DSA - Seminar 1 – ADT Bag

ADT Bag (also called MultiSet)

- A Bag is similar to a set, but the elements do not have to be unique.
- It is similar to a shopping bag/cart: I can buy multiple pieces of the same product and the order in which I buy the elements is not important.
- The order of the elements is not important
- There are no positions in a Bag:
  - There are no operations that take a position as parameter or that return a position.
  - The added elements are not neccesarily stored in the order in which they were added (they can be stored in this way, but there is no guarantee, and we should not make any assumptions regarding the order of the elements).
  - For example:
    - We add to an initially empty Bag the following elements: 1, 3, 2, 6, 2, 5, 2
    - If we print the content of the Bag, the elements can be printed in any order:
      - 1, 2, 2, 2, 3, 6, 5
      - 1, 3, 2, 6, 2, 5, 2
      - 1, 5, 6, 2, 3, 2, 2
      - Etc.

Domain: $\mathcal{B}$ = {b | b is a Bag with elements of the type TElem}

Interface (set of operations):

init(b)

pre : true

post: b ∈ $\mathcal{B}$, b is an empty Bag

add(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: b' ∈ $\mathcal{B}$, b' = b U {e} (Telem *e* is added to the Bag)

remove(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: b' ∈ $\mathcal{B}$, b' = b \ {e} (one ocurrence of e was removed from the Bag).

$$\text{remove} \leftarrow \begin{cases} \text{true, if an element was removed } (\text{size(b')} < size(b)) \\ \text{false, if e was not present in b } (\text{size(b')} = \text{size(b)}) \end{cases}$$

search(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ TElem

post: $search \leftarrow \begin{cases} true, if\ e \in \mathbf{B} \\ false, otherwise \end{cases}$

size(b)

pre: b ∈ $\mathcal{B}$

post: size ← $the\ number\ of\ elements\ from\ b$

nrOccurrences(b, e)

pre: b ∈ $\mathcal{B}$, e ∈ Telem

post: $nrOccurrences\ \leftarrow the\ number\ of\ occurrences\ of\ e\ in\ b$

destroy(b)

pre: b ∈ $\mathcal{B}$

post: b was destroyed

iterator(b, i)

pre: b ∈ $\mathcal{B}$

post: i ∈ $\mathcal{I}$, i is an iterator over b

ADT Iterator

- Has access to the interior structure of the Bag and it has a current element from the Bag.

Domain: $\mathcal{I}$ = {i | i is an iterator over b ∈ $\mathcal{B}$ }

Interface:

init(i, b)

pre: b ∈ $\mathcal{B}$

post: i ∈ $\mathcal{I}$, i is an iterator over b. i refers to the first element of b, or it is invalid if b is empty

valid(i)

pre: i ∈ $\mathcal{I}$

post: $valid\ \leftarrow \begin{cases} true, if\ the\ current\ element\ from\ i\ is\ a\ valid\ one \\ false, otherwise \end{cases}$

first(i)

pre: i ∈ $\mathcal{I}$

post: i' ∈ $\mathcal{I}$, the current element from i' refers to the first element from the bag or i is invalid if the bag is empty

next(i)

pre: i ∈ $\mathcal{I}$, valid(i)

post: i' ∈ $\mathcal{I}$, the current element from i' refers to the next element from the bag b.

throws: exception if i is not valid

getCurrent(i, e)

pre: i ∈ $\mathcal{I}$, valid(i)

post: e ϵ TElem, e is the current element from i
throws: exception if i is not valid

Reprezentation:

1.
   - A dynamic array of elements, where every element can appear multiple times.
   - The iterator will contain a current position from the dynamic array

1	3	2	6	2	5	2

2.
   - A dynamic array of unique elements, and for each element its frequency (either two dynamic arrays, or one single dynamic array of pairs).
   - The iterator will have a current position and a current frequency for the element.

(1,1)	(3,1)	(2,3)	(5,1)	(6,1)

Python implementation

- **Note:** lists from Python are actually Dynamic Arrays

Version 1:

```
class Bag:

    def __init__(self):
        self.__elems = []

    def add(self, e):
        self.__elems.append(e)

    def remove(self, e):
        if e in self.__elems:
            self.__elems.remove(e)
            return True
        return False


    def search(self, e):
        return e in self.__elems

    def size(self):
        return len(self.__elems)

    def nrOccurrences(self, e):
        count = 0
        for elem in self.__elems:
            if elem == e:
                count = count + 1
        return count
```

```
    def iterator(self):
        return BagIterator(self)


class BagIterator:

    def __init__(self, bag):
        self.__bag = bag
        self.__current = 0

    def first(self):
        self.__current = 0

    def valid(self):
        return self.__current < self.__bag.size()

    def next(self):
        if self.__current == self.__bag.size():
            raise ValueError() #in Python e raise, nu throw. ValueError e
eroare built-in in Python, puteti folosi orice
        self.__current += 1

    def getCurrent(self):
        if self.__current == self.__bag.size():
            raise ValueError()
        return self.__bag._Bag__elems[self.__current]
```

Main program:

```
from Bag import Bag, BagIterator

def createIntBag():
    intBag = Bag()
    intBag.add(6)
    intBag.add(11)
    intBag.add(77)
    intBag.add(7)
    intBag.add(4)
    intBag.add(6)
    intBag.add(77)
    intBag.add(6)
    return intBag

def createStringBag():
    stringBag = Bag()
    stringBag.add("data")
    stringBag.add("structure")
    stringBag.add("and")
    stringBag.add("algorithms")
    stringBag.add("data")
    stringBag.add("data")
    stringBag.add("algorithms")
    return stringBag

def printBag(bag):

    it = bag.iterator()
    while it.valid():
        print(it.getCurrent())
        it.next()
    print("Over. Let's start again")
    it.first()
    while it.valid():
        print(it.getCurrent())
        it.next()


def main():

    b1 = createIntBag()
    printBag(b1)
    print("Number of occurrences for 77: ", b1.nrOccurrences(77))
    b2 = createStringBag()
    printBag(b2)
    print("Number of occurrences for data: ", b2.nrOccurrences("data"))

main()
```

Version 2:

```
class BagF:

    def __init__(self):
        self.__elems = []
        self.__freq = []

    def add(self, e):
        found = False
        for i in range(len(self.__elems)):
            if self.__elems[i] == e:
                self.__freq[i] += 1
                found = True
        if not found:
            self.__elems.append(e)
            self.__freq.append(1)

    def remove(self, e):
        for i in range(len(self.__elems)):
            if self.__elems[i] == e:
                self.__freq[i] -= 1
                if self.__freq[i] == 0:
                    del self.__elems[i]
                    del self.__freq[i]
                return True
        return False

    def search(self, e):
        return e in self.__elems

    def size(self):
        contor = 0
        for fr in self.__freq:
            contor += fr
        return contor

    def nrOccurrences(self, e):
        for i in range(len(self.__elems)):
            if e == self.__elems[i]:
                return self.__freq[i]
        return 0

    def iterator(self):
        return BagFIterator(self)


class BagFIterator:

    def __init__(self, bag):
        self.__bag = bag
        self.__currentE = 0
        self.__currentFr = 1

    def first(self):
        self.__currentE = 0
```

```
        self.__currentFr = 1

    def valid(self):
        return self.__currentE < len(self.__bag._BagF__elems)

    def next(self):
        if self.__currentE == len(self.__bag._BagF__elems):
            raise ValueError()
        self.__currentFr += 1
        if self.__currentFr > self.__bag._BagF__freq[self.__currentE]:
            self.__currentE += 1
            self.__currentFr = 1

    def getCurrent(self):
        if self.__currentE == len(self.__bag._BagF__elems):
            raise ValueError()
        return self.__bag._BagF__elems[self.__currentE]
```