Gonda Dávid OAF 1.beadandó

Feladat

Valósítsa meg az egész számokat tartalmazó zsák típust! A zsákot dinamikusan lefoglalt tömb segítségével ábrázolja! Implementálja a szokásos műveleteket (elem betevése, kivétele, üres-e a halmaz, egy elem hányszor van a zsákban), valamint két zsák metszetét (a közös elemek a kisebb előfordulási számmal maradnak meg), továbbá egy zsák kiírását, és végül a másoló konstruktort és az értékadás operátort! Törekedjen a metszetképzés műveletigényének minimalizálására, a dokumentációban mutasson rá a saját megoldásának műveletigényére!

Zsák típus

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

Típusérték-halmaz

Olyan számokat (ebben az esetben egész számokat: $\mathbb{Z}$) tartalmazó zsákkal akarunk dolgozni, amelyekben az különböző értékű elemek számát tartjuk nyilván. Az elemek sorrendje lényegtelen. E zsákoknak a mérete ($n \in N$) lényeges paraméter a megvalósítás szempontjából, de lényegtelen a felhasználó számára.

Típus-műveletek

1. Érték betétele

Adott érték betétele a zsákba.
Ha a tomb még üres, az elemet beillesztjük a tömb első helyére, majd számlálóját 1-re váltjuk, ezután a méretet megnöveljük egyel.
Ha az adott elem már létezik a zsákban, akkor az elem számlálójához adunk csak egyet, ha nem létezik még, akkora dinamikus tomb méretét megnöveljük egyel, majd az új üres helyre illesztjük az elemet, számlálóját pedig 1-re állítjuk.

2. Érték kivétele

Adott érték kivétele a zsákban.
Ha az elem benne van a zsákban, és a számlálója nagyobb mint 1, akkor cask kivonunk a számlálójából 1-et. Ha benne van a zsákban, de a számlálója 1, akkor a tömbben lévő utolsó elemet az aktuális elemre másoljuk, majd a tomb méretét csökkentjük egyel.
Ha az elem nem szerepel a zsákban, akkor békénhagyjuk.

3. Üres-e a tomb

Ha a méret = 0, akkor igazat ad, egyébként hamisat.

4. Érték darabszáma

A zsákban az adott érték előfordulása.
Ha a zsákban szerepel az adott érték, kiírja a darabszámát, egyébként kiírja hogy “Nincs ilyen elem”. Szükség esetén visszatérési értékként az elemszámot visszaadja.

5. Metszet

Megadja két zsák metszetét.
Elindít egy ciklust az egyik zsákra, ebbe ágyazva pedig a másikra. Ha a két zsák valamely eleme megegyezik, beilleszti az adott zsákba ezt az elemet, majd a kisebb darabszámot be.

Implementáció

1. Érték betétele

```
void Bag::insert(int e) {
    if(isEmpty() == 1) {
        meret = meret + 1;
        elems[0].darab++;
        elems[0].ertek = e;
    }else{
        bool l = 0;
        for(int i = 0; i < meret + 1; i++) {
            if(elems[i].ertek == e) {
                l = 1;
                elems[i].darab++;
            }
        }
        if(l == 0) {

            zsElem *p = elems;
            elems = new zsElem[meret+1];

            for(int i = 0; i < meret; i++) {
                elems[i].ertek = p[i].ertek;
                elems[i].darab = p[i].darab;
            }
            delete [] p;


            elems[meret].darab = 1;
            elems[meret].ertek = e;
            meret = meret + 1;
        }
    }
}
```

2. Elem kivétele

```
void Bag::erase(int e) {
    for(int i=0; i<meret; i++) {
        if (elems[i].ertek==e) {
            elems[i].darab = elems[i].darab - 1;
            if (elems[i].darab==0) {
                elems[i]=elems[meret-1];
                meret = meret - 1;
            }
        }
    }
}
```

3. Üres-e a tömb

```
bool isEmpty() {return meret == 0;};
```

4. Érték darabszáma

```
int Bag::elemCount(int e) {
    bool l = 0;
    int i = 0;
    while(l == 0 && i < meret) {
        if(elems[i].ertek == e) l = 1;
        else i++;
    }
    if(l == 0) {
        std::cout << "Nincs ilyen elem" << std::endl << std::endl;
        return 0;
    }
    else std::cout << "A megadott elem darabszama: " << elems[i].darab << std::endl << std::endl;
    return elems[i].darab;
}
```

5. Metszet

```
void Bag::intersection(Bag x, Bag a){
    for(int i = 0; i < x.meret; i++){
        for(int j = 0; j < a.meret; j++){
            if(x.elems[i].ertek == a.elems[j].ertek){
                this->insert(x.elems[i].ertek);
                if(x.elems[i].darab <= a.elems[j].darab) this->elems[meret-1].darab = x.elems[i].darab;
                else this->elems[meret-1].darab = a.elems[j].darab;
            }
        }
    }
}
```

A metszet műveletigénye

Legjobb esetben nincs közös érték, ekkor a műveletigény csak a két ciklus, azaz *x.meret * a.meret*. Legrosszabb esetben az összes érték metszet, ekkor a műveletigény (*x.meret * a.meret * 2összehasonlítás) + (x.meret * a.meret * 2értékadás).*

Osztály

A zsák egy osztály és egy struktúra segítségével jön létre:

```
//Zsák elemei struktúra
struct zsElem{
    int ertek;
    int darab;
    };

//Zsák osztály
//Műveletek: Zsákba beszúrás, törlés, üres-e, megadott elem darab száma, képernyőre írás, metszet
//Reprezentáció: Zsák elemei struktúrával vannak megadva (elem értéke, darabszáma), ezen elemek pedig egy dinamikus tömbben szerepelnek
class Bag{
public:
    Bag();
    ~Bag();

    void insert(int e);
    void erase(int e);
    bool isEmpty() {return meret == 0;};
    int elemCount(int e);
    void write();
    void intersection(Bag x, Bag a);

//Tesztekhez
    int _ertek(int e) { return elems[e].ertek;}
    int _darab(int e) { return elems[e].darab;}

    Bag(const Bag& x);
    Bag& operator=(const Bag& x);

private:
    zsElem* elems;
    int meret;
};
```

Ez a definíció a *Bag.h*-ban van.

Tesztelési terv

1. Érték betétele
    - inp.txt fájlból való betétel : 9 0 3 3 0 0

2. Érték kivétele
    - Érték ami többször van bent
    - Érték ami egyszer van bent
    - Érték ami nincs bent

3. Üres-e
    - Lekérdezés üres zsákon
    - Lekérdezés nem üres zsákon

4. Érték darabszáma
    - Érték ami többször benne van
    - Érték ami egyszer van benne
    - Érték ami nincs benne

(ez a 3 egy tesztesetben benne van)

5. Két zsák metszete
    - Nincs közös érték
    - Van közös érték
    - Egy üres zsák, egy nem üres zsák
    - Két üres zsák

6. Értékadás
    - a = b