14. ADT Bag on Doubly Linked List on an Array

Resiga Alexandru

916

**Domain:**

**B =** {b | b is a Bag with elements of type TElem}

**Interface (set of operations):**

* init(b)

*pre*: true

*post*: b ∈ **B**, b = φ

* add(b, e)

*pre*: b ∈ **B**, e ∈ TElem

*post*: b’ ∈ **B**, b’ = b U {e} (TElem e is added to the Bag)

* remove(b, e)

*pre*: b ∈ **B**, e ∈ TElem

*post*: b’ ∈ **B**, b’ = b \ {e} (one ocurrence of e was removed from the Bag)

* search(b, e)

*pre*: b ∈ **B**, e ∈ TElem

*post*: search = {true, if e ∈ **B**, false, otherwise}

* size(b)

*pre*: b ∈ **B**

*post*: size = the number of elements from b

* destroy(b)

*pre*: b ∈ **B**

*post*: b was destroyed

* iterator(b, i)

*pre*: b ∈ **B**

*post*: i ∈ **I**, i is an iterator over b

**Bag Iterator:**

**Domain:**

**I** = {i | i is an iterator over b ∈ **B**}

**Interface:**

* init (i, b)

*pre*: b ϵ **B**

*post*: i ϵ **I**, i is an iterator over b

* valid(i)

*pre*: i ϵ **I**

*post*: valid = {true, if the current element from i is a valid one, false, otherwise}

* next(i)

*pre*: i ϵ **I**, valid(i)

*post*: i’ ϵ **I**, the current element from i’ refers to the next element from the bag b.

* getCurrent(i, e)

*pre*: i ϵ **I**, valid(i)

*post*: e ϵ TElem, e is the current element from i

**Representation:**

**DLLANode:**

info: TElem

next: Integer

prev: Integer

**DLLA:**

bag: DLLLANode[]

cap: Integer

head: Integer

tail: Integer

firstEmpty: Integer

**DLLAIterator:**

list: DLLA

currentElement: Integer

**Problem:**

*You are running an online shopping platform and in order to process the items added to the shopping cart by clients you need to implement a container to facilitate basic operations such as adding, removing, searching items in the cart.*

**Explanation:**

In this case, we would use a bag because users can add items which are not unique, the order in which users add items to the shopping cart is not relevant and we can simply iterate through the elements in order to find a specific one.

**Implementation of operations:**

**subalgorithm init(b)** is:

b.cap ← INIT\_CAPACITY

b.bag[0].prev <- -1

b.bag[0].next <- 1

b.bag[INIT\_CAPACITY].next <- -1

b.bag[INIT\_CAPACITY].prev <- INIT\_CAPACITY -2

for i ← 1, b.cap-1 execute

b.bag [i].next ← i + 1

b.bag[i].prev ← i - 1

end-for

b.head <- -1

b.tail = <- -1

b.firstEmpty ← 1

**end-subalgorithm**

**Complexity:** Θ(INIT\_CAPACITY) ϵ Θ(1)

**subalgorithm add(b, e)** is:

newElem ← b.firstEmpty

b.firstEmpty <- b.bag[b.firstEmpty].next

b.bag[newElem].info = e

if newElem = 1 then

if b.head = -1

b.head = newElem

b.tail = newElem

else

b.bag[b.head].prev <- newElem

end-if

else

nodC <- b.head

pos = 1

while b.bag[nodC].next != b.bag[b.firstEmpty].next do

pos <- pos+1

nodC <- b.bag[nodC].next

end-while

nodC <- b.bag[nodC].prev

pos <- pos -1

if nodC != b.firstEmpty then

nodNext = b.bag[nodC[.next

b.bag[newElem].next <- nodNext

b.bag[newElem].prev <- nodC -1

b.bag[nodC-1].next <- newElem

if nodNext != b.bag[b.firstEmpty].next then

b.tail <- newElem

else

b.bag[nodNext].prev <- newElem

end-if

end-if

end-if

**end-subalgorithm**

**Complexity:** Θ(1)

**function search (b, e)** is:

current ← b.head

if current = -1

search <- False

while current != b.firstEmpty && b.bag[current].info != e do

current ← b.bag[current].next

end-while

if current != b.firstEmpty then

search ← True

else

search ← False

end-if

**end-function**

**Complexity:**

Best Case: Θ(1) – when the sought element is b.head

Average Case: Θ(n)

Worst Case: Θ(n) – when the sought element is not in the bag

**subalgorithm remove(b, e)** is:

if b.search(e) = True

nodC ← b.head

prevNode ← -1

while nodC != b.bag[b.firstEmpty].next && b.bag[nodC].info != p do

prevNode ← nodC

nodC ← b.bag [nodC].next

end-while

if nodC != b.firstEmpty then

if nodC = b.head then

if b.head = b.tail

b.head 🡨 -1

b.tail 🡨 -1

b.firstEmpty <- 1

else

b.firstEmpty = b.head

b.head = b.bag[b.head].next

b.bag[b.firstEmpty].next = b.bag[b.tail].next

b.bag[b.tail].next = b.firstEmpty

end-if

else

b.firstEmpty <- nodC

if nodC = tail then

b.tail = prevNode

else

b.bag[prevNode].next <- b.bag[nodC].next

b.bag[b.bag[nodC].next].prev <- b.bag[nodC].prev

b.bag[b.bag[nodC].next].next 🡨 b.firstEmpty

end-if

end-if

end-if

end-if

**end-subalgorithm**

**Complexity:**

Best Case: **Θ(1)**

Average Case: **Θ(n)**

Worst Case: **Θ(n)**

**subalgorithm size(b)** is:

count 🡨 0

nodC = b.head

while nodC != b.firstEmpty do

nodC 🡨 b.bag[nodC].next

count 🡨 count +1

end-while

size 🡨 count

**end-subalgorithm**

**Complexity: Θ(1)**

**subalgorithm destroy(b)** is:

while b.head != b.firstEmpty do

@ create empty TElem e

b.bag[b.head].info = e

b.head 🡨 b.bag[b.head].next

end-while

**end-subalgorithm**

**Complexity: Θ(1)**

**Implementation of operations for DLLAIterator:**

**subalgorithm init(it, b)** is:

it.list 🡨 b.bag

it.currentElement 🡨 b.head

**end-subalgorithm**

**subalgorithm getCurrent(it, e)** is:

e 🡨 it.list[it.currentElement].info

**end-subalgorithm**

**subalgorithm next(it)** is:

it.currentElement = it.list[currentElement].next

**end-subalgorithm**

**function valid(it)** is:

if it.currentElement != it.list.firstEmpty then

valid 🡨 True

else

valid 🡨 False

end-if

**end-function**

**Tests**

Bag bag{};

Product p1{ 1, "car", "nice description" };

Product p2{ 2, "house", "another description" };

Product p3{ 3, "lorem", "ipsum" };

bag.add(p1);

assert(bag.search(p1) == true);

assert(bag.firstEmpty == 2);

assert(bag.head == 1);

assert(bag.search(p2) == false);

bag.add(p2);

assert(bag.tail == 2);

assert(bag.firstEmpty == 3);

bag.remove(p2);

assert(bag.firstEmpty == 2);

bag.remove(p1);

assert(bag.tail == -1);

assert(bag.head == -1);

bag.remove(p3);

assert(bag.firstEmpty == 1);

bag.add(p1);

bag.add(p2);

DLLAIterator it{ bag };

assert(it.getCurrent().getID() == p1.getID());

it.next();

assert(it.getCurrent().getID() == p2.getID());

it.next();

assert(it.valid() == false);

**Solution:**

@create Bag b

while true do

command: int

@printMenu

@input command

if command = 1 then

@input id, name, description

@ create Product p

bag.add(p)

else if command = 2 then

@input id, name, description

@create Product p

bag.remove(p)

else if command = 3 then

@input id, name, description

@create Product p

answer: bool

answer 🡨 bag.search(p)

@print answer

else if command = 4 then

@create DLLAIterator it on Bag b

while it.valid() do

@print it.getCurrent()

it.next()

end-while

end-if

end-while