Jason Tieu

304047667

Project 2 Report

For this project, we were required to use a doubly-linked list. Mine was very standard. In order to navigate through the list, I would make a node pointer p that points to head. It would move to the next element as long as it is not pointing to null. One I insert or erase a node, I would make sure to set the next and previous pointers to their new nodes.

**Pseudocode:**

bool Map::insert(const KeyType& key, const ValueType& value)

{

if map already contains key

return false

repeatedly:

move to the last node in the list

if at last node (list is not empty)

make new node

set node's key and value to arguments

add new node to end of list

increment size

return true

if list is empty

make a new node

set new node's key and value to arguments

assign head to new node

increment size

return true

}

bool Map::erase(const KeyType& key)

{

if map does not contain key

return false

repeatedly:

go through map and find the key specified

if list is not empty

head points to node /w key's next node

if node /w key's prev is not null

make prev node's next the one after current node

if node /w key's next is not null

make next node's prev the node before current node

delete current node

decrease size of map

return true

}

void Map::swap(Map& other)

{

make new node ptr

set new node ptr to m\_head

set m\_head to other m\_head

set other m\_head to ptr

}

bool combine (const Map& m1, const Map& m2, Map& result)

{

repeatedly:

get key and value from m1

if m2 doesn't contain key from m1

insert key and value to result

otherwise

get key's value from m2

if m2 value equals m1 value

insert key and value to result

otherwise

do not insert and mark as false

repeatedly:

get key and value from m2

if m1 doesn't contain key from m2

insert key and value to result

otherwise

get key's value from m1

if m1 value equals m2 value

insert key and value to result

otherwise

do not insert and mark as false

}

void subtract(const Map& m1, const Map& m2, Map& result)

{

repeatedly:

get key and value from m1

if m2 doesn't contain key from m1

insert key and value into result

}

**Test cases:**

for test cases, I also used a temporary member function:

void Map::print() const

{

for (Node\* p = m\_head; p != NULL; p = p->m\_next)

{

if (p != NULL)

cout << p->m\_key << " " << p->m\_value << endl;

else

cout << "p points to NULL" << endl;

}

}

int main()

{

Map m;

assert(m.empty()); //check map is empty

assert(m.size() == 0); //check size is 0

assert(!m.erase("Ricky")); //check cannot erase

m.insert("Jason", 100);

m.print();

cout << endl;

assert(m.size() == 1); // check size is 1

assert(m.update("Jason", 500)); //check update Jason

m.print();

cout << endl;

assert(m.insertOrUpdate("Bob", 2020)); //check insert Bob

m.print();

cout << endl;

assert(m.size() == 2); // check size is 2

assert(m.insertOrUpdate("Jason", 1)); //check update Jason

assert(m.size() == 2); // check size is 2

m.print();

cout << endl;

//get test

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

KeyType key;

ValueType value;

assert(m.get("Jason", value)); //check get value from Jason

cout << "Jason's value = " << value << endl;

assert(m.get("Bob", value)); //check get value from Bob

cout << "Bob's value = " << value << endl;

value = 0;

key = "Bob";

int i = 0;

m.get(i, key, value);

cout << i << " element is " << value << endl;

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

assert(m.erase("Jason")); //check erase Jason

assert(m.size() == 1); //check size 1 from 2

cout << "Jason deleted." << endl;

assert(m.erase("Bob")); //check erase Bob

cout << "Bob deleted." << endl;

assert(m.empty());//check map is empty

cout << "m is empty." << endl;

assert(m.size() == 0);//check size is 0

cout << "m\_size = 0" << endl;

m.print();

cout << "Print was empty" << endl;

cout << endl;

assert(!m.contains("Jason"));//check doesn't contain Jason

assert(!m.contains("Bob"));//check doesn't contain Bob

//swap test

Map m1, m2;

assert(m1.insert("Jason", 1));//check insert Jason

assert(m1.insert("Al", 2));//check insert Al

assert(m1.insert("Bob", 3));//check insert Bob

cout << "m1 elements: " << endl;

m1.print();

cout << endl;

assert(m2.insert("Car", 4));//check insert Car

assert(m2.insert("Dog", 5));//check insert Dog

assert(m2.insert("Ear", 6));//check insert Ear

assert(m2.insert("Fart", 7));//check insert Fart

assert(m2.insert("God", 8));//check insert God

cout << "m2 elements: " << endl;

m2.print();

cout << endl;

m1.swap(m2);

cout << "m1 elements: " << endl;

m1.print();

cout << endl;

cout << "m2 elements: " << endl;

m2.print();

cout << endl;

//copy constructor test

Map m3(m2);//copying a non-empty map

m3.print();

Map m4;

Map m5(m4); //copying an empty map

m5.print();

cout << "m5 is empty" << endl;

cout << endl;

cout << "equal operator overloading test" << endl;

Map m100 = m1;//equal operator

m100.print();

m1.print();

cout << endl << endl << endl;

//subtract test

Map m6;

assert(m6.insert("Fred", 123));//check insert Fred

assert(m6.insert("Ethel", 456)); //check insert Ethel

assert(m6.insert("Lucy", 789));//Check insert Lucy

cout << "m6 content: " << endl;

m6.print();

cout << endl;

Map m7;

assert(m7.insert("Lucy", 654));//check insert Lucy

assert(m7.insert("Ricky", 321));//check insert Ricky

assert(m7.insert("Ethel", 654));//check insert Ethel

cout << "m7 content: " << endl;

m7.print();

cout << endl;

Map result;

subtract(m6, m7, result);//check subtract

result.print();

//combine test

Map m8;

assert(m8.insert("Fred", 123));//check insert Fred

assert(m8.insert("Ethel", 456));//check insert Ethel

assert(m8.insert("Lucy", 789));//check insert Lucy

cout << "m8 content: " << endl;

m8.print();

cout << endl;

Map m9;

assert(m9.insert("Lucy", 789));//check insert Lucy

assert(m9.insert("Ricky", 321));//check insert Ricky

//assert(m9.insert("Ethel", 654)); //check insert Ethel

cout << "m9 content: " << endl;

m9.print();

cout << endl;

Map result2;

bool check = combine(m8, m9, result2);//check combine

cout << "result content" << endl;

result2.print();

cout << check << endl; //check bool for combine

return 0;

}