Homework 9: Iterators and Optimization

CSCI 60

Due 5pm Thursday, March 12, 2020 / before class Friday if using a late day

Krehbiel

Overview. Your final homework is a mix of questions related to iterators and putting the finishing touches on your linked list implementation of your bag data structure.

Submission instructions. Modify `lbag.h` and `bag.h` as instructed below and submit these files with their original names. If you download all starter files for the assignment into the same directory, your starter files and completed code should compile as follows:

```
g++ hw9main.cpp lbag.cpp bag.h -std=c++11
```

If you don't change the main file at all (though it would be a good idea to add tests to it!), your sample runs before and after finishing all the tasks should look like the following:

```
Q1 test:

Q2 test:
1000 2's and 1000 4's
1000 4's erased
1000 2's and 0 4's

Q3 test:
```

```
Q1 test:
(4 3 2 1 0 ) is not sorted
(1 2 3 4 5 6 7 8 9 10 ) is sorted

Q2 test:
1000 2's and 1000 4's
1000 4's erased
1000 2's and 0 4's

Q3 test:
0
1
4
9
16
```

Question 1. (5 points) Write a member function for `LBag` to check whether the underlying list is sorted. The function should return true if and only if the values are listed in non-decreasing order from head to tail, e.g., $2 \rightarrow 3 \rightarrow 3$ is sorted but $3 \rightarrow 4 \rightarrow 2$ is not.

Question 2. (10 points) Re-implement `erase` to remove all nodes with a given value in a single pass of the list as discussed in class. Note that this dramatically reduces the runtime from quadratic to linear!

Question 3. (25 points) In class we wrote an iterator for our `LBag` data structure built on top of a node-based linked list. For this final programming assignment, you will write an iterator for our dynamic array implementation of `Bag`. As with our `L_iterator`, implementing your `A_iterator` for use with `Bag` will involve adding `begin` and `end` member functions in the definition of the `Bag` class to produce iterators associated with the `Bag` object as well as designing a class `A_iterator` that overloads the operators `*`, ++ (pre- and post-), !=, and ==.

The design challenge is figuring out what private member variables `A_iterator` will need. When we designed `L_iterator`, it sufficed to keep track of a current pointer to a node, because this pointer could be used to access the current element in the list, to update the pointer when it was incremented, and to check (by comparing itself to the `end` iterator) when the end of the list was reached. What information does `A_iterator` need to <u>access</u> the current element in the list, <u>update</u> itself when incremented, and <u>check</u> when the end of the array is reached?