CS 014: Introduction to Data Structures and Algorithms
Fall 2017
Lab 9

Objectives

In this lab, you will implement an expansive open addressing hashtable.

Deliverables

- (10%) attending the lab on Monday.
- (65%) The final source code.
- (5%) You must adhere to the following submission format. You need to submit a single file on iLearn named "CS014_lab9_<UCR Net ID>.zip" where <UCR Net ID> is the student UCR Net ID. The ZIP file should contain the source code in .cc or .hh files. The ZIP file should also contain a PDF report that contains the answers to any questions in the lab. On the cover page, your report should mention:
    - Your TA name.
    - Your lab section number.
    - You names and UCR Net IDs.
- (20%) A PDF report with the answers to the questions below.

Groups

- This lab should be done individually.

Due date

- The deliverables are due on Tuesday 12/05 by 11:59 PM Pacific Time. However, you are highly encouraged to deliver it during or before the lab on Monday 12/04 to save your time.

Problem overview

Open Addressing hashtables are implemented using arrays. A hash function maps a key to an index in the array, called bucket, where the key is stored. If two keys map to the same bucket, a collision happens. To resolve collisions, a resolution function $f(i)$ is used. In this lab, you will implement two resolution functions, linear probing and quadratic probing. Please refer to the textbook and lectures for more details. In this lab, you are required to modify an existing implementation of a hashtable to allow it to expand as needed. For simplicity, you are only asked to expand the hashtable if more elements are inserted. You do NOT have to shrink it back after deletions happen.

Detailed steps

1. Create a workspace in Cloud9 for this lab and set the value 'https://github.com/aseldawy/CS014-Lab9.git' in the field 'Clone from Git or Mercurial URL' when you create a Cloud9 workspace. Instead, you can download the source code from iLearn and upload it to an empty workspace in Cloud9.
2. Take a look at the code to understand how it works. Unlike the generic hashtable that you implemented in Lab 8, this hashtable only works with integer keys and uses linear probing resolution function.
3. The main function creates a hashtable of initial size `hashtableSize` and then inserts that number of elements into it. For example, at the first iteration it initializes the hashtable with 100 elements and inserts 100 elements into it.
4. Run the code and make sure that it works correct.
5. Change the following line:

```
Hashtable hashtable(hashtableSize);
```

To

```
Hashtable hashtable(10);
```

6. Run the code again. What happens to the program? Why?
7. Now, as the hashtable is initialized with only 10 elements, you need to modify it to make it dynamically expansive.
8. We will follow the guidelines we learned in the class and trigger an expansion when the load factor $\lambda \geq 0.5$. To calculate the load factory efficiently, you will need to add a new attribute, `size`, in the class Hashtable which stores the total number of occupied buckets. You will also need to adjust the `insert` and `erase` functions to update the `size` correctly.
9. Add a test at the beginning of the `insert` function to test if an expansion and rehash should be triggered. If the load factor is larger than or equal to 0.5, call the function `expandAndRehash` which should do the following:
    a. Calculate the new capacity of the hashtable. The new capacity should be the value in the array `GoodPrimeNumbers` next to the current capacity.
    b. Create two new local variables of type `vector` for buckets and status with the new capacity.
    c. Swap the two buckets vectors and the two status vectors. At this point, the hashtable is empty as the new vectors are not filled yet.
    d. Scan the keys that were in the hashtable which are now stored in the local variable. For each key, call the insert function to insert it into the hashtable. Question: Is it possible that one of these inserts trigger another rehash? Why or why not? How should we adjust the size and capacity of the hashtable before calling the insert function?
10. If the rehash is implemented correctly, the code should now run fine. Verify that the code runs correctly and observe the number of collisions. Question: How many rehashes happen when you initialize the hashtable with 10 elements and insert 10,000 elements?