Data structure and Algorithms

Project

Retegan Mihai-Gheorghe
Gr. 813
MIE1

Project topic: ADT MultiMap – implementation on a singly lined list on an array. Every node contains a key and a doubly linked list with dynamic allocation of all the values belonging to that key.

Problem statement and justification:

Problem statement: Implement an application which is a synonyms dictionary. For example: "beautiful" is synonym to "pretty", "handsome", "good-looking" "lovely". We also would like to be able to add words and synonyms to that words or delete certain words or synonyms and to be able to see all words.

Justification: the problem obviously requires to associate a word with synonyms, so we need a container that can hold key-values pair. Words are the keys and the synonyms are the values. A key can have multiple values.

ADT MultiMap – Domain and Interface

Domain:

MM = {mm|mm is a Multimap with <TKey, TValue> pairs; a Tkey can have multiple associated Tvalue}

Interface:

-init (mm):

pre: true

post: mm ∈ MM, mm is an empty multimap

-destroy(mm):

pre: mm ∈ MM

post: the multimap was destroyed

-add(mm, k, v):

pre: mm ∈ MM, k − TKey, v − TValue

post: mm' ∈ MM, mm' = mm∪ < k, v >

-remove(mm, k, v):

pre: mm ∈ MM, k − TKey, v – Tvalue

post:

$$\text{remove} \leftarrow \begin{cases} \text{true, if } < k, v > \in mm, mm' \in MM, mm' = mm - < k, v > \\ \text{false, otherwise} \end{cases}$$

-search(mm, k, l):

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 the empty list.

-iterator(mm, it):

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):

pre: mm ∈ MM

post: size ← the number of pairs from mm

ADT Iterator – Domain *and* Interface:

Domain:

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

Interface:

- init(it, mm) :

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) :
    pre: it∈ I, it is valid
    post: getCurrent ← e, e ∈MM, e is the current element from it
    throws: an exception if it is not valid

-next(it) :
    pre: it ∈ I, it is valid(it)
    post: it'∈ I, the current element from it' points to the next element from the multimap , or, if the current element from it was the last element from the multimap , it' is invalid
    throws: an exception if it is not valid

- valid(it) :
    pre: it I ∈

    post: { True, if the current element from it is valid
          { false, otherwise

ADT Representation:

VNode: (node for the doubly linked list)
    value: TValue
    next: ↑ VNode
    previous: ↑ VNode

DLL: (doubly linked list for values)
    tail: ↑ VNode
    head: ↑ VNode

TElem: (key and values list)
    key: TKey
    values: DLL

TElemITERATOR(key -value)
    Key:TKey
    Value:TValue

MultiMap: (container with an static array of TElem)

Elems: TElem[cap]
nextPost: Integer[cap]
cap: Integer // capacity of arrays
sizeKV: Integer // how many values are in total
headList: Integer // head of singley linked list
firstEmpty: Integer // first empty position on elems array

MultiMapIterator:

mm:MM
currentElement: Integer
currentNode: ↑VNode
counter : Integer // to count how many values that were “read” from MultiMap

Implementation in pseudocode (and complexity)

ADT MultiMap:

```
Subalgorithm init(mm) is:
    mm.cap ← 30
    mm.len←0
    mm.headList←-1
    mm.firstEmpty←0
    mm.sizeKV←0
    for i←0,29 execute
        mm.elems[i] ← {NILL, NILL}
        mm.nextPos[i] ←i+1
    end-for
    mm.nextPos[mm.cap] ←-1

end-subalgorithm
```

//Ω(cap) – cap is capacity of arrays

```
// auxiliary function
Subalgorithm findKeyPos(mm,k) is:
        current←mm.headList
        while current ≠ -1 and mm.elems[current].first ≠ k execute
                current ← mm.nextPos[current]
        end-while
        if current ≠ -1 then
                findKeyPos ← current
        else
                findKeyPos←-1
        end-if
end-subalgorithm

// O(cap) – cap is capacity of array

Subalgorithm add(mm, k, v) is:
        If mm.firstEmpty = -1 then
                @throw “full”
        Else
                Pos ← mm.findKeyPos(k)
                If pos = -1 then
                        If mm.isEmpty() then:
                                newPosition←mm.firstEmpty
                                mm.elems[newPosition].first←c
                                n = allocate()
                                [n].value←v
                                [n].next←NILL
                                [n].previous←NILL
                                mm.elems[newPosition].second.head←n
                                mm.elems[newPosition].second.tail←n
                                mm.firstEmpty←mm.nextPos[mm.firstEmpty]
                                mm.nextPos[newPosition] ←mm.headList
                                mm.headList←newPosition
                                mm.len←mm.len+1
                                mm.sizeKV←mm.sizeKV+1
                        else:
                                pozCurrent←1
                                nodCurrent←mm.headList
                                while nodCurrent ≠ -1 and pozCurrent<mm.firstEmpty-1 execute:
                                        pozCurrent←pozCurrent+1
                                        nodCurrent←mm.nextPos[nodCurrent]
                                end-while
                                if nodCurrent ≠ -1 then:
                                        newElem←mm.firstEmpty
```

```
                                        mm.firstEmpty←mm.nextPos[mm.firstEmpty]
                                        mm.elems[newElem].first←k
                                        n=allocate()
                                        [n].value←v;
                                        [n].next←NILL
                                        [n].previous←NILL
                                        mm.elems[newElem].second.head←n
                                        mm.elems[newEelm].second.tail←n
                                        mm.len←mm.len+1
                                        mm.nextPos[newElem] ←mm.nextPos[nodCurrent]
                                        mm.nextPos[nodCurrent] ← newElem
                                end-if
                                mm.sizeKV←mm.sizeKV+1
                        end-if
                else:
                        n←allocate()
                        [n].value←v
                        [n].previous←mm.elems[pos].second.tail
                        [mm.elems[pos].second.tail].next←n
                        mm.elems[pos].second.tail←n
                        [mm.elems[pos].second.tail].next←n
                        mm.sizeKV←mm.sizeKV+1
                end-if
end-subalgorithm

//WC=O(capacity) BC= Ω(1)


Subalgorithm remove(mm, k, v) is:
        pos←mm.findKeyPos(k)
        if pos = -1 then:
                remove←false
        else:
                if mm.elems[pos].second.tail=mm.elems[pos].second.head then:
                        nodC←mm.headList
                        prevNode←-1
                        while nodC ≠ -1 and mm.elems[nodC].first≠k execute:
                                prevNode←nodC
                                nodC←mm.nextPos[nodC]
                        end-while
                        if nodC=mm.headList then:
                                mm.elems[mm.headList].first←NILL
                                mm.headList← mm.nextPos[mm.headList]
                                mm.sizeKV←mm.sizeKV-1
```

```
                    else
                        mm.elems[nodC].first←NILL
                        mm.nextPos[prevNode] ←mm.nextPos[nodC]
                        mm.sizeKV←mm.sizeKV-1
                    end-if
                    mm.nextPos[nodC] ←mm.firstEmpty
                    mm.firstEmpty←nodC
                    mm.len←mm.len-1
                else
                    [currentNode] ← mm.elems[pos].second.head
                    While currentNode≠ NILL and [currentNode].value ≠ v execute:
                        currentNode ← [currentNode].next
                    end-while
                    [deletedNode]=currentNode
                    If currentNode ≠ NILL then:
                        If currentNode = mm.elems[pos].second.head then:
                            mm.elems[pos].second.head←mm.elems[pos].second.head.next
                            [mm.elems[pos].second.head].previous ← NILL
                            mm.sizeKV←mm.sizeKV-1
                            remove←true
                        else:
                            if currentNode = mm.elems[pos].second.tail then:
                                mm.elems[pos].second.tail←mm.elems[pos].second.tail.previous
                                mm.elems[pos].second.tail.next←NILL
                                mm.sizeKV←mm.sizeKV-1
                                remove←true
                            else:
                                [[currentNode].next].previous ← [[currentNode].previous
                                [[currentNode].previous].next ← [[currentNode].next
                                mm.sizeKV←mm.sizeKV-1
                                return true
                            end-if
                        end-if
                    else:
                        remove←false
                    end-if
                end-if
            end-if
end-subalgorithm
//WC=O(capacity) BC= Ω (1)
```

```
Subalgorithm search(mm, k) is:
        pos ← mm.findKeyPos(k)
        if pos = -1  then:
                search← @empty vector
        else
                current=allocate()
                current←mm.elems[pos].second.head
                values (values vector<TValue>)
                while current ≠ NILL execute:
                        values.push_back([current].value)
                        current← current.next
                end-while
                search←values
        end-if
end-subalgorithm
// BC= Ω(1)   WC=AC=O(capacity)


Subalgorithm size(mm) is:
        s←mm.sizeKV
end-subalgorithm
// Ω(1)

Subalgorithm isEmpty(mm) is:
        If mm.sizeKV=0 then:
                isEmpty←1
        else
                isEmpty←0
end-subalgorithm
// Ω(1)

Subalgorithm iterator(mm) is:
        iterator← MultiMapIterator([mm])
end-subalgorithm
// Ω(1)
```

MultiMapIterator:

```
Subalgorithm init(it,mm) is:
        It.currentElement←0
        It.currentNode←mm.elems[it.currentElement].second.head
        It.counter←0
End-subalgorithm
// Ω(1)

Subalgorithm getCurrent(it) is:
        k←mm.elems[it.currentElement].first
        v←[it.currentNode].value
        getCurrent← {k,v}
end-subalgorithm
// Ω(1)

Subalgorithm valid(it) is:
        If it.counter=mm.sizeKV then:
                valid←false
        else:
                valid←true
        end-if
end-subalgorithm
// Ω(1)

Subalgorithm next() is:
        If [it.currentNode].next=NILL then:
                It.current←mm.nextPos[it.currentElement]
                It.currentNode← mm.elems[it.currentElement].second.head
                It.counter←it.counter+1
        Else:
                It.currentNode←[it.currentNode].next
                It.counter←it.counter+1
        End-if
End-subalgorithm
// Ω(1)

Subalgorithm first(it) is:
        It.currentElement←mm.headList
        It.currentNode←mm.elems[it.currentElement].second.head
        It.counter←0
End-subalgorithm
// Ω(1)
```

Tests:

```
#include "Tests.h"
#include "MultiMap.h"
#include "MultiMapIterator.h"
#include <assert.h>
#include <vector>
#include<iostream>

void testAll() {
	MultiMap m;
	m.add(1, 100);
	m.add(2, 200);
	m.add(3, 300);
	m.add(1, 500);
	m.add(1, 600);
	m.add(1, 300);
	m.add(1, 200);
	m.add(2, 800);
	m.add(3, 500);
	m.add(2, 600);
	m.add(3, 700);
	m.add(4, 800);
	m.add(5, 900);
	m.add(4, 8000);

	assert(m.size() == 14);

	assert(m.remove(6, 600) == false);
	assert(m.remove(1, 500) == true);
	assert(m.remove(2, 200) == true);
	assert(m.remove(1, 5100) == false);
	assert(m.remove(3, 700) == true);
	assert(m.remove(4, 8000) == true);
	assert(m.remove(4, 800) == true);
	assert(m.remove(5, 900) == true);
	assert(m.remove(2, 800) == true);
	assert(m.remove(2, 600) == true);
	assert(m.size() == 6);
	m.add(1, 500);
	m.add(1, 700);
	m.add(2, 300);

	vector<TValue> v,v1;
	v=m.search(6);
	assert(v.size()==0);

	v=m.search(1);
	assert(v.size()==6);

	m.add(1, 500);
	m.add(1, 500);

	v = m.search(1);
	assert(v.size() == 8);
	v1 = { 100, 600, 300, 200, 500, 700, 500, 500 };
	assert(v == v1);
```

```
	m.add(4, 800);

	v = m.search(4);
	v1 = { 800 };

	assert(v.size() == 1);
	assert(v == v1);

	v = m.search(1);
	assert(v != v1);

	assert(m.isEmpty() == false);

	MultiMapIterator im = m.iterator();
	assert(im.valid() == true);
	while (im.valid()) {
		im.getCurrent();
		im.next();
	}
	assert(im.valid() == false);
	im.first();
	assert(im.valid() == true);
}
```