Documentation

MultiMap on a Singly Linked list

ADT Specification:

A MultiMap is a container in which the elements are key value pairs and because it’s a multimap the key are not necessarily unique.

ADT Representation:

MMEntry:

* Key: Tkey
* Values: LinkedList

Node:

* Info: MMEntry/Tval (for the MMEntries list)
* Next: ↑Node

LinkedList:

* Head: ↑Node
* Tail: ↑Node

MultiMap:

* Entries: LinkedList

Iterator Representation:

Iterator:

* Current: Node↑(w/ Tval as Info)
* CurrentList: Node↑(w/ MMEntry)

Interface:

LinkedList:

init(list):Complexity:𝛉(1)

* descr: creates a new empty linked list
* pre: true
* post: list ϵ SLL, list is an empty linked list

list.head<-NILL

list.tail<-NILL

end-subalgorithm

destroy(list):Complexity:𝛉(1)

* descr: destroys a linked list
* pre: list ϵ SLL
* post: list is destroyed

temp<-list.head

While head!=NILL execute:

temp<-list.head

list.head<-list.head.next

destroy(temp)

end-while

insertFirst(list, elem):Complexity(Best:𝛉(1),Worst:𝛉(n),Average:O(n))

* descr: add a new element elem to the beginning of the list
* pre: list ϵ SLL, elem ϵ TElem
* post: list’ ϵ SLL, list’ = list ∪ {elem}, elem is the first element in the list

If head=NILL

list.head<-allocate()

setInfo(list.head,elem)

list.tail=list.head

Return

end-if

list.temp<-head;

list.head=allocate(elem);

setNext(list.head,temp)

end-subalgorithm

insertLast(list, elem):Complexity:𝛉(1)

* descr: add a new element elem to the end of the list
* pre: list ϵ SLL, elem ϵ TElem
* post: list’ ϵ SLL, list’ = list ∪ {elem}, elem is the last element in the list

If tail=NILL

head<-allocate()

setInfo(list.head,elem)

list.tail=list.head

Return

end-if

something<-allocate(elem);

setNext(list.tail,something)

end-subalgorithm

deleteElement(list, elem) is:Complexity:𝛉(1)

* descr: removes a given element from the list
* pre: list ϵ SLL, elem ϵ TElem
* post: list’ ϵ SLL, list’ = list / {el}

If [list.head]=elem

t<-list.head

list.head<-list.head.next

detroy(t)

Return

end-if

temp<-list.head

While temp.next !=NILL and not temp.next.info=elem execute:

temp<-temp.next;

end-while

If temp.next != NILL

t<-temp.next

temp.next=temp.next.next

destroy(t)

End-if

end-subalgorithm

search(list, elem):Complexity(Best:𝛉(1),Worst:𝛉(n),Average:O(n))

* descr: searches for the given element in the list
* pre: list ϵ SLL, elem ϵ TElem
* post: search ← true if elem is in the list, false otherwise

temp<-list.head;

While temp!=NILL execute:

if([temp]==elem)

search<-true

End-if

temp<-temp.next

End-while

End-function

getWI(list, elem):Complexity(Best:𝛉(1),Worst:𝛉(n),Average:O(n))

* descr: searches for the given element in the list
* pre: list ϵ SLL, elem ϵ TElem
* post: search ← true if elem is in the list, false otherwise

temp<-list.head;

While temp!=NILL execute:

if([temp]==elem)

getWI<-temp

End-if

temp<-temp.next

End-while

getWI<-NILL

end-function

getHead(list):Complexity:𝛉(1)

* descr: returns the head of the list
* pre: list ϵ SLL
* post: getHead ← head of the list

getHead<-list.head

end-function

getTail(list):Complexity:𝛉(1)

* descr: returns the tail of the list
* pre: list ϵ SLL
* post: getTail ← tail of the list

getTail<-list.tail

end-function

MultiMap:

init(mm):Complexity:𝛉(1)

* descr: creates a new empty multimap
* pre: true
* post: mm ϵ MM, mm is an empty multimap

enteries<-allocate

end-subalgorithm

destroy(mm):Complexity:𝛉(1)

* descr: destroys a multimap
* pre: mm ϵ MM
* post: mm is destroyed

enteries<-destroy

end-subalgorithm

add(mm, key, value):Cmplexity(Best:𝛉(1),Worst:𝛉(n/𝜶),Average:O(n/𝞪))

* descr: add a key-value pair to the multimap
* pre: mm ϵ MM, key ϵ TKey, value ϵ TVal
* post: mm’ ϵ MM, mm’ = mm ∪ < key, value > or is added to the list of values of the certain key

ne:MMEntry

ne<-allocate(key,value)

if mm.enteries.search([ne]) then:

temp<-getWI(enteries,value)

insertLast(temp.info.values,value)

else

insertLast(enteries,[ne])

end-subalgorithm

remove(mm, key, value) is:Complexity:𝛉(1)

* descr: removes a given pair <key, value> from the multimap
* pre: mm ϵ MM, key ϵ TKey, value ϵ TValue
* post: mm’ ϵ MM, mm’ = mm / <key,value> or is removed from the list of values the a certain key

m:MMEnrty

m<-allocate()

m.key=<-key

temp<-getWI(enteries,[m])

If search(temp.info.values,value) then

deletElement(temp.info.values,value)

end-subalgorithm

search(mm, key):Complexity(Best:𝛉(1),Worst:𝛉(n/𝞪),Average:O(n/𝞪))

* descr: searches for the values associated with a given key in the map
* pre: mm ϵ MM, key ϵ TKey
* post: search ← a list of elements of type TVal, containing the element stored in the map at the key key

m:MMEnrty

m<-allocate()

m.key=<-key

If search(enteries,[m]) then

search<-getWI(enteries,[m]).info.values

Else

search<-NILL

end-function

Iterator

init(it, mm):Complexity:𝛉(1)

* descr: creates an iterator over the given multimap
* pre: mmt ϵ MultiMap
* post: it ϵ Iterator, it is an iterator over the given list

it.currentList<-mm.enteries.head

it.current<-mm.enteries.head.info.values.head

end-subalgorithm

destroy(it):Complexity:𝛉(1)

* descr: destroys the given iterator
* pre: it ϵ Iterator
* post: it was destroyed

destroy(it.current)

destroy(it.currentList)

end-subalgorithm

getCurrent(it):Complexity:𝛉(1)

* descr: returns the current element in the list
* pre: it ϵ Iterator
* post: getCurrent ← the current element in the list, of type TElem

getCurrent<-current<-info

end-function

isValid(it):Complexity:𝛉(1)

* descr: checks if the iterator is valid
* pre: it ϵ Iterator
* post: isValid ← true if the iterator is still valid, false otherwise

isValid<-(currentList=NILL)

end-function

next(it):Complexity:𝛉(1)

* descr: moves the iterator to the next position in the list, for the multimap the iterator first goes though the elements of a list stored in the MMEntries list.
* pre: it ϵ Iterator
* post: it’ ϵ Iterator, it’ is positioned on the next element in the list

If current.next !=NILL then:

current<-current.next

Else

currentList=currentList.next

If currentList!=NILL

current=currentList.info.values.head

end-if

End-if

end-subalgorithm

Chosen problem:

A data “encryption” utility which separates files into multiple files. The program should be able to reconstruct the files as well as display the separations,separate and reunite files.

This problem suits the multimap as a base ADT really well because the data is separated into multiple parts for one file so the parts should be stored together for each file like in a multimap, but because the software doesn’t just rename the files it has to store multiple things for each.

Tests:

Node<int> n1, n2(1), n3(&n2), n4(4, &n3), n5(5, &n4);

std::cout << n1;

assert(n4 != n3);

n4++;

assert(n4 == n3);

++n4;

assert(n4 == n2);

assert(n5 > n2);

assert(n2 < n5);

assert(n5 >= n2);

assert(n2 <= n5); assert(n5 == 5);

n5.setInfo(4);

n5.setNext(&n2);

n5 = n4;

assert(n5 == n4);

SLL<int> sll, sll2;

sll.insertFirst(1);

assert(sll.head->info == 1);

sll.insertLast(2);

assert(sll.tail->info == 2);

sll2.insertLast(2);

assert(sll2.tail->info == 2);

sll2.insertFirst(1);

assert(sll2.head->info == 1);

assert(sll.getWI(2) != NULL);

sll.removeElement(2);

assert(sll.search(1));

sll.removeElement(1);

assert(!sll.search(2));

assert(sll2.search(2));

assert(sll2.getHead() == sll2.head);

assert(sll2.getTail() == sll2.tail);

MMEntry mme1, mm2("a", "2");

mme1 = mm2;

assert(mm2 == mme1);

mm2.key = "0";

assert(mm2 < mme1);

assert(mme1 > mm2);

MultiMap mmap;

mmap.add("1", "a");

mmap.add("1", "b");

assert(mmap.search("1") != NULL);

mmap.remove("1", "a");

assert(mmap.search("1") != NULL);

mmap.remove("1");

assert(mmap.search("1") == NULL);