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.

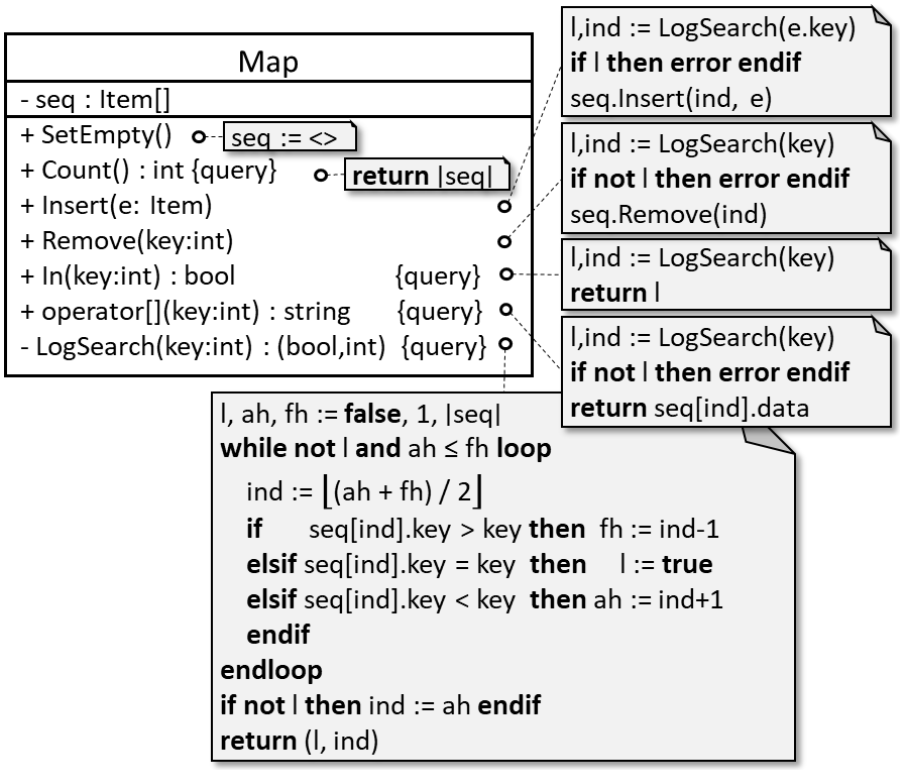
Map

<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 := Get(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)
map.seq := <>
c := Count(map)
c := |map.seq|
map := Insert(map,e) l : $\mathbb{L}$, ind : $\mathbb{N}$
l, ind := logSearch(map.seq, e.key)
¬l
map.seq.Insert(ind, e) | Hiba: már létező kulcs
map := Remove(map, key) l : $\mathbb{L}$, ind : $\mathbb{N}$
l, ind := logSearch(map.seq, key)
l
map.seq.Remove(ind) | Hiba: nem létező kulcs
l := In(map, key)
l, ind := logSearch(map.seq, key) ind : $\mathbb{N}$
data := Get(map, key)
l, ind := logSearch(map.seq, key)
l
data := map.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.



Asszociatív tömb megvalósításának tesztelése:

Hogyan kellene tesztelni például az Insert() metódust? Segítene, ha letudnánk kérdezni a reprezentáló sorozatot (C#-ban ezt elvégezhetné a ToString() metódus). Ezután minden Insert() hívás után ellenőrizhetnénk a reprezentáló sorozatot. Az Insert() hívásokat úgy kellene végrehajtani, hogy sor kerüljön a már meglevő sorozat elejére, végére, közepére történő beszúrásra, valamint már meglévő kulccsal történő beszúrásra.

Vegyük most sorra, milyen tesztelést képzelnének el az egyes metódusokhoz:

- SetEmpty() (végrehajtása után a ToString() üres sorozatot ad)
- Count() (üres / nem üres állapotra kipróbáljuk)
- Insert(Item e) (beszúrás a reprezentáló sorozat elejére, közepére, végére)
- Remove(int key) (törlés a reprezentáló sorozat elejéről, közepéről, végéről)
- In(int key) (amikor benne van az elején, közepén, végén, amikor nincs benne)
- Select(int key) (a sorozat elején, közepén végén létező, vagy nem létező kulcs)
- LogSearch(int key) (a sorozat elején, közepén végén létező, vagy nem létező kulcs)

Teszteljük a hibás eseteket is.

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 := RemMax(pq)	// *ha pq nem üres*

A reprezentáció megválasztása előtt is kitalálhatunk már az egyes műveletekhez teszteseteket:

- SetEmpty() : végrehajtása után az isEmpty() igazat ad.
- isEmpty() : kezdeti (üres) / Add() utáni (nem üres) állapotra is kipróbáljuk
- Add(Item e) : egymás után berakunk elemeket egyre növekedő priorítású elemeket, majd ezeket fordított sorrendben kapjuk vissza ugyanennyi RemMax()-szal
- MaxSelect() : maximum kiválasztás szürkedoboz tesztesetei (lásd előadás)
- GetMax() : a maximum kiválasztás tesztesetei mellett hibaesetet is tesztelni kell
- RemMax() : a maximum kiválasztás tesztesetei mellett hibaesetet is tesztelni kell) n darab Add() után n darab RemMax() kiüríti a tárolót

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

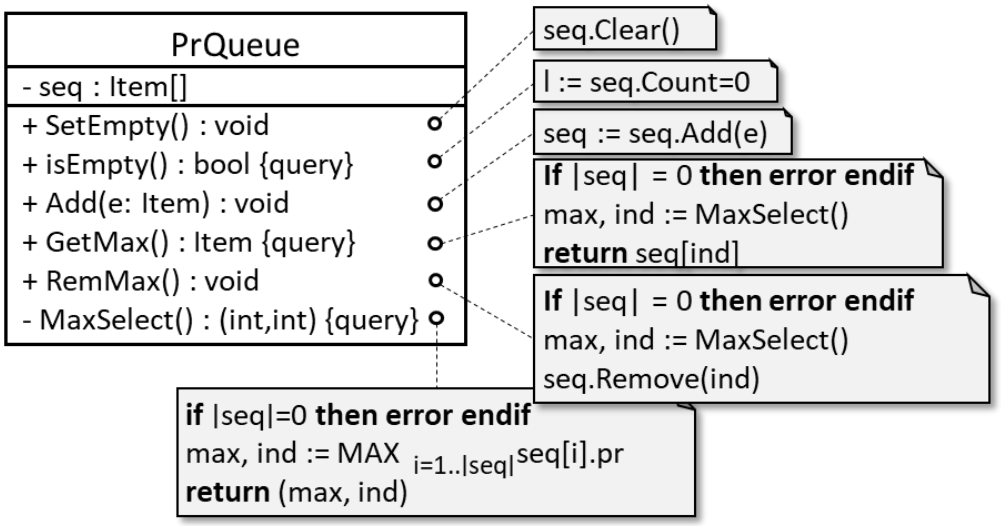
n hosszú sorozat	**rendezetlen**	**rendezett**
SetEmpty()	$\Theta(1)$	$\Theta(1)$
isEmpty()	$\Theta(1)$	$\Theta(1)$
Add()	$\Theta(1)$	$\Theta(\log_2 n)$
GetMax()	$\Theta(n)$	$\Theta(1)$
RemMax()	$\Theta(n)$	$\Theta(1)$

Habár a rendezett sorozattal történő reprezentáció összességében hatékonyabbnak tűnik, az alábbiakban a rendezetlen sorozattal való reprezentációt mutatjuk be.

Típusmegvalósítás:

seq: Item* – a prioritásos sor elemeit <u>rendezetlenül</u> tároló sorozat, ahol Item = rec(pr : ℤ, data : 𝕊)	**pq := SetEmpty(pq)**) pq.seq := <> vagy pq.seq.Clear() **l := isEmpty(pq)**) l := |pq.seq| =0 vagy l := pq.seq.Count() **pq := Add(pq, e)** pq.seq := pq.seq ⊕ <e> vagy pq.seq.Add(e) **e := GetMax(pq)** |pq.seq|>0 max, ind := $MAX_{i=1..[pq.seq|}$(pq.seq[i].pr) e := pq.seq[ind] **Hiba:** üres prioritásos sor **pq := RemMax(pq)** | pq.seq|>0 max, ind := $MAX_{i=1..[pq.seq|}$(pq.seq[i].pr) pq.seq:= pq.seq[1..ind-1] ⊕ pqseq[ind+1..|pq.seq|] **Hiba** üres prioritásos sor vagy pq.seq.Remove(ind)

Osztály diagram:



A reprezentáció ismeretében további teszteseteket lehet a műveletekhez felállítani. Ezek ellenőrzéséhez azonban szükség lesz a reprezentációt adó sorozat kiolvasására. (Ez lehet például egy olyan ToString() metódus, amely <(adat, prioritás),(adat,prioritás), … > formájú sztringben adja meg a tároló tartalmát. Például:

RemMax() tesztelése		
teszteset	**prioritásos sor**	**új prioritásos sor**
üres sor	<>	<> hiba (kivétel dobás)
egy elemű	<(alma,3)>	<>
több elemű esetek:		
első a legnagyobb	<(alma,5),(körte,2),(barack,3)>	<(körte,2),(barack,3)>
utolsó a legnagyobb	<(barack,3),(körte,2),(alma,5)>	<(körte,2),(barack,3)>
belső a legnagyobb	<(barack,3),(alma,5),(körte,2)>	<(körte,2),(barack,3)>
nem egyértelmű, első és utolsó a legnagyobb	<(alma,5),(körte,2),(eper,5)>	<(körte,2),(eper,5)> vagy <(alma,5),(körte,2)>
nem egyértelmű, belső és utolsó a legnagyobb	<(alma,5) ,(eper,5),(körte,2)>	<(eper,5), (körte,2)> vagy <(alma,5),(körte,2)>
mind egyforma	<(alma,5) ,(eper,5),(körte,5)>	<(eper,5), (körte,5)> vagy <(alma,5),(körte,5)> vagy <(alma,5) ,(eper,5)>

Fontos még több művelet kölcsönhatását is vizsgálni: például több egymás utáni RemMax(), majd Add() együttes hatását.

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.)

Specifikáció

A = (x : Item*, y : 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 = $\bigcup_{e \text{ in } x} \{e\} \wedge y = \bigoplus_{e \text{ in } pq} <e>$)

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	~ $\mathbb{S}$*, $\bigoplus$, <>

pq.SetEmpty()
i = 1 .. n
pq.Add(t[i])
y := <>
¬pq.isEmpty()
y := y $\bigoplus$ pq.GetMax()
pq.RemMax()