# Gonda Dávid OAF 2.beadandó

## Feladat

Készítsen egy zsák típust! A zsákokat rendezett láncolt listával ábrázolja! Implementálja a szokásos műveleteket (elem betétele, kivétele, benne van-e egy adott elem, hányszorosan van benne, üres-e), egészítse ki az osztályt a zsák tartalmát kiíró operátor<<-ral! Definiáljon olyan barát-operátorokat is, amely kiszámítja két zsák szimmetrikus differenciáját (a közös elemek a multiplicitása az eredményben az eredeti multiplicitások különbségének abszolút értéke legyen) és metszetét (a közös elemeket a kisebb előfordulási számmal)! A szimmetrikus differencia és a metszet műveletigénye: $\mathcal{O}(m+n)$, ahol m és n a két zsáknak megfelelő halmazok elemszáma.

## Zsák típus

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

## Típusérték-halmaz

Olyan számokat (ebben az esetben egész számokat: $\mathbb{Z}$) tartalmazó zsákkal akarunk dolgozni, amelyekben a különböző értékű elemek számát tartjuk nyilván. Az elemek növekvő sorrendben helyezkednek el egy láncolt listán, amit a zsák első és utolsó listaelemeivel érhetünk el.

## Típus-műveletek

### *1. Érték betétele*

Adott érték betétele a zsákba.
Először beolvas egy inputot egy egész változóba, ha beolvasás hibás, dob egy kivételt. Ha nem, akkor megnézi, hogy üres-e a lista, vagy hogy az első elem nagyobb vagy egyenlő-e az inputtal, ha ebből valamelyik igaz, akkor megnézi hogy az első elem nagyobb-e mint az input vagy hogy üres-e a lista, ha ezekből valamelyik igaz, akkor beilleszti az inputot az első elem elé egy egyes számlálóval, és egyenlővé teszi az első elemet a beszúrt inputtal. Ha az input egyenlő az első elemmel, hozzáad az első elem számlálójához egyet, ha ezek közül semmi sem igaz, akkor beilleszti az inputot növekvő sorrend szerint. Hogyha valahol szerepel az input esetleg, akkor a számlálójához hozzáad egyet.

### *2. Üres-e a tomb*

Ha a zsák első pointere NULL, igazat ad vissza.

### *3. Érték darabszáma*

A zsákban az adott érték előfordulása.
Ha a zsákban szerepel az adott érték, visszaadja a darabszámát, egyébként nullát. A lista bejárását itt egy enumerátorral valósítjuk meg.

### *4. Metszet*

Megadja két zsák metszetét.
Ha az egyik zsák üres, üres listát ad. Különben bemásolja a közös elemeket ha vannak.
Egyezés esetén bemásolja az egyező értéket, és a kisebb elemszámot.

### *5. Szimmetrikus Differencia*

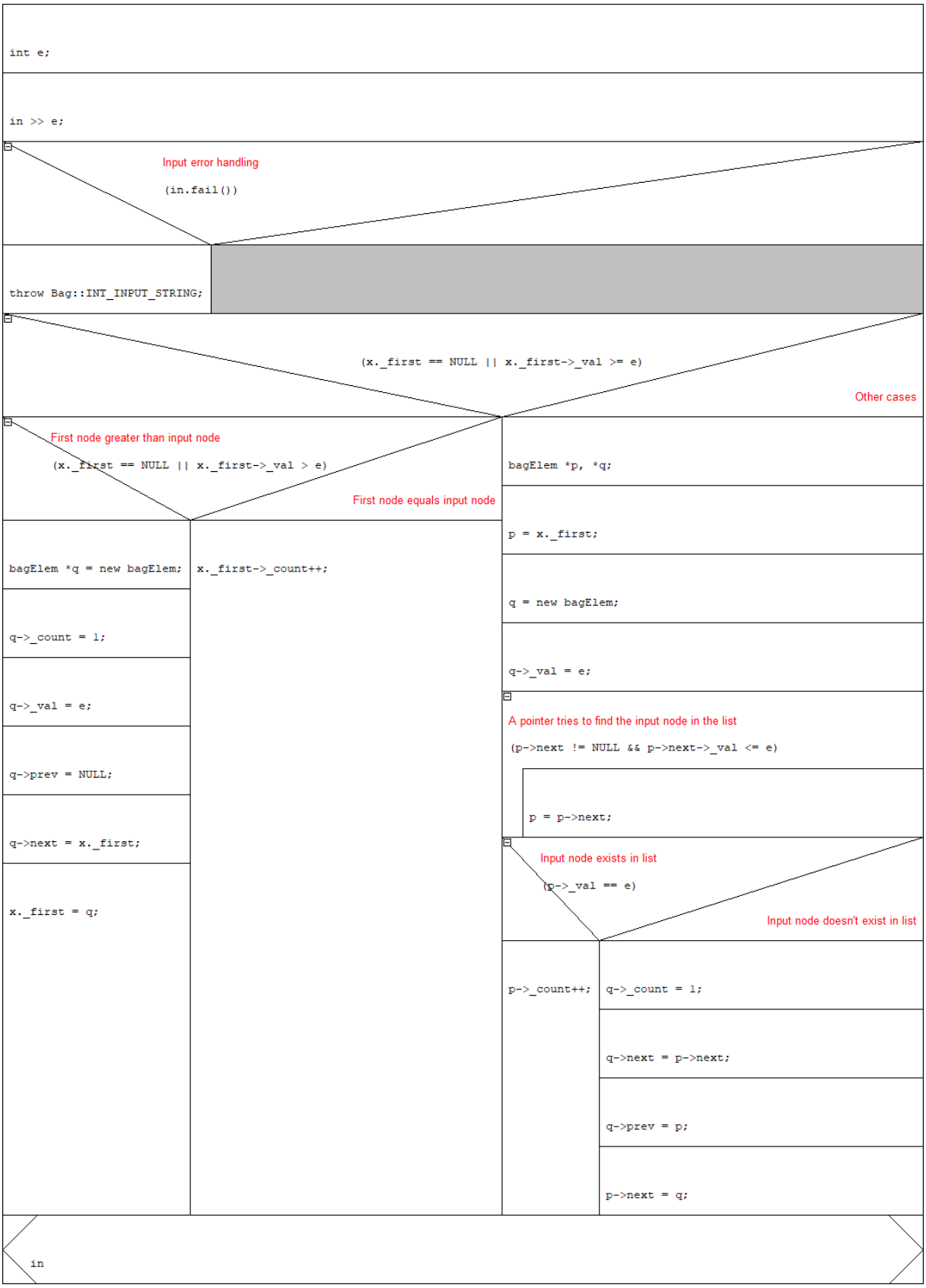
Megadja két zsák szimmetrikus differenciáját.
Ha valamelyik zsák üres, akkor a nem üres lista lesz a szimmetrikus differencia. Különben végigmegy a két zsákon két pointerrel. Ha az értékük és a számlálójuk megegyezik, akkor tovább lépteti mind a kettő pointert, ha a számlálójuk nem egyezik meg, akkor a szimmetrikus differencia zsákba behelyezi az értéket, és a két számláló különbségének abszolút értékét. Ha az egyik pointer értéke (a) nagyobb mint a másiké (b), akkor beilleszti b-t és tovább lépteti. Ha fordítva van, akkor ugyanez történik a-val. Ezek után ha az egyik pointer eléri a NULL-t, akkor a függvény a másikat hozzáláncolja a szimmetrikus differencia zsákhoz.

### *6. Elem törlése*

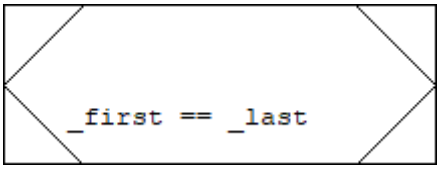
Töröl egy elemet a listából
A függvény megkeresi az input értéket a zsákban, és ha létezik, és a számlálója nagyobb mint egy, akkor azt csökkenti egyel, ellenkező esetben az előtte lévő elemet az utána lévőhöz fűzi.

# Implementáció

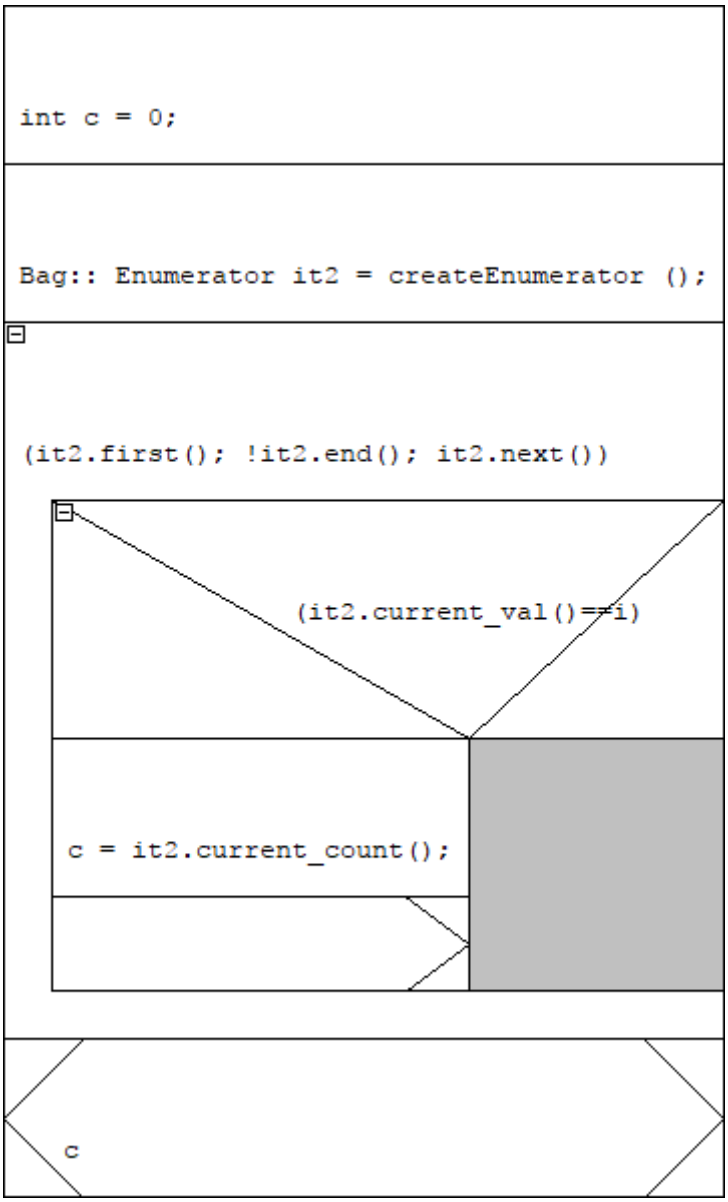
## *1. Érték betétele*



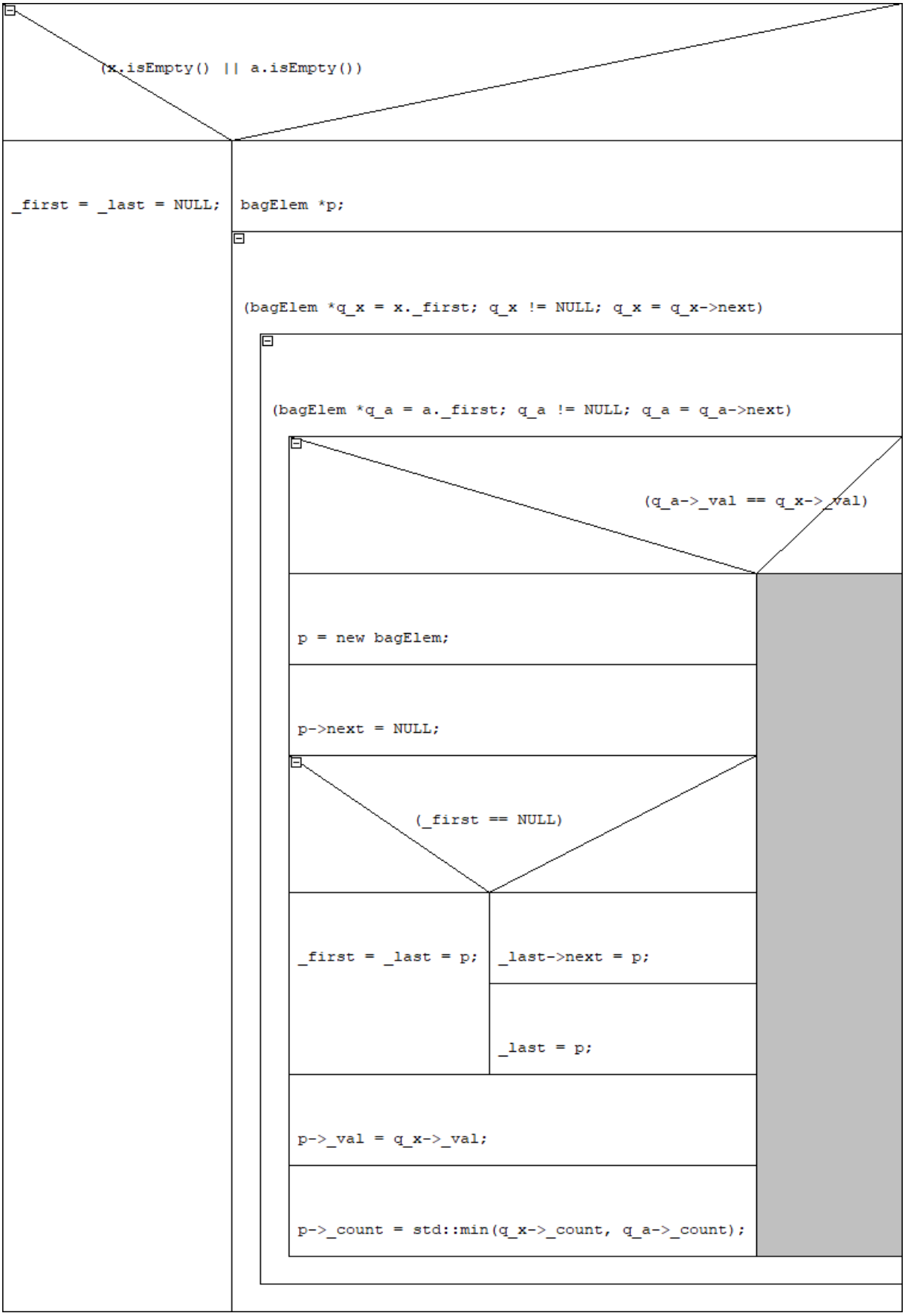
*2. Üres-e a tömb*



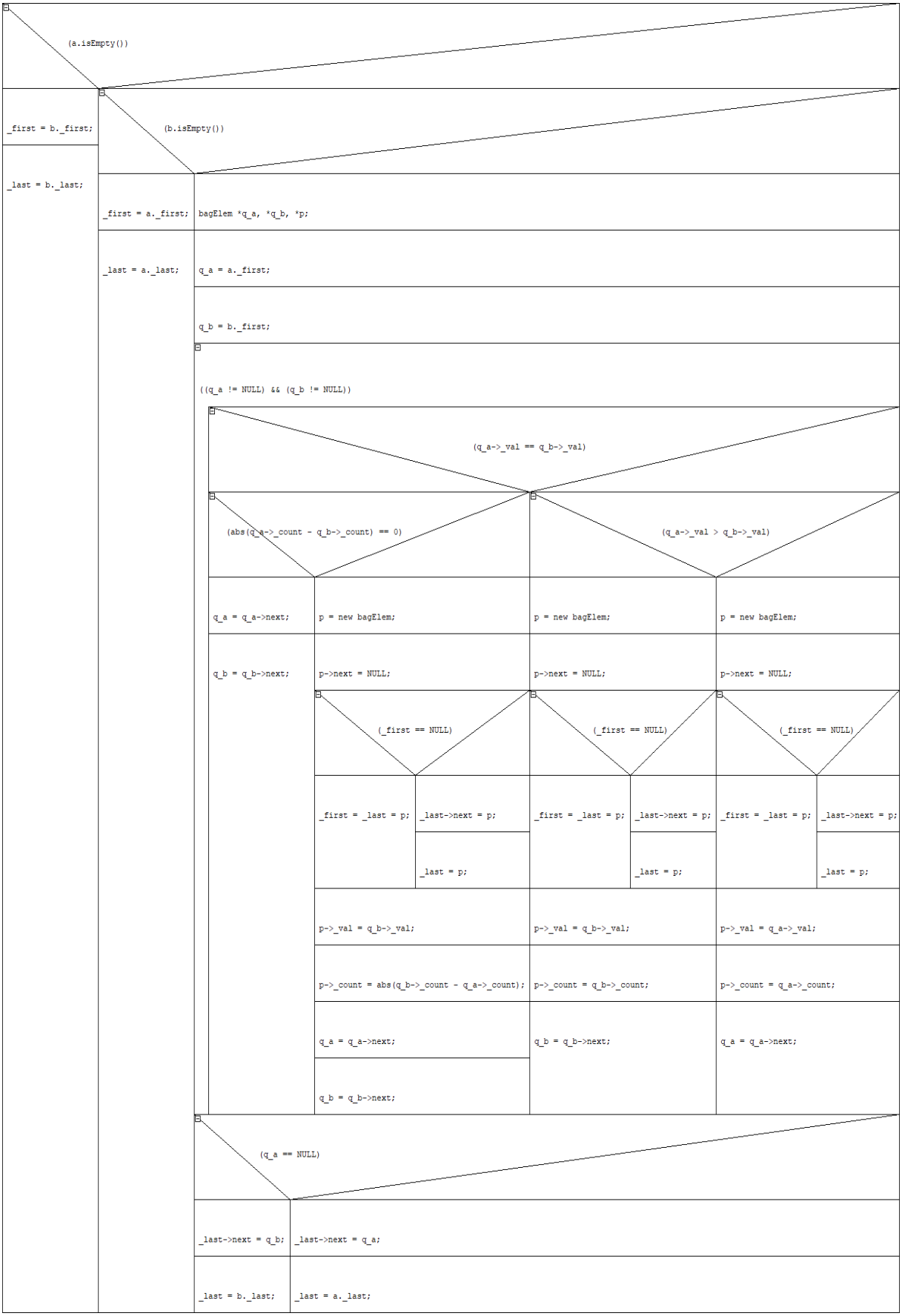

*3. Érték darabszáma*



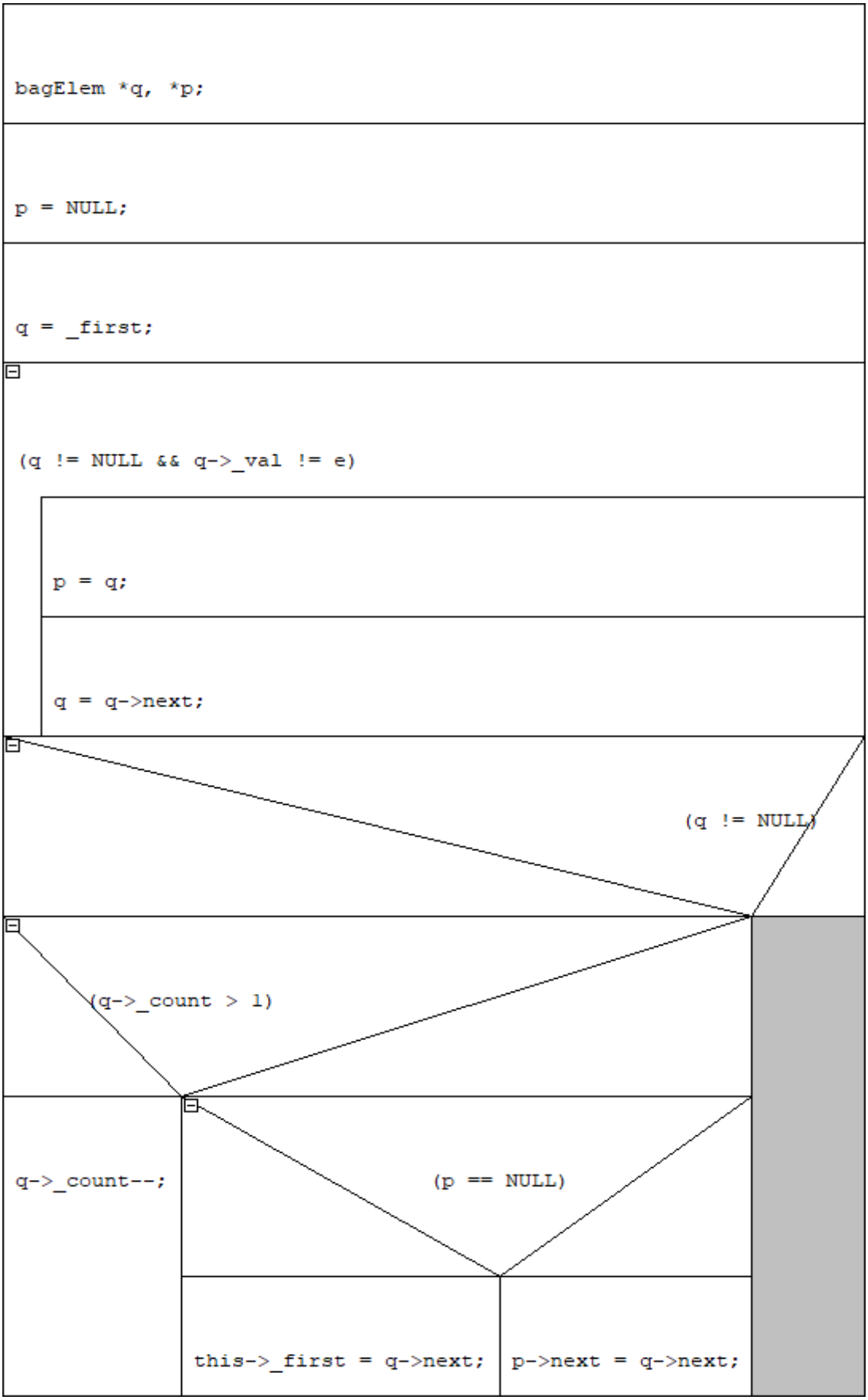
*4. Metszet*


(x.isEmpty() || a.isEmpty())
_first = _last = NULL;
bagElem *p;
(bagElem *q_x = x._first; q_x != NULL; q_x = q_x->next)
(bagElem *q_a = a._first; q_a != NULL; q_a = q_a->next)
(q_a->_val == q_x->_val)
p = new bagElem;
p->next = NULL;
(_first == NULL)
_first = _last = p;
_last->next = p;
_last = p;
p->_val = q_x->_val;
p->_count = std::min(q_x->_count, q_a->_count);

*5. Szimmetrikus Differencia*



*6. Elem törlése*



## Osztály

A zsák egy osztály és egy struktúra segítségével jön létre:

```
class Bag{
public:
    enum Exceptions{ INT_INPUT_STRING };
    Bag();
    ~Bag();

    void erase(int e);
    bool isEmpty() const { return _first == _last; };
    int count(int e);
    bool isElem(int i);
    friend std::ostream& operator<<(std::ostream& out, Bag& x);
    friend Bag operator-(const Bag& a, const Bag& b);
    friend Bag operator+(const Bag& x, const Bag& a);

    Bag(const Bag& x);
    Bag& operator=(const Bag& x);
    friend std::istream& operator>>(std::istream& in, Bag& x);

class Enumerator {
    public:
        Enumerator(Bag *p):_bq(p),_current(NULL) {
            ++(_bq->_enumaratorCount);
        }
        ~Enumerator() {
            --(_bq->_enumaratorCount);
        }
        int current_val()const {
            return _current->_val;
        }
        int current_count()const {
            return _current->_count;
        }
        void first()  {
            _current = _bq->_first;
        }
        bool end()    const {
            return _current==NULL;
        }
        void next()   {
            _current = _current->next;
        }

    private:

        Bag *_bq;
        bagElem *_current;
    };

    Enumerator createEnumerator() {
        return Enumerator(this);
    }

private:
    bagElem *_first;
    bagElem *_last;

    int _enumaratorCount;
};
```

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. Két zsák szimmetrikus differenciája
    - Nincs közös érték
    - Van közös érték
    - Egy üres zsák, egy nem üres zsák
    - Két üres zsák

7. Értékadás
    - a = b