Diana Elisa Noveanu

group 915/2, project 5

Documentation

***Container***

SLLA MultiMap

A MultiMap is a container where the elements are pairs of keys and values, (<key,value>), where each key can have multiple values (unlike a regular map, in which case, one key can be associated to only one value). The ADT MultiMap has no positions, therefore the values can only be reached through making use of the keys.

Domain

MM = {mm | mm is a MultiMap with elements of type e = (k, v), where k ∈ TKey and v ∈ TValue}

Representation

*TElem*

key: Tkey (string)

value: Tvalue (string)

*Str*

len: Integer

s: String[]

*SLLNode*

info: TElem

next: Integer

*MultiMap*

multiMap: sll

*SLL*

elems: SLLNode[]

capacity: Integer

head: Integer

firstEmpty: Integer

*Iterator*

mm: MultiMap

current: Integer

Interface:

*TElem*

init(TElem)

destroy(TElem)

setKey(TElem, k)

setValue(TElem, v)

getKey(TElem)

getValue(TElem)

*SLLNode*

init(SLLNode)

destroy(SLLNode)

setInfo(SLLNode, TElem)

setNext(SLLNode, int)

getInfo(SLLNode)

getNext(SLLNode)

*SLL*

init(SLL)

destroy(SLL)

addFirst(SLL, TElem)

remove(SLL, TElem)

search(SLL, TElem)

getAll(SLL)

getHead(SLL)

seHead(SLL, int)

*Iterator*

init(Iterator)

destroy(Iterator)

isValid(Iterator)

next(Iterator)

getCurrent(Iterator)

*Str*

init(str)

destroy(str)

add(str, k)

*MultiMap*

init(MultiMap)

destroy(MultiMap)

add(MultiMap, k, v)

remove(MultiMap, k, v)

search(MultiMap, k)

iterator(MultiMap)

getSLL(MultiMap)

head(MultiMap)

Representation in Pseudocode

• TElem

***func init(t)***

*descr: creates a new TElem*

*pre: true*

*post: TElem* ∈ *TElem, key and value are empty strings*

alloc(key)

alloc(value)

init<-TElem(key, value)

***end-function***

Complexity: θ(1)

***func init(t, k, v)***

*descr: creates a new TElem*

*pre: true, k* ∈ *Tkey, v* ∈ *TValue*

*post: t* ∈ *TElem*

alloc(t)

t.key key

t.value value

initt

***end-function***

Complexity: θ(1)

***func destroy(TElem t)***

*descr: destroys TElem*

*pre: TElem* ∈ *TElem*

*post: TElem was destroyed*

***end-function***

Complexity: θ(1)

***func getKey(t)***

*descr: returns key of a TElem*

*pre: t* ∈ *TElem, key* ∈ *TKey*

*post: key*

getKeyt.key

***end-function***

Complexity: θ(1)

***func getValue(t)***

*descr: returns value of a TElem*

*pre: t* ∈ *TElem, value* ∈ *TValue(string)*

*post: value*

getValuet.value

***end-function***

Complexity: θ(1)

***func setKey(t, k)***

*descr: sets the key of a TElem*

*pre: t* ∈ *TElem, key* ∈ *TKey(string)*

*post: key of t is changed*

t.key k

***end-function***

Complexity: θ(1)

***func setValue(t, v)***

*descr: sets the value of a TElem*

*pre: t* ∈ *TElem, v* ∈ *TValue(string)*

*post: value of t is changed*

t.value value

***end-function***

Complexity: θ(1)

2. SLL Node

***func init(n)***

*descr: creates a new SLLNode*

*pre: true*

*post: n*∈ *SLLNode, t is an empty TElem, next is -1*

alloc(info)

info TElem()

alloc(next)

initSLL(info, next)

***end-function***

Complexity: θ(1)

***func init(n, t, nex)***

*descr: creates a new SLLNode*

*pre: true, t* ∈ *TElem, next* ∈ *Integer*

*post: n* ∈ *SLLNode*

alloc(n)

n.info t

n.next nex

initn

***end-function***

Complexity: θ(1)

***func destroy(n)***

*descr: destroys SLLNode*

*pre: n* ∈ *SLLNode*

*post: n was destroyed*

***end-function***

Complexity: θ(1)

***func getInfo(n)***

*descr: returns info of a SLLNode*

*pre: n* ∈ *SLLNode*

*post: info*

getInfo n.info

***end-function***

Complexity: θ(1)

***func getNext(n)***

*descr: returns next of a SLLNode*

*pre: n* ∈ *SLLNode*

*post: next*

getNextn.next

***end-function***

Complexity: θ(1)

***func getInfo(n, inf)***

*descr: sets info of a SLLNode*

*pre: n* ∈ *SLLNode, inf* ∈ TElem

*post: info of n changed*

n.info inf

***end-function***

Complexity: θ(1)

***func setNext(n, nex)***

*descr: sets next of SLLNode*

*pre: n* ∈ *SLLNode, next* ∈ Integer

*post: next of n changed*

n.next nex

***end-function***

Complexity: θ(1)

3. SLL

***func init(sll)***

*descr: creates a new sll*

*pre: true*

*post: sll* ∈ *SLL, sll is an empty SLL*

sll.head -1

sll.firstEmpty 0

init sll

***end-function***

Complexity: θ(1)

***func destroy(sll)***

*descr: destroys a SLL*

*pre: sll* ∈ *SLL*

*post: sll was destroyed*

***end-function***

Complexity: θ(1)

***func addFirst(sll, t)***

*descr: add a new TElem to the sll*

*pre: sll* ∈ *SLL, t* ∈ *TElem*

*post: sll’* ∈ *SLL, sll’ = sll* ∪ *t*

if sll.head = -1 then

sll.nodes.setInfo(t)

sll.headsll.firstEmpty

sll.firstEmpty sll.getFirstEmpty()

end-if

if nodes.firstEmpty = -1

@resize

end-if

sll.nodes[firstEmpty].setInfo(t)

sll.nodes[firstEmpty].setNext(sll.head)

sll.head sll.firstEmpty

sll.firstEmpty sll.getFirstEmpty()

***end-function***

Complexity: θ(1)

***func remove(sll, t)***

*descr: removes a TElem from sll*

*pre: sll* ∈ *SLL, t* ∈ *TElem*

*post: the pair was removed*

currentsll.head

prev -1

while current ≠ -1 and [sll.nodes[current]].info ≠ t execute

prev current

current [sll.nodes[current]].getNext()

end-while

if current ≠ -1 then

sll.head sll.nodes[current].getNext()

else

sll.nodes[prev].setNext(sll.nodes[current].getNext())

end-if

***end-function***

Complexity computations:

Best case: θ(1) the element we want to remove is on the first position

Worst case: θ(n) the element is on the last position in the multimap: ∑1 = n

Average case: θ(n) ∑1/(n+1) \* ∑i = ∑ 1/(n+1) \* n(n+1)/2 = n (chance of element e to be on a certain position)

***func search(sll, t)***

*descr: checks if an element is in the sll*

*pre: sll* ∈ *SLL, t* ∈ *TElem*

*post: true if it exists, false otherwise*

alloc(current)

current sll.getHead()

while current ≠ -1 and sll.nodes[current].getInfo() ≠ t execute

current sll.nodes[current].getNext()

end-while

if current ≠ -1 then

return true

else

return false

end-if

***end-function***

Complexity: θ(n)

***func getAll(SLL)***

*descr: returns all elems of a SLL*

*pre: sll* ∈ *SLL*

*post: all nodes*

getAll<-sll.nodes[]

***end-function***

Complexity: θ(1)

***func getHead(SLL)***

*descr: returns head of a SLL*

*pre: sll* ∈ *SLL*

*post: head*

getHead<-sll.head

***end-function***

Complexity: θ(1)

***func seHead(SLL, int)***

*descr: sets the head of a SLL*

*pre: sll* ∈ *SLL, int* ∈ Integet

*post: head was set*

sll.head <- int

***end-function***

Complexity: θ(1)

4. Iterator

SLLA MultiMap Iterator

An iterator is a structure that is used to iterate throughout the elements of a container, in this case, a MultiMap. An iterator usually contains: a reference to the container it iterates over and another reference to a current element from the container. Iterating through the elements of the container means actually moving said current element from one element to another until the iterator becomes invalid.

Domain

I = {it | it is an interator over a MultiMap with elements of type TElem}

***func init(it, mm)***

*descr: creates a new iterator for a container*

*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*

***alloc(it)***

it.current <- mm.getHead()

***end-function***

Complexity: θ(1)

***func destroy(it)***

*descr: destroys an iterator*

*pre: it is an Iterator*

*post: it was destroyed*

***end-function***

Complexity: θ(1)

***func getCurrent(it)***

*descr: returns the current element from the iterator*

*pre: it* ∈ *I, it is valid*

*post: e* ∈ *TElem, e is the current element from it (of form <key,value>)*

getCurrent <- it.getAll[it.current]

***end-function***

Complexity: θ(1)

***func next(it)***

*descr: moves the current element from the multimap 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 mm*

it.current <- [it.sll.getAll[it.current]].next

***end-function***

Complexity: θ(1)

***func valid(it)***

*description: verifies if the iterator is valid*

*pre: it* ∈ *I*

*post: valid ←*

***end-function***

Complexity: θ(1)

5. Str

***func init(str)***

*descr: initializes a str*

*pre: true*

*post: empty str*

init<-str

***end-function***

Complexity: θ(1)

***func destroy(str)***

*descr: destroys a str*

*pre: str is a STR post: str was destroyed*

***end-function***

Complexity: θ(1)

***func add(str, k)***

*descr: adds a value to str*

*pre: str is a Str, value is a string*

*post: value was added*

str.len = s.len + 1

str[len]<-k

***end-function***

Complexity: θ(1)

6. MultiMap

***func init(mm, sll)***

*descr: creates a new mm*

*pre: true*

*post: mm* ∈ *MM*

mm.sll sll

***end-function***

Complexity: θ(1)

***func destroy(mm)***

*descr: destroys a multiMap*

*pre: mm* ∈ *MM*

*post: mm was destroyed*

***end-function***

Complexity: θ(1)

***func iterator(mm)***

*descr: returns an iterator for a multimap*

*pre: mm* ∈ *MM*

*post: it* ∈ *I, it is an iterator over mm*

iteratorIterator(mm.sll)

***end-function***

Complexity: θ(1)

***func 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 >*

alloc(TElem t)

t.value v

t.keyk

[mm.sll].addFirst(t)

***end-function***

Complexity: θ(1)

***func remove(mm, k, v)***

*descr: removes a pair with a given key from the multimap*

*pre: mm* ∈ *MM, k* ∈ *Tkey, v* ∈ *TValue*

*post: the pair was removed*

pre -1

current [mm.sll].head

ititerator(mm)

while it.isValid() execute

if [[it.getCurrent()].info].value = v and [it.getCurrent().info].key = k then

if current = [mm.sl].head then

[mm.sll].setHead([it.geCurrent()].getNext())

else

[[mm.getAll()][prev]].setNext([it.geCurrent()].getNext())

end-if

end-if

prev current

current [it.getCurrent()].getNext()

it.next()

end-while

***end-function***

Complexity: θ(n)

***func search(mm, k)***

*descr: returns all values with a given key from the multimap*

*pre: mm* ∈ *MM, k* ∈ *TKey*

*post: v* ∈ *TValue[], all the values corresponding to k (empty, if there are none)*

alloc(str s)

ititerator(mm)

while it.isValid execute

if [[it.current].info].key = key then

s.add([[it.current].info].value)

end-if

it.next

end-while

searchs

***end-function***

Complexity: θ(n)

Container Tests

Tests::Tests()

{

this->run();

}

void Tests::run() {

testTElem();

testSLLNode();

testIterator();

testStr();

testSLL();

testMultiMap();

}

void Tests::testTElem() {

TElem t1{};

TElem t2{ "key","value" };

assert(t1.getKey() == "");

assert(t1.getValue() != "value");

assert(t2.getKey() == "key");

assert(t2.getKey() != "");

t1.setKey("another key");

t1.setValue("");

t2.setKey("");

t2.setValue("and another value");

assert(t1.getKey() == "another key");

assert(t1.getValue() == "");

assert(t2.getKey() != "key");

assert(t2.getValue() == "and another value");

}

void Tests::testSLLNode() {

SLLNode n{ TElem{"a","m"},9 };

SLLNode n2{};

assert(n.getNext() == 9);

assert(n2.getNext() == -1);

assert(n.getInfo().getValue() == "m");

assert(n.getInfo().getKey() != "m");

assert(n2.getInfo().getKey() == "");

n2.setNext(7);

n2.setInfo(TElem{ "r","s" });

assert(n2.getInfo().getKey() == "r");

assert(n2.getInfo().getValue() == "s");

assert(n2.getNext() == 7);

}

void Tests::testSLL() {

SLL ss{};

assert(ss.getHead() == -1);

ss.addFirst(TElem{ "f", "g" });

assert(ss.getHead() == 0);

assert(ss.getAll()[ss.getHead()].getInfo().getKey() != "");

assert(ss.getAll()[ss.getHead()].getInfo().getValue() == "g");

ss.addFirst(TElem("a", "b"));

assert(ss.getFirstEmpty() == 2);

assert(ss.getHead() == 1);

assert(ss.search(TElem("a", "b")) == true);

ss.remove(TElem("a", "b"));

assert(ss.search(TElem("a", "b")) == false);

ss.addFirst(TElem("a", "b"));

ss.remove(TElem("f", "g"));

ss.setHead(9);

assert(ss.getHead() == 9);

}

void Tests::testIterator() {

MultiMap m{};

m.add("noi", "voi");

m.add("ei", "noi");

m.add("noi", "boi");

m.add("lor", "heo");

m.add("noi", "eu");

Iterator i = m.iterator();

assert(i.getCurrent().getInfo().getKey() == "noi");

i.next();

assert(i.getCurrent().getInfo().getKey() == "lor");

i.next();

assert(i.getCurrent().getInfo().getKey() == "noi");

while (i.isValid() && i.getCurrent().getInfo().getKey() != "ei")

i.next();

assert(i.getCurrent().getInfo().getValue() == "noi");

}

void Tests::testStr() {

str s;

s.add("first");

assert(s.len == 1);

s.add("second");

assert(s.len == 2);

str s2{};

assert(s2.len == 0);

s2.add("third but first in second str");

assert(s2.s[s2.len - 1] == "third but first in second str");

}

void Tests::testMultiMap() {

MultiMap m{};

m.add("noi", "voi");

m.add("ei", "noi");

m.add("noi", "boi");

m.add("lor", "heo");

m.add("noi", "eu");

str s = m.search("noi");

assert(s.len == 3);

str s2 = m.search("ei");

assert(s2.len == 1);

str s3 = m.search("a");

assert(s3.len == 0);

m.remove("noi", "voi");

str s5 = m.search("noi");

assert(s5.len == 2);

assert(m.getSLL().getAll()[m.getSLL().getHead()].getInfo().getKey() == "noi");

m.remove("noi", "eu");

assert(m.getSLL().getAll()[m.getSLL().getHead()].getInfo().getKey() == "lor");

}

***Problem statement***

Tech Dictionary

As university freshmen students, the majority of us have learnt a large number of new words throughout this past year, although not all of us have gotten to know the same words.

As a solution to this “communication” problem, we decided it was time to implement our own TechDictionary, which will store any new/useful words that will prove (or have already proved) to be necessary in our academic career, and also to aid us in understanding each other and what our professors want from us.

Therefore, the application will allow any user to add new words, remove mistaken words, search for a word, and add/remove only definitions of a word already stored in the dictionary. Each word can have up to 10 definitions, that will all be shown when the word is looked up.

Choice

The MultiMap is the perfect ADT for creating such a dictionary because, having unique keys and multiple possible values associated to said keys will make easier having every key represented by a word, and the list of values with that key will contain the several (or none) definitions of each word (as in a regular dictionary).

Representation in Pseudocode of Application

• Main

***func main()***

mm alloc(MultiMap)

mm.populate()

ui alloc(UI)

ui.start()

***end-func***

Complexity: θ(1)

• UI

***func start(ui)***

*descr: starts the application*

@print menu

@read command

if command = 1 then

ui.commandOne()

else if command = 2 then

ui.commandTwo()

else if command = 3 then

ui.commandThree()

else if command = 4 then

ui.commandFour()

else

@throw error

***end-func***

Complexity: θ(1)

***func commandOne(ui)***

*descr: adds word to dictionary*

@read word

@read key

[ui.mm].add(key,word)

***end-func***

Complexity: θ(1)

***func commandTwo(ui)***

*descr: removes word (and all its definitions) from dictionary)*

@read word

alloc(str s)

s[ui.mm].search(word)

for i0, s.len do

ui.removeWord(word,s[i])

end-for

***end-func*** Complexity: θ(n)

***func commandThree(ui)***

*descr: removes/adds only a definition of a word*

@read word

alloc(str s)

s[ui.mm].search(word)

@printDefinitions of w

@read command

if command = 1 then

@read definiton

[ui.mm].add(word, definition)

else

@choose definition to delete

[ui.mm].remove(word, definition)

end-if

***end-func***

Complexity: θ(1)

***func commandFour(ui***)

*descr: search for a word*

@read word

alloc(str s)

s[ui.mm].search(word)

@printDefinitions of w

***end-func***

Complexity: θ(1)

***func removeWord(ui, key, value)***

*descr: removes a word from the multiMap*

[ui.mm].remove(key, value)

***end-func***

Complexity: θ(1)