**SMap**

|  |  |
| --- | --- |
| asszociatív tömbök, azaz speciális tárolók halmaza,  ahol az elemek ℤ ⨯ 𝕊  (kulcs-adat) típusú párok | **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 : ℕ , e : ℤ ⨯ 𝕊, data : 𝕊 key : ℤ, l : 𝕃, data : 𝕊 |
| seq: Item\*  Item = rec(key:ℤ, data:𝕊)  Invariáns:  a sorozat kulcsuk szerint rendezetten tárolója az az Item típusú elemeket | **map:=SetEmpty(map)**   |  | | --- | | seq := <> |   **c := Count(map)**   |  | | --- | | c := ∣seq∣ |   **map := Insert(map,e)** l : 𝕃, ind : ℕ   |  |  | | --- | --- | | l, ind := logSearch(seq, e.key) | | | Øl | | | seq.Insert(ind, e) | **Hiba**: már létező kulcs |   **map := Remove(map, key)** l : 𝕃, ind : ℕ   |  |  | | --- | --- | | l, ind := logSearch(seq, key) | | | l | | | seq.Remove(ind) | **Hiba**: nem létező kulcs |   **l := In(map, key)**   |  |  | | --- | --- | | l, ind := logSearch(seq, key) | ind : ℕ |   **data := Select(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, 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.

A képen szöveg, képernyőkép, Betűtípus, szám látható

Előfordulhat, hogy a mesterséges intelligencia által létrehozott tartalom helytelen.

*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.

**PrQueue**

|  |  |
| --- | --- |
| a maximum prioritásos sorok, azaz olyan gyűjtemények, amelyek elemei ℤ ⨯ 𝕊 típusú párok. | **pq:=SetEmpty(pq)** pq : PrQueue  **l := isEmpty(pq)** l : 𝕃  **pq := Add(pq, e)** e : ℤ ⨯ 𝕊  **e := GetMax(pq)** // *pq nem üres*  **pq, e := RemMax(pq)** // *pq nem üres* |

|  |  |  |
| --- | --- | --- |
| n hosszú sorozat | **rendezetlen** | **rendezett** |
| **SetEmpty()** | Q(1) | Q(1) |
| **isEmpty()** | Q(1) | Q(1) |
| **Add()** | Q(1) | Q(log2 n) |
| **GetMax()** | Q(n) | Q(1) |
| **RemMax()** | Q(n) | Q(1) |

*Típusmegvalósítás:*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| seq: Element\*   * a prioritásos sor elemeit   rendezetlenül tároló sorozat, ahol Element = rec(pr : ℤ, data : 𝕊) | **pq := SetEmpty(pq)**)  seq := <>  **l := isEmpty(pq)**)  l := ∣seq∣ =0  **pq := Add(pq, e)**  seq := seq ⨁ <e>  **e := GetMax(pq)**   |  |  | | --- | --- | | |seq|>0 | | | max, ind := MAXi=1..[seq|(seq[i].pr) | **Hiba:** üres prioritásos sor | | e := seq[ind] |   **pq, e := RemMax(pq)**   |  |  | | --- | --- | | |seq|>0 | | | max, ind := MAXi=1..[seq|(seq[i].pr) | **Hiba** üres prioritásos sor | | e := seq[ind] | | seq[ind] := seq[∣seq∣] | | pop\_back(seq) | |

*Osztály diagram:*

A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

*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\*, cout : Item\*)

*Ef* = (t = t0)

*Uf* = (t = t0 Ù cout = ( t elemeit tartalmazza monoton csökkenően felsorolva) )

=(t = t0 Ù pq : PrQueue Ù pq = ⋃ i=1..n{t[i]}Ù cout = ⨁e in pq <e> )

*Összegzés* („uniózás”)

enor(t) ~ i=1 ..|t|

e ~ {t[i]}

s ~ pq

H, +, 0 ~ PrQueue, ⋃, <>

pr := pr ⋃ {e} ~ pr.Add(e)

*Összegzés* (összefűzés)

enor(t) ~ first, next, current: pq.RemMax()

end: pq.isEmpty()

f(i) ~ <e>

s ~ cout

H, +, 0 ~ 𝕊\*, ⨁, <>

cout := cout ⨁ <e> ~ cout.Write(e)

|  |  |
| --- | --- |
| pq.SetEmpty() | |
| i = 1 .. n | |
|  | pq.Add(t[i]) |
| cout := <> | |
| Øpq.isEmpty() | |
|  | cout := cout ⨁ pq.RemMax() |

*Prioritásos sor megvalósításának tesztelése***:**

* 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> |