MultiMap

implementation on a hash table, collision resolution by separate chaining

- Project no. 1 -

``

Cremene Delia

Group 811

ADT MultiMap

A multimap is a container which stores <key, value> pairs. The keys are not unique, therefore a key can have multiple associated values. The order in which the pairs are kept is not important.

MultiMap Domain

MM = { mm | mm is a multimap with < k, v > pairs, where k ∈ TKey and v ∈ TValue }

MultiMap Interface

init(mm):
- descr: creates a new empty multimap
- pre: true
- post: mm ∈ MM, mm is an empty multimap

destroy(mm):
- descr: destroys a multimap
- pre: mm ∈ MM
- post: the multimap was destroyed

add(mm, k, v):
- descr: add a new key-value pair to the multimap
- pre: mm ∈ MM, k ∈ TKey, v ∈ TValue
- post: mm' ∈ MM, mm' = mm ∪ < k, v >

remove(mm, k, v):
- descr: removes a <key, value> pair from the multimap
- pre: mm ∈ MM, k ∈ TKey, v ∈ TValue
- post: $remove \leftarrow \begin{cases} true, & \text{if } < k, v > \in mm, mm' \in MM, mm' = mm - < k, v > \\ false, & \text{otherwise} \end{cases}$

search(mm, k, l):
- descr: returns a list with all the values associated to a key
- pre: mm ∈ MM, k ∈ Tkey
- post: l ∈ L, l is the list of values associated to the key k. If k is not in the multimap, l is an empty list

iterator(mm, it):
- descr: returns an iterator over the multimap
- pre: mm ∈ MM
- post: it ∈ I, it is an iterator over mm, the current element from it is the first pair from mm, or it is invalid if mm is empty

size(mm):
- descr: returns the number of pairs from the multimap
- pre: mm ∈ MM
- post: size ← the number of pairs from mm

isEmpty(m):
- descr: verifies if the map is empty
- pre: m ∈ M
- post: $isEmpty \leftarrow \begin{cases} true, & \text{if mm contains no pairs} \\ false, & \text{otherwise} \end{cases}$

Iterator Domain

I = { it | it is an iterator over a multimap }

Iterator Interface

init(it, mm):

descr: creates a new iterator for a multimap
pre: mm is a multimap
post: it ∈ I and it points to the first element in mm if mm is not empty or it is not valid

getCurrent(it):

descr: returns the current element from the iterator
pre: it ∈ I, it is valid
post: getCurrent ← <k, v>, k ∈ TKey, v ∈ TValue, <k, v> is the current pair from it
throws: an exception if it is invalid

next(it):

descr: moves the current element from the container to the next pair or makes the iterator invalid if no pairs are left
pre: it ∈ I, it is valid
post: the current element from it points to the next element from the multimap
throws: an exception if it is invalid

valid(it):

descr: verifies if the iterator is valid
pre: it ∈ I
post: $\text{valid} \leftarrow \begin{cases} \text{True}, & \text{if it points to a valid element from the container} \\ \text{False}, & \text{otherwise} \end{cases}$

MultiMap representation

Pair:

key: TKey
value: TValue

Node:

info: Pair
next: ↑ Node

Multimap:

table: ↑Node[ ]
m: Integer
h: TFunction

Iterator representation

Iterator:

mm: MultiMap
currentPos: Integer
currentNode: ↑ Node

Multimap implementation

```
subalgorithm init(mm) is:
    mm.m ← @initial capacity
    mm.table ← @allocate an array with mm.m positions
    mm.hf ← hf
    for i ← 0, mm.m execute
        mm.table[i] ← NIL
    end-for
end-subalgorithm
```

Complexity: Θ(m)

```
subalgorithm destroy(mm) is:
    for i ← 0, mm.m execute
        currentNode ←mm.table[i]
        while (currentNode != NIL) execute
            prevNode ← currentNode
            currentNode ← [currentNode].next
            free(prevNode)
        end-while
    mm.m ← 0
    free(mm.table)
    end-for
end-subalgorithm
```

Complexity: Θ(m)

```
subalgorithm add(mm, k, v) is:
    allocate (newNode)
    [newNode].info.key ← k
    [newNode]. info.value ← v
    [newNode].next ← mm.table[mm.hf(k, mm.m)]
    mm.table[mm.hf(k, mm.m)] ← newNode
end-subalgorithm
```

Complexity: Θ(1)

```
function remove(mm, k, v) is:
    currentNode ← mm.table[mm.hf(k, mm.m)]
    prevNode ← NIL

    if currentNode = NIL then
        remove ← false
    end-if
    found ← false
    while currentNode != NIL and found = false execute
        if [currentNode].info.key = k and [currentNode]. info.value = v then
            found ← true
        else
            prevNode ← currentNode
            currentNode ← [currentNode].next
        end-if
    end-while
```

```
        if found = false then
                remove ← false
        end-if
        if prevNode = NIL and [currentNode].next = NIL then
                mm.table[mm.hf(k, mm.m)] ← NIL
        else
                if prevNode = NIL and [currentNode].next != NIL then
                        mm.table[mm.hf(k, mm.m)] ← [currentNode].next
                else
                        [prevNode].next ← [currentNode].next
                end-if
        end-if
        remove ← true
end-function
```

Complexity: Θ (1 + α)

```
function search(mm, k, l) is:
        currentNode ← mm.table[mm.hf(k, mm.m)]
        i ← 0
        while currentNode != NIL execute
                if [currentNode].info.key = k then
                        l[i] ← [currentNode]. info.value
                        i++
                end-if
                currentNode ← [currentNode].next
        end-while
search ← l
end-function
```

Complexity: Θ (1 + α)

```
function size(mm) is:
        count ← 0
        for i ← 0, mm.m execute
                currentNode ←  mm.table[i]
                while currentNode != NIL execute
                        currentNode ← [currentNode].next
                        count = count + 1
                end-while
        end-for
        size ← count
end-function
```

Complexity: Θ (m + number of pairs in the table)

```
function isEmpty(mm) is:
        for i=0, mm.m execute
                if mm.table[i] != NIL then
                        isEmpty ← false
                end-if
        end-for
        isEmpty ← true
end-function
```

Complexity: Θ (m)

function iterator(mm, it) is:
    iterator ← init(it, mm)   //init from iterator init
end-function

Complexity: O(m)

Where α = n / m is the load factor of the table with m slots containing n elements, that is, the average number of elements stored in a chain. In case of separate chaining α can be less than, equal to, or greater than 1.

function hashFunction(s, m) is:
//descr: is a function that maps a key k to a slot in the table
//pre: s is a string, m is an integer representing the multimap's capacity
//post: returns an integer between 0 and m-1 representing the allocated slot for the string in the multimap's table
    pos ← 0
    for i ← 0, length(s) execute
        pos ← pos + int(s[i]) * 31 ^ (length(s) - i – 1)
    end-for
    if pos < 0 then
        pos ← -pos
    end-if
    hashFunction ← pos % m
end-function

Complexity: Θ (length of string)

Iterator implementation

```
subalgorithm init(it, mm) is:
    it.mm ← mm
    it.currentPos ← 0
    it.currentNode ← NIL
    while it.currentPos < it.mm.m and it.currentNode = NIL execute
        it.currentNode ← it.mm.table[currentPos]
        it.currentPos = it.currentPos + 1
    end-while
    if valid(it) then
        it.currentPos ← it.currentPos - 1
    else
        it.currentNode ← NIL
    end-if
end-subalgorithm
```

Complexity: O(m)

```
function getCurrent(it) is:
    if valid(it) then
        getCurrent ← [it.currentNode].info
    else
        @throw exception for invalid iterator
    end-if
end-function
```

Complexity: Θ (1)

```
subalgorithm next(it) is:
    it.currentNode ← [it.currentNode].next
    while it.currentPos < it.mm.m and it.currentNode = NIL execute
        it.currentPos ←  it.currentPos + 1
        it.currentNode ← it.mm.table[it.currentPos]
    end-while
    if valid(it) = false  then
        @throw exception for invalid iterator
    end-if
end-subalgorithm
```

Complexity: O (m)

```
function valid(it) is:
    if it.currentNode != NIL then
        valid ←  true
    valid ← false
    end-if
end-function
```

Complexity: Θ (1)

Problem statement

The local bookstore of a small town had become increasingly popular and the business is thriving. To be more efficient the bookstore now needs an app that stores their stock of books. Each book has an author and title. An author can write more than one book. The app should allow the staff to search by author, add a new book and delete an existing book really fast. Two or more occurrences of the same book (same author and title) will be kept separately as different entrances (also, when we delete a book only one occurrence will be deleted).

Justification

In this case the multimap is a great choice for a container because each book consists of a <key, value> pair where the key is the author of the book and the value represents its title. Furthermore, since an author can write more than one book and we can also have more than one book having the same author and title in stock the keys are not unique.

Implementation

main.cpp

```
subalgorithm populate(mm) is:
//descr: populates the multimap with some given pairs
//pre: mm is a multimap
//post: the multimap was populated with some given elements
        add(mm, author, title) //multiple times for different values
end-subalgorithm
                                                        Complexity: Θ (1)

subalgorithm printMenu() is:
//descr: prints the menu for the user interface
end-subalgorithm
                                                        Complexity: Θ (1)

function main() is:

        init (mm)                                       Complexity: Θ (m)
        populate(mm)                                    Complexity: Θ (1)

        while true execute:
                printMenu()
                read command

                if command = 0 then
                        @ exit console
                end-if

                else if command = 1 then
                        read author
                        read title
                        add(mm, author, title)
                end-if
                                                        Complexity: Θ (1)
```

```
        else if command = 2 then
            read author
            read title
            if remove(mm, author, title) = true
                print "The book was removed"
            else
                print "The book did not exist"
            end-if
        end-if
```

Complexity: Θ (1 + α)

```
        else if command = 3 then
            read author
            search(mm, author, books)
            for i ← 0, length(books) execute
                print books[i].title
            end-for
        end-if
```

Complexity: Θ (1 + α)

```
        else if command = 4 then
            @try
            imm ← iterator(imm, mm)
            while valid(imm) execute
                b ← getCurrent(imm)
                print b
                next(imm)
            end-while
            @catch iterator error
        end-if
```

Complexity: Θ (m + number of pairs in the table)

```
    end-while

    main ← 0
end-function
```

Where α = n / m is the load factor of the table with m slots containing n elements, that is, the average number of elements stored in a chain. In case of separate chaining α can be less than, equal to, or greater than 1.

Tests

Test.h

```
#pragma once

class Test
{
public:
	Test() {};
	~Test() {};

	void createTest();
	void addTest();
	void removeTest();
	void searchTest();
	void sizeTest();
	void isEmptyTest();
};
```

Test.cpp

```
#include "Test.h"
#include "MultiMap.h"
#include <assert.h>


void Test::createTest()
{
	MultiMap mm{  };
	Node* l[100];
	mm.getLibrary(l);
	for (int i = 0; i < mm.getCapacity(); i++)
		assert(l[i] == nullptr);
}

void Test::addTest()
{
		MultiMap mm{ };
		assert(mm.size() == 0);
		mm.add("F. Scott Fitzgerald", "The Great Gatsby");
		assert(mm.size() == 1);
		mm.add("Margaret Atwood", "The Handmaid's Tale");
		assert(mm.size() == 2);
		mm.add("Margaret Atwood", "The Handmaid's Tale");
		assert(mm.size() == 3);
		mm.add("Daniel Keyes", "Flowers For Algernon");
		mm.add("Amy Harmon", "From Sand and Ash");
		mm.add("Jane Austen", "Emma");
		mm.add("Daniel Keyes", "The Minds of Billy Milligan");
		mm.add("Amy Harmon", "From Sand and Ash");
```

```
            mm.add("Jane Austen", "Persuasion");
            mm.add("Charlotte Bronte", "Jane Eyre");
            assert(mm.size() == 10);
            mm.add("Daniel Keyes", "Flowers For Algernon");
            mm.add("Amy Harmon", "From Sand and Ash");
            mm.add("Jane Austen", "Emma");
            mm.add("Daniel Keyes", "The Minds of Billy Milligan");
            mm.add("Amy Harmon", "From Sand and Ash");
            mm.add("Jane Austen", "Persuasion");
            mm.add("Charlotte Bronte", "Jane Eyre");
            assert(mm.size() == 17);
}

void Test::removeTest()
{
      MultiMap mm{  };
      assert(mm.size() == 0);
      mm.add("F. Scott Fitzgerald", "The Great Gatsby");
      assert(mm.size() == 1);
      assert(mm.remove("F. Scott Fitzgerald", "The Great Gatsby")==true);
      assert(mm.size() == 0);
      assert(mm.remove("F. Scott Fitzgerald", "The Great Gatsby")==false);
      assert(mm.size() == 0);
      mm.add("Margaret Atwood", "The Handmaid's Tale");
      assert(mm.size() == 1);
      mm.add("Margaret Atwood", "The Handmaid's Tale");
      assert(mm.size() == 2);
      assert(mm.remove("F. Scott Fitzgerald", "The Great Gatsby")==false);
      assert(mm.size() == 2);
      assert(mm.remove("Margaret Atwood", "The Handmaid's Tale")==true);
      assert(mm.size() == 1);
      assert(mm.remove("Margaret Atwood", "The Handmaid's Tale")==true);
      assert(mm.size() == 0);
}

void Test::searchTest()
{
      MultiMap mm{ };
      assert(mm.size() == 0);
      mm.add("Daniel Keyes", "Flowers For Algernon");
      mm.add("Amy Harmon", "From Sand and Ash");
      mm.add("Jane Austen", "Emma");
      mm.add("Daniel Keyes", "The Minds of Billy Milligan");
      mm.add("Amy Harmon", "From Sand and Ash");
      mm.add("Jane Austen", "Persuasion");
      mm.add("Charlotte Bronte", "Jane Eyre");
      mm.add("Daniel Keyes", "Flowers For Algernon");
      mm.add("Amy Harmon", "From Sand and Ash");
      mm.add("Jane Austen", "Emma");
      mm.add("Daniel Keyes", "The Minds of Billy Milligan");
      mm.add("Amy Harmon", "From Sand and Ash");
      mm.add("Jane Austen", "Persuasion");
```

```
	mm.add("Charlotte Bronte", "Jane Eyre");
	mm.add("F. Scott Fitzgerald", "The Great Gatsby");
	mm.add("Jane Austen", "Persuasion");

	vector <string> v;
	mm.search("Kristin Hannah",v);
	assert(v.size() == 0);

	mm.search("Daniel Keyes",v);
	assert(v.size() == 4);

	mm.search("Charlotte Bronte",v);
	assert(v.size() == 2);

	mm.search("Jane Austen",v);
	assert(v.size() == 5);

	mm.search("F. Scott Fitzgerald",v);
	assert(v.size() == 1);
}

void Test::sizeTest()
{
	//contained in the others
}

void Test::isEmptyTest()
{
	MultiMap mm{  };
	assert(mm.size() == 0);
	assert(mm.isEmpty() == 1);
	mm.add("F. Scott Fitzgerald", "The Great Gatsby");
	assert(mm.size() == 1);
	assert(mm.isEmpty() == 0);
	mm.remove("F. Scott Fitzgerald", "The Great Gatsby");
	assert(mm.size() == 0);
	assert(mm.isEmpty() == 1);
	mm.add("Margaret Atwood", "The Handmaid's Tale");
	assert(mm.size() == 1);
	assert(mm.isEmpty() == 0);
}
```