3. Sorozattal reprezentált típusok

1. 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 halmaza, azaz olyan gyűjtemények, amelyek elemei $\mathbb{Z} \times \mathbb{S}$ típusú párok.	**pq:=SetEmpty(pq)**	pq : PrQueue
	l := isEmpty(pq)	pq : PrQueue, l : $\mathbb{L}$
	pq := Add(pq, e)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	e := GetMax(pq)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	// pq sor nem lehet üres	
	pq, e := RemMax(pq)	pq : PrQueue, e : $\mathbb{Z} \times \mathbb{S}$
	// pq sor nem lehet üres	

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

(1) **Rendezetlen** (n hosszú) **sorozat**:
   a. **SetEmpty** : üres sorozatot készít. $\Theta(1)$ (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)
   b. **isEmpty:** hosszból azonnal eldönthető. $\Theta(1)$
   c. **Add:** az új elemet a sorozat végéhez fűzzük. $\Theta(1)$
   d. **GetMax:** ha nem üres s sor, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és visszaadjuk. A sorozat nem változik. $\Theta(n)$
   e. **RemMax:** ha nem üres a sorozat, a tanult maximum kiválasztás algoritmussal megkeressük az egyik legnagyobb prioritású elemet, és kivesszük. $\Theta(n)$

(2) **Rendezett** (n hosszú) **sorozat.** Elemek prioritás szerint növekvő a sorrendben vannak.
   a. **SetEmpty** : üres sorozatot készít. $\Theta(1)$ (Habár nem tudjuk, hogy a vector clear() metódusa mit is csinál pontosan.)
   b. **isEmpty:** hosszból azonnal eldönthető $\Theta(1)$
   c. **Add:** az új elemet betesszük a rendezettség szerinti helyére, amelyet lineáris kereséssel vagy kiválasztással kell megkeresnünk $\mathrm{O}(n)$
   d. **GetMax:** ha nem üres a sorozat, akkor a rendezettség miatt a sorozat utolsó eleme az egyik legnagyobb prioritású elem. A sorozat nem változik. $\Theta(1)$
   e. **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. $\Theta(1)$

n hosszú sorozat	**rendezetlen**	**rendezett**
SetEmpty()	$\Theta(1)$	$\Theta(1)$
isEmpty()	$\Theta(1)$	$\Theta(1)$
Add()	$\Theta(1)$ [1]	$\Theta(n)$ [*]
GetMax()	$\Theta(n)$	$\Theta(1)$
RemMax()	$\Theta(n)$ [2]	$\Theta(1)$ [3]

(1) + sorozat végéhez új elem hozzáírása (*) + sorozat közepére kell betenni egy elemet

(2) nem kell léptetni a sorozat elemeit (3) + sorozat végéről elveszünk elemet

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, de a beszúrást az elemek léptetésével érhetjük el. Rendezetlen esetben viszont a GetMax()-ban nem kell léptetni a sorozat elemeit.

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)) [3.kvíz]

l := |seq| =0

pq := Add(pq, e)

seq := seq ⊕ <e>

e := GetMax(pq)

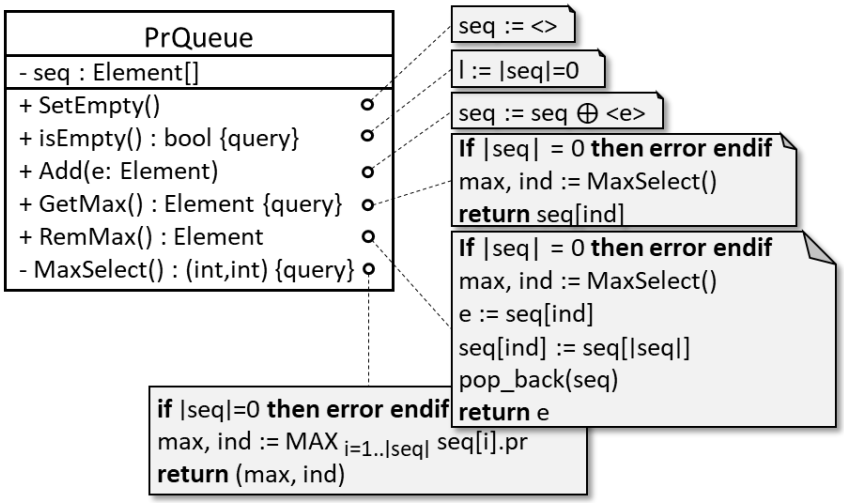
|seq|>0	
max, ind := $MAX_{i=1..\|seq\|}$(seq[i].pr)	**Hiba:** üres prioritásos sor
e := seq[ind]	

pq, e := RemMax(pq) [4.kvíz]

|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)	

A max, ind := MAX$_{i=1..|seq|}$(seq[i].pr) segédművelet specifikációja és algoritmusa: [5.kvíz]

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 = (t : Item^n, cout : Item*)

Ef = (t = t_0)

Uf = (t = t_0 ∧ cout = (t elemeit tartalmazza monoton csökkenően felsorolva))

Uf=(t = t_0 ∧ pq : PrQueue ∧ pq = $\bigcup_{i=1..n} \{t[i]\}$ ∧ cout = $\bigoplus^{\neg isEmpty()pq}$ RemMax(pq))

Mindkét lépés egy összegzés. Az összefűzés addig tart, amíg a pq prioritásos sor nem lesz üres. (Ez már előre vetíti a felsorolós változatát az összegzésnek.)

Összegzés („uniózás”)

i=m..n	~ i=1..n
f(i)	~ {t[i]}
s	~ pq
H, +, 0	~ Item*, ∪, <>

Összegzés (összefűzés)

i=m..n	~ pq elemeinek felsorolása
f(i)	~ RemMax(pq)
s	~ cout
H, +, 0	~ $\mathbb{S}$*, ⊕, <>

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

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() (max és az ind vizsgálandó)
- GetMax() (a maxsearch()-höz képest még a hibás esetet kell tesztelni)
- RemMax() (a max()-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> <1,2,3>	RemMax() RemMax() Add(3) Add(2) Add(1) RemMax()	2 1 3 2 1 3	<1> <> <3> <2,3> <1,2,3> <1,2>

Visszavezetés szekvenciális inputfájl felsorolásával

2. Egy szekvenciális inputfájlban kaktuszfajtákról tárolunk adatokat: név, őshaza, virágszín, méret.
   a. Számoljuk meg a piros virágú kaktuszokat!

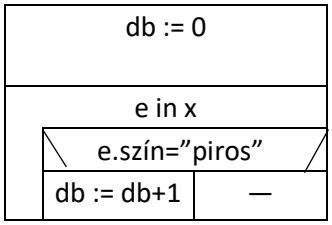
Specifikáció:

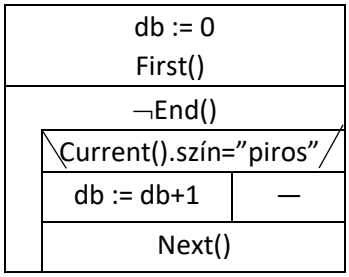
A = (x:infile(Kaktusz), db:$\mathbb{N}$)
Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)
Ef = ($x=x_0$)
Uf = (db = $\sum_{e \in x_0,\ e.szín=”piros”} 1$)

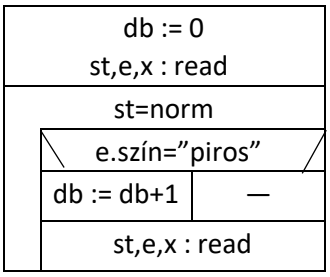
Számlálás

t:enor(E) ~ x:infile(Kaktusz) (st,e,x:read)
felt(e) ~ e.szín=”piros”
c ~ db

Algoritmus:







   b. Válogassuk ki egy szekvenciális outputfájlba a piros virágú kaktuszok, egy másikba a mexikói őshazájú kaktuszok neveit!

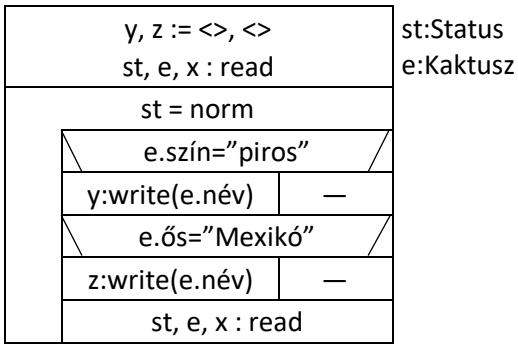
Specifikáció:

A = (x:infile(Kaktusz), y, z :outfile($\mathbb{S}$))
Kaktusz=rec(név:$\mathbb{S}$, szín:$\mathbb{S}$, ős:$\mathbb{S}$, méret:$\mathbb{N}$)
Ef = ($x=x_0$)
Uf = ($y=\bigoplus_{e \in x_0,\ e.szín=”piros”} e.név$ $\wedge$ $z=\bigoplus_{e \in x_0,\ e.ős=”Mexikó”} e.név$)

2 összegzés (kiválogatás) közös ciklusba vonva
- vagy 1 duplaösszegzés

H, +, 0 ~ ($\mathbb{S}$*, $\oplus$, <>), ($\mathbb{S}$*, $\oplus$, <>)
t:enor(E) ~ x:infile(Kaktusz) (st,e,x:read)
f_1(e) ~ <e.név> ha e.szín=”piros”
f_2(e) ~ <e.név> ha e.ős=”Mexikó”
s ~ y, z

Algoritmus:



3.kvíz [Egészítsük ki az algoritmust a piros virágú mexikói kaktuszok kiválogatásával.]