Documentation

Data Structures and Algorithms

★ Requirement

29. ADT Set – implementation on a doubly linked list on an array.

★ Definitions

A Set is a container in which the elements are unique, and their order is not important (they do not have positions).

A doubly linked list on array is a data structure that simulates a dynamical doubly linked list with the following: an array in which we will store the elements, the capacity of the array, two indexes to tell where the head and the tail of the list are and an index to tell where the first empty position in the array is. The links to next and previous element can be stored using two arrays, or storing for each element his next and previous(DLLANode: info:TElem, next,prev:Integer).

❖ Problem statement

The eccentric Zepherin Xirdal will organize a charity party in order to raise funds for the poor children in Agadez, Niger. He is an art passionate and he will put up for auction his collection of paintings. His famous friends that are invited can buy paintings or they can also bring their own to be sold. Zephy has only original pieces, so he has to check if the painting brought by one of the guests is not fake. This way if the painting is in his collection already means that he had received a fake and it should not be added. Because the decision of choosing a painting may be difficult, he allows his friends to change their choice, but only if is the last painting that was sold. This way the buyer can choose another piece and return the one bought first. Help Zepherin by writting an aplication which allows him to manage the collection of paintings: adding a paint when a friend want to donate, removing a paint when is sold, searching for a paint in order to ensure it is original, updating the price of a painting in an auction, reinserting a painting if the last buyer changes his mind and keeping track of the earnings.

❖ Motivations

The problem is suitable to exemplify the main qualities of a Set, highlighting the need of different elements(masterpieces are unique). A painting will be uniquelly identifiable by its name. We will simply make use of the operations provided by this ADT to add, remove and update the data. The order of the paintings in the collection is not important(they don't have positions),we only care of them to be original, before adding them to the auction collection, so the Set provides us a great way of storing them.

❖ ADT Specification and Interface

a)ADT Set

Domain

S={s | s is a set with elements of the type TElem}

Operations - ADT Set Interface

- **init(s)**

  Descr: creates a new empty set

  Pre: true

  Post: s ∈ S, s is an empty set

- **add(s,e)**

  Descr: adds a new elemnt into the set

  Pre: s ∈ S, e ∈ TElem

  Post: s'∈ S, s'=s U {e} (e is added only if it is not in s yet. If s contains the element e already, no change is made.)

- **remove(s,e)**

  Descr: removes an element from the set

  Pre: s ∈ S, e ∈ TElem

  Post: s'∈ S, s'=s \ {e} (if e is not in s, no change is made)

- **find(s,e)**

  Descr: verifies if an element is in the set

  Pre: s ∈ S, e ∈ TElem

  Post: find =true, if e ∈ S

  =false, otherwise

- **size(s)**

Descr: returns the number of elements from the set

Pre: s ∈ S

Post: size= the number of elements from s

- **iterator(s,it)**

Descr: returns an iterator for the set s

Pre: s ∈ S

Post: it ∈ I, is an iterator over the set s

- **destroy(s)**

Descr: destroys a set

Pre: s ∈ S

Post: the set s was destroyed

b) ADT Iterator

Domain

I = {i | i is an iterator over a set s}

Operations – ADT Iterator Interface

- **createIterator(it,s)**

Descr: returns an iterator over the set s

Pre: s ∈ S

Post: it ∈ I, it is an iterator over s

- **valid(it)**

Descr: checks if an iterator is valid or not

Pre: it ∈ I,

Post: valid= true, if the current element from it is a valid one

= false, otherwise

- **getCurrent(it,e)**

Descr: returns the current element from s

Pre: it ∈ I, valid(it)

Post: e ∈ TElem, e is the current element from s

- **next(it)**

Descr: it sets the iterator to the next element from s

Pre: it ∈ I, valid(it)

Post: it' ∈ I, the current element from it' is the next element from it

❖ Representation of the container and iterator

Set:	DLLANode:	*Iterator:*
elems:DLLANode[]	info:TElem	s:Set
cap: Integer	prev:Integer	current: Integer
head: Integer	next:Integer	
tail: Integer		
firstEmpty: Integer		

❖ Representation of other structures

Nod:	LinkedList:	Painting
data:Integer	head: ↑Nod	name: String
next: ↑Nod		author:String
		year: Integer
		price: Double

★ Implementation

❖ Operations ADT Set

SubAlgorithm init(s) is: **Θ(1)**

```
        s.size ←0
        s.capacity ←cap   {cap is a number we give , the capacity of the set}
        s.head ← -1
        s.tail ← -1
        s.free.add(-1)
        For ( i ←s.capacity-1, -1, -1) execute
            s.free.add(i)
        EndFor
        s.firstEmpty ← getData(getHead(s.free))
        @allocate an array of capacity elements of type Node in arraynodes
        s.elems ←arraynodes
EndSubAlgorithm
```

SubAlgorithm add(s,e) is: **O(n)**

```
  If (find(s,e)=false) then
        If (s.firstEmpty != -1) then
              s.size←s.size+1
              If (s.head = -1) then
                 createNode(n,e,-1,-1)
                 s.elems[s.firstEmpty] ← n
                 s.head ←s.firstEmpty
                 s.tail ←s.firstEmpty
              Else
                 createNode(n,e,s.tail,-1)
                 s.elems[s.firstEmpty]← n
                 SetNext(s.elems[s.tail], s.firstEmpty)
                 s.tail ←s.firstEmpty
              EndIf
              s.free.remove(s.firstEmpty)
              s.firstEmpty  ←getData(getHead(s.free))
        EndIf
  EndIf
EndSubAlgorithm
```

SubAlgorithm remove(s,e) is: **O(n)**

```
curent ← s.head
While ((curent ! = -1) and (getInfo(s.elems[curent])!=e))  execute
   curent ←getNext(s.elems[curent])
EndWhile
If (curent != -1) then
   s.size ←s.size-1
   If (curent =s.head) then
      nou ←getNext(s.elems[curent])
      setPrev(s.elems[nou], -1)
      s.head←nou
   Else
   If (curent = s.tail) then
      nou ←getPrev(s.elems[curent])
      setNext(s.elems[nou], -1)
      s.tail ←nou
   Else
      Inainte←getPrev(s.elems[curent])
      dupa← getNext(s.elems[curent])
      setNext(s.elems[inainte],dupa)
      setPrev(s.elems[dupa], inainte)
   EndIf
   EndIf
   s.free.add(curent)
   s.firstEmpty ←getData(getHead(s.free))
 EndIf
EndSubAlgorithm
```

Function find(s,e) is: **O(n)**

```
      found ← false
      curent ← s.head
      While ((curent != -1) and  (found = false)) execute
         If (getInfo(s.elems[curent].getInfo) = e) then
            found ←true
         Else
            curent ←getNext(s.elems[curent])
        EndIf
      EndWhile
      find←found
EndFunction
```

Function size(s) is: **Θ(1)**

```
    size←s.size
EndFunction
```

SubAlgorithm Iterator(s,it) is: **Θ(1)**

```
    createIterator(it,s)
EndSubAlgorithm
```

SubAlgorithm destroy(s) is : **Θ(1)**

```
    @deallocate the set s
EndSubAlgorithm
```

❖ Operations ADT Iterator

SubAlgorithm createIterator(it,s) is: **Θ(1)**

```
    it.s←s
    it.current← getHead(s)
EndSubAlgorithm
```

Function valid(it) is: **Θ(1)**

```
    If (s.current = -1)
       valid←false
    Else
       valid←true
    EndIf
EndFunction
```

Function getCurrent(it,e) is: **Θ(1)**

```
    all←getAllElems(it.set)
    GetCurrent←getInfo(all[it.current])
EndFunction
```

SubAlgorithm next(it) is: **Θ(1)**

```
    all←getAllElems(it.set)
    it.current← getNext(all[it.current])
```

EndSubAlgorithm

❖ Operations Node

SubAlgorithm createNode(n,info, prev, next) is: **Θ(1)**

n.info← info
n.next ← next
n.prev ← prev
EndSubAlgorithm

SubAlgorithm setNext(n,next) is: **Θ(1)**

n.next← next
EndSubAlgorithm

SubAlgorithm setPrev(n,prev) is: **Θ(1)**

n.prev← prev
EndSubAlgorithm

SubAlgorithm setInfo(n,info) is: **Θ(1)**

n.info← info
EndSubAlgorithm

Function getNext(e) is : **Θ(1)**

getNext←e.next
EndFunction

Function getPrev(e) is : **Θ(1)**

getPrev←e.prev
EndFunction

Function getInfo(e) is : **Θ(1)**

getInfo←e.info
EndFunction

- **Operations Painting**

SubAlgorithm createPaint(p,name,author,year,price) is: **Θ(1)**

p.name ← name
p.author ← author
p.year ← year
p.price ← price
EndSubAlgorithm

Function getName(p) is : **Θ(1)**

getName← p.name
EndFunction

Function getAuthor(p) is : **Θ(1)**

getAuthor← p.author
EndFunction

Function getYear(p) is : **Θ(1)**

getYear← p.year
EndFunction

Function getPrice(p) is : **Θ(1)**

getPrice← p.price
EndFunction

Function toString(p) is : **Θ(1)**

@returns a string with all the fields of a Painting, one afther the other parated by space
EndFunction

- **Operations Linked List**

SubAlgorithm init(sll) is: **Θ(1)**

sll.head ← NIL
EndSubAlgorithm

SubAlgorithm add(sll,data) is: **Θ(1)**

```
    createNod(nod,data)
    setNext(node, sll.head)
    sll.head ← node
EndSubAlgorithm
```

SubAlgorithm remove(sll,data) is: **O(n)**

```
    curent ← sll.head
     prev = ← NIL
    While ((curent != NIL) and (getData(curent) != data)) execute
        prev ← curent
        curent ← getNext(curent)
     EndWhile
     If (prev = NIL) then
        If (sll.head != NIL) then
            Sll.head ← getNext(sll.head)
        EndIf
    Else
       If (curent != NIL) then
        setNext(prev,curent.getNext())
        setNext(curent, NIL)
    EndIf
EndSubAlgorithm
```

Function getHead(sll) is : **Θ(1)**

```
    getHead← sll.head
EndFunction
```

- **Operations Nod**

SubAlgorithm createNod(sll,data) is: **Θ(1)**

```
    @allocate a new element of type Nod
EndSubAlgorithm
```

Function getData(e) is : **Θ(1)**

```
    getData←e.data
EndFunction
```

Function getNext(e) is : **Θ(1)**

```
    getNext←e.next
EndFunction
```

SubAlgorithm setData(n,data) is: **Θ(1)**

```
    n.data← data
EndSubAlgorithm
```

SubAlgorithm setNext(n,next) is: **Θ(1)**

```
    n.next← next
EndSubAlgorithm
```

- **Ui Operations**

SubAlgorithm createUi(s,total,last) is: **Θ(1)**

```
    ui.st←s
    ui.total←total
    ui.last←last
EndSubAlgorithm
```

SubAlgorithm printMenu() is: **Θ(1)**

```
    @print the list with available options
EndSubAlgorithm
```

SubAlgorithm addPaint() is: **O(n)**

```
    @read name, author, year and price
    createPaint(p,name, author, year, price)
     add(ui.s,p)
    EndSubAlgorithm
```

SubAlgorithm removePaint() is: **O(n)**

```
    @read name
    p ← findObject(name)
   If (getName(p)!="") then
      ui.last ← p
      ui.total ← ui.total + getPrice(p)
      remove(ui.s,p)
   EndIf
   Else
        @print"The painting was already sold!"
EndSubAlgorithm
```

Function findObject(name) is: **O(n)**

```
    iterator(ui.s,it)
    While (valid(it)=true) execute
        If (getName(getCurrent(it)) =name)
            findObject ←getCurrent(it)
        EndIf
        next(it)
    EndWhile
    @if the object was not found return empty Painting
EndFunction
```

SubAlgorithm printEarnings() is: **Θ(1)**

```
    @print the value of ui.total
EndSubAlgorithm
```

SubAlgorithm updatePrice() is: **O(n)**

```
    @read name and price
    p← findObject(name)
    If (getName(p) != "") then
        CreatePaint( name, getAuthor(p), getYear(p), price)
        remove(ui.s, p)
        add(ui.s, nou)
    EndIf
    Else
        @print "The painting was already sold!"
EndSubAlgorithm
```

SubAlgorithm reinsert() is: **O(n)**

```
    @read name of the painting
    If (name = getName(ui.last) then
        @read name of the painting to change with in newname
        paint←indObject(newname)
        If (paint.getName()!="")
            ui.total ←ui.total - getPrice(ui.last)
            add(ui.s, ui.last)
            ui.last ← paint
            ui.total ← ui.total + getPrice(paint)
            remove(ui.s, paint)
            @print "The change was successfull!"
```

```
            Else
                @print "The painting was already sold!"
            EndIf
        Else
            @print "This is not the last sold painting!"
    EndIf
EndSubAlgorithm
```

SubAlgorithm display() is: **O(n)**

```
        iterator(ui.s,it)
        While (valid(it)) execute
            @print toString(getCurrent(it))
            next(it)
        EndWhile
EndSubAlgorithm
```

SubAlgorithm run() is:

```
        @print the menu
        @read the option
        @call the subalgorithms described above
EndSubAlgorithm
```

★ **Tests for the ADT**

```
Painting p1 = Painting{ "Mona Lisa","Leonardo daVinci",1756,800 };
assert(p1.getName() == "Mona Lisa");
assert(p1.getAuthor() == "Leonardo daVinci");
assert(p1.getYear()==1756);
assert(p1.getPrice()==800);
assert(p1.toString() == "Mona Lisa Leonardo daVinci 1756 800 mill");
Painting p2= Painting{ "Venus Birth","Sandro Boticelli",1600,800};
Painting p3 = Painting{ "The Kiss","Gustav Klimt",1600,800 };
LinkedList* list = new LinkedList();
list->add(1);
```

```
list->add(2);
list->add(3);
list->remove(3);
list->remove(1);
assert(list->getHead()->getData()==2);
delete list;
Node n{ p1,-1,-1 };
assert(n.getInfo().getName()=="Mona Lisa");
assert(n.getNext()==-1);
n.setInfo(p2);
assert(n.getInfo().getName() == "Venus Birth");
Set lista;
lista.add(p1);
assert(lista.getSize() == 1);
lista.add(p2);
assert(lista.find(p2) == true);
assert(lista.getSize() == 2);
lista.add(p3);
assert(lista.getSize()==3);
Iterator it = lista.iterator();
assert(it.getCurrent().getName() == "Mona Lisa");
it.next();
assert(it.valid()==true);
assert(it.getCurrent().getName() == "Venus Birth");
lista.remove(p2);
assert(lista.getSize() == 2);
```

```
lista.remove(p3);
assert(lista.getSize() == 1);
assert(lista.find(p2) == false);
assert(lista.find(p3) == false);
Iterator It2 = lista.iterator();
assert(It2.getCurrent().getName() ==p1.getName());
assert(It2.valid() == true);
lista.remove(p1);
assert(lista.getSize() == 0);
Iterator It = lista.iterator();
assert(It.valid() == false);
```