Vivianne Dinh

Nachenberg

Project 2 Report

**Doubly Linked List Implementation**

My doubly-linked list includes nodes that have a next and prev pointer which point to the next and previous nodes respectively. The nodes also have a key and value. The list has a head pointer that initially points to nullptr when the list is empty and the list is linear (not circular). The last node in the list is found when the next pointer points to nullptr.

**Pseudocode**

**copy constructor**

create empty map

for every node in other map,

insert node into new map

**assignment operator**

delete all nodes of left map to make an empty map

for every node in right map,

copy right node into left map

return the left hand map (object)

**erase**

traverse map until key is found

if key is in first node

make head pointer point to next node

if key is in last node

make previous node’s next pointer point to nullptr

if key is in a middle node

make previous node’s next pointer point to next node

make next node’s prev pointer point to previous node

delete node with key given

decrement size

**reassign**

create a copy of map m

if theres only one node

swap copy with result and return

for all nodes

update value with the next value

for last node

update value with first node

sway copy with result

**combine**

create a copy of the first map

for every node in second map

check if key in first map

if key is in first map

check value

if value is different

delete node in copy map

in this case, function will return false

if key is not in first map

insert node into copy

swap copy with result

**Test Cases**

// test insert and constructor

Map m1;

m1.insert("ben", 58);

m1.insert("vanessa", 50);

m1.insert("krystal", 24);

m1.insert("austin", 20);

m1.insert("vivianne", 18);

m1.insert("matthew", 15);

m1.dump();

// test copy constructor

Map m2(m1);

m2.dump();

// test assignment operator

Map m3;

m3.insert("julia", 20);

m3.insert("tyler", 16);

m3 = m1;

m3.dump();

// test empty function

**if** (m3.empty())

cout << "map is empty" << endl;

**else**

cout << "map is not empty" << endl;

// test size function

Map m4;

m3.dump();

cout << "map has " << m3.size() << " elements" << endl; // not empty map

cout << "map has " << m3.size() << " elements" << endl; // empty map, returns 0

// test update function

m1.update("vivianne", 19);

m1.update("trevor", 20); // key not in the map

m1.dump();

// test insertOrUpdate function

m1.insertOrUpdate("christopher", 27); // insert

m1.insertOrUpdate("vivianne", 20); // update

m1.dump();

// test erase function

m1.erase("austin");

m1.erase("hello"); // erase a key that isnt in the map

m1.dump();

// test contains function

**if** (m1.contains("vivianne")) // key is in map

cout << "map contains the key" << endl;

**else**

cout << "map does not contain the key" << endl;

**if** (m1.contains("austin")) // key is not in map

cout << "map contains the key" << endl;

**else**

cout << "map does not contain the key" << endl;

// test first get function

**double** val;

**bool** nm = m1.get("vivianne", val);

cout << "the value is " << val << endl;

nm = m1.get("henry", val); // key not in the map

// test second get function

string ns;

nm = m1.get(8, ns, val); // i is not in the map (over size())

cout << "key: " << ns << " / value: " << val << endl;

nm = m1.get(4, ns, val);

cout << "key: " << ns << " / value: " << val << endl;

// test swap function

m1.dump();

cout << endl;

m2.dump();

cout << endl;

m1.swap(m2);

m1.dump();

cout << endl;

m2.dump();

// test reassign

m1.dump();

reassign(m1, m3);

m3.dump();

// test combine

combine(m1, m2, m3);

m3.dump();