My doubly-linked list is a straight linked list where you can traverse forward or backwards using the next and prev pointers. It has both a head and tail pointer so you can access either the front or the back. The nodes are in a backwards order from the way we added it (like a stack). For example, if we inserted “Alpha” and then inserted “Beta”, “Beta” would come before “Alpha” in our linked list.

Pseudocode

erase and eraseAll call the helper function : int doErase(bool all, const ItemType& value);

erase calls this with all set to false and eraseAll calls it with all set to true.

doErase pseudocode…

declare a Node pointer pointing to head (traverser)

declare a bool deleted representing whether or not we deleted an item

declare an int numDeleted representing the number of items we deleted

traverse through linkedlist until we find a node with the same value as the parameter or nullptr{

if(value is found){

if all is true{

set numDeleted to one and decrement numItems

}

Else{

Set numDeleted to numItems and after set numItems to 0

}

Set deleted to true

break from the loop

}

Update traverser

}

Reset traverser

Loop through the linkedlist and check if numItems is zero for any nodes{

If numItems is zero{

Link next and prev

Update head and tail if they need to be updated

Delete the node

Break from the loop

}

Update traverser

}

Check if any items were deleted(bool)

If so{

Return numDeleted

}

Else{

Return 0;

}

void combine(const Multiset& ms1, const Multiset& ms2, Multiset& result);

combine pseudocode…

Initialize a Multiset using the copy constructor taking in ms1 as an argument

Loop through ms2’s uniqueSize{

Declare an ItemType

Use the get function to change ItemType into a value on the node

Get the count of that ItemType

Loop through the count and insert ItemType into the Multiset

}

Using the assignment operator, set result to copy

void subtract(const Multiset& ms1, const Multiset& ms2, Multiset& result);

subtract pseudocode…

Initialize a Multiset using the copy constructor taking in ms1 as an argument

Loop through ms2’s uniqueSize{

Declare an ItemType

Use the get function to change ItemType into a value on the node

Get the count of that ItemType

Loop through the count and erase ItemType into the Multiset

}

Using the assignment operator, set result to copy

Test Code:

#include "Multiset.h"

#include <iostream>

#include <cassert>

using namespace std;

void test()

{

Multiset first;

assert(first.empty()); // Tests empty

assert(first.size() == 0); // Tests size

assert(first.insert("cumin")); // Tests insert

assert(first.insert("turmeric"));

assert(first.insert("cumin"));

assert(first.insert("coriander"));

assert(first.insert("cumin"));

assert(first.insert("turmeric"));

assert(first.size() == 6 && first.uniqueSize() == 3); // Tests uniqueSize

assert(first.count("turmeric") == 2); // Tests count

assert(first.count("cumin") == 3);

assert(first.count("coriander") == 1);

assert(first.count("cardamom") == 0); // Tests count

Multiset second(first); // Tests copy constructor

Multiset third;

assert(second.size() == 6 && second.uniqueSize() == 3);

assert(second.count("turmeric") == 2);

assert(second.count("cumin") == 3);

assert(second.count("coriander") == 1);

assert(second.count("cardamom") == 0);

assert(second.erase("coriander")); // Tests erase

assert(second.count("coriander") == 0);

assert(second.eraseAll("turmeric") == 2); // Tests eraseAll

assert(!second.contains("turmeric"));

assert(second.contains("cumin"));

for (int i = 0; i < 3; i++)

{

assert(second.insert("cardamom"));

}

ItemType val;

assert(second.get(0, val) == 3); // Tests get

assert(val == "cardamom"); // Tests get

third = first; // Tests assignment operator

assert(third.size() == 6 && third.uniqueSize() == 3);

assert(third.count("turmeric") == 2);

assert(third.count("cumin") == 3);

assert(third.count("coriander") == 1);

assert(third.count("cardamom") == 0);

assert(third.size() == 6 && third.uniqueSize() == 3);

assert(third.count("turmeric") == 2);

assert(third.count("cumin") == 3);

assert(third.count("coriander") == 1);

assert(third.count("cardamom") == 0);

assert(second.count("turmeric") == 0);

assert(second.count("cumin") == 3);

assert(second.count("coriander") == 0);

assert(second.count("cardamom") == 3);

third.swap(second); // Tests swap

assert(second.size() == 6 && second.uniqueSize() == 3);

assert(second.count("turmeric") == 2);

assert(second.count("cumin") == 3);

assert(second.count("coriander") == 1);

assert(second.count("cardamom") == 0);

assert(third.count("turmeric") == 0);

assert(third.count("cumin") == 3);

assert(third.count("coriander") == 0);

assert(third.count("cardamom") == 3);

Multiset fourth;

combine(second, third, fourth); // Tests combine

assert(fourth.count("turmeric") == 2);

assert(fourth.count("cumin") == 6);

assert(fourth.count("coriander") == 1);

assert(fourth.count("cardamom") == 3);

combine(third, second, fourth); // Tests combine

assert(fourth.count("turmeric") == 2);

assert(fourth.count("cumin") == 6);

assert(fourth.count("coriander") == 1);

assert(fourth.count("cardamom") == 3);

Multiset fifth;

subtract(second, third, fifth); // Tests subtract

assert(fifth.count("turmeric") == 2);

assert(fifth.count("cumin") == 0);

assert(fifth.count("coriander") == 1);

assert(fifth.count("cardamom") == 0);

subtract(third, second, fifth); // Tests subtract

assert(fifth.count("turmeric") == 0);

assert(fifth.count("cumin") == 0);

assert(fifth.count("coriander") == 0);

assert(fifth.count("cardamom") == 3);

combine(second, third, second); // Checks for aliasing in combine

assert(second.count("turmeric") == 2);

assert(second.count("cumin") == 6);

assert(second.count("coriander") == 1);

assert(second.count("cardamom") == 3);

combine(second, third, third); // Checks for aliasing in combine

assert(third.count("turmeric") == 2);

assert(third.count("cumin") == 9);

assert(third.count("coriander") == 1);

assert(third.count("cardamom") == 6);

subtract(third, second, second); // Checks for aliasing in subtract

assert(second.count("turmeric") == 0);

assert(second.count("cumin") == 3);

assert(second.count("coriander") == 0);

assert(second.count("cardamom") == 3);

subtract(third, second, third); // Checks for aliasing in subtract

assert(third.count("turmeric") == 2);

assert(third.count("cumin") == 6);

assert(third.count("coriander") == 1);

assert(third.count("cardamom") == 3);

}

int main()

{

test();

cout << "Passed all tests" << endl;

}