1. Linked list implementation:

I implemented my linked list as a doubly linked list with a dummy node in the front as well as a head pointing to the dummy node. It is not circular and does not have a tail or last dummy node. Each of my nodes have a value, a next node pointer, and a previous node pointer. My list is in a way that when an item is inserted, it will insert in order from decreasing to increasing.

1. Test cases: using std::string

Set ss;

assert(ss.empty()); // ensure set is initialized as empty

assert(ss.size() == 0); // test counter is initially set to 0 with an empty set

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

assert(!ss.insert("maddy")); // shouldn't insert the same value

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

assert(ss.insert("Gabby")); // should still insert this string regardless of case

assert(ss.size() == 3);

assert(!ss.empty()); // tests that dummy node correctly linked to list

assert(!ss.contains("kha"));

Set s2 = ss;

assert(!s2.empty()); // new set should not be empty, should be filled with copied values

assert(s2.contains("maddy")); // make sure all values copied into copied set and copy constructor works

assert(s2.contains("Gabby"));

assert(s2.contains("gabby"));

assert(s2.size() == 3); // the sizes member correctly copied over from copy constructor

assert(!s2.contains("simon"));

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

assert(!ss.contains("maddy")); // testing value correctly erase

assert(ss.size() == 2); // testing if counter correctly decremented

assert(!ss.erase("maddy")); //testing shouldn't be able to erase value that isn't there

assert(ss.contains("gabby")); // testing other nodes arent effected

assert(!ss.contains("maddy")); //testing the value is gone

ItemType value;

assert(!ss.get(3,value)); //testing a out of range position, size is only 2 wanting something greater than 3 others

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

assert(ss.get(1,value)); // testing get function

assert(value == "alexis"); // value correctly assigned as the item greater than 0 other items

assert(!ss.get(-60, value)); //testing a invalid position of get function

Set s3;

s3.swap(ss); //testing to swap an empty set

assert(s3.contains("gabby")); // correctly swap values into new set

assert(s3.contains("alexis"));

assert(s3.size() == 3); //test sizes are correctly swapped

assert(ss.size() == 0); // original array should now be empty

assert(!ss.contains("alexis")); //test contents are all empty

Set result;

result.insert("simon"); //initiliazing result so it isnt empty when passed in

unite(s2,s3,result); //uniting two sets that are identical, should only contain unique values

assert(result.size() == 4);//testing only unique values got put into result

assert(result.contains("alexis"));

assert(!result.contains("simon")); //testing result doesn't contain a the value it initially had, should be deleted

assert(s2.size() == 3); //testing the sets passed in are unaffected and same size

assert(s3.size() == 3);

assert(s2.contains("maddy")); //should still have original value

assert(s3.contains("alexis")); //should still have original value

butNot(s2, s3, result);

assert(!result.contains("alexis")); // tests butNot function should have zero values because passed in 2 identical sets

assert(result.contains("maddy")); // should be empty with no unique values

assert(s2.contains("maddy")); //tests sets passed in remain unaffected

assert(s3.contains("gabby"));

s2.insert("kha");

butNot(s2,s3,result);

assert(result.contains("kha")); // tests butNot function -- should contain one unique value

s2 = s3; //tests operator equals function

assert(s2.contains("alexis")); // testing the values got correctly assigned to the other set

assert(s2.size() == 3); // size correctly passed

assert(s3.size() == 3); // size remains unchanged

assert(!s2.contains("maddy")); // testing it does not have its old value

assert(s3.contains("gabby")); //testing original set remains unchanged

Set s4;

s4=s2; // test operator equals on empty set

assert(s4.contains("alexis"));

assert(s4.size() == 3);

Set s5;

unite(s5,s4,result); // testing unite with empty set

assert(result.contains("alexis"));

assert(result.contains("gabby"));

assert(result.size() == 3);

Set s6;

assert(!s6.erase("baba")); //testing erase on an empty set

s6.swap(s4); //swap with an empty function

assert(s5.empty());

assert(s6.size() == 3);

1. Psudeocode

insert:

Find where new node should be inserted to be alphabetically/decreasing to increasing order

Add values into new node

Reassign previous pointers, following pointers, and the new nodes pointers

erase:

Repeatedly

Check each item in the list to see if it matches the item wanting to delete

If found item

Delete item

Else keep looking through list until no more values to check

contains:

Repeatedly

Check each value in list to see if it matches the desired value

get:

Repeatedly

Keep traversing through list one by one because it is sorted to find the word the is exactly x bigger than other items

If the index matches the desired number, we have an exact value

Else, no value that is bigger than x items

swap:

switch the set’s counter variables with one another

switch the node dummy pointers with one another by creating a local temp node pointer

butNot:

Erase the existing values in the result set

Repeatedly

if there are items in the copied set 2

Check if result contains this value

Result has this value, delete it because not unique

Proceed to the next item in set 2

unite:

Erase items in results if there are existing items

Repeatedly

While there are items in set 1 and set 2

Start at both set 1 and set 2’s first items and increment over items together

Insert set 1 item

Check if set 2 contains this item

If set 2 does not have this item

Add set 2’s item to the result

If both set 1 and set 2 have gone through their entire list

Break