Sorozattal reprezentált típusok megvalósítása II. rész

1. Feladat – 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 értékek:	**Típus műveletek:**
Map azon asszociatív tömbök halmaza, amely tömböknek az elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok	**map:=setEmpty(map)** map : Map *//kiüríti az asszociatív tömböt* **c := count(map)** map : Map, c : $\mathbb{N}$ *//megadja az elemek számát* **map := insert(map,e)** map : Map, e : $\mathbb{Z} \times \mathbb{S}$ *//új elemet tesz be, ha a kulcsa még nem létezik* **map := erase(map, key)** map : Map, key : $\mathbb{Z}$ *// törli az adott kulcsú elemet, ha a kulcs létezik, különben hiba* **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*

A kulcs-adat párok tárolásához egy **rendezett sorozatot** fogunk használni. 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.

(Megjegyzés: található hatékonyabb tárolás (bináris keresőfák, hash tábla), lásd majd Algoritmusok tárgyban.)

Típus reprezentáció:

seq: Item* – az elemeket kulcsuk szerint rendezetten tároló sorozat,

ahol Item = rec(key: $\mathbb{Z}$, data: $\mathbb{S}$)

Típusműveletek implementációja:

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 := seq[1 .. ind-1] ⊕ e ⊕ seq[ind .. |seq|])	—

map := erase(map, key) l : $\mathbb{L}$, ind : $\mathbb{N}$

l, ind := logSearch(seq, key)	
l	
seq := seq[1 .. ind-1] ⊕ seq[ind+1 .. |seq|])	**Hiba**: nem létező kulcs

l := in(map, key)

l, ind := logSearch(seq, key)	ind : $\mathbb{N}$

data := operator[](map, key)

l, ind := logSearch(seq, key)	
l	
data := seq[ind].data	**Hiba**: nem létező kulcs

Több művelet egy kulcs szerinti kereséssel indul, amelyhez a logaritmikus keresés algoritmusát használjuk. Ez egy fontos privát algoritmusa lesz a típusunknak.

Egy lépéssel még kiegészítjük az algoritmust: ha nincs benne a keresett kulcs a tömbben, akkor az első nála nagyobb elem sorszámát adja meg. Ezzel ügyesebb lesz az insert művelet: nem kulcs-összehasonlításra támaszkodik az eltolás ciklusa, hiszen tudjuk, hogy mettől meddig kell a tömb elemeit jobbra tolni. Ha nincs a keresett kulcsnál nagyobb a tömbben, akkor az utolsó elem utáni sorszámot kapjuk vissza.

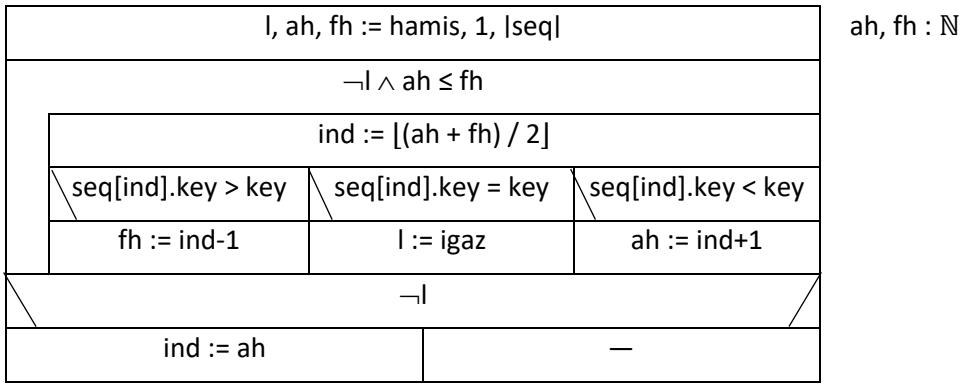
Kulcs szerinti keresés

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)

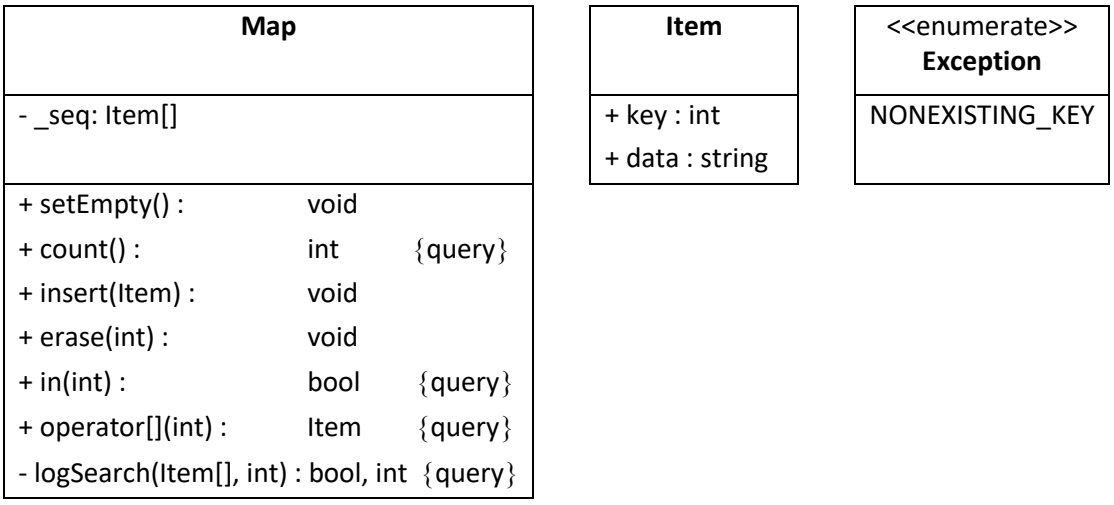
Uf=(Ef $\wedge$ l = $\exists i \in [1 .. |seq|]$: seq[i].key = key $\wedge$
(l $\rightarrow$ ind$\in$[1 .. |seq|] $\wedge$ seq[ind].key = key) $\wedge$
($\neg$l $\rightarrow$ $\forall i \in$ [1..ind-1]: seq[i].key < key $\wedge$ $\forall i \in$[ind.. |seq|] : seq[i].key > key))

l, ind := logSearch (seq, key)



ah, fh : $\mathbb{N}$

Asszociatív tömb megvalósítása C++ osztállyal: UML diagram:



2. Feladat – zsák típus

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

<table>
<tr>
<td>Bag

azon zsákok halmaza, amelyek elemei természetes számok (ℤ)</td>
<td>b := ∅ b : Bag // kiüríti a zsákot
b := putIn(b, e) b : Bag, e : ℤ // elemet tesz be
m := mostFrequent(b) b : Bag, m : ℤ
// leggyakoribb elem
b:=takeOut(b, e) b : Bag, e : ℤ // elemet vesz ki</td>
</tr>
<tr>
<td>vec : Item* – az elemeket tartalmuk (data) szerint rendezetten tároló sorozat,

ahol Item = rec(data: ℤ, count: ℕ)

maxind : ℕ – a vec sorozat legnagyobb count értékű elemének indexe</td>
<td>
b := ∅ b : Bag
vec := <>

b := putIn(b,e) b : Bag, e : ℤ
l, ind := logSearch(vec, e) // data szerint
l
vec[ind].count := vec[ind].count+1 | vec := vec[1..ind-1] ⊕ (e,1) ⊕ vec[ind..|vec|]
vec[ind].count > vec[maxind].count | maxind>=ind
maxind := ind | – | maxind:=maxind+1 | –

m := mostFrequent(b) b : Bag, m : ℤ
|vec| > 0
m := vec[maxind].data | hiba

b:=takeOut(b,e) b : Bag, e : ℤ
l, ind := logSearch(vec, e) // data szerint
l
vec[ind].count > 1 | vec[ind].count = 1
vec[ind].count := vec[ind].count–1 | vec := vec[1..ind-1] ⊕ vec[ind+1..|vec|]
|vec| > 0 | –
max, maxind := MAX_{i=1..|vec|} (vec[i].count) | skip
</td>
</tr>
</table>

Megjegyzés: Kiegészíthető egyéb műveletekkel (elemek száma, üres-e)