4. Gyűjtemények II.

1. Asszociatív tömb
   Valósítsuk meg az asszociatív tömb típust típusát, amely olyan kulcs-adat párokat tároló gyűjteményt jellemez, amelyben egyedi kulcs alapján lehet visszakeresni az értékeket. Kulcsnak válasszuk most az egész számokat, adatként pedig sztringeket tároljunk.

   Típusspecifikáció: **Map**

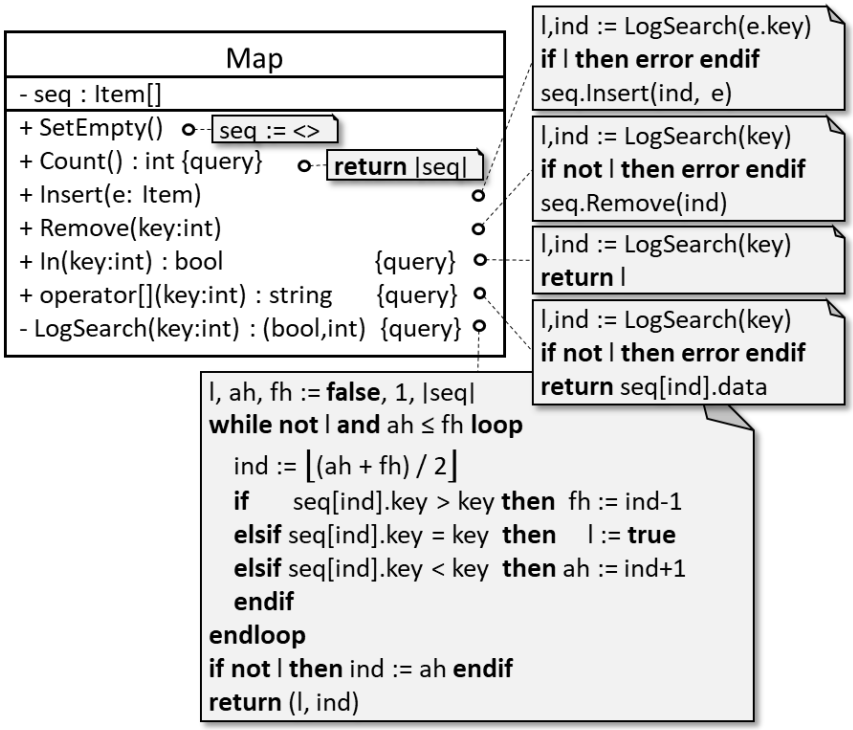
<table>
<tr>
<td>asszociatív tömbök, azaz speciális tárolók halmaza,
ahol az elemek $\mathbb{Z} \times \mathbb{S}$ (kulcs-adat) típusú párok</td>
<td>map := SetEmpty(map)
c := Count(map)
map := Insert(map,e) // ha e kulcs-része egyedi
map := Remove(map, key) // ha key létezik
l := In(map, key)
data := Select(map, key) // ha key létezik

map : Map, c : $\mathbb{N}$, e : $\mathbb{Z} \times \mathbb{S}$, data : $\mathbb{S}$ key : $\mathbb{Z}$, l : $\mathbb{L}$, data : $\mathbb{S}$</td>
</tr>
<tr>
<td>seq: Item*
Item = rec(key:$\mathbb{Z}$, data:$\mathbb{S}$)

Invariáns:
a sorozat kulcsuk szerint rendezetten tárolója az az Item típusú elemeket</td>
<td>map:=SetEmpty(map)
seq := <>
c := Count(map)
c := |seq|
map := Insert(map,e) l : $\mathbb{L}$, ind : $\mathbb{N}$
l, ind := logSearch(seq, e.key)
¬l | seq.Insert(ind, e) | Hiba: már létező kulcs
map := Remove(map, key) l : $\mathbb{L}$, ind : $\mathbb{N}$
l, ind := logSearch(seq, key)
l | seq.Remove(ind) | Hiba: nem létező kulcs
l := In(map, key)
l, ind := logSearch(seq, key) ind : $\mathbb{N}$
data := Select(map, key)
l, ind := logSearch(seq, key)
l | data := seq[ind].data | Hiba: nem létező kulcs</td>
</tr>
</table>

Több művelet egy kulcs szerinti kereséssel indul, amelyhez a logaritmikus keresés algoritmusát használjuk. (Ez, ha nincs a sorozatban keresett kulcsú elem, akkor az első olyan elem indexét adja vissza, amelynek kulcsa nagyobb a keresett kulcsnál, vagy ha nem lenne ilyen, akkor a sorozat hossza plusz egyet.) Ez egy privát metódusa lesz a típusunknak.

Osztály diagram:



2. Prioritásos sor

Készítsünk maximum prioritásos sor típust. Ennek elemei két mezőből állnak（prioritás (egész szám), adat (szöveg)）. A sorból mindig a legnagyobb prioritású elemet vesszük ki (több legnagyobb esetén nem meghatározott, hogy melyiket).

Típusspecifikáció: **PrQueue**

a maximum prioritásos sorok, azaz olyan gyűjtemények, amelyek elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok.	**pq:=SetEmpty(pq)**	pq : PrQueue
	l := isEmpty(pq)	l : $\mathbb{L}$
	pq := Add(pq, e)	e : $\mathbb{Z} \times \mathbb{S}$
	e := GetMax(pq)	// *ha pq nem üres*
	pq, e := RemMax(pq)	// *ha pq nem üres*

A reprezentációhoz rendezetlen vagy rendezett sorozatot használhatunk, és a műveletek futási ideje ettől függően alakulhat:

n hosszú sorozat	**rendezetlen**	**rendezett**
SetEmpty()	$\Theta(1)$	$\Theta(1)$
isEmpty()	$\Theta(1)$	$\Theta(1)$
Add()	$\Theta(1)$	$\Theta(\log_2 n)$
GetMax()	$\Theta(n)$	$\Theta(1)$
RemMax()	$\Theta(n)$	$\Theta(1)$

Típusmegvalósítás:

seq: Element*

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

pq := SetEmpty(pq))

seq := <>

l := isEmpty(pq))

l := |seq| =0

pq := Add(pq, e)

seq := seq $\oplus$ <e>

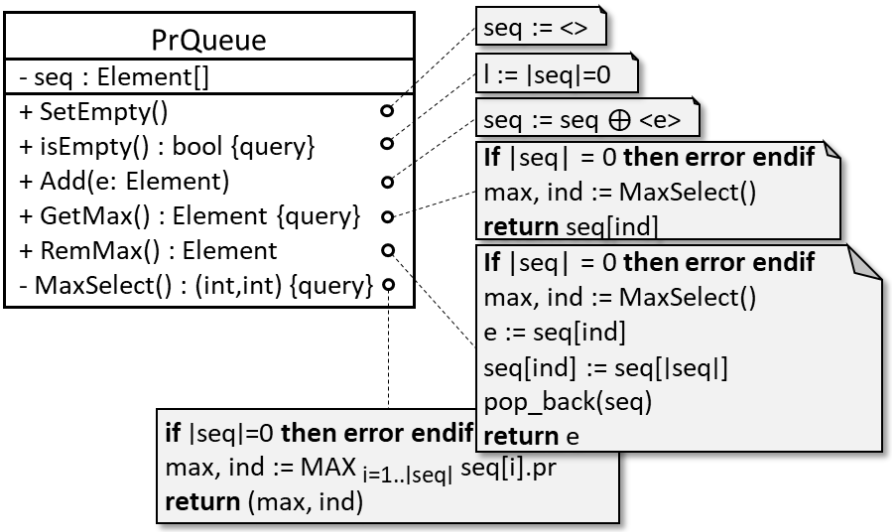
e := GetMax(pq)

|seq|>0	
max, ind := $MAX_{i=1..[seq]}$(seq[i].pr)	**Hiba:** üres prioritásos sor
e := seq[ind]	

pq, e := RemMax(pq)

|seq|>0	
max, ind := $MAX_{i=1..[seq]}$(seq[i].pr)	**Hiba** üres prioritásos sor
e := seq[ind]	
seq[ind] := seq[|seq|]	
pop_back(seq)	

Osztály diagram:



Egy feladat megoldása prioritásos sorral:

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 = (x : Item*, cout : Item*)

Ef = ($x = x_0$)

Uf = ($x = x_0 \wedge$ cout = (t elemeit tartalmazza monoton csökkenően felsorolva))

=($x = x_0 \wedge$ pq : PrQueue $\wedge$ pq = $\bigcup_{e \text{ in } x} \{e\} \wedge$ cout = $\bigoplus_{e \text{ in } pq}$ <e>)

Összegzés („uniózás”)

enor(t) ~ i=1 ..|t|, x[i]

s ~ pq

H, +, 0 ~ PrQueue, ∪, ∅

ahol

∅ ~ pr.SetEmpty()

pr := pr ∪ {e} ~ pr.Add(e)

Összegzés (összefűzés)

enor(t) ~ first: -
next: pq.RemMax()
current:pq.GetMax()
end: pq.isEmpty()

f(i) ~ <e>

s ~ cout

H, +, 0 ~ $\mathbb{S}$*, ⊕, <>

cout := cout ⊕ <e> ~ cout.Write(e)

pq.SetEmpty()	
i = 1 .. n	
	pq.Add(t[i])
cout := <>	
¬pq.isEmpty()	
	cout := cout ⊕ pq.RemMax()

cout := cout ⊕ pq.GetMax(); pq.RemMax() helyett

Prioritásos sor megvalósításának tesztelése:

Mivel a gyakorlat egyik témaköre a tesztelés megtervezése, így, tekintsük át, mit, hogyan kellene tesztelni. A tényleges tesztelés megvalósítása a géptermi gyakorlaton történik.

Vegyük sorra, milyen tesztelést képzelnének el az egyes metódusokhoz:

- SetEmpty() (végrehajtása után az isEmpty() igazat ad)
- isEmpty() (üres / nem üres állapotra kipróbáljuk)
- Add(Item e) (egymás után berakunk elemeket, majd ellenőrizzük az elhelyezésüket)
- MaxSelect() (maximum kiválasztás szürkedoboz tesztesetei)
- GetMax() (a MaxSelect() tesztesetei mellett hibaesetet is tesztelni kell)
- RemMax() (a GetMax()-hoz képest még a tömb átrendeződését is ellenőrizzük)

RemMax() tesztelése				
teszteset	**prioritásos sor** (sor elemei a prioritás szerint csökkenő sorrendben)		**eredmény**	**új prioritásos sor** (sor elemei a prioritás szerint csökkenő sorrendben)
üres sor	<>		hiba (kivétel dobás)	<>
egy elemű	<3>		3	<>
több elemű esetek:				
első a legnagyobb	<5,2,3>		5	<3,2>
utolsó a legnagyobb	<1,2,3>		3	<1,2>
belső a legnagyobb	<1,3,2>		3	<1,2>
nem egyértelmű, első és utolsó a legnagyobb	<5,2,5'>		5	<5',2> (az adat rész segítségével ellenőrizhető, hogy az elsőt vettük ki)
nem egyértelmű, belső és utolsó a legnagyobb	<1,3,3'>		3	<1,3'>
mind egyforma	<3,3',3''>		3	<3'',3'>
több egymás utáni RemMax(), majd Add() együttes hatása	<2,1> <1> <> <3> <2,3> <3,2,1>	RemMax() RemMax() Add(3) Add(2) Add(1) RemMax()	2 1 3 2 1 3	<1> <> <3> <2,3> <2,3,1> <2,1>