HashTable

CMS 230, Fall 2017

Due Wednesday, November 1, at 11:59 PM

Description

Implement a hash table using structures in C.

- Your table must support initialization, insert, lookup, remove, and print operations
- The keys and values in the table will both be of type `char *`. The keys will be null-terminated strings.

Use the included `hashtable.c` as a starting point. The `hashtable.h` header contains function prototypes and structure definitions.

This project will let you practice using memory allocation and structure to implement a non-trivial data structure. Along the way, you'll get to work with hash function (always an important topic) and arrays of pointers.

Background

Recall that a hash table is used to store key-value pairs and allows for average $O(1)$ insert and lookup times if the load on the table is not too high.

A chained hash table consists of an array of *buckets*. Each bucket contains a linked list of nodes that store the key-value pairs.

To insert into the table, calculate an integer from the key and use it to determine the bucket that should hold the new key-value pair, as in the following pseudo-C implementation:

```
void hashtableInsert(HashTable *h, char *key, char *value) {
  unsigned long int hashCode = hash(key);
  int b = hashCode % h->size;

  // Insert key-value pair into h->buckets[b]
}
```

You must implement functions to insert, lookup, and remove from the table.

Rehashing. Recall that many hash table implementations, like Java's `HashMap`, track the load on the table as items are inserted and removed, where the load is defined to be the average number of items per bucket. If the load factor becomes too high—say, greater than .75—the implementation creates a new, larger table and rehashes all of the entries. **You do not need to implement load-based rebalancing**.

Duplicates. The table can contain duplicate keys. A lookup or remove operation will always be satisfied by the first matching key that it encounters.

Hash Function. `hashtable.c` contains a `hash` function that implements Java's basic `String` hashing function. Don't modify `hash`—if you do, your program will probably hash everything to different buckets and your outputs won't match the tests' expected outputs.

Testing and Grading

Use the included Python script `test.py` to test your code. The fraction of tests you pass determines your grade.

Note that `hashtable.c` does not contain a `main`. Instead, your program will be combined with another file in the `tests` directory to create an executable. The test program captures the output of the executable and compares it the expected output. You may, of course, look at the C source files in the test directory to understand what they're doing, but don't modify any of them.

The project includes a `Makefile`. Edit the file to uncomment the build commands for each test as you complete your implementation.