DSA - Seminar 2 – 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). If e is not in b, b is not changed.

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 ← $the\ number\ of\ elements\ from\ b$

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

valid(i)

pre: i ∈ $\mathcal{I}$

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

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.

getCurrent(i, e)

pre: i ∈ $\mathcal{I}$, valid(i)

post: e ∈ TElem, e is the current element from i

Reprezentation:

1.

- A dynamic array of elements, where every element can appear multiple times.
- The iterator will have 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 hve 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)

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

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

    def iterator(self):
        return Iterator(self)


class Iterator:

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

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

    def next(self):
        self.__current += 1

    def getCurrent(self):
        return self.__bag._Bag__elems[self.__current]
```

Main program:

```
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()


def main():

    b1 = createIntBag()
    printBag(b1)
    b2 = createStringBag()
    printBag(b2)

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]

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

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

    def iterator(self):
        return IteratorF(self)


class IteratorF:

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

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

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

    def getCurrent(self):
```

```
        return self.__bag._BagF__elems[self.__currentE]
```