Algorithms & Data Structures: Lab 08

week of 26th November 2018

1 Setup

1.1 Saving your work from last week

As with all previous weeks, you will use `git` to download a bundle of lab code. You will probably have made modifications in your downloaded copy; if you have not already done so, you need to save those modifications. First examine the changes present in your downloaded copy by issuing the following commands from the labs directory:

```
git status
git diff
```

and if you are satisfied with the changes, store them in the git version control system by doing

```
git commit -a
```

and writing a suitable commit message

1.2 Downloading this week's distribution

Once you have successfully saved your changes from last week, you can get my updates by doing

```
git pull
```

which *should* automatically merge in new content. After the `git pull` command, you should have a new directory containing this week's material (named `08/`) alongside the existing directories.

2 Hash tables

2.1 Data structure

Implement a hash table, using

$$h_{a,c,m}(x) = (ax + c) \bmod m$$

as the hash and reduction function. The client of the hash table will specify the parameters a and c, and the capacity of the storage table m. You **must** keep the `buckets` member variable `public` and use values of the hash function h as indexes into that array; in a real implementation `buckets` would be declared as `private`, but it needs to be exposed for testing purposes. (You may of course add other `private` member variables).

2.2 Basic operations

Implement the `insert` and `find` operations, for **strictly positive** (*i.e.* non-zero and non-negative) keys. Collision resolution for this hash table implementation uses linear probing: if a hash bucket is occupied on insert, move on to the next one until there is an unoccupied one to store the key in.

Also, implement the `loadFactor` function, which should return as a float the fraction of total hash buckets that are occupied.

Once these three methods are correctly implemented, the tests distributed along with the lab bundle should pass.

2.3 Further work

As it stands, your implementation of a hash table is incomplete compared with a fully-functional one:

- it doesn't support `delete`;
- it has a fixed capacity, rather than being able to expand beyond its initial size;
- its collision resolution strategy is not best-of-breed.

Extend your implementation to rectify these or other problems. Be aware that for the upcoming submission of this work, you will need to be able to support the basic operations above precisely; any work that you do here (including adapting the collision resolution strategy) must not detract from being able to support `find`, `loadFactor` and `insert` with a simple linear probing collision strategy.

3 List visualiser

3.1 Submission

You must submit your `ListVisualiser` source code, along with any auxiliary files it needs to build (*e.g.* a header file) but **not** your `SLList` implementation, to the List Visualiser Activity on learn.gold by **16:00** on **Friday 30th November**.

You should include in your submission, which can be a ZIP file, any instructions for compilation or use of your code which would be helpful to another member of the class, as they will need to be able to compile and use your code in order to assess it and provide you with feedback.

Failure to submit your source code will lead to an **automatic mark of zero** for both the submission **and** the assessment portion of this activity. There **no extensions** to the deadline of 16:00 on Friday 30th November.