Raj Kumar

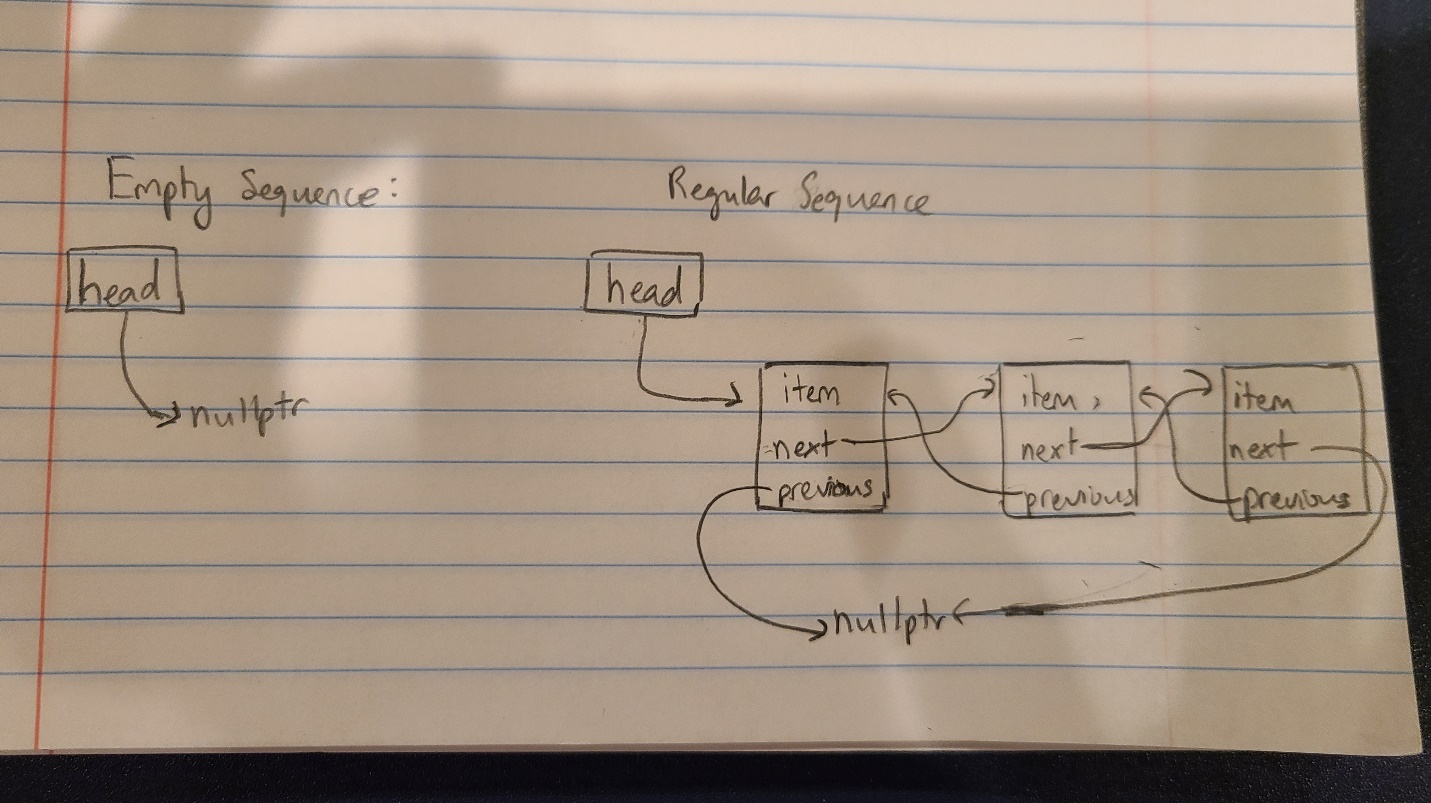
Professor Smallberg

Computer Science 32

26 January 2021

Project 2 Report

My implementation of the doubly-linked list used a standard doubly-linked list with no dummy node. An empty sequence consisted of a head pointer with a nullptr value, and there were no nodes in such a scenario (hence no dummy node). Each node had three components: one pointer pointing to the *next* node in the sequence, one pointer pointing to the *previous* node in the sequence, and one variable *item* of the type Itemtype. A picture is below:



**Pseudocode**

Insert

Make new node

Find node one before insertion position

New node previous pointer now points to the node found

Find node one after insertion position

New node next pointer now points to the node found

Erase

Reach correct position by going through linked-list

Go back one node

this node’s next pointer now points to the node after the to-be-deleted node

Go forward two nodes

This node’s previous pointer now points to the node before the to-be-deleted node

Go back one node

Delete this node

Remove

Loop through-linked list

Check if item in node matches value

True: *erase* that node

Subsequence

Loop through sequence 1’s nodes

If match between sequence 1’s node’s item and sequence 2 item

True: if sequence 2 has no more items

True: Return position in sequence 1

False: Check if next node’s item in sequence 2 matches corresponding next node’s item in sequence 1

False: go to next node in sequence 1

*Loop is done*

Return -1 to indicate no subsequence

Interleave

Make temporary sequence

Loop until shorter sequence of sequence 1 and 2 ends

Insert xth node of sequence 1 into end of temporary sequence

Insert xth node of sequence 2 into end of temporary sequence

Increment x

Loop until longer sequence of sequence 1 and 2 ends

Insert xth node of longer sequence out of 1 and 2 into end of temporary sequence

Increment x

Assign temporary sequence to result sequence

**Tests**

**Note: These are my original tests I used (excluding given tests by Professor Smallberg).**

// using Itemtype = int tests

Sequence s; //default constructor

assert(s.insert(5) == 0); //test insert, test if correct position is returned

assert(s.insert(1,6) == 1); //test insert with two parameters, test if correct position is returned

int x = 4;

assert(s.get(0, x)); //tests get function

assert(x == 5); //tests get function, make sure x is now 5 because of previous line

assert(s.get(1, x)); //tests get function, makes sure get returns true

assert(x == 6); //tests get function, make sure x is now 6 because of previous line

assert(s.get(2, x) == false); //tests get function, make sure x is not changed

assert(s.erase(1)); //tests erase function, makes sure erase returns true

assert(s.get(1, x) == false); //tests get function and makes sure nothing is found

assert(x == 6); //make sure x is not changed

Sequence t; //default constructor

assert(t.remove(5) == false);//nothing to remove

assert(t.empty()); //test empty

assert(t.size() == 0); //test size

assert(t.insert(3) == 0); // test insert

assert(t.insert(1, 5) == 1); //test insert

assert(t.insert(1, 5) == 1); //test insert

assert(t.find(3) == 0); //test find

assert(t.find(5) == 1); //test find returning first case of found input

assert(t.size() == 3); //test size

assert(t.remove(5) == 2); //test remove and its returning of nodes erased

assert(t.size() == 1); //test size and if remove in previous line actually did remove

assert(t.get(0, x));

assert(x == 3);

assert(t.erase(0));//test erase

assert(t.size() == 0); //test size and if erase in previous line actually did erase

assert(t.find(3) == -1); //can't find something that isn't there

Sequence u;

assert(u.insert(0, 0) == 0); //test insert, make sequence u ready for future tests

assert(u.insert(1, 1) == 1); //test insert, make sequence u ready for future tests

assert(u.insert(2, 2) == 2); //test insert, make sequence u ready for future tests

assert(u.insert(3, 3) == 3); //test insert, make sequence u ready for future tests

assert(u.insert(4, 4) == 4); //test insert, make sequence u ready for future tests

assert(u.insert(5, 5) == 5); //test insert, make sequence u ready for future tests

assert(u.set(5, 6) == true); //test set

assert(u.get(5, x)); // test get

assert(x == 6); //test if set actually set the 5th Node properly

assert(u.set(6, 6) == false); //can't set out of bounds

assert(u.set(-1, 6) == false); //can't set out of bounds

assert(u.find(6) == 5); //test find

//copy constructor tests, makes sure both w and u are exact copies, and also end at the same place

Sequence w(u); //copy constructor

int y;

assert(u.get(0, x));

assert(w.get(0, y));

assert(x == y);

assert(u.get(1, x));

assert(w.get(1, y));

assert(x == y);

assert(u.get(2, x));

assert(w.get(2, y));

assert(x == y);

assert(u.get(3, x));

assert(w.get(3, y));

assert(x == y);

assert(u.get(4, x));

assert(w.get(4, y));

assert(x == y);

assert(u.get(5, x));

assert(w.get(5, y));

assert(x == y);

assert(u.get(6, x) == false);

assert(w.get(6, y) == false);

Sequence z(s); //copy constructor

//test swap and copy constructor makes sure both u and z are exact copies, and also end at the same place

z.swap(w);

assert(u.get(0, x));

assert(z.get(0, y));

assert(x == y); //see if 0th nodes match

assert(u.get(1, x));

assert(z.get(1, y));

assert(x == y);

assert(u.get(2, x));

assert(z.get(2, y));

assert(x == y);

assert(u.get(3, x));

assert(z.get(3, y));

assert(x == y);

assert(u.get(4, x));

assert(z.get(4, y));

assert(x == y);

assert(u.get(5, x));

assert(z.get(5, y));

assert(x == y);

assert(u.get(6, x) == false);

assert(z.get(6, y) == false);

//tests assignment and if a is now exactly the same as z

Sequence a;

a = z;

assert(a.get(0, x));

assert(z.get(0, y));

assert(x == y); //see if 0th nodes match

assert(a.get(1, x));

assert(z.get(1, y));

assert(x == y);

assert(a.get(2, x));

assert(z.get(2, y));

assert(x == y);

assert(a.get(3, x));

assert(z.get(3, y));

assert(x == y);

assert(a.get(4, x));

assert(z.get(4, y));

assert(x == y);

assert(a.get(5, x));

assert(z.get(5, y));

assert(x == y);

assert(a.get(6, x) == false);

assert(z.get(6, y) == false);

//test subsequence

assert(subsequence(a, z) == 0); //test if a sequence is a subsequence of a copy of itself starting at 0

assert(a.insert(0, 1) == 0); //test insert

assert(subsequence(a, z) == 1); //test subsequence finding a match at correct Node number

assert(z.insert(2, 4) == 2); //test insert

assert(subsequence(a, z) == -1); //test if subsequence can't find matches if there are no matches

assert(subsequence(a, a) == 0); //test aliasing

Sequence b; //default constructor

assert(b.insert(0, 1) == 0); //test insert

Sequence c(b); //copy constructor

assert(b.insert(0, 1) == 0); //test insert

assert(subsequence(b, c) == 0); //test if number returned is the first match's index

//test interleave and see if the interleaved sequence is what is expected

Sequence d; //default constructor

interleave(b,c,d);

assert(d.get(0, x));

assert(b.get(0, y));

assert(x == y); //see if 0th nodes match

assert(d.get(1, x));

assert(b.get(0, y));

assert(x == y); //see if 1st nodes match

assert(d.get(2, x));

assert(b.get(1, y));

assert(x == y); //see if 2nd nodes match

//test interleave in case of aliasing and see if the interleaved sequence is what is expected

interleave(b, b, b);

assert(b.get(0, x));

assert(x == 1); //see if 0th nodes match

assert(b.get(1, x));

assert(x == 1); //see if 1st nodes match

assert(b.get(2, x));

assert(x == 1); //see if 2nd nodes match

assert(b.get(3, x));

assert(x == 1); //see if 3rd nodes match

//test interleave with empty sequences and aliasing

Sequence e; //default constructor

interleave(e,e,e);

assert(e.size() == 0); //see if sequence e is still empty