OEP

3. táblás gyakorlat

adattípus definiálás

sorozattal reprezentált típusok

Tartalom

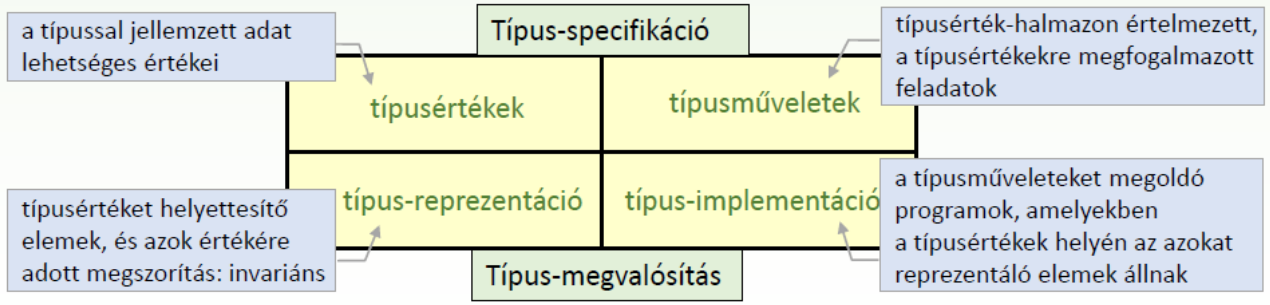
- Diagonális mátrix
- Alsó háromszög mátrix
- Prioritásos sor

Adattípus fogalma (1. előadás)

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

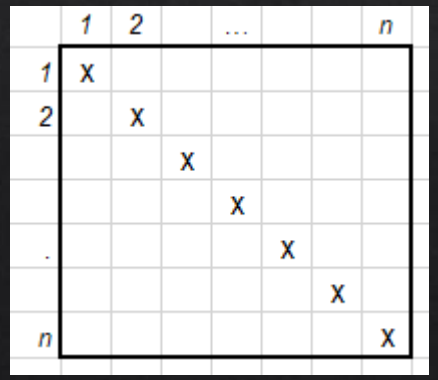
Adattípus fogalma

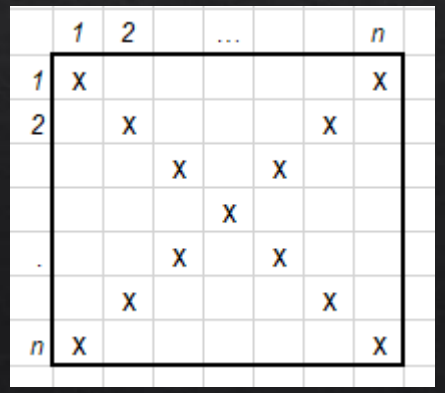
- Egy adat (változó) típusának definiálásához szükség van a típus specifikációjára és annak megvalósítására.
- A típus-specifikáció megadja:
  - az adat által felvehető értékek halmazát
  - a típusértékekkel végezhető műveleteket
- A típus-megvalósítás megmutatja:
  - hogyan ábrázoljuk (reprezentáljuk)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

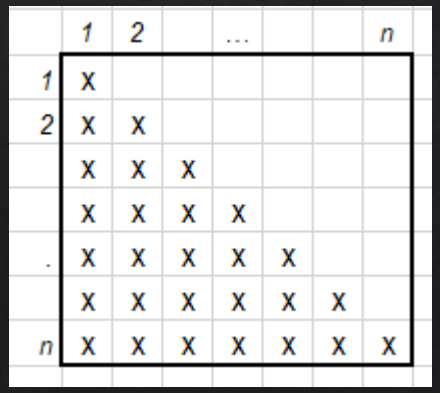


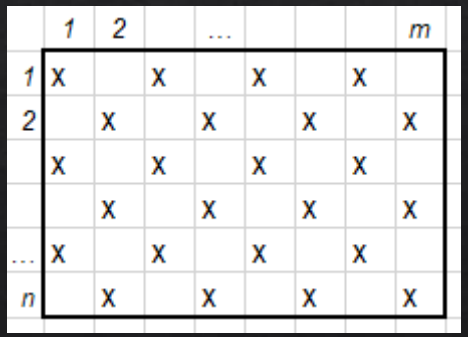
Ritka mátrixok helytakarékos ábrázolása

- Ritka mátrixok:
  - Bizonyos elemek mindig nulla értékűek.
  - A „nem nulla” elemek szabályos elhelyezkedésűek.
- Helytakarékos ábrázolás:
  - Nagy méret esetén érdemes csak azokat a mátrix elemeket tárolni, amelyek nullától különböző értéket (is) felvehetnek.



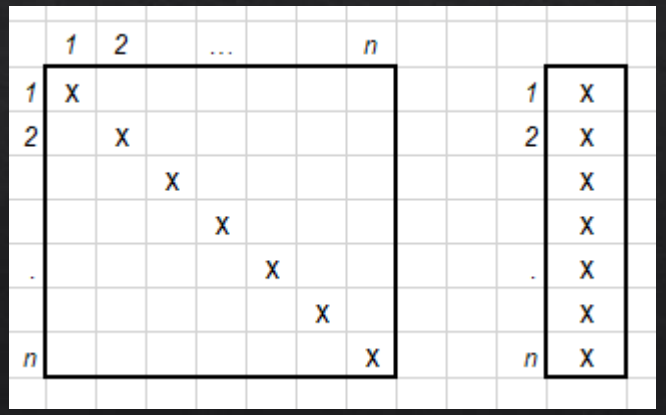






Diagonális mátrix

- Négyzetes mátrix
- Csak a főátlójában szerepelhetnek nullától különböző elemek:
  - A[i,j] = 0, $\forall$ 1≤i,j≤n, i≠j esetén
  - A[i,j] ≠ 0, $\forall$ 1≤i,j≤n, i=j esetén
- Tárolás: n méretű egydimenziós tömbben lehetséges:



index(i,j) = i

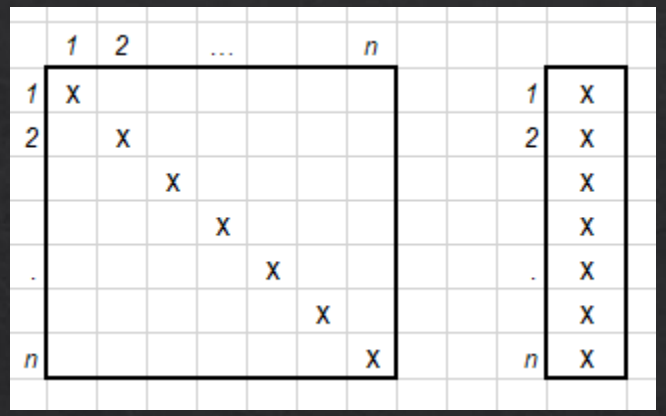
Diagonális mátrix

- Típus specifikáció (típus értékek, műveletek)

$\mathrm{Diag}(\mathbb{R}^{n\times n})$

c := a+b	$(a, b, c : \mathrm{Diag}(\mathbb{R}^{n\times n}))$	
c := a·b	$(a, b, c : \mathrm{Diag}(\mathbb{R}^{n\times n}))$	
e := a[i,j]	$(a : \mathrm{Diag}(\mathbb{R}^{n\times n}), i,j : \mathbb{N}, e : \mathbb{R})$	
a[i,j] := e	$(a : \mathrm{Diag}(\mathbb{R}^{n\times n}), i,j : \mathbb{N}, e : \mathbb{R})$	//ha i=j

Diagonális mátrix



- Típus reprezentáció (típus értékek, műveletek)

$x:\mathbb{R}^n$

$\forall i \in [1..n]$: c.x[i] := a.x[i]+b.x[i]
Kvíz
ha i=j akkor e := a.x[i] különben e := 0.0
ha i=j akkor a.x[i] := e

Diagonális mátrix típus

$Diag(\mathbb{R}^{n\times n})$	c := a+b (a, b, c : $Diag(\mathbb{R}^{n\times n})$)
	c := a·b (a, b, c : $Diag(\mathbb{R}^{n\times n})$)
	e := a[i,j] (a : $Diag(\mathbb{R}^{n\times n})$, i,j : $\mathbb{N}$, e : $\mathbb{R}$)
	a[i,j] := e (a : $Diag(\mathbb{R}^{n\times n})$, i,j : $\mathbb{N}$, e : $\mathbb{R}$) //ha i=j
$x:\mathbb{R}^n$	$\forall i\in[1..n]$: c.x[i] := a.x[i]+b.x[i]
	$\forall i\in[1..n]$: c.x[i] := a.x[i]·b.x[i]
	ha i=j akkor e := a.x[i] különben e := 0.0
	ha i=j akkor a.x[i] := e

Nézzük meg az első beadandó minta dokumentációt!

Alsó háromszög mátrix

- Valósítsuk meg az alsó háromszög mátrix típust (a mátrixok a főátlójuk felett csak nullát tartalmaznak)! Ilyenkor elegendő csak a főátló és az alatti elemeket reprezentálni egy sorozatban. Implementáljuk a mátrix i-edik sorának j-edik elemét visszaadó műveletet, valamint két mátrix összegét és szorzatát!

7 x 7 -es alsó háromszög mátrix

	1	2	3	4	5	6	7
1	1						
2	2	3					
3	4	5	6				
4	7	8	9	10			
5	11	12	13	14	15		
6	16	17	18	19	20	21	
7	22	23	24	25	26	27	28

A sorszámok mutatják, hogy hányadik lesz a mátrix elem a tömbben.

A mátrix elhelyezése sorfolytonosan egy egydimenziós tömbben

(a biztosan nulla értékű elemeket felesleges tárolni)

1	[1,1]
2	[2,1]
3	[2,2]
4	[3,1]
5	[3,2]
6	[3,3]
27	[7,6]
28	[7,7]
29	0

a nulla értéket is tárolhatjuk egy példányban

$A[i,j] \neq 0$, ha $1 \leq j \leq i \leq n$

$X[n(n+1)/2]$

Alsó háromszög mátrix

- Index függvény

7 x 7 -es alsó háromszög mátrix

	1	2	3	4	5	6	7
1	1						
2	2	3					
3	4	5	6				
4	7	8	9	10			
5	11	12	13	14	15		
6	16	17	18	19	20	21	
7	22	23	24	25	26	27	28

A sorszámok mutatják, hogy hányadik lesz a mátrix elem a tömbben.

A mátrix elhelyezése sorfolytonosan egy egydimenziós tömbben
(a biztosan nulla értékű elemeket felesleges tárolni)

1	[1,1]
2	[2,1]
3	[2,2]
4	[3,1]
5	[3,2]
6	[3,3]
27	[7,6]
28	[7,7]
29	0

a nulla értéket is tárolhatjuk egy példányban

$$ind(i,j) = j + \sum_{k=1}^{i-1} k = j + \frac{i(i-1)}{2} \quad , ha\ 1 \le j \le i \le n$$

Próbák:
i=1, j=1 index(1,1)=1
i=4, j=2 index(4,2)=8
i=7, j=7 index(7,7)=28

Alsó háromszög mátrix

- Típus specifikáció

AHM($\mathbb{R}^{n\times n}$)	c := a+b	(a, b, c : AHM($\mathbb{R}^{n\times n}$)) // azonos n-re
	c := a*b	(a, b, c : AHM($\mathbb{R}^{n\times n}$)) // azonos n-re
	e := a[i,j]	(a : AHM($\mathbb{R}^{n\times n}$), i,j : $\mathbb{N}$, e : $\mathbb{R}$)
	a[i,j] := e	(a : AHM($\mathbb{R}^{n\times n}$), i,j : $\mathbb{N}$, e : $\mathbb{R}$) //ha i ≥ j

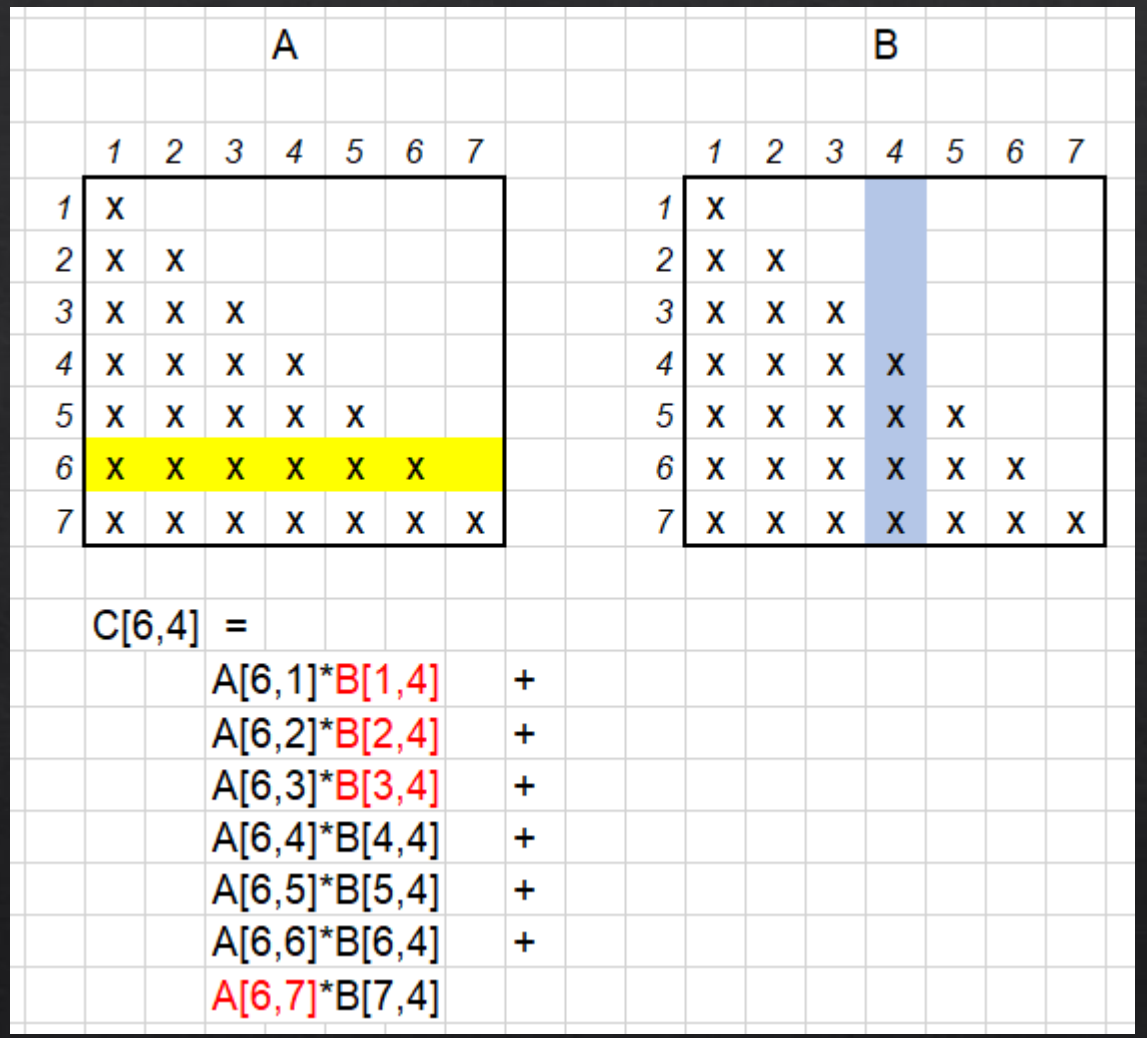
- Típus implementáció

x:$\mathbb{R}^{n(n+1)/2}$	$\forall i \in [1..n(n+1)/2]$: c.x[i] := a.x[i]+b.x[i]
	Mi lenne a leghatékonyabb?
	ha i≥j akkor e := a.x$\left[\frac{i(i-1)}{2}+j\right]$ különben e := 0.0
	ha i≥j akkor a.x$\left[\frac{i(i-1)}{2}+j\right]$:= e

Alsó háromszög mátrix

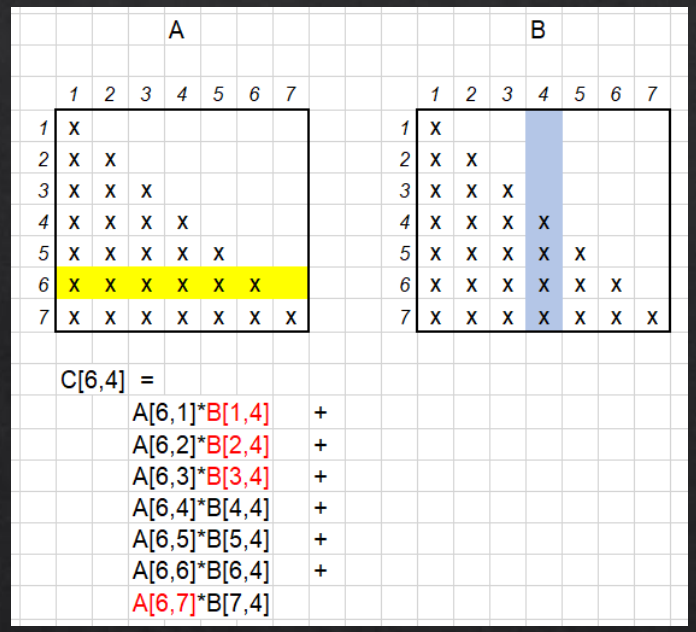
- Szorzás művelet

Kvíz



Alsó háromszög mátrix

- Szorzás művelet



$$\forall i,j \in [1..n]: \text{ha } i \geq j \text{ akkor}$$

$$c.x\left[\frac{i(i-1)}{2}+j\right] := \sum_{k=j}^{i} a.x\left[\frac{i(i-1)}{2}+k\right] \cdot b.x\left[\frac{k(k-1)}{2}+j\right]$$

Alsó háromszög mátrix típus

AHM($\mathbb{R}^{n\times n}$)	c := a+b (a, b, c : AHM($\mathbb{R}^{n\times n}$)) // azonos n-re
	c := a*b (a, b, c : AHM($\mathbb{R}^{n\times n}$)) // azonos n-re
	e := a[i,j] (a : AHM($\mathbb{R}^{n\times n}$), i,j : $\mathbb{N}$, e : $\mathbb{R}$)
	a[i,j] := e (a : AHM($\mathbb{R}^{n\times n}$), i,j : $\mathbb{N}$, e : $\mathbb{R}$) //ha i ≥ j
x:$\mathbb{R}^{n(n+1)/2}$	$\forall i \in [1..n(n+1)/2]$: c.x[i] := a.x[i]+b.x[i]
	$\forall i,j \in [1..n]$: ha i≥j akkor $c.x\left[\frac{i(i-1)}{2}+j\right] := \sum_{k=j}^{i} a.x\left[\frac{i(i-1)}{2}+k\right] \cdot b.x\left[\frac{k(k-1)}{2}+j\right]$
	ha i≥j akkor e := $a.x\left[\frac{i(i-1)}{2}+j\right]$ különben e := 0.0
	ha i≥j akkor $a.x\left[\frac{i(i-1)}{2}+j\right]$:= e

Prioritásos sor

- Készítsünk maximum prioritásos sort.
- Az elemek két mezőből állnak (prioritás (egész), adat (szöveg)).
- A sorból mindig a legnagyobb prioritású elemet vesszük ki (több legnagyobb esetén nem meghatározott, hogy melyiket).

Prioritásos sor

- Típus specifikáció

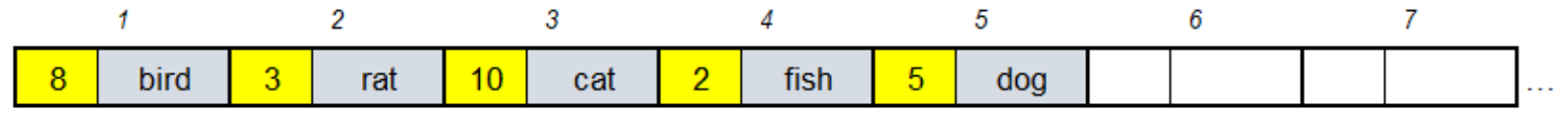
Típus értékek:	**Típus műveletek:**	
PrQueue a maximum prioritásos sorok halmaza, amely soroknak az elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok.	**setEmpty(pq)**	pq : PrQueue
	// Kiüríti a pr sort	
	l := isEmpty(pq)	pq : PrQueue, l : $\mathbb{L}$
	//Igazat ad, ha üres a pr sor, hamisat ha nem.	
	pq := add(pq, e)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	//Betesz egy új elemet a pr sorba.	
	pq, e := remMax(pq)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	//Kiveszi az egyik legnagyobb prioritású elemet. Fontos, hogy a sor nem lehet üres.	
	e := getMax(pq)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	//Visszadja az egyik legnagyobb prioritású elemet, nem veszi ki. Fontos, hogy a sor nem lehet üres.	

Műveletek alternatív megfogalmazása:

pq.setEmpty()	
l := pq.empty()	{query}
pq.add(e)	
e := pq.remMax()	
e:= pq.getMax()	{query}

Prioritásos sor

- Típus reprezentáció (vector típusra építve)
- **Rendezetlen sorozatot használunk. Az elemeket folyamatosan helyezzük el a sorozatban. Nem tudjuk, melyik közülük a legnagyobb prioritású.**



a. **setEmpty** : üres sorozatot készít. Θ(1) (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)

b. **isEmpty:** hosszból azonnal eldönthető. Θ(1)

c. **add:** az új elemet a sorozat végéhez fűzzük. Θ(1)

d. **remMax:** ha nem üres a sorozat, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és kivesszük. Θ(n)

e. **getMax:** ha nem üres s sor, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és visszaadjuk. A sorozat nem változik. Θ(n)

Θ(1)-re javítható, ha a legnagyobb prioritású elem indexét tároljuk.

Prioritásos sor

- Típus reprezentáció (vector típusra építve)
- **Rendezett sorozatot használunk. A sorozatban az elemek prioritás szerint növekvő a sorrendben vannak.**

a. **setEmpty** : üres sorozatot készít. Θ(1) (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)

b. **isEmpty:** hosszból azonnal eldönthető Θ(1)

c. **add:** az új elemet betesszük a rendezettség szerinti helyére, amelyet lineáris kereséssel vagy kiválasztással kell megkeresnünk O(n)

d. **remMax:** ha nem üres a sorozat, akkor a rendezettség miatt a sorozat utolsó eleme az egyik legnagyobb prioritású elem. Azt ki is kell vennünk a sorozatból. Θ(1)

e. **getMax:** ha nem üres a sorozat, akkor a rendezettség miatt a sorozat utolsó eleme az egyik legnagyobb prioritású elem. A sorozat nem változik. Θ(1)

Prioritásos sor

- Látjuk, hogy a prioritásos sor ábrázolásánál nem tudjuk a konstans műveletigényt tartani mindegyik művelet esetén.
- Mindkét módszernél legalább az egyik művelet „lineáris” műveletigényű, azaz az elemek számával arányos.
- Van egy pici különbség a két lineáris igényű művelet között: míg a maximum kiválasztás minden esetben megvizsgálja az összes elemet, addig a rendezett részbe való beszúráshoz egy keresés kell, amely korábban is leállhat.
- Ha egy előre megadott méretű tömböt használnánk, amelynél külön megadjuk, hogy az első hány eleme van kitöltve, akkor a beszúrás néhány elem hátra csúsztatását igényli, amely legrosszabb esetben az összes elemet odébb csúsztatja.
- A remMax() tehát rendezett esetben O(n). Igaz, hogy átlagosan az elemek felét érinti majd a hátrébb csúsztatás, tehát itt várhatóan fele annyi lesz az átlagos lépésszám, viszont az elemek hátrébb csúsztatása jóval költségesebb, mintha csak kiolvassuk őket, amelyet a remMax() rendezetlen esetben csinál.
- Ez az érv a rendezetlen ábrázolás mellett szól.

Prioritásos sor

- Ha C++ nyelven a vectort használjuk, akkor az erase(), illetve az insert() műveletekkel tetszés szerint változtathatjuk a sorozatot.
- Azt, hogy melyik a jobb, annak függvényében lehet eldönteni, hogy melyik művelet lesz gyakoribb, és azt megvalósítani hatékonyan.
- (Megjegyzés: Tovább növelhető a hatékonyság kupac adatszerkezettel, erről majd az Algoritmusok és adatszerkezetek tárgyban lesz szó.)

Rendezetlen sorozattal való megvalósítást választjuk
(egy másik példában fogunk rendezett sorozatot használni)

Prioritásos sor reprezentáció

- Típus értékek ábrázolása:

Típus reprezentáció:

vec: Item*

- a prioritásos sor elemeit rendezetlenül tároló sorozat, ahol Item = rec(pr : $\mathbb{Z}$, data : $\mathbb{S}$)

- Típus műveletek:
  - Pr. sor üresre állítása:

setEmpty(pq))

vec := <> (C++ vector: clear())

  - Pr. sor üres-e [kvíz]

l := isEmpty(pq))

l := |vec| =0 (C++ vector: size())

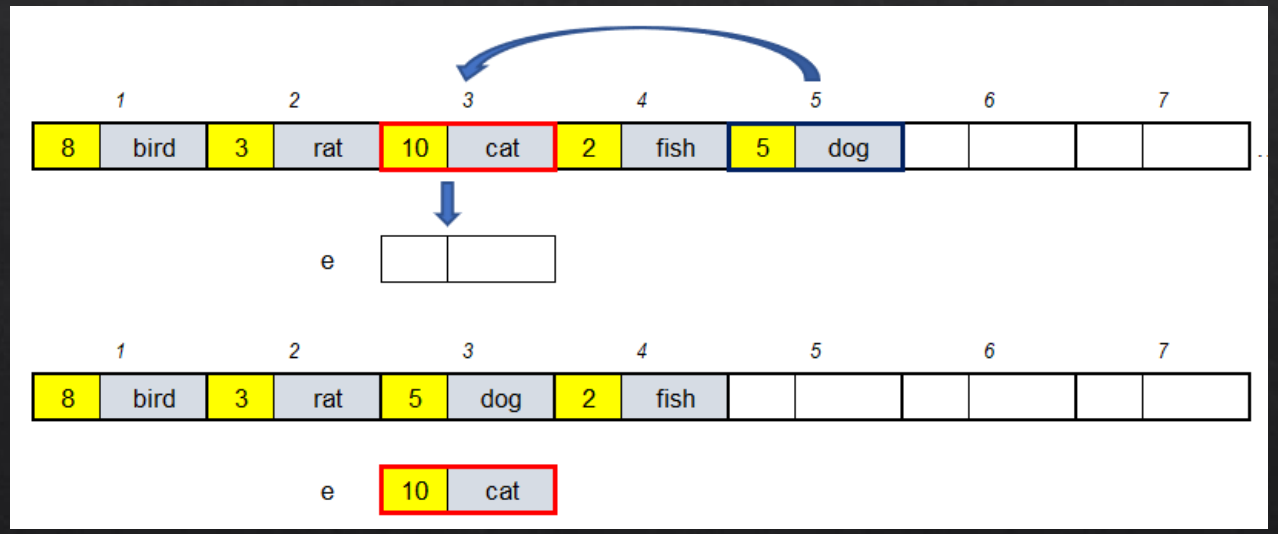
Prioritásos sor reprezentáció

- Típus műveletek:
  - Pr. sorba betevés:

pq := add(pq, e)

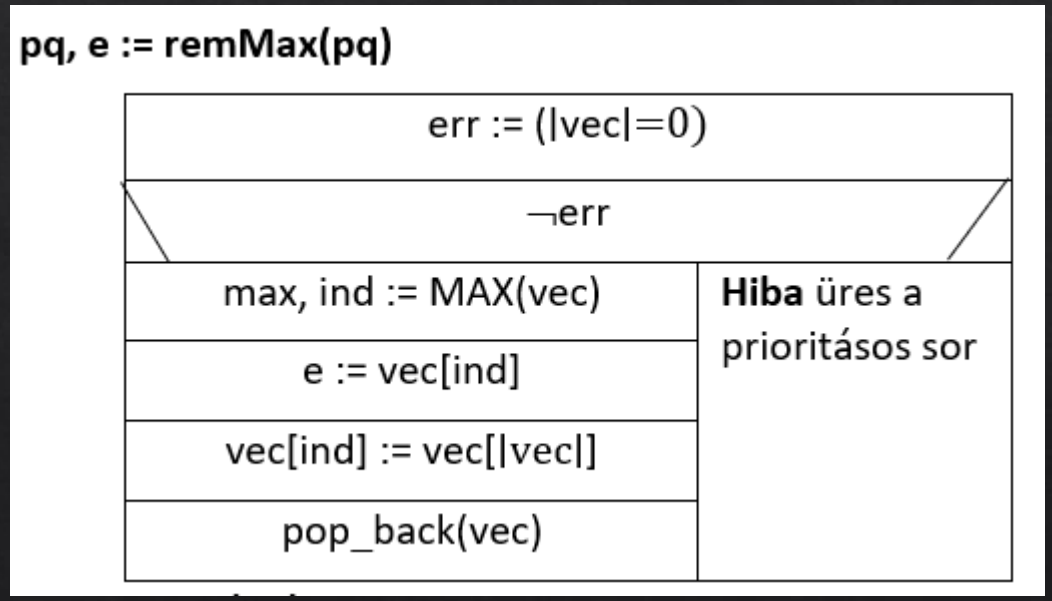
vec := vec ⊕ <e> (C++ vector: push_back())

  - Pr. sorból legnagyobb prioritású elem kiszedése [kvíz] Terv:
    - Maximum kiválasztással meghatározzuk a legnagyobb prioritású elemet, kivesszük egy változóba,
    - A helyén keletkező „lyukba” a vector utolsó elemét bemásoljuk,
    - A vector utolsó elemét eldobjuk.
    - Hibát kell jelezni, ha üres a prioritásos sor!



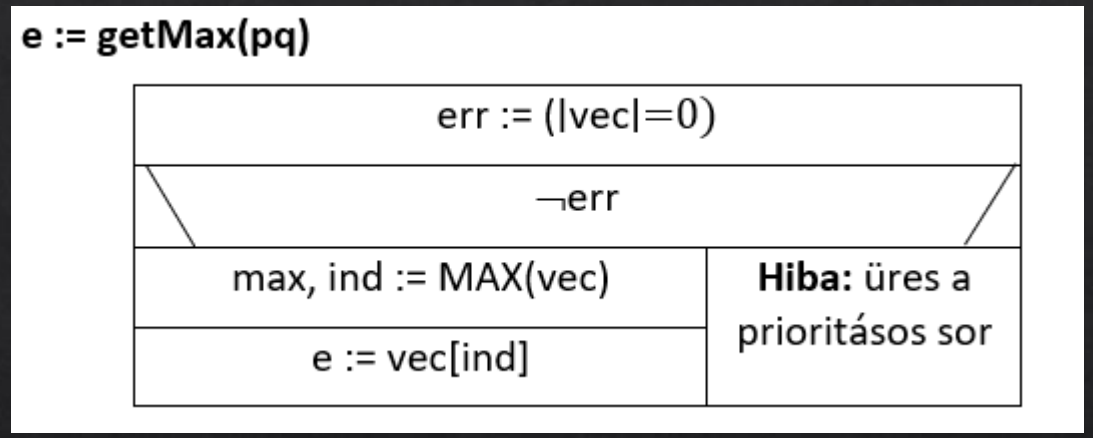
Prioritásos sor reprezentáció

- Típus műveletek:
  - Pr. sorból legnagyobb prioritású elem kiszedése:



Prioritásos sor reprezentáció

- Típus műveletek:
  - Pr. sorból legnagyobb prioritású elem lekérdezése:



Prioritásos sor reprezentáció

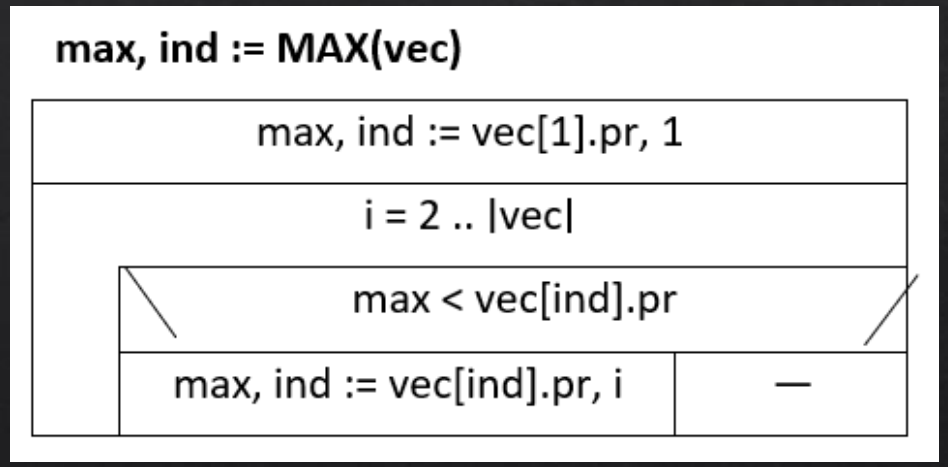
- Maximum kiválasztó segéd függvény
  - Több művelet is használja a max, ind := MAX(vec) segédműveletet (privát metódust).
  - Készítsük el a specifikációját: [kvíz]

A = (vec:(Item)*, e:Item) Item=rec(pr:$\mathbb{Z}$, data:$\mathbb{S}$)

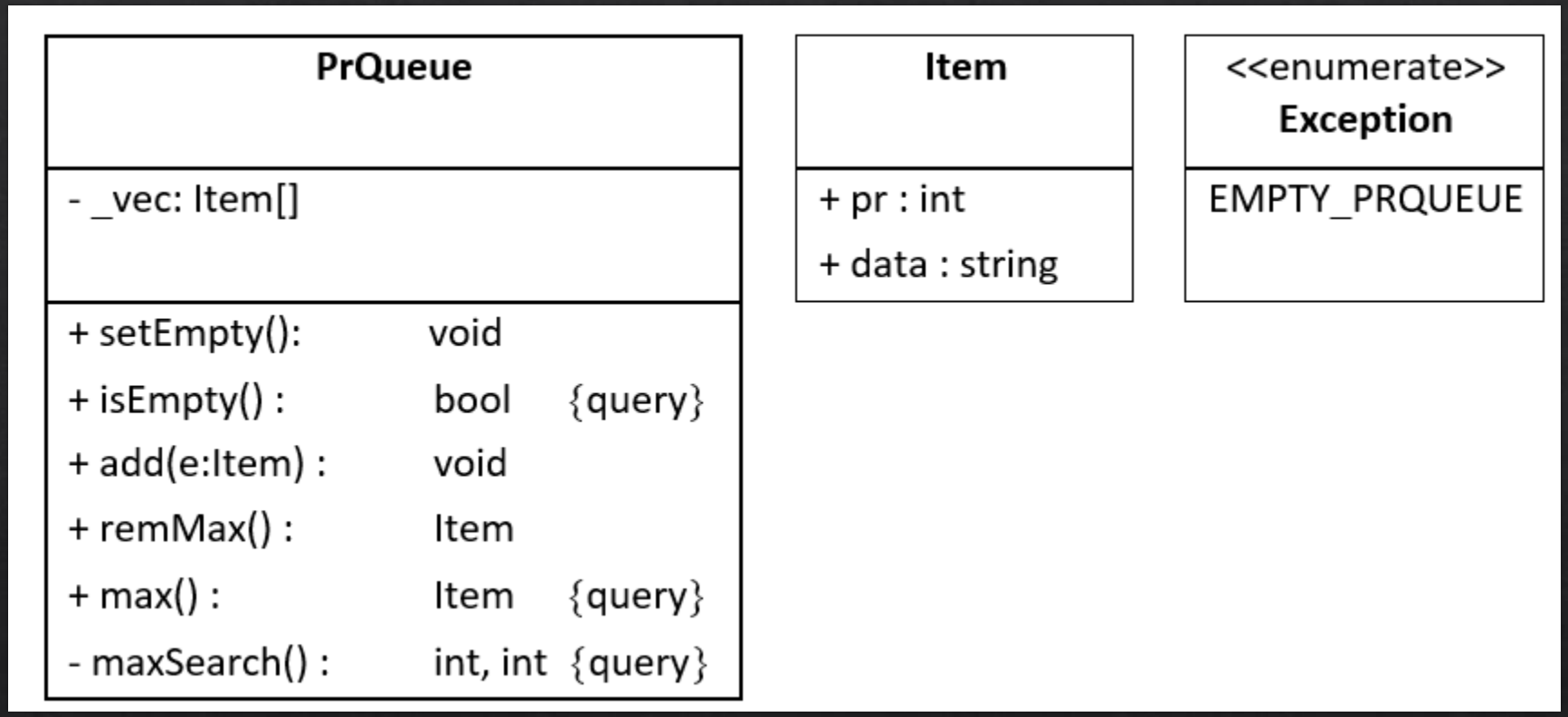
Ef = (vec=vec' $\wedge$ |vec|>0)

Uf = (Ef $\wedge$ (max, ind) = $MAX_{i=1\ ..\ |vec|}$ vec[i].pr $\wedge$
$\wedge$ e = vec[ind])

ahol max:$\mathbb{Z}$ és ind:$\mathbb{N}$



Prioritásos sor UML ábra



Prioritásos sor használata egy feladatban

- Egy programozási versenyen csapatok indultak. Ismerjük a csapatok nevét, és a versenyen elért pontszámukat. Készítsünk listát a csapatok eredményéről csökkenő sorrendben. (Feltehető, hogy a csapatok neve egyedi.)
- A = (t : Item^n, cout : (Item)*) Item=rec(pr:$\mathbb{Z}$, data:$\mathbb{S}$)
- Ef = (t = t_0)
- Uf = (t = $t_0 \wedge$ cout = (t elemeit tartalmazza, pr értékei szerint monoton csökkenően felsorolva))
- Prioritásos sorral történő megoldás:
- Minden elemet berakunk egy prioritásos sorba
  pq : PrQueue $\wedge$ pq = $\bigcup_{i=1..n} \{t[i]\}$
- **Az elemeket kivesszük a prioritásos sorból és kiírjuk.**
  cout = $\bigoplus^{\neg isEmpty(pq)}$ remMax(pq)

- Tétel: két összegzés
- pq : PrQueue $\wedge$ pq = $\bigcup_{i=1..n} \{t[i]\}$
  Az unió művelet a prioritásos sorba tesz be egy új elemet (azaz szemantikailag megegyezik az add() művelettel) úgy, mintha a prioritásos sorhoz unióznánk az új elemet tartalmazó prioritásos sort. Ennek a műveletnek a neutrális eleme az üres prioritásos sor.
- cout = $\bigoplus^{\neg isEmpty(pq)}$ remMax(pq)
  $\bigoplus$ - összefűző művelet (kimenetre történő kiírás szemléltetése), neutrális elem: <>
  Az összefűzés addig tart, amíg a pq prioritásos sor nem lesz üres.