Edmund Zhi

UID: 805328871

Project 2 Report

**Description:**

I have implemented a circular double-linked list. An empty set consists of only a head/dummy node whose previous and next pointers point to itself. When an item is added to our set, the first item is at position 0, and m\_head->m\_next points to that item. My m\_head->prev points to the very last item in our Set, which is at position (m\_size-1). In other words, the node at the end of the list is the one that m\_head->m\_previous points to and whose m\_next is the head/dummy node.

My list of nodes is sorted in descending order (which is determined by a “<” sign in line 109 of Set.cpp). This was to make the get() function easier to write. A typical set looks something like this:

Diagram

Description automatically generated

**Pseudocode for non-trivial algorithms:**

**insert:**

*if set is empty*

*create node with value and add node to set*

*increase size*

*if value is already in set*

*return false*

*search through the set*

*if item in set is less than value*

*remember item*

*stop searching*

*add create new node with value to set right before remembered item*

*increase size*

**erase:**

*if set doesn’t have target value*

*return false*

*traverse through the set until we reach target value*

*unlink item from set*

*delete item*

**get:**

*if desired position is out of bounds*

*return false*

*traverse through the Set until we reach target value*

*store target value in another variable*

*return true*

**swap:**

*switch sizes*

*switch head nodes*

***unite:***

*allocate new Set*

*insert everything from Set 1*

*insert everything from Set 2 that is not already added*

*replace target set with new allocated Set*

**difference:**

*allocate new Set*

*insert everything from Set 1 that is not in Set 2*

*insert everything from Set 2 that is not in Set 1*

*replace target set with new allocated Set*

**Test Cases:**

All of my tests were run on ItemType = std::string.

This test tested the housekeeping functions

**void** test6() {

Set s1; //tests constructor

Set s2;

Set result;

result = s1; //tests assignment operator

assert(s1.size() == s2.size() == result.size());

assert(s1.size() == 0); // tests size()

assert(s1.empty()); // tests empty()

s1.insert("1");

s1.insert("2");

s1.insert("3");

s2.insert("4");

Set s3(s2);

assert(s3.size() == 1); //checks copy constructor for single node

Set s4(s1);

assert(s4.size() == s1.size()); // checks copy constructor for more than one node

}

This test tested to make sure that insert and get were working as intended. Insert should not be true if Set already contains value. Set should be organized in descending order so get works properly.

**void** test2() {

Set ss;

assert(ss.insert("1"));

assert(ss.insert("2"));

assert(!ss.insert("2"));

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

ItemType x = "laobing";

assert(ss.get(0, x) && x == "2");

assert(ss.get(1, x) && x == "1");

}

This tested the erase function.

**void** test()

{

Set ss;

assert(ss.insert("pita"));

assert(ss.insert("roti"));

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

assert(ss.contains("roti"));

assert(ss.erase("roti"));

assert(ss.insert("roti"));

ItemType x = "laobing";

assert(ss.get(0, x) && x == "roti");

assert(ss.get(1, x) && x == "pita");

}

The following two tests tested the unite and difference functions:

**void** test3() {

Set s1;

Set s2;

assert(s1.insert("2"));

assert(s1.insert("8"));

assert(s1.insert("3"));

assert(s1.insert("9"));

assert(s1.insert("5"));

ItemType x = "laobing";

assert(s1.get(0, x) && x == "9");

assert(s1.get(1, x) && x == "8");

assert(s1.get(2, x) && x == "5");

assert(s1.get(3, x) && x == "3");

assert(s1.get(4, x) && x == "2");

assert(s2.insert("10"));

assert(s2.insert("6"));

assert(s2.insert("3"));

assert(s2.insert("8"));

assert(s2.insert("5"));

assert(s2.size() == 5);

assert(s2.get(0, x) && x == "8");

assert(s2.get(1, x) && x == "6");

assert(s2.get(2, x) && x == "5");

assert(s2.get(3, x) && x == "3");

assert(s2.get(4, x) && x == "10");

Set result;

unite(s1, s2, result);

assert(result.size() == 7);

assert(result.get(0, x) && x == "9");

assert(result.get(1, x) && x == "8");

assert(result.get(2, x) && x == "6");

assert(result.get(3, x) && x == "5");

assert(result.get(4, x) && x == "3");

assert(result.get(5, x) && x == "2");

assert(result.get(6, x) && x == "10");

}

**void** test4() {

Set s1;

Set s2;

assert(s1.insert("2"));

assert(s1.insert("8"));

assert(s1.insert("3"));

assert(s1.insert("9"));

assert(s1.insert("5"));

assert(s2.insert("10"));

assert(s2.insert("6"));

assert(s2.insert("3"));

assert(s2.insert("8"));

assert(s2.insert("5"));

Set result;

difference(s1, s2, result);

ItemType x = "laobing";

assert(result.size() == 4);

assert(result.get(0, x) && x == "9");

assert(result.get(1, x) && x == "6");

assert(result.get(2, x) && x == "2");

assert(result.get(3, x) && x == "10");

}

Since my contains function is included in all of these functions, if these test cases work, then contains() must work as well. Moreover, since the unite() and difference() functions incorporate swap, then swap will work if these test cases work.