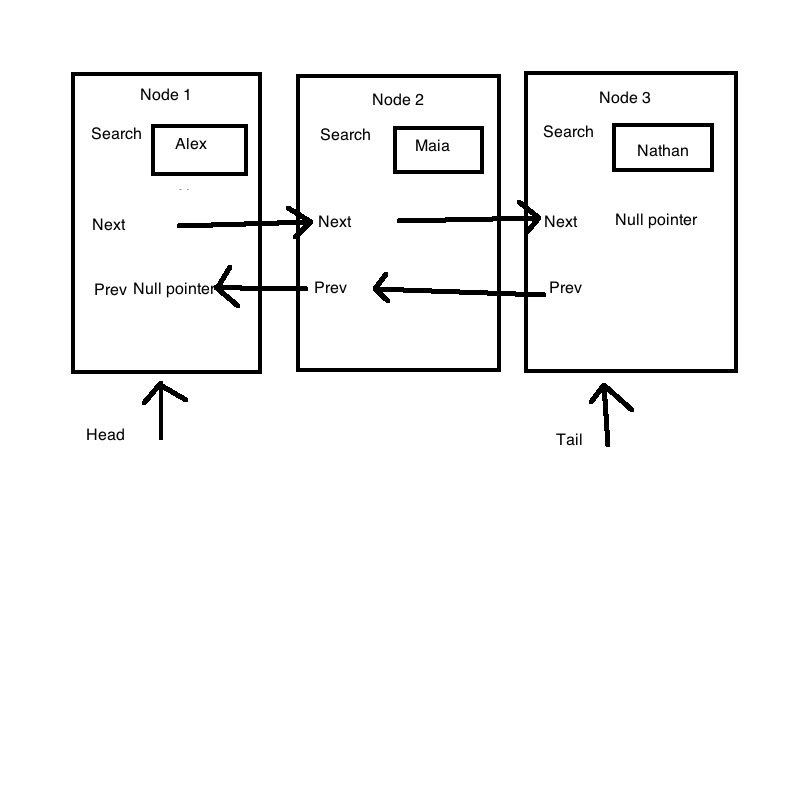
Report

1. My doubly-linked list is a list that contains both a head and a tail pointer. This list is not a circular linked list nor does it have a dummy node; it is just a regular doubly linked list. I have a head and a tail pointer pointing to the first and last node in the list, and to access the nodes in between, I mostly use the “next” Node pointer to access the next one, or I would use the “previous” Node pointer to access the one before. The value of the node is declared with the search variable. In my project, the list nodes are not in any particular order, so I am aware that my project may spend time searching for a value because it is not in order.

Unemptied Set:



Empty Set:

Head = null pointer, tail = null pointer

Size = 0

1. Insert:

Check if size is greater than maximum or if a value is already inserted

Return false if true

Create a new node and initialize it with the value passed in insert

If current set is empty:

Set head and tail to the new node

Else if not empty:

Insert new Node after tail node

Set tail node to after new node

Increase size and return true

Erase:

Check if empty set or does not contain the attempt to erase value

Return false if true

If list only has one item:

Assign kill node to item

Delete item

Create a new node to head

Going through list before the last node:

Once found the value we want to delete:

Assign kill node to delete node

Assign the pointer node to the node past the delete node

Delete node

Return true

Get:

Test boundaries and false inputs

Return false if true

Assign node pointer to head

Going through loop:

Create counter variable to count how many nodes are greater than value

Create new node pointer2 to head to compare value with previous node pointer

If node pointer1 is greater than node pointer2:

Add it to counter

If counter matches the pos number passed it

Return true

Unite:

Create an ItemType and Set variable

While going through set1:

Add values into our newly created set

While going through set2:

Add values not already in new created set

Set new created set to result

Subtract:

Create an ItemType and Set variable

While going through set1:

add values into our newly created set

While going through set2:

subtract values that are already in the newly created set

Set new created set to result

1. Test Cases

Set skaters;

assert(skaters.size() == 0); //checks size

assert(skaters.insert("maia"));

assert(skaters.insert("alex"));

assert(skaters.insert("nathan"));

assert(skaters.insert("karen")); //inserts values into list

assert(skaters.contains("nathan")); //checks if we inserted a value

assert(skaters.size() == 4); //rechecks size

ItemType x;

assert(skaters.insert("karen") == false); //checks if we can insert a node already there

assert(skaters.get(0, x) && x == "alex"); //checks for correct value alphabetically

assert(skaters.get(2, x) && x == "maia"); //checks for correct value with get

assert(skaters.get(-5, x) == false); //checks for out of bounds

assert(skaters.erase("alex")); //erase an item

assert(skaters.size() == 3); //checks size after erasing

assert(skaters.get(1, x) && x == "maia"); //rechecks that nodes are bumped up

assert(skaters.contains("alex") == false); //checks if set still contains deleted item

Set favs(skaters); //checks copy constructor

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

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

assert(favs.insert("satoko"));

Set stanned = skaters; //checks assignment operator

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

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

Set meh;

meh.insert("yuzuru");

meh.insert("shoma");

favs.swap(meh); //checks swap method

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

assert(favs.get(0, x) && x == "shoma"); //ensures that fav now has meh values

assert(favs.insert("rika")); //insert into facs

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

assert(meh.get(1, x) && x == "maia"); //checks that meh has fav values

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

favs.swap(meh); //swap back

unite(favs, meh, skaters); //checks for unite function

assert(skaters.size() == 7); //rechecks size

assert(skaters.get(0, x) && x == "karen");

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

assert(skaters.get(6, x) && x == "yuzuru");

subtract(favs, meh, skaters); //checks subtract method

assert(skaters.size() == 4); //rechecks size

assert(skaters.get(0, x) && x == "karen");

assert(skaters.get(2, x) && x == "nathan"); //checks for certain elements