Daniel Levi-Minzi

1/28/20

Project 2

004923946

I implemented a circular doubly linked list with a dummy node.

[] -><- [] -><- [] -><- []

\ /

\ /

[]

Note that the / and \ are both pointed both ways. The bottom node is the

dummy node. An empty map is just the dummy node.

Erase:

check if key is in map

if the size is greater one

if its in there take its next and prev and set them to point to

each other

else there must be one node

set the dummy to null

delete the node

Reassign:

erase everything in the result

then assign each node’s value to the next node’s value

if its last node, set it to the first node’s value

Here are all of the test cases:

// note that functions like combine and reassign should be tested with

// many different number of nodes like none, one, etc.

#include "Map.h"

#include <iostream>

#include <cassert>

using namespace std;

int main () {

Map x;

x.insert("lala", 15);

cout << "false: " << x.empty() << endl;

cout << "size = 1: " << x.size() << endl;

cout << "contains = " << x.contains("lala") << endl;

cout << "!contains = " << x.contains("RAH") << endl;

cout << "update? " << x.update("lala", 20) << endl;

KeyType k;

ValueType v1;

x.get(0, k, v1);

cout << k << " " << v1 << endl;

Map l;

cout << "true: " << l.empty() << endl;

Map La;

cout << "should be 1: " << La.empty() << endl;

cout << "should be 0: " << La.size() << endl;

La.insert("Laura", 10);

cout << "should be 1: " << La.size() << endl;

cout << "should be 0: " << La.empty() << endl;

cout << "should be 0: " << La.insert("Laura", 15) << endl;

Map m;

m.insert("A", 10);

m.insert("B", 44);

m.insert("C", 10);

string all;

double total = 0;

for (int n = 0; n < m.size(); n++) {

string k;

double v;

m.get(n, k, v);

all += k;

total += v;

}

cout << all << total << endl;

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

assert(m.contains("A"));

assert(!m.update("Z", 123));

assert(m.update("A", 12));

string all2;

double total2 = 0;

for (int n = 0; n < m.size(); n++) {

string k;

double v;

m.get(n, k, v);

all2 += k;

total2 += v;

}

cout << all2 << total2 << endl;

Map g;

assert(g.empty());

ValueType v = -1234.5;

assert( !g.get("abc", v) && v == -1234.5); // v unchanged by get failure

g.insert("xyz", 9876.5);

cout << "should be 1: " << g.insertOrUpdate("xyz", 100) << endl;

double l1;

g.get("xyz", l1);

cout << "should be 100: " << l1 << endl;

assert(g.insertOrUpdate("lala", 5));

assert(g.size() == 2);

assert(g.insertOrUpdate("lala", 5));

assert(g.insertOrUpdate("lala", 6));

Map gpas;

gpas.insert("Fred", 2.956);

assert(!gpas.contains(""));

gpas.insert("Ethel", 3.538);

gpas.insert("", 4.000);

gpas.insert("Lucy", 2.956);

assert(gpas.contains(""));

gpas.erase("Fred");

assert(gpas.size() == 3 && gpas.contains("Lucy") && gpas.contains("Ethel") &&

gpas.contains(""));

assert(!gpas.contains("Fred"));

for (int n = 0; n < gpas.size(); n++) {

std::string k2;

double v2;

gpas.get(n, k2, v2);

std::cout << "gpas: " << k2 << " " << v2 << std::endl;

}

Map gpas2;

gpas2.insert("Fred", 2.956);

gpas2.insert("Ethel", 3.538);

double q;

std::string ke1;

assert(gpas2.get(1,ke1,q) && (ke1 == "Fred" || ke1 == "Ethel"));

std::string ke2;

assert(gpas2.get(1,ke2,q) && ke2 == ke1);

Map m1;

m1.insert("m1a", 1);

m1.insert("m1b", 1);

Map m2;

m2.insert("m2a", 2);

m2.insert("m2b", 2);

m2.insert("m2c", 2);

m2.insert("m2d", 2);

std::cout << "m2 size: " << m2.size() << "\n" << "m1 size: " << m1.size() << std::endl;

std::cout << "SWAP" << std::endl;

m1.swap(m2);

std::cout << "m2 size: " << m2.size() << "\n" << "m1 size: " << m1.size() << std::endl;

for (int n = 0; n < m1.size(); n++) {

std::string ke3;

double ve3;

m1.get(n, ke3, ve3);

std::cout << "M1: " << ke3 << " " << ve3 << std::endl;

}

for (int n = 0; n < m2.size(); n++) {

std::string kde;

double vde;

m2.get(n, kde, vde);

std::cout << "M2: " << kde << " " << vde << std::endl;

}

assert(m1.size() == 4 && m1.contains("m2a") && m1.contains("m2b") &&

m2.size() == 2 && m2.contains("m1a"));

std::cout << "UNSWAP" << std::endl;

m2.swap(m1);

std::cout << "m2 size: " << m2.size() << "\n" << "m1 size: " << m1.size() << std::endl;

for (int n = 0; n < m1.size(); n++) {

std::string kde1;

double vde1;

m1.get(n, kde1, vde1);

std::cout << "M1: " << kde1 << " " << vde1 << std::endl;

}

for (int n = 0; n < m2.size(); n++) {

std::string kde3;

double vde3;

m2.get(n, kde3, vde3);

std::cout << "M2: " << kde3 << " " << vde3 << std::endl;

}

assert(m2.size() == 4 && m2.contains("m2a") && m2.contains("m2b") &&

m1.size() == 2 && m1.contains("m1a"));

m1 = m2;

cout << m1.get(0, k, v) << k << endl;

assert(m1.size() == 4 && m1.contains("m2a") && m1.contains("m2b"));

Map az(m2);

assert(az.size() == 4 && az.contains("m2a") && az.contains("m2b") && az.contains("m2c") && az.contains("m2d"));

Map TC1;

TC1.insert("Fred", 123);

TC1.insert("Ethel", 456);

TC1.insert("Lucy", 789);

Map TC2;

TC2.insert("Lucy", 789);

TC2.insert("Ricky", 321);

Map TC\_COMB;

TC\_COMB.insert("THIS", 12090);

cout << combine(TC1, TC2, TC\_COMB) << endl;

for (int n = 0; n < TC\_COMB.size(); n++) {

std::string kde3;

double vde3;

TC\_COMB.get(n, kde3, vde3);

std::cout << "TC\_COMB: " << kde3 << " " << vde3 << std::endl;

}

Map TC3;

TC3.insert("Lucy", 653);

TC3.insert("Ricky", 321);

cout << combine(TC1, TC3, TC\_COMB) << endl;

for (int n = 0; n < TC\_COMB.size(); n++) {

std::string kde3;

double vde3;

TC\_COMB.get(n, kde3, vde3);

std::cout << "TC\_COMB: " << kde3 << " " << vde3 << std::endl;

}

// reassign test //

Map REASS;

REASS.insert("Lola", 5);

for (int n = 0; n < REASS.size(); n++) {

std::string kde3;

double vde3;

REASS.get(n, kde3, vde3);

std::cout << "REASS: " << kde3 << " " << vde3 << std::endl;

}

Map REASSRESULT;

reassign(REASS, REASSRESULT);

for (int n = 0; n < REASSRESULT.size(); n++) {

std::string kde3;

double vde3;

REASSRESULT.get(n, kde3, vde3);

std::cout << "REASSRESULT: " << kde3 << " " << vde3 << std::endl;

}

}