lab_hash

Hellish Hash Tables

Due: Mar 31, 23:59 PM

Doxygen | Lab handout | Lab slides

Assignment Description

In this lab you will be implementing functions on hash tables with three different collision resolution strategies — separate chaining, linear probing, and double hashing. These hash tables serve an implementation of the dictionary abstract data type.

Lab Insight

Hashing is very powerful as it enables us to build data structure like hash tables and maps. On top of which, there are variations of hashing that can be used to help encrypt data. If you are interested in learning more about the applications of hashing, you can take CS 498 Applied Cryptography, CS 461 Computer Security I, and CS 463 Computer Security II.

Getting Set Up

From your CS 225 git directory, run the following on EWS:

```
git fetch release
git merge release/lab_hash -m "Merging initial lab_hash files"
```

If you're on your own machine, you may need to run:

```
git fetch release
git merge --allow-unrelated-histories release/lab_hash -m "Merging initial lab_hash files"
```

Upon a successful merge, your lab_hash files are now in your `lab_hash` directory.

The code for this activity resides in the `lab_hash/` directory. Get there by typing this in your working directory:

```
cd lab_hash/
```

ℹ If you want to speed up compile time on a `make`, try using `make -j <target>`, ie `make -j test`

Notes About `list` Iterators

When you are working with the Separate Chaining Hash Table, you will need to iterate over the linked list of a given bucket. Since the hash tables are templatized, however, this causes us a slight headache syntactically in C++. To define a `list` iterator on a given bucket, you will need to declare it as follows:

```
typename list< pair<K,V> >::iterator it = table[i].begin();
```

If you use the `list::erase()` function, be advised that if you erase the element pointed to by an iterator that the parameter iterator is no longer valid. For instance:

```
typename list< pair<K,V> >::iterator it = table[i].begin();
table[i].erase(it);
it++;
```

is invalid because `it` is invalidated after the call to `erase()`. **So, if you are looping with an iterator, remember a `break` statement after you call `erase()`!**

Separate Chaining Hash Table

Open your `schashtable.cpp`. In this file, several functions have not been implemented—your job is to implement them.

`insert`

- `insert`, given a `key` and a `value`, should insert the `(key, value)` pair into the hash table.
- You do not need to concern yourself with duplicate keys. When in client code and using our hash tables, the proper procedure for updating a key is to first remove the key, then re-insert the key with the new data value.
- Here is the Doxygen for `insert`.

`find`

- given a `key`, should return the corresponding `value` associated with that key
- Here is the Doxygen for `find`.

`remove`

- Given a key, remove it from the hash table.
- If the given key is not in the hash table, do nothing.
- You may find the Doxygen for `remove` helpful.

`resizeTable`

- This is called when the load factor for our table is $ge0.7$.
- It should resize the internal array for the hash table. Use the return value of `findPrime` with a parameter of double the current size to set the size. See other calls to `resize` for reference.
- Here is the Doxygen for `resizeTable`.

Linear Probing Hash Table

Open your `lphashtable.cpp`. In this file, you will be implementing the following functions.

`insert`

- `insert`, given a `key` and a `value`, should insert the `(key, value)` pair into the hash table.
- Remember the collision handling strategy for linear probing! (To maintain compatibility with our outputs, you should probe by moving forwards through the internal array, not backwards).
- You do not need to concern yourself with duplicate keys. When in client code and using our hash tables, the proper procedure for updating a key is to first remove the key, then re-insert the key with the new data value.
- Here is the Doxygen for `insert`.
- You MUST handle collisions in your `insert` function, or your hash table will be broken!

`findIndex`

- given a `key`, should return the corresponding `index` associated with that key
- Here is the Doxygen for `findIndex`.

`remove`

- Given a key, remove it from the hash table.
- If the given key is not in the hash table, do nothing.
- You may find the Doxygen for `remove` helpful.

`resizeTable`

- This is called when the load factor for our table is $ge0.7$.
- It should resize the internal array for the hash table. Use the return value of `findPrime` with a parameter of double the current size to set the size. See other calls to `resize` for reference.
- Here is the Doxygen for `resizeTable`.

Double Hashing Hash Table

Open your `dhhashtable.cpp`. In this file, you will be implementing the following functions.

`insert`

- `insert`, given a `key` and a `value`, should insert the `(key, value)` pair into the hash table.
- Remember the collision handling strategy for double hashing! (To maintain compatibility with our outputs, you should probe by moving forwards through the internal array, not backwards).
- You do not need to concern yourself with duplicate keys. When in client code and using our hash tables, the proper procedure for updating a key is to first remove the key, then re-insert the key with the new data value.
- Here is the Doxygen for `insert`.
- You MUST handle collisions in your `insert` function, or your hash table will be broken!

`findIndex`

- given a `key`, should return the corresponding `index` associated with that key
- Here is the Doxygen for `findIndex`.

`remove`

- Given a key, remove it from the hash table.
- If the given key is not in the hash table, do nothing.
- You may find the Doxygen for `remove` helpful.

Committing Your Code

Guide: How to submit CS 225 work using git

Grading Information

The following files (and ONLY those files!!) are used for grading this lab:

- `dhhashtable.cpp`
- `lphashtable.cpp`
- `schashtable.cpp`

If you modify any other files, they will not be grabbed for grading and you may end up with a “stupid zero.”