Shaan Mathur

Smallberg

CS32

April 19, 2016

Project 2 Report

My doubly-linked list was implemented without a dummy node and was not circular. It was a standard doubly linked list as described in class: it had a head Node pointer and a tail Node pointer, and Nodes for each element in the list. The head pointer points at the first Node, and the tail pointer points at the last Node. If there are no Nodes, the head and tail pointers are nullptrs. Each is indexed from 0 to size() – 1 inclusive, starting from the head node to the tail node respectively. All new nodes are inserted at the end.

Tail

Head

Data

Next

Prev:nullptr

Data

Next

prev

Data

Next

prev

Data

Next

Prev:nullptr

Pseudocode:

bool Set::insert(const ItemType& value)

if tail is a nullptr

Create a new Node

Make head and tail point to the new Node

Else

If the Set doesn’t contains value

Point the last Node to a new Node

Make the tail point to this new Node

Increment the size

bool Set::erase(const ItemType& value)

If the Set contains value

If only one element in Set

Delete the element

Make head and tail nullptrs

Decrement size

Else if item is at first Node

Delete the Node

Move the head pointer to the next Node

Make the previous pointer for the second Node a nullptr

Decrement size

Else

Adjust pointers of Node on left and right to skip current Node

Delete current Node

Decrement size

bool Set::contains(const ItemType& value) const

while haven’t found a nullptr

if data equals value

return true

Go to next Node

Return false

bool Set::get(int pos, ItemType& value) const

if a valid position

repeatedly pos times:

move iterator to next Node

assign value data at that Node

void Set::swap(Set& other)

Assign each data field to temporary variable

Assign other variable to this variable

Assign temporary variable to other variable

Set::~Set()

While not a nullptr

Delete Node

Move to the next Node

void unite(const Set& s1, const Set& s2, Set& result)

if s1 and s2 are the same

if s1 is the same as result

return

else

assign s2 to result

else

Create a temporary set

repeatedly for the size of the larger Set:

if first set can be accessed at this index

insert value into temporary set

if second set can be accessed at this index

insert value into temporary set

Assign temporary set to result

void subtract(const Set& s1, const Set& s2, Set& result)

Create temporary set

repeatedly for s2 size:

get value in s2 at index

erase value from temporary set

assign temporary set to result

Set::Set(const Set& s)

Invoke default constructor

While iterator is not a nullptr

Insert value into Set

Move forward

Test Cases:

Set s;

assert(s.empty());

assert(s.size() == 0);

assert(s.insert(0));

assert(!s.insert(0));

assert(s.size() == 1);

assert(s.erase(0));

assert(!s.erase(0));

assert(s.size() == 0 && s.empty());

//Tests for empty()

Set a;

assert(a.empty());

assert(a.size() == 0);

assert(a.insert(0));

assert(!a.empty());

//Tests for size()

assert(a.size() == 1);

assert(a.erase(0));

assert(a.size() == 0);

//Tests for insert()

Set b;

assert(b.insert(0));

assert(!b.insert(0));

assert(b.insert(1));

assert(!b.insert(0) && !b.insert(1));

assert(b.insert(2));

assert(!b.insert(0) && !b.insert(1) && !b.insert(2));

for (int i = 3; i <= 100; i++) {

assert(b.insert(i));

assert(!b.insert(i));

}

assert(b.size() == 101);

//Tests for erase()

assert(b.erase(0));

assert(!b.erase(0));

assert(b.erase(1));

assert(!b.erase(0) && !b.erase(1));

assert(b.erase(3));

assert(!b.erase(3));

assert(b.erase(100));

assert(!b.erase(100));

assert(b.erase(2));

for (int i = 4; i < 100; i++) {

assert(b.erase(i));

assert(!b.erase(i));

}

assert(b.empty());

//Tests for contains()

Set c;

for (int i = 1; i < 101; i++) {

assert(c.insert(i));

assert(!c.insert(i));

}

for (int i = 100; i > 0; i--) {

assert(c.contains(i));

}

assert(!c.contains(0));

assert(c.contains(50));

assert(c.contains(1));

assert(c.contains(100));

assert(!c.contains(101));

assert(c.erase(100) && !c.contains(100));

//Tests for get()

Set x;

for (int i = 0; i < 100; i++) {

assert(x.insert(i));

assert(!x.insert(i));

}

for (int i = 99; i >= 0; i--) {

ItemType temp;

assert(x.get(i, temp) && temp == i);

}

for (int i = -1; i > -50; i--) {

ItemType temp;

assert(!x.get(i, temp));

}

ItemType temp;

assert(!x.get(100, temp));

assert(x.insert(100) && x.get(100, temp) && temp == 100);

//Tests for swap()

Set a1, a2;

a1.swap(a2);

assert(!a1.get(0, temp) && !a2.get(0, temp));

a1.insert(0);

a1.swap(a2);

assert(a2.get(0, temp) && temp == 0 && !a1.get(0, temp));

a2.erase(0);

a1.swap(a2);

assert(!a1.get(0, temp) && !a2.get(0, temp));

for (int i = 0; i < 100; i++) {

a1.insert(i);

a2.insert(i + 100);

assert(a1.contains(i));

assert(a2.contains(i + 100));

}

a1.swap(a2);

for (int i = 0; i < 100; i++) {

assert(a1.contains(i + 100));

assert(a2.contains(i));

}

//Tests for Copy Constructor

Set b1;

Set b3(b1);

for (int i = 0; i < 100; i++) {

assert(b1.insert(i));

}

Set b2(b1);

for (int i = 0; i < 100; i++) {

b1.erase(i);

}

for (int i = 0; i < 100; i++) {

assert(b2.get(i, temp) && temp == i);

}

assert(!b3.get(0, temp));

assert(b1.insert(0));

assert(!b3.get(0, temp));

//Tests for Assignment Operator Overload

Set c1;

for (int i = 0; i < 100; i++) {

assert(c1.insert(i));

}

Set c2;

c2 = c1;

for (int i = 0; i < 100; i++) {

assert(c1.erase(i));

assert(c2.get(i, temp) && temp == i);

}

//Tests for unite()

Set s1, s2, result;

s1.insert(0);

s1.insert(1);

s2.insert(2);

s2.insert(3);

s2.insert(4);

s2.insert(5);

unite(s1, s2, result);

for (int i = 0; i < 6; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

unite(s1, result, result);

cout << endl;

for (int i = 0; i < 6; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

unite(result, s2, result);

cout << endl;

for (int i = 0; i < 6; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

unite(result, result, result);

cout << endl;

for (int i = 0; i < 6; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

unite(s2, s1, result);

cout << endl;

for (int i = 0; i < 6; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

unite(s1, s1, result);

cout << endl;

for (int i = 0; i < 2; i++) {

assert(result.get(i, temp));

cout << temp << endl;

}

assert(!result.get(3, temp));

Set x1, x2;

x1.insert(2);

x1.insert(8);

x1.insert(3);

x1.insert(9);

x1.insert(5);

x2.insert(6);

x2.insert(3);

x2.insert(8);

x2.insert(5);

x2.insert(10);

subtract(x1, x2, result);

for (int i = 0; i < 2; i++) {

assert(result.get(i, temp));

cout << "NUM:\t" << temp << endl;

}

assert(!result.get(2, temp));

//Tests for subtract()

Set a;

a.insert("a");

a.insert("b");

a.insert("c");

Set b, temp;

b.insert("a");

b.insert("c");

b.insert("abcdh");

temp.insert("abcdefg");

subtract(a, temp, temp);

for (int i = 0; i < temp.size(); i++) {

ItemType t;

assert(temp.get(i, t));

cout << t << endl;

}

cout << "Passed All Tests" << endl;