CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean | Fall 2021
Webpage: `http://www.cs.clemson.edu/~bcdean/` | MWF 9:05-9:55
Handout 4: Lab #3 | Powers 208

1 Building a Hash Table Class

The goal of this lab is to practice building a hash table, to practice writing code involving a C++ class that implements a data structure, and to practice using abstraction by substituting out the "back end" of a class while keeping its "front end" interface the same, allowing code calling the class to not have to change.

In the directory:

`/group/course/cpsc212/f21/lab03`

you will find three files, `main.cpp`, `intset.cpp`, and `intset.h`. Make a copy of these files in your own directory space.

If you look at `intset.h`, you will find the definition of a simple class that implements a data structure for representing a set of integers. The members of this class allow you to perform the following operations:

- `find(key)` : Return true if key is present in the set, false otherwise.
- `insert(key)` : Insert a new integer into the set. This function checks if the key being inserted is already in the set and if so, does nothing (so the set contains at most one copy of each integer).
- `remove(key)` : Remove an integer from the set. This function checks if the key being removed is in the set, and if it is not, the function does nothing.
- `print()` : Print the contents of the set in sorted order.

The code in `intset.h` and `intset.cpp` currently provides a straightforward implementation of these functions using a *sorted array* as the underlying representation of the set. If the memory allocated for the array fills up due to a large number of *insert* operations, a new array is allocated of twice the original size.

In this lab, your goal is to replace the underlying implementation of the integer set with one based on a chained *hash table*. You will need to change most of the code in `intset.cpp`, and some of the code in `intset.h` (the changes you will need to make to the `intset.h` file are indicated in comments in that file). You should *not* change the definition of any of the public member functions (e.g., `find`) in `intset.h`, so that code that uses your integer set class should not need to be changed

in any way (this is one of the benefits of object-oriented software design – the ability to change underlying implementation of a class while keeping the same public interface intact, making the change largely transparent to anyone using the class).

2 Hash Table Expansion

As a key part of the implementation of your *insert* function, you should expand the size of the underlying array for the hash table periodically to ensure performance stays good (so that the average length of a linked list in the hash table is short). Specifically, your hash table should keep track of two key numbers:

- n, the number of keys currently stored in the table.
- m, the size of the array currently allocated for the table.

Any time n grows to be as large as m, you should upsize the table so m doubles. This entails allocating a new table, re-inserting (and therefore re-hashing) everything into the new table, and then de-allocating the old table and all of the elements stored in its associated linked lists.

Note that your hash table will still function properly without the code to do upsizing; it will merely slow down.

3 Memory Issues

Your hash table class should use a constructor to initialize itself and a destructor to free its memory, just as with the initial array implementation. All memory allocated by your code should therefore be released by the time your program exits.

4 Compiling and Running

To compile on the Unix lab machines, use the command:

```
g++ -o main main.cpp intset.cpp
```

You can then execute your code by running:

```
./main
```

Recall that the "-o main" part of the command line tells the compiler what to name your executable file, in case you want to give it a different name.

5 Testing

The file `main.cpp` provides a simple command-line interface for testing your code.

It allows you to type, at the command line, commands like:

```
insert 7
remove 3
find 2
print
quit
```

These commands invoke the corresponding methods from the integer set class, so you can use them to debug your work. You may also want to put several of these commands in a text file (e.g., `input.txt`) and execute them all at once using

```
./main < input.txt
```

since this way you won't need to re-type the same commands over and over during testing.

The `main.cpp` file also defines a command "stress" that performs stress testing on your underlying class, executing 1 million calls to *insert*, *find*, and *remove*. This should run slowly with the initial array representation, since the *insert* operation takes $\Theta(n)$ time. However, once you have substituted a hash table, stress testing should easily run in less than 1 second.

6 Dining Preferences

An upcoming class demo will make use of data collected from everyone in the class in terms of their favorite restaurants in town. The file `dining-prefs.txt` contains a number of lines like

"Brian Dean" likes "All In Coffee Shop"

each describing a local restaurant that you like. Please modify this file so that it lists a small number (say, 3 to 5) of your favorite local restaurants, one per line, exactly in the format above. Then be sure to include this file in your submission.

7 Submission and Grading

Submit your `intset.cpp` and `dining-prefs.txt` files using handin.cs.clemson.edu just as with the previous lab. Zero points will be awarded for code that does not compile, so make sure your code compiles on the lab machines before submitting!

Final submissions are due by 11:59pm on the evening of Sunday, September 12. No late submissions will be accepted.