq f(j{+}1) )$

$Uf = ( Ef \wedge l{=}( \exists j{\in}[m..n]{:}f(j){=}h ) \wedge l{\rightarrow}( ind{\in}[m..n] \wedge f(ind){=}h ) )$

*Algoritmus*:

| $ah,fh,l := m,n,hamis$ | | | $ah, fh{:}\mathbb{Z}$ |
|---|---|---|---|
| $\neg l \wedge ah \leq fh$ | | | |
| $ind := (ah+fh)\ div\ 2$ | | | |
| $f(ind)>h$ | $f(ind)<h$ | $f(ind)=h$ | |
| $fh := ind\text{-}1$ | $ah := ind{+}1$ | $l := igaz$ | |

kvíz2



# Amikor nem találjuk…

*Algoritmus*:

| *ah,fh,l :=m,n,hamis* | | | *ah, fh*:ℤ |
|---|---|---|---|
| ¬*l* ∧ *ah*≤*fh* | | | |
| *ind :=(ah+fh) div 2* | | | |
| *f(ind)>h* | *f(ind)<h* | *f(ind)=h* | |
| *fh := ind-1* | *ah := ind+1* | *l := igaz* | |

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 5 | 7 | 8 | 11 | 15 | 21 | 23 | 25 | 30 |
| 3 | 4 | 2 | 3 | 4 | 1 | 3 | 4 | 2 | 3 | 4 |

- Keresett kulcs 17: ah,fh=8, ind=8; h<A[8] → ah=8, fh=7
- Keresett kulcs 22: ah,fh=8, ind=8; h>A[8] → ah=9, fh=8
- Keresett kulcs 1: ah=1,fh=2, ind=1; h<A[1] → ah=1, fh=0
- Keresett kulcs 32: ah,fh=11, ind=11; h>A[1] → ah=12, fh=11
- Hova illik (melyik elem elé): **ah**

# l, ind := logSearch (seq, key)

- Specifikáció:
- A = (seq : Item*, key : $\mathbb{Z}$, l : $\mathbb{L}$, ind : $\mathbb{N}$)
- Ef = (seq=$seq_0$ $\wedge$ key=$key_0$ $\wedge$ $\forall i \in [1 .. |seq|-1]$ : seq[i].key < seq[i+1].key )
- Uf=(Ef $\wedge$ l = $\exists i \in [1 .. |seq|]$ : seq[i].key = key $\wedge$

  ( l $\rightarrow$ ind$\in$[1 .. |seq|] $\wedge$ seq[ind].key = key ) $\wedge$

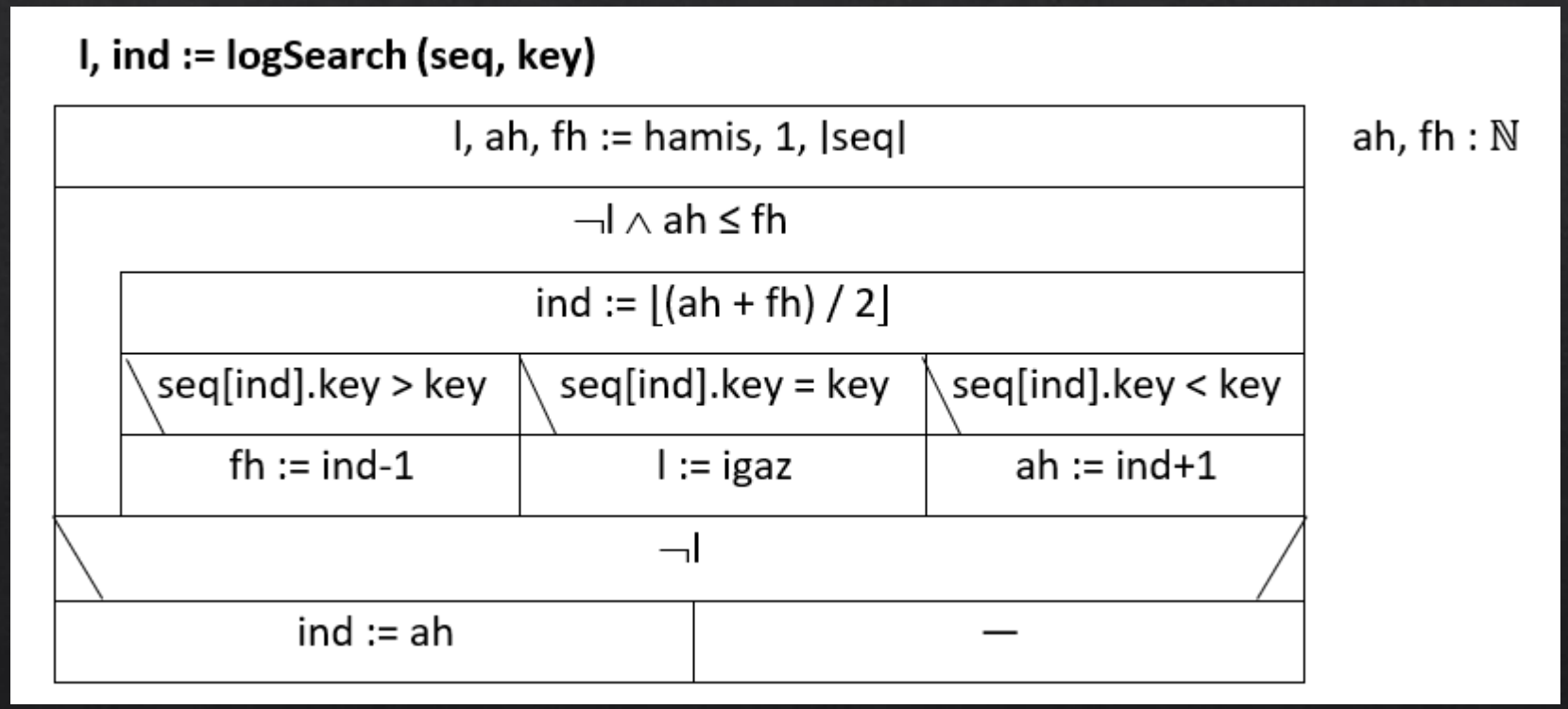
  ($\neg$l $\rightarrow$ $\forall i \in$ [1..ind-1]: seq[i].key < key $\wedge$ $\forall i \in$[ind.. |seq|] : seq[i].key > key) )

# l, ind := logSearch (seq, key)

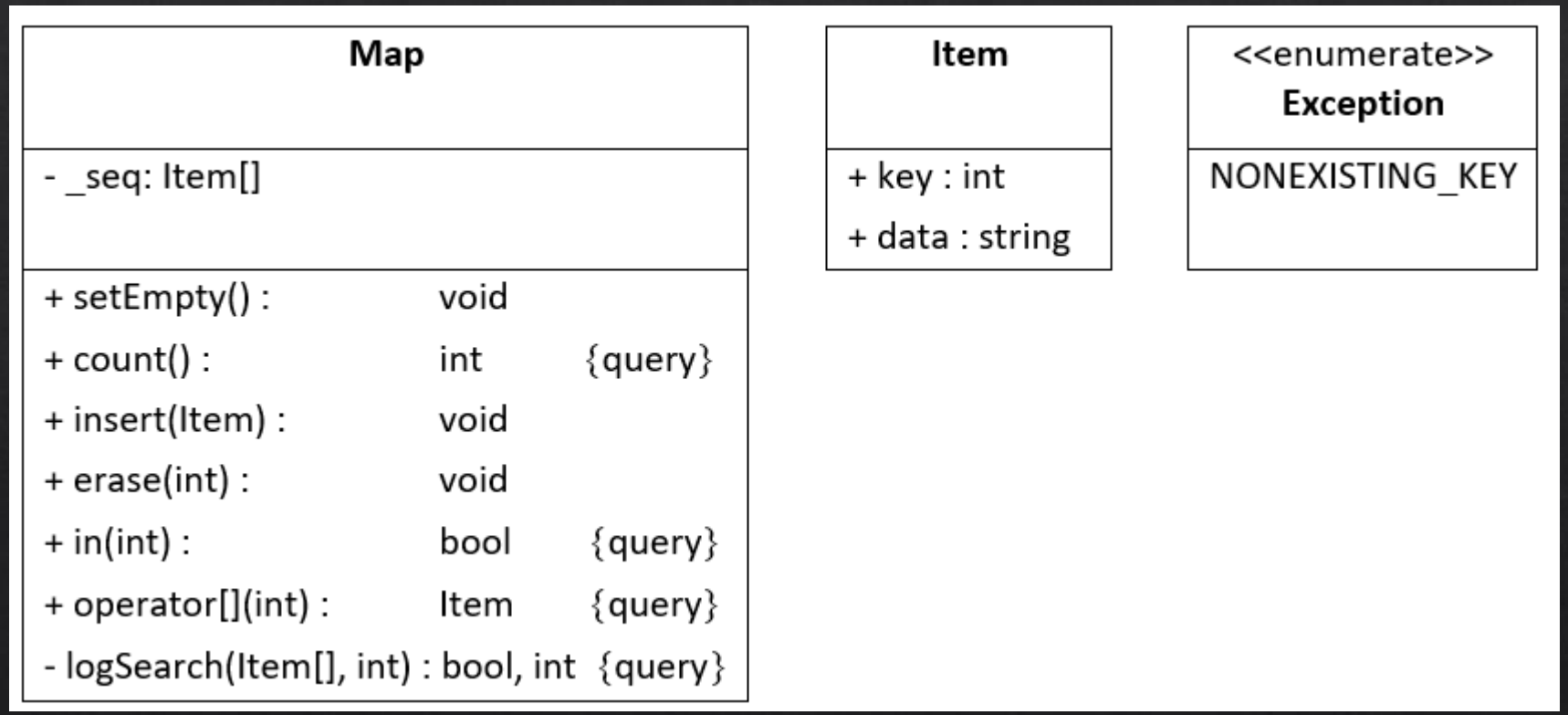
A = (seq : Item*, key : $\mathbb{Z}$, l : $\mathbb{L}$, ind : $\mathbb{N}$)

Ef = (seq=$seq_0$ $\wedge$ key=$key_0$ $\wedge$ $\forall i \in [1 .. |seq|-1]$ : seq[i].key < seq[i+1].key )

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



# Asszociatív tömb UML ábra



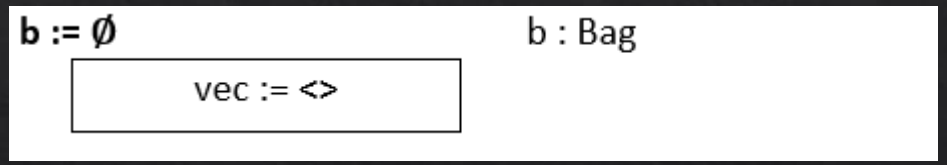
# Zsák típus

- Az előadáson bemutatott változattal szemben, most egy olyan zsák típust kellene definiálni, ahol nincs felső korlát a zsákba bekerülő természetes számokra.
  (Esetleg a zsák elemeinek típusa lehet akármilyen típus, csak lehessen annak értékeit sorba rendezni.)

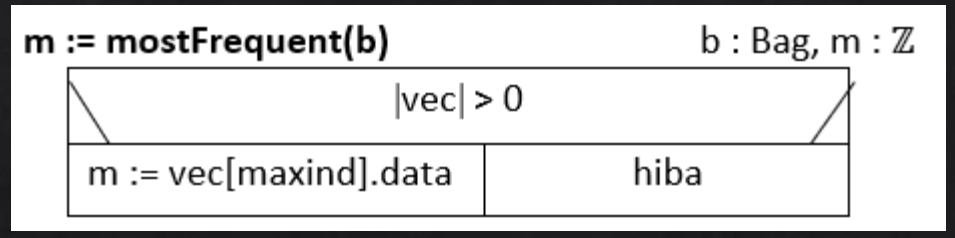
| **Bag**<br>**azon zsákok halmaza, amelyek elemei egész számok (ℤ)** | **b := ∅** **b : Bag** ***// kiüríti a zsákot***<br>**b := putIn(b, e)** **b : Bag, e : ℤ** ***// elemet tesz be***<br>**m := mostFrequent(b)** **b : Bag, m : ℤ** ***// leggyakoribb elem***<br>**b:=takeOut(b, e)** **b : Bag, e : ℤ** ***// elemet vesz ki*** |
|---|---|

# Zsák típus reprezentáció

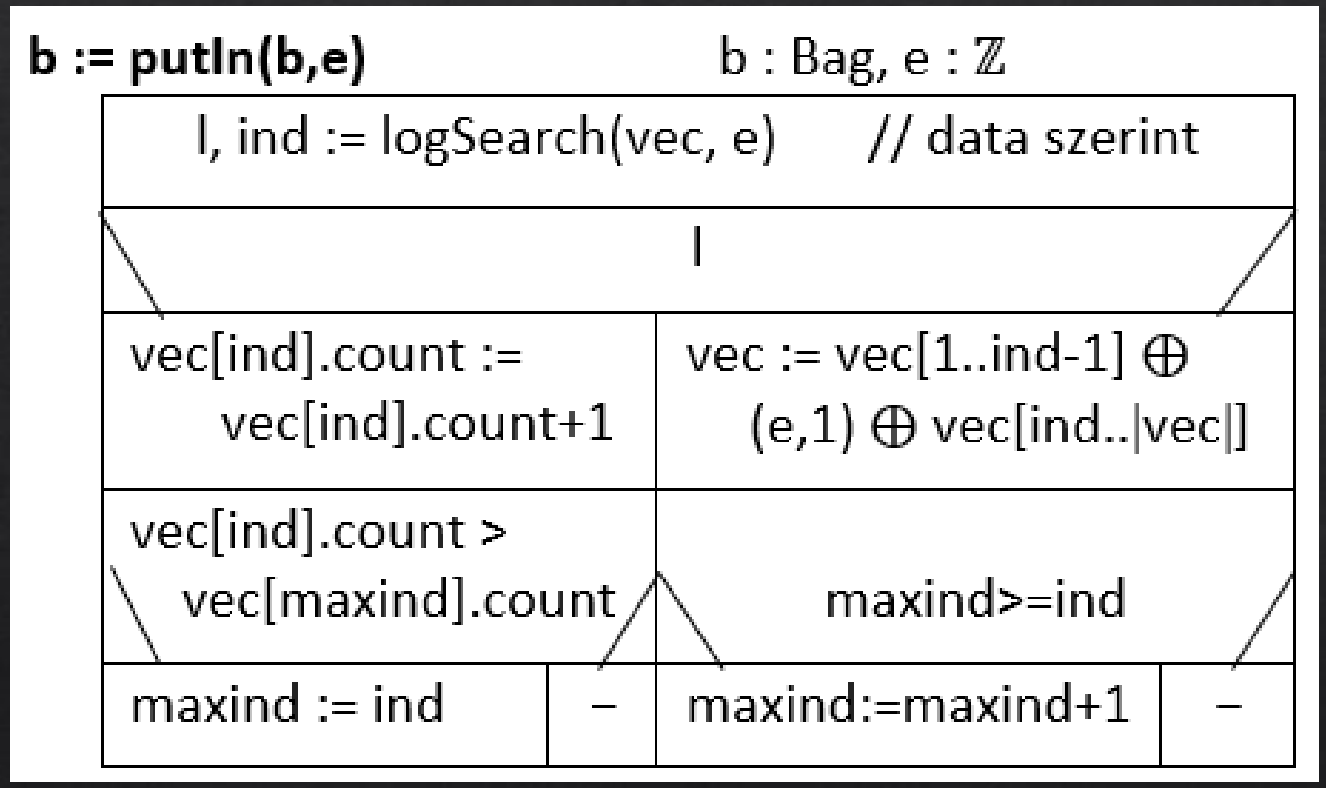
- Típus ábrázolása:
  - vec : Item* – az elemeket tartalmuk (data) szerint rendezetten tároló sorozat,
  - ahol  Item = rec(data: $\mathbb{Z}$, count: $\mathbb{N}$)
  - [kvíz4]
  - maxind : $\mathbb{N}$  – a seq sorozat legnagyobb count értékű elemének indexe.
- Műveletek implementációja:
  - Zsákot üressé tesz:




  - Legyakoribb elem a zsákban:



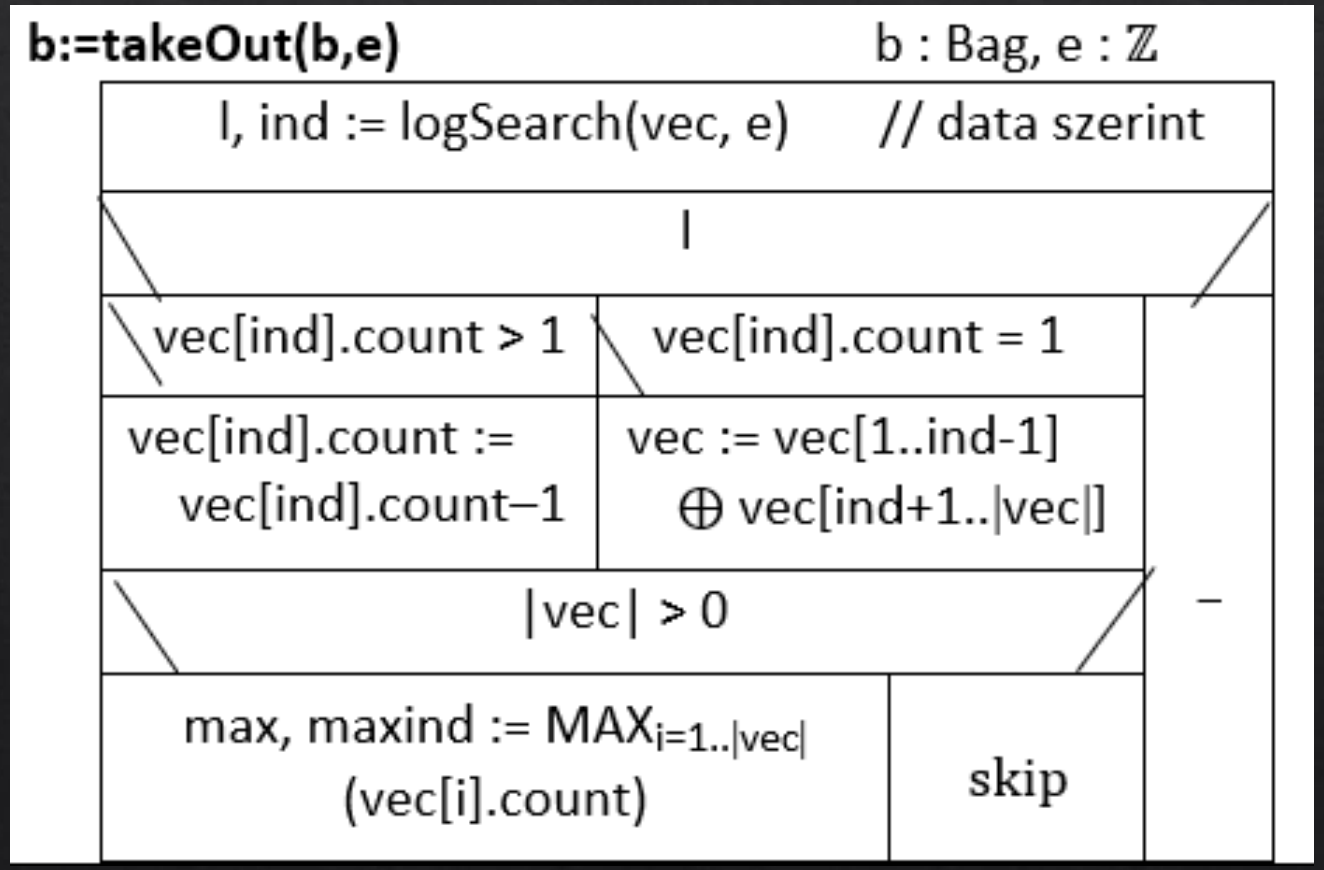
# Zsák típus reprezentáció - betevés




Lépések:

- Logaritmikus keresés, van-e zsákban már „e” érték?
- Igen: count-ot növeljük eggyel, majd ellenőrizzük, hogy kell-e módosítani a leggyakoribb elem indexét.
- Nem: beszúrjuk az elemet a rendezettség szerinti helyére. Ha maxind egy nagyobb indexű elemre mutat, akkor eggyel növelni kell

# Zsák típus reprezentáció - kivevés




Lépések:

- Logaritmikus keresés, van-e zsákban „e” érték?
- Van, és több mint egy: count-ot csökkentjük eggyel.
- Van, de csak egy: az adott elmet kihagyjuk a sorozatból.
- Nincs: hatástalan.
- Ha volt törlés, a leggyakoribb elem megváltozhatott, frissíteni kell.

# Önálló feladat

- Készítsünk UML diagramot a zsák típushoz!
  (Induló fájl: Bag_UML.docx)