Harsh Chobisa

Prof. Carey Nachenberg

CS 31

January 26, 2019

Project 2 Report

I used a doubly linked list with a dummy node. I chose to use a dummy node because I feel that it simplifies the code, as it takes away the edge case of having to insert a node at the start of the linked list, because the dummy node is the start of the list. Essentially, it makes the code easier to write because there are less cases to cover. The list is made up of dynamically allocated struct Node, and each node has three data members: a pointer to the next node, a pointer to the previous node, and a value. The list is in order of insertion; the first item inserted will be first in the list, and the last item inserted will be last in the list.

Here’s what a typical linked list for a set looks like:

NULLPTR

NULLPTR

Node2

Node1

Dummy

Here’s what an empty set with an empty linked list looks like. The dummy node points directly to a null pointer:

Dummy

NULLPTR

NULLPTR

Pseudocode:

Unite:

Set results equal to s1

Loop through s2

If a value in s2 is not in s1

insert that value into result

Subtract

Set result equal to a an empty Set

Loop through s1

If a value is not in s2 and is in s1

Insert into result

Get

Loop through entire list

For each node (named dummy)

Get its value

Loop through entire list

Count how many nodes have values greater than dummy’s value

If count is equal to pos

Return true

Else move on to next node

Insert

Loop to end of list

Create new node storing value

Link node to end of list

Erase

Loop until you find value

Link previous node to next node

If value to be erased is not last in list

Link next node to previous node

Delete current node

Test Cases:

Set s = Set();

//tests the empty function

assert(s.empty());

//ItemType x = "9876543";

s.insert("123456789");

//tests the size function

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

s.insert("99810823");

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

//tests the insert function

assert(!(s.insert("123456789"))); //cannot insert the same value twice

Set n = Set();

assert(n.insert("10")); //testing adding to an empty list

assert(n.insert("20")); //testing adding to the end of a list

n.insert("30");

n.insert("40");

n.insert("50");

//tests the remove function

assert(n.erase("10")); //testing erasing from the front of a list

assert(n.erase("50")); //testing erasing the last item in a list

assert(n.erase("30")); //testing erasing an item from the middle

assert(!(n.erase("30"))); //cannot remove an item thats not in the list

n.erase("40");

assert(n.erase("20")); //testing deleting the last item in a list

//tests the contains function

assert(!(n.contains("20")));

n.insert("60");

assert(n.contains("60"));

n.erase("60");

n.insert("10");

n.insert("20");

n.insert("30");

n.insert("40");

n.insert("50");

//tests the get function

ItemType checker;

assert(n.get(0, checker) && checker == "10");

assert(n.get(1, checker) && checker == "20");

assert(n.get(2, checker) && checker == "30");

assert((!n.get(8, checker))); //returns false if pos is greater than list size

assert((!n.get(-1, checker))); //returns false if pos is less than list size;

assert((!n.get(n.size(), checker))); //returns false if pos is equal to list size;

//test swap function

Set t;

t.insert("1"); t.insert("2"); t.insert("3");

s = Set();

s.insert("5");

t.swap(s);

assert(t.size() == 1 && s.size() == 3);

ItemType y;

assert(t.get(0, y) && y == "5");

assert(s.get(0, y) && y == "1");

assert(s.get(1, y) && y == "2");

assert(s.get(2, y) && y == "3");

//test assignment operator

t = s;

assert(t.get(0, y) && y == "1");

assert(t.get(1, y) && y == "2");

assert(t.get(2, y) && y == "3");

//test copy constructor

Set k = s;

assert(k.get(0, y) && y == "1");

assert(k.get(1, y) && y == "2");

assert(k.get(2, y) && y == "3");

//test unite

Set j = Set();

j.insert("3");

j.insert("4");

j.insert("5");

Set r = Set();

r.insert("100000");

unite(k, j, r);

assert(r.contains("1") && r.contains("2") && r.contains("3")

&& r.contains("4") && r.contains("5") && r.size()==5);

//test unite for aliasing

unite(k, j, j);

assert(j.contains("1") && j.contains("2") && j.contains("3")

&& j.contains("4") && j.contains("5") && j.size()==5);

//test subtract

subtract(k, j, r);

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

Set x = Set();

x.insert("1");

x.insert("2");

x.insert("3");

Set z = Set();

z.insert("2");

z.insert("3");

z.insert("4");

subtract(x, z, z);

assert(z.contains("1") && z.size()==1);

cerr << "Passed all tests" << endl;