Crișan Alexandra Bianca
Group 912

*Data Structures and Algorithms – Project*

**Statement of the problem & Why is this ADT suitable?**

An Art Gallery has to store a list of artists and the number of pieces they created, in an alphabetical order. Clients must be able to browse through the Portfolio in whatever direction they like (they can go forward or back). A ***Map*** is a suitable option for solving this problem mainly because we can store pairs of ArtistName and nrOfPieces, the name being the key, and the number of pieces the associated value. In addition, making the map a ***sorted*** one solves the order problem. Lastly, the ***doubly linked list*** allows us to browse through the list with the help of the next/previous operations and the iterator, just like flipping the pages of a real Portfolio.

**ADT Sorted Map**

**I. Specification**

A Map is a container with <key, value> pairs. If it is a Sorted Map, we also have a relation(order) on the set of possible keys.

Domain:

M={m | m is a map with elements e = (k, v), where k ∈ TComp and v ∈ TValue}

**II. Interface**

init(sm, r)
“””
creates a new empty sorted map
pre: r∈R is a relation
post: sm∈M, sm is an empty sorted map
“””

destroy(sm)
“””
destroys a sorted map
pre: sm∈M
post: sm was destroyed
“””

add(sm, k, v)
“””
adds a new key-value pair to the sorted map
pre: sm∈M, k∈TComp, v∈TValue
post: (k ,v) is added to the sorted map sm
“””

Crișan Alexandra Bianca
Group 912

delete(sm, k, v)
“””
deletes a pair with a given key from the sorted map
pre: sm∈M, k∈TComp
post: v∈TValue

where v is v’ , if k was found, or 0 otherwise
“””

search(sm, k, v)
“””
searches for the value associated with the given key in the sorted map
pre: sm∈M, k∈TComp
post: v∈TValue

where v is v’ , if k was found, or 0 otherwise
“””

iterator(sm, it)
“””
returns an iterator for a sorted map
pre: sm∈M
post: it∈I is an iterator over sm
“””

size(sm)
“””
returns the number of pairs from the sorted map
pre: sm∈M
post: size <- the number of pairs from the sorted map
“””

keys(sm, s)
“””
returns the sorted set of keys from the sorted map
pre: sm∈M
post: s∈S is the set of keys from sm
“””

values(sm, b)
“””
returns a bag with all the values from the sorted map
pre: sm∈M
post: b∈B is the bag of all values from sm
“””

Crișan Alexandra Bianca
Group 912

pairs(sm, s)
""
returns the sorted set of all the pairs from the sorted map
pre: sm∈M
post: s∈S is the set of all the pairs from sm
""

## III. Representation

We implement the Sorted Map over a doubly linked list with dynamic allocation.

| *Pair* | *Node* | *SM* |
|---|---|---|
| key : String | info: Pair | head: ↑ Node |
| value: Integer | next: ↑ Node | tail: ↑ Node |
| | prev: ↑ Node | r: Relation |

## IV. Implementation

- ***subalgorithm init***(sm, r) Complexity: O(1)

```
        sm.head <- NIL
        sm.tail <- NIL
        sm.r <- r
end subalgorithm
```

- ***subalgorithm destroy***(sm) Complexity: O(n)

```
        while sm.head != NIL do
                aux <- sm.head
                sm.head <- [sm.head].next
                free(aux)
        end-while
end subalgorithm
```

- ***function search***(sm, key) Complexity: BC: O(1), WC: O(n), AC: O(n)

```
        currentNode <- sm.head
        found <- false
        while currentNode != NIL and found == false
                if [currentNode].info.key == key
                        found <- true
                else
                        currentNode <- [currentNode].next
                end-if
        end-while
        if found == true
                search <- currentNode
        else
                search <- NIL
        end-if
end function
```

Crişan Alexandra Bianca
Group 912

- ***subalgorithm add***(sm, key, value)     Complexity: BC: O(1), WC: O(n), AC: O(n)

```
ok <- 0
find <- search(sm, key)
currentNode <- sm.head

if currentNode == NIL
//the list is empty
        [toAdd].info <- Pair(key, value)
        [toAdd].next <- NIL
        [toAdd].prev <- NIL
        sm.head <- toAdd
        sm.tail <- toAdd
end-if

if sm.r([currentNode].info.key, key) == 0
//insert it first
        [toAdd].info <- Pair(key, value)
        [toAdd].next <- currentNode
        [curentNode].prev <- toAdd
        [toAdd].prev <- NIL
        sm.head <- toAdd
        ok <- 1
end-if

while currentNode != NIL and ok == 0
//the list is not empy

        if [currentNode].next != NIL
        //it has more than one element
                if sm.r([[currentNode].next].info.key, key) == 0
                        and sm.r([currentNode].info.key, key) == 1)
                        [toAdd].info <- Pair(key, value)
                        [toAdd].next <- [currentNode].next
                        [curentNode].next <- toAdd
                        [[toAdd].next].prev<- toAdd
                        [toAdd].prev <- currentNode
                        ok <- 1
                        currentNode <- [currentNode].next
                end-if
        else
        //it has only one element
                if sm.r([currentNode].info.key, key) == 1
                //insert it after the currentNode
                        [toAdd].info <- Pair(key, value)
                        [toAdd].next <- NIL
                        [curentNode].next <- toAdd
                        [toAdd].prev <- currentNode
```

Crişan Alexandra Bianca
Group 912

```
                    sm.tail <- toAdd
                    ok <- 1
                    currentNode <- [currentNode].next
                else
                //insert it before the currentNode
                    [toAdd].info <- Pair(key, value)
                    [toAdd].next <- currentNode
                    [curentNode].prev <- toAdd
                    [toAdd].prev <- NIL
                    sm.head <- toAdd
                    ok <- 1
                    currentNode <- [currentNode].next
                end-if
            end-if
            currentNode <- [currentNode].next
        end-while
    end subalgorithm
```

- ***subalgorithm delete***(sm, key)  Complexity: BC: O(1) WC: O(n), AC: O(n)

```
        ok <- 0
        find <- search(sm, key)
        currentNode <- sm.head

        if currentNode == find
        //the first element is the one we want
            if [currentNode].next == NIL
            //it's the only element in the list
                sm.head <- NIL
                sm.tail <- NIL
                delete find
            else
            //there are more elements
                [[find].next].prev <- NIL
                sm.head <- [find].next
                delete find
            end-if
        end-if

        while currentNode != NIL
            if currentNode == find
                if [currentNode].next == NIL
                //the searched element is the last one
                    [[find].prev].next <- NIL
                    sm.tail <- [find].prev
                    delete find
                    currentNode = NIL
                else
                //there are more elements after it
```

```
                    aux <- [currentNode].next
                    [aux].prev <- [find].prev
                    [[find].prev].next <- aux
                    delete find
                    currentNode <- aux
                end-if
            else
                currentNode <- [currentNode].next
            end-if
        end-while
    end subalgorithm
```

- ***function size***(sm) Complexity: O(n)

```
        nr <- 0
        currentNode <- sm.head
        while currentNode != NIL
            nr <- nr + 1
            currentNode <- [currentNode].next
        end-while
        size <- nr
    end function
```

- ***function keys***(sm, s) Complexity: O(n)

```
        currentNode <- sm.head
        while currentNode != NIL
            add(s, [currentNode].info.key)
            curren tNode <- [currentNode].next
        end-while
        keys <- s
    end function
```

- ***function values***(sm, b) Complexity: O(n)

```
        currentNode <- sm.head
        while currentNode != NIL
            add(b, [currentNode].info.key)
            curren tNode <- [currentNode].next
        end-while
        values <- b
    end function
```

- ***function pairs***(sm, s) Complexity: O(n)

```
        currentNode <- sm.head
        while currentNode != NIL
            add(s, [currentNode].info.key)
            curren tNode <- [currentNode].next
        end-while
        pairs <- s
    end function
```

Crișan Alexandra Bianca
Group 912

- ***function iterator***(sm, it) Complexity: O(1)
  iterator <- it(sm)
  ***end function***

**<u>Iterator</u>**

1) **<u>Interface</u>**

Domain: I = {it | it is an iterator over a sorted map with elements of type Pair }

init(it, sm)
“””
creates a new iterator for a sorted map
pre: sm is a sorted map
post: it ∈ I and it points to the first element in sm if sm is not empty or it is not valid
“””

getCurrent(it, e)
“””
returns the current element from the iterator
pre: it ∈ I, it is valid
post: e is a Pair, the current element from it
“””

next(it)
“””
moves the current element from the sorted map to the next element or makes the iterator invalid if no elements are left
pre: it ∈ I, it is valid
post: the current element from it points to the next element from the sorted map
“””

previous(it)
“””
moves the current element from the sorted map to the previous element or makes the iterator invalid if no elements are left
pre: it ∈ I, it is valid
post: the current element from it points to the previous element from the sorted map
“””

valid(it)
“””
verifies if the iterator is valid
pre: it ∈ I
post: valid <- True if it points to a valid element in the sorted map, False otherwise
“””

Crișan Alexandra Bianca
Group 912

setFirst(it, sm)
"""
sets the iterator to the head of the sorted map
pre: it ∈ I, sm∈M
post: the iterator points to the head of the sm
"""

setLast(it, sm)
"""
sets the iterator to the tail of the sorted map
pre: it ∈ I, sm∈M
post: the iterator points to the tail of the sm
"""

## 2) Representation

*Iterator*
currentNode: ↑ Node
sm: SM

## 3) Implementation

- ***subalgorithm init***(it, sm) Complexity: O(1)
  it.sm <- sm
  it.currentNode <- sm.head
  ***end subalgorithm***

- ***subalgorithm next***(it) Complexity: O(1)
  it.currentNode <- [it.currentNode].next
  ***end subalgorithm***

- ***subalgorithm prev***(it) Complexity: O(1)
  it.currentNode <- [it.currentNode].prev
  ***end subalgorithm***

- ***function getCurrent***(it) Complexity: O(1)
  getCurrent <- it.currentNode
  ***end function***

- ***function valid***(it) Complexity: O(1)
  valid <- it.currentNode != NIL
  ***end function***

- ***subalgorithm setFirst***(it, sm) Complexity: O(1)
  it.currentNode <- sm.head
  ***end subalgorithm***

- ***subalgorithm setLast***(it, sm) Complexity: O(1)
  it.currentNode <- sm.tail
  ***end subalgorithm***

**<u>Solution of the problem</u>**

*Menu:*

1. Add an (artist, number of pieces) pair to the portfolio
2. See the whole portfolio
3. Delete an artist with a given name
4. Search for an artist with a given name
5. View the list from the start
6. View the list from the end
7. Next artist
8. Previous artist
0. Exit

“””
Method that prints a Node’s information (Pair – the artist’s name and the number of his pieces)
“””

- ***subalgorithm toString***(node) Complexity: O(1)
  print “Name: “
  print [Node].info.key
  print “Number of pieces: “
  print [Node].info.value
  ***end subalgorithm***

“””
Starts viewing the list from the beginning
Pre: it ∈ I, sm∈M
Post: the viewing starts and the first artist is being printed
“””

- ***subalgorithm first***(it, sm) Complexity: O(1)
  setFirst(it, sm)
  toString(getCurrent(it))
  ***end subalgorithm***

“””
Starts viewing the list from the end
Pre: it ∈ I, sm∈M
Post: the viewing starts and the last artist is being printed
“””

Crișan Alexandra Bianca
Group 912

- ***subalgorithm last***(it, sm) Complexity: O(1)
    setLast(it, sm)
    toString(getCurrent(it))
  ***end subalgorithm***

  “””
  Moves the iterator to the next element in the list and prints it
  Pre: it ∈ I, sm∈M
  Post: the iterator points to the next artist(which is printed)
  “””
- ***subalgorithm next***(it, sm) Complexity: O(1)
    next(it)
    if valid(it)
        toString(getCurrent(it))
  ***end subalgorithm***

  “””
  Moves the iterator to the previous element in the list and prints it
  Pre: it ∈ I, sm∈M
  Post: the iterator points to the previous artist (which is printed)
  “””

- ***subalgorithm prev***(it, sm) Complexity: O(1)
    previous(it)
    if valid(it)
        toString(getCurrent(it))
  ***end subalgorithm***

  “””
  Function that prints the Portfolio using the iterator
  “””
- ***subalgorithm printList***(it) Complexity: O(n)
    while valid(it)
        aux <- getCurrent(it)
        toString(aux)
        next(it)
  ***end subalgorithm***

  “””
  Starts the application and lets you choose an option from the menu
  “””
- ***subalgorithm start***(it, sm)
    printMenu() //prints the menu of the app
    initList() //initializes the Portfolio
    while option != 0
        print “Enter an option”
        read option
        if option == 1

```
                //Add an artist and the number of his pieces (pair)
                print "Name: "
                read name

                Print "Number of pieces: "
                read nr
                add(sm, name, nr)

            else if option == 2
                //See the portfolio
                printList(it)

            else if option == 3
                //Delete an artist
                print "Name: "
                read name

                delete(sm, name)

            else if option == 4
                //Search for an artist
                print "Name: "
                read name

                node <- search(sm, name)
                if node != NIL
                    toString(node)
                else print "No matches"
            else if option == 5
                //From the start
                first(it, sm)
            else if option == 6
                //From the end
                last(it, sm)
            else if option == 7
                //Next artist
                next(it, sm)
            else if option == 8
                //Previous artist
                prev(it, sm)
    end subalgorithm
```

**Tests**

```
class Tests
{
public:

  /*
```

Crișan Alexandra Bianca
Group 912

```
 Default constructor for Tests
 */
Tests(){}

/*
 Calls all the test functions
 */
void testAll()
{
   this->testSearch();
   this->testDelete();
   this->testAdd();
   this->testgetSize();
   this->testgetKeys();
   this->testgetValues();
   this->testgetPairs();
   this->testgetters();
}

/*
 Initializes the list for the tests
 Returns a DLL
 */
DLL initList()
{
   DLL l;
   l.add("Anne", 20);
   l.add("David", 1);
   l.add("Brianna", 10);
   return l;
}

/*
 Tests the search function from the DLL
 */
void testSearch()
{
   DLL l = this->initList();
   Node *n = l.search("David");
   assert(n->getInfo().getKey() == "David");
   assert(n->getInfo().getValue() == 1);

   Node *m = l.search("Lily");
   assert(m == NULL);
}

/*
 Tests the delete function from the DLL
 */
```

Crișan Alexandra Bianca
Group 912

```
void testDelete()
{
   DLL l = this->initList();
   //delete the last element
   l.del("Brianna");
   assert(l.getSize() == 2);
   //delete the first element
   l.del("Anne");
   assert(l.getSize() == 1);

   DLL s{};
   //delete the only element
   s.add("Ana", 2);
   s.del("Ana");
   assert(s.getSize() == 0);

   DLL c = this->initList();
   //delete an element from the middle
   c.del("David");
   assert(c.getSize() == 2);

   //try to delete something that doesn't exist
   try
   {
      c.del("Serena");
   }
   catch (Exception& e)
   {

   }
}

/*
 Tests the add function from the DLL
 */
void testAdd()
{
   DLL l{};
   //empty list
   l.add("Maria", 21);
   assert(l.getSize() == 1);

   DLL s{};
   s.add("Dave", 22);
   //add something before
   s.add("Andreea", 10);
   assert(s.getSize() == 2);
   assert(s.getHead()->getInfo().getKey() == "Andreea");
```

Crișan Alexandra Bianca
Group 912

```
    DLL a{};
    a.add("Alina", 3);
    //add something after
    a.add("Matt", 8);
    assert(a.getSize() == 2);
    assert(a.getTail()->getInfo().getKey() == "Matt");

    DLL b = this->initList();
    //add something in the middle
    b.add("Chris", 24);
    assert(b.getSize() == 4);
    assert(b.getKeys().at(2) == "Chris");
}

/*
 Tests the getSize function from the DLL
 */
void testgetSize()
{
    DLL l = this->initList();
    assert(l.getSize() == 3);
}

/*
 Tests the getKeys function from the DLL
 */
void testgetKeys()
{
    DLL l = this->initList();
    vector<string> s = l.getKeys();

    assert(s.at(0) == "Anne");

}

/*
 Tests the getValues function from the DLL
 */
void testgetValues()
{
    DLL l = this->initList();
    vector<int> v = l.getValues();

    assert(v.at(0) == 20);

}

/*
 Tests the getPairs function from the DLL
```

```
 */
void testgetPairs()
{
   DLL l = this->initList();
   vector<Pair> p = l.getPairs();

   assert(p.at(0).getKey() == "Anne");
   assert(p.at(0).getValue() == 20);

}

/*
 Tests the getHead and getTail functions from the DLL
 */
void testgetters()
{
   DLL l = this->initList();
   assert(l.getHead()->getInfo().getKey() == "Anne");
   assert(l.getTail()->getInfo().getKey() == "David");
}

/*
 Default destructor for Tests
 */
~Tests(){}

};
```