Angel Jimenez

CS 32

Project 2

Spring 18

Maybe took me like 30 hours

**Linked List Description:**

For my implementation I researched the four types of linked list and their implementation methods, structure, advantages and disadvantages. I found that the Circular Doubly Linked List was superior in structure due to the sentinel node’s existence removing the problem of special cases.

The general structure of a circular doubly linked list with a “dummy node” looks like this:

\begin{picture}(120,30)
\put(-80,15){\makebox(0,0){dummy}}
\put(-40,20){\line(2,...
...150}}
\put(-45,-2.5){\oval(15,15)[l]}
\put(-45,5){\vector(1,0){5}}
\end{picture}

This image was taken from:

<http://www.cs.uwm.edu/faculty/boyland/classes-archive/fa15.cs351/www/linked-list-variations.html>

Noticeably, we no longer deal with null pointers and thus we remove certain special cases, when we want to loop through our list, we simply make the escape condition such that the current pointer is not equal to the head pointer and to begin, we assign the current pointer to the position following head. Head always points to the dummy node, which has no values but its next and previous pointers do have values.

In my list nodes are 3 items: m\_value, which stores the ‘ItemType’ data; next, which points to the upcoming node, and prev, which points to the node before.

**Pseudo Code:**

Sequence::Sequence() // Constructor

{

Initialize list size variable to zero

create new node, named ‘head’

assign next and prev pointers to itself

}

Sequence::~Sequence() // Destructor

{

While loop, exit when head point no longer pointing to self

{

Create Node ptr, assign to head previous node

Relinking previous to node ahead

Relinking head prev to current prev

delete curr;

}

delete head;

}

Sequence& Sequence::operator=(const Sequence& rhs) // Assignment operator

{

If statement, check for alias

{

Create temporary list

Call swap function, pass temporary list

}

return list

}

Sequence::Sequence(const Sequence& other) // Copy constructor

{

Assign passed objects size to primary object

Create dummy node pointing to self

For loop: node pointer to passed list, loop until head node, reassign curr pointer to next

{

Create a temp node to copy member value

Copy passed member value

Assign temp node next to head

Assign temp node prev to head prev

Assign old last node next to temp node

Assign head prev ptr to temp node

}

}

bool Sequence::empty() const

{

Return true if size equal to zero

}

int Sequence::size() const

{

Return member size

}

bool Sequence::insert(int pos, const ItemType& value)

{

If statement: true if out of bounds position

return false;

Create Node ptr and assign it to node after head

For loop: increment value until less than position

Update node pointer

Create temporary node

Input value into node member value

Link new node ‘prev’ ptr to node before it

Link new node ‘next’ to current node

Link previous node to new node

Link old node prev to new node

Update member size variable

return true;

}

int Sequence::insert(const ItemType& value)

{

Initialize position variable to zero

Temp node ptr assigned to first ‘real’ node

While loop: condition, current ptr not head node

If condition: true if passed value less than member value

Break

Update current ptr position

Update position variable

Call insert function, pass position and value

return count;

}

bool Sequence::erase(int pos)

{

If statement: true if out of bounds position

Temporary node ptr

For loop: increment until less than position {

Update temp node ptr

Link previous node to node after current

Link forward node to node before temp

delete temp ptr

update size variable (decrease)

return true;

return false;

}

int Sequence::remove(const ItemType& value)

{

Initialize removed items variable to zero

Temp node ptr

While loop: exit when temp = head

If statement: member value equivalent to passed value

Temp node on current position

Place current ptr to node before temp

Remove link dependency on temp

delete temp

update member size and removal count

Update temp pt

Return removal count

}

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

{

if position in bounds

Create temp node ptr

For loop until pos, update node ptr

Copy member value at node at position

return true;

return false;

}

bool Sequence::set(int pos, const ItemType& value)

{

if statement to catch good case

Node pointer to cycle through linked list

for loop to get to desired position in linked list

update node next pointer

assign member data to value

}

int Sequence::find(const ItemType& value) const

{

initialize p, set to zero

create node to traverse linked list

while loop to circulate the linked list

if statement to catch equivalence condition

return p;

update curr position

increment position variable

return -1 if while loop changes nothing

}

void Sequence::swap(Sequence& other)

{

create a temporary node pointer, assign to LL1

reassign LL1 head to LL2 head

reassign LL2 head to temp head (LL1)

create temporary size variable, assign it LL1 size

assign LL1 size to LL2 size

Assign LL2 size to temp Size

}

int subsequence(const Sequence& seq1, const Sequence& seq2)

{

if statement: bad size parameters (

return -1

for loop: exit condition, i reaches the size of sequence 1.

temporary place holder for our ItemType

use get to get the value of the 0th item in seq 2,

use get to obtain the value of the ith item in seq 1,

create variable to store start position of subseq

if statement: compare seq1 and seq2 values

compounded for loop, update multiple variables

bool subseq variable defaulted to true

get next value in seq1

get next value in seq 2

if condition: values are different

update bool subseq to false, break loop.

if condition: subseq (if its true, will proceed)

return kStartPos

return -1; cycled through linked list, condition unmet

}

void interleave(const Sequence& seq1, const Sequence& seq2, Sequence& result)

{

new linked list

if statements to catch empty list cases

intialize length and size variables to zero

if seq1 larger, set length to seq1

else length is seq2

temp datatype for value storage

for loop, if get possible, insert value at position of size into result

increment local size variable with insertion

update passed list to temp result

}

void Sequence::dump() const {

temp node ptr

while loop exiting when temp is head

print cerr node data

print node size

}

**Test Cases:**

#include "Sequence.h"

#include <iostream>

#include <cassert>

using namespace std;

int main() {

Sequence seq1;

// testing trivial functions

cerr << "testing trivial" << endl;

assert(seq1.empty() == true);

assert(seq1.erase(0) == false);

// testing insert

cerr << "testing insert" << endl;

assert(seq1.insert("a") == 0);

assert(seq1.insert("ab") == 1);

assert(seq1.insert("abd") == 2);

assert(seq1.insert("abc") == 2);

assert(seq1.insert(0, "doggy"));

assert(seq1.insert(4, "cow")); // pushes abd to pos 5

seq1.dump();

cerr << endl;

assert(seq1.insert(15, "nope") == false);

assert(seq1.insert(-1, "nooo") == false);

// testing erase

cerr << "testing erase" << endl;

assert(seq1.erase(0) == true);

assert(seq1.erase(-1) == false);

assert(seq1.erase(14) == false);

assert(seq1.erase(4) == true);

seq1.dump();

cerr << endl;

// testing remove

cerr << "testing remove" << endl;

assert(seq1.insert("bug"));

assert(seq1.insert("bug"));

assert(seq1.insert(4, "bug"));

assert(seq1.insert(3, "BAD\_CODE"));

assert(seq1.insert(6, "BAD\_CODE"));

assert(seq1.insert(0, "bug"));

assert(seq1.insert(10, "bug"));

seq1.dump();

cerr << endl;

// too many bugs in our code, there are 5

assert(seq1.remove("bug") == 5); // remove the 5 bugs

// lets get rid of the bad code

assert(seq1.remove("BAD\_CODE") == 2);

seq1.dump();

cerr << endl;

cerr << "testing copy and assignment" << endl;

Sequence seq2;

assert(seq2.empty() == true);

Sequence seq3(seq1);

assert(seq1.size() == seq3.size());

seq3.dump();

cerr << endl;

seq3 = seq2;

assert(seq3.size() == 0);

seq3.dump();

cerr << endl;

// testing find

cerr << "testing find" << endl;

seq2.insert("hay");

seq2.insert("hay");

seq2.insert("hay");

seq2.insert("hey");

seq2.insert("hey");

seq2.insert("hey");

assert(seq2.insert(3, "needle") == true);

assert(seq1.find("needle") == -1);

assert(seq3.find("needle") == -1);

assert(seq2.find("needle") == 3);

seq2.dump(); // found a needle in hay stack

// testing swap

cerr << "testing swap" << endl;

cerr << endl; // i should have added this to dump

cerr << "seq2" << endl;

seq2.dump();

cerr << "seq3" << endl;

seq3.dump();

seq3.swap(seq2);

cerr << "new seq2" << endl;

seq2.dump();

cerr << "new seq3" << endl;

seq3.dump();

// testing get and set

cerr << "testing get and set" << endl;

seq1.dump();

ItemType test;

assert(seq1.get(0, test) == true);

assert(seq1.get(10, test) == false);

assert(seq1.get(-4, test) == false);

cerr << test << endl;

test = "Mouse";

cerr << endl;

seq1.dump();

assert(seq1.set(3, test) == true); // change cow to mouse

assert(seq1.set(-3, test) == false);

assert(seq1.set(39, test) == false);

test = "elephant";

assert(seq1.set(0, test) == true); // change a to elephant

seq1.dump();

// testing subsequence

Sequence seq7;

seq7.insert("needle");

seq7.insert(0, "hay");

seq7.insert(1, "hey");

seq3.dump();

seq7.dump();

cerr << subsequence(seq3, seq7) << endl << endl; // returns -1

seq7.set(0, "hay");

seq7.set(1, "needle");

seq7.set(2, "hey");

seq7.dump();

cerr << subsequence(seq3, seq7) << endl; // returns 2

cerr << endl << "testing interleave" << endl;

Sequence seq8;

seq8.insert("a");

seq8.insert("a");

seq8.insert("a");

seq8.dump();

Sequence seq9;

seq9.insert("b");

seq9.insert("b");

seq9.insert("b");

seq9.insert("b");

seq9.insert("b");

seq9.dump();

seq2.dump();

interleave(seq8, seq9, seq2);

seq2.dump();

}