Preethika Kiruveedula

Computer Science 32

Discussion 1A

204912895

Project 2 Report

1.

Tail Pointer

Nullptr

Prev Pointer

Next Pointer

Node 2

Head Pointer

Nullptr

Next Pointer

Prev Pointer

Node 1

Value

Key

Value

Key

If Empty:

Nullptr

Head Pointer

Tail Pointer

I created a linear doubly linked list. There is a head pointer pointing to the first node and a tail pointer pointing to the last node. Each node has a KeyType value and a ValueType value along with a next pointer and a previous pointer. The next pointer points to the next node and the prev pointer points to the previous node. Every time a node is inserted into the linked list it is inserted to the top of the linked list. It is not a circular linked list and does not have a dummy node. The nodes are in the order of newest to oldest.

2. Pseudocode:

* Default Constructor:

set map\_size = 0

set head = nullptr

set tail = nullptr

* Copy Constructor:

Initialize head, tail, and map size

Repeatedly go through passed in map parameter

Copy its into newly map

* Assignment Operator:

If parameter is equal to current map

Return

Create temp map with value of parameter

repeatedly

Swap each node of two maps

* Destructor:

Repeatedly for every node

Set temp node to node after the one you want to delete

Delete node

Traverse to next node

* Bool insert()

If it already contains key

Return false

Allocate an new node, initialize its values

If empty change values of head and tail

Else add node to the top or front of the list

Return true

* Bool update()

Initialize new node to head

Traverse through map

If current node key is equal to the parameter

Change parameter value to node value

Return true

Return false

* Bool insertOrUpdate():

If map contains the passed in key

Update and return true

Else

Insert into map and return true

* Bool erase()

If list is empty

Return false

If key is not in the map

Return false

If there is only one node

Delete the node, decrement the size, make head point to nullptr

Return true

If deleting head node

Change where the prev pointer of the next node and the head pointer point

Delete node

Return true

If deleting tail node

Change the prev pointer to one before tail node and change the tail pointer

Delete Node

Return true

If deleting any other node

Set the next pointer to node before it to one after the target node

Set prev pointer of the one after the target node to the one before it

Delete node

Return false

* Bool contains()

Traverse through list

If key is equal to parameter

Return true

Return false

* Bool get()

Traverse through list

If key of the node matches parameter

Make passed in value equal to this node’s value

Return true

Return false

* Bool get()

If i is less then 0 or greater than size

Return false

Initialize int variable

Traverse through list

If int == i

Change key and value to those of the node

Return true

Increment int

Return false

* Void swap

Initialize local variables for head, tail, and size

Swap head and tail pointers

Swap size

* Bool combine()

Assign result to the value of m1

Obtains sizes of m2 and result

Initialize a bool value to false

Repeatedly

Get values from result aka m1 and check against values from m2

If they have the same key and value erase from result

If a key from m2 isn’t found in result insert it into result

* Void Subtract()

Assign result to m1 (to get rid of any previous values in

Initialize a count variable to the count of m2

Repeatedly

Using two local variables

Repeatedly get the key/value for each node

Repeatedly go through m2

If a key in m2 is in m1

Erase this key in the result map

Break out of loop

3. Test Cases:

//Default constructor

Map original;

//test empty

assert(original.empty());

//test nothing to erase

assert(!original.erase("blank"));

//test inserting

original.insert("A", 1.23);

original.insert("B", 2.34);

original.insert("C", 3.45);

//test size

assert(original.size() == 3);

//Setting values and keys

KeyType key = "B";

ValueType value = 9.87;

KeyType key1 = "D";

ValueType value1 = 2.36;

KeyType key2 = "A";

ValueType value2 = 5.67;

KeyType key3 = "B";

ValueType value4 = 5.89;

ValueType value3;

KeyType k;

ValueType val;

//testing get

assert(original.get(0, k, val) && key == "C" && val == 3.45);

//testing invalid i in get

assert(!(original.get(-1, k, val)&& k == "A" && val == 1.23));

assert(original.contains("A") && original.contains("B") && !original.contains("D"));

//testing update

assert(original.update(key, value1) && original.get(key, value3) && value3 == 9.87);

//testing invalid insert

assert(!original.insert(key3, value4) && original.get(key3, value3) && value3 == 9.87);

//testing insertOrUpdate

assert(original.insertOrUpdate(key2, value2) && original.get(key2, value3) && value3 == 5.67);

assert(original.size() == 3);

assert(original.insertOrUpdate(key1, value1) && original.get(key1, value3) && value3 == 2.36);

assert(original.size() == 4);

//testing erase

assert(original.erase("C") && original.size() == 3);

//testing erase/invalid get

assert(!original.get("C", value3));

assert(value3 == 2.36);

//Testing swap

Map test1;

test1.insert("Z", 3.21);

test1.insert("Y", 5.43);

original.swap(test1); //testing swap

assert(original.contains("Z") && original.contains("Y"));

assert(test1.contains("A") && test1.size() == 3);

//Testing assignment operator and copy constructor

ValueType value4;

Map test2(test1); //testing copy constructor

assert(test2.contains("A") && test2.contains("B") && test2.get("B", value4) && value4 == 9.87 && test2.size() == 3);

Map test3;

test3.insert("K", 1.23);

test3 = test1; //testing assignment operator

assert(test3.contains("D") && test3.contains("B") && test3.get("D", value4) && value4 == 2.36 && test3.size() == 3);

assert(test3.erase("A") && test3.size() == 2); //testing erasing head

test3 = test2;

assert(test3.erase("B") && !test3.contains("B")); //testing erasing middle

test3 = test1;

assert(test3.erase("D") && test3.size() == 2); //testing erasing tail

//testing combine and subtract functions

Map m1;

assert(m1.empty());

assert(m1.insert("Ashley", 456));

assert(m1.insert("Kevin", 789));

assert(m1.insert("Michael", 789));

assert(m1.insert("Preethi", 123));

Map m2;

assert(m2.empty());

assert(m2.insert("Lauren", 458));

assert(m2.insert("Preethi", 123));

assert(m2.insert("Claire", 321));

assert(m2.insert("Steph", 567));

Map m3;

assert(m3.empty());

assert(m3.insert("Preethi", 123));

assert(m3.insert("Lauren", 475));

assert(m3.insert("Bruce", 586));

assert(m3.insert("Makenna", 734));

Map m4;

assert(m4.empty());

assert(m4.insert("Nicole", 729));

assert(m4.insert("Bruce", 586));

assert(m4.insert("John", 621));

Map result;

assert(result.combine(m1, m2, result));

assert(result.contains("Kevin"));

Map difference;

assert(difference.subtract(m1, m3, difference));

assert(!difference.contains("Preethi"));

Map honey;

assert(subtract(m2, m3, honey));

Map gold;

assert(combine(m3, m4, gold));

cout << "Passed all Test Cases!" << endl;