|  |  |  |
| --- | --- | --- |
| **Sipos Richárd**  K9P7QZ  k9p7qz@inf.elte.hu  8.csoport | **1. beadandó/8.feladat** | 2020. március 7. |

# Feladat

*Valósítsa meg az egész számokat tartalmazó zsák típust! Ábrázolja a zsák elemeit ((az előfordulás számukkal együtt) egy sorozatban! Implementálja a szokásos műveleteket (adott elem betevése, kivétele, egy adott elem hányszor van a zsákban), valamint a zsák leggyakoribb elemét visszaadó műveletet (ehhez célszerű a zsákot reprezentáló sorozat mellett külön eltárolni a leggyakoribb elemet, és ezt karbantartani), továbbá egy zsák kiírását!*

# Zsák típus

A feladat lényege egy felhasználói típusnak a zsák típusnak a megvalósítása.

## Típusérték-halmaz

Olyan számokat (ebben az esetben egész számokat: ℤ) tartalmazó zsák típussal akarunk dolgozni, mely két, különböző egész számokat tárol egy indexen: magát a számot, valamint az előfordulási mennyiségét.

## Típus-műveletek

### 1. Adott elem betevése

Zsákba betenni egy adott elemet, majd növelni egyel az előfordulási számát.

azaz *Bag(n).times = Bag(n).times + 1*

### 2. Adott elem kivétele

Zsákból kivenni egy elemet, csökkenteni az előfordulási számát egyel, ha az utána nem nulla lesz.

azaz *Bag(n).times = Bag(n).times - 1*

Abban az esetben, ha utána az előfordulási száma nulla, akkor a helyére berakni a sorozat utolsó tagját, majd pop\_backelni a vectort.

Ha üres a zsák, vagy nincs benne az adott elem, exceptiont fog dobni, lekezelve.

*3. Egy adott elem hányszor van a zsákban*

Egy egész számot ad vissza (result), amely megmondja, hogy a paraméterül adott szám (n) hányszor szerepel a zsákban.

azaz *result = Bag(n).times*

Ha az elem nincs a zsákban, vagy a zsák üres, akkor a program azt megfelelően tudja kezelni, majd jelzi a problémát a felhasználó számára.

### 4. Zsák leggyakoribb eleme

Visszaadja (max), egy egyszerű maximum kereséssel (ha a zsák(Bag) nem üres), hogy melyik elem szerepel benne legtöbbször.

Ha a zsák üres, azt a program megfelelően kezeli, majd szól a felhasználónak a problémáról.

## Reprezentáció

Egy *n* elemű zsák, egy *T\_comm* típusú *vektor*(\_vec) segítségével van ábrázolva.

T\_comm:( number : ℤ, times : ℤ)

vector<T\_comm> \_vec

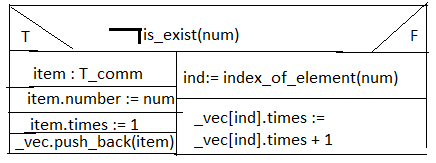
Ennek megfelelően:

\_vec[𝑖] = { number : ℤ, times : ℤ }

## Implementáció

### 1. Adott elem betevése

A \_vec vektorral ábrázolt zsákba egy *num* egész szám betevése az alábbi programmal implementálható:



Ahol az *is\_exist(num)* ellenőrzi, hogy a szám benne van-e a zsákban, az

*index\_of\_element(num)* pedig visszatér a már létező elem indexével.

### 2. Adott elem kivétele

A \_vec vektorral ábrázolt zsákba egy *num* egész szám kivétele az alábbi programmal implementálható:

### 3. Összeadás

A *v* tömbbel ábrázolt *a* mátrix és a *t* tömbbel ábrázolt *b* mátrix összege az *u* tömbbel ábrázolt *c* mátrixba kerül, ha az alábbi programot végrehajtjuk. A végrehajtás előtt ellenőrizni kell, hogy mindhárom mátrix, pontosabban az őket reprezentáló tömb azonos méretű-e.

*i*[*0*..*n-1*]: *u*[*i*]:= *v*[*i*] + *t*[*i*]

### 4. Szorzás

A *v* tömbbel ábrázolt *a* mátrix és a *t* tömbbel ábrázolt *b* mátrix szorzata az *u* tömbbel ábrázolt *c* mátrixba kerül, ha az alábbi programot végrehajtjuk. A végrehajtás előtt ellenőrizni kell, hogy mindhárom mátrix, pontosabban az őket reprezentáló tömb azonos méretű-e.

*i*[*0*..*n-1*]: *u*[*i*]:= *v*[*i*] \* *t*[*i*]

# Tesztelési terv

Megvalósított műveletek tesztelése (fekete doboz tesztelés)

1. Különféle méretű mátrixok létrehozása, feltöltése és kiírása.
   1. 0, 1, 2, 5 dimenziójú mátrix
2. Mátrix adott pozíciójú értékének lekérdezése és megváltoztatása.
   1. Diagonálisra eső elem lekérdezése és megváltoztatása
   2. Diagonálison kívüli elem lekérdezése és megváltoztatása
   3. Illegális index megadása, 0 dimenziós mátrix indexelése 3) A másoló konstruktor kipróbálása.

a) A *b* mátrix létrehozása az *a* mátrix mintájára, majd a két mátrix tartalmának összehasonlítása, majd az egyik mátrix megváltoztatása és a két mátrix tartalmának összehasonlítása.

4) Az értékadás operátor kipróbálása.

1. A *b=a* értékadás végrehajtása az *a* és *b* mátrixokra (az *a* és *b* mérete azonos illetve különbözik: egyik, illetve másik a nagyobb), majd a két mátrix tartalmának összehasonlítása, majd az egyik mátrix megváltoztatása és a két mátrix tartalmának összehasonlítása.
2. A *c=b=a* értékadás végrehajtása az *a*, *b* és *c* mátrixokra (ezek mérete lehet különböző), majd *a* és *c* mátrixok tartalmának összehasonlítása, majd az egyik mátrix megváltoztatása és a mátrixok tartalmának összehasonlítása.
3. Az *a=a* értékadás végrehajtása az *a* mátrixra, majd az *a* mátrix tartalmának ellenőrzése. 5) A *c*:=*a*+*b* mátrixösszeadás kipróbálása.
4. Eltérő méretű mátrixokkal (az *a* és *b* mérete különbözik, a *c* és *a* mérete különbözik)
5. Kommutativitás ellenőrzése (a + b == b + a)
6. Asszociativitás ellenőrzése (a + b + c == (a + b) + c == a + (b + c))
7. Null elem vizsgálata (a + 0 == a, ahol 0 a null mátrix) 6) A *c*:=*a*\**b* mátrixszorzás kipróbálása.
8. Eltérő méretű mátrixokkal. (az *a* és *b* mérete különbözik, a *c* és *a* mérete különbözik)
9. Kommutativitás ellenőrzése (a \* b == b \* a)
10. Asszociativitás ellenőrzése (a \* b \* c == (a \* b) \* c == a \* (b \* c))
11. Null elem vizsgálata (a \* 0 == 0, ahol 0 a null mátrix)
12. Egység elem vizsgálata (a \* 1 == a, ahol 1 az egység mátrix)

Megj: A beolvasó és kiíró operátorok teszteléséhez elég, hogy ezeket a fenti esetek tesztelésénél intenzíven használjuk.

Tesztesetek a kód alapján (fehér doboz tesztelés) 1. Extrém méretű (-1, 0, 1, 1000) mátrix létrehozása.

2. Kivételek generálása és elkapása.