Hash Table Implementation

Homework #9

By

John Robertson

CS 303 Algorithms and Data Structures

October 27, 2019

1. **Problem Specification**

This assignment requires implementing a hash table, thus reimplementing Python's dictionary type, as well as testing it on a real-world application.

2. **Program Design**

This program has three files: a pre-made HashEntry.py, a Hashmap.py for which stubs had to be filled, and a driver.py to test HashMap. The HashMap works by enforcing a bound on the keys for the data that it stores without actually changing the keys, only calculating internally where to put and later find them.

The following steps were required to develop this program:

* Fill out the HashMap stub
* Edit HashEntry to make debugging easier by giving it a string representation
* Write the driver
* Debug HashMap
* Debug the driver

For the record, I wrote a script to count the number of lines in UPC.csv, but it is too trivial to include in the submission.

3. **Testing Plan**

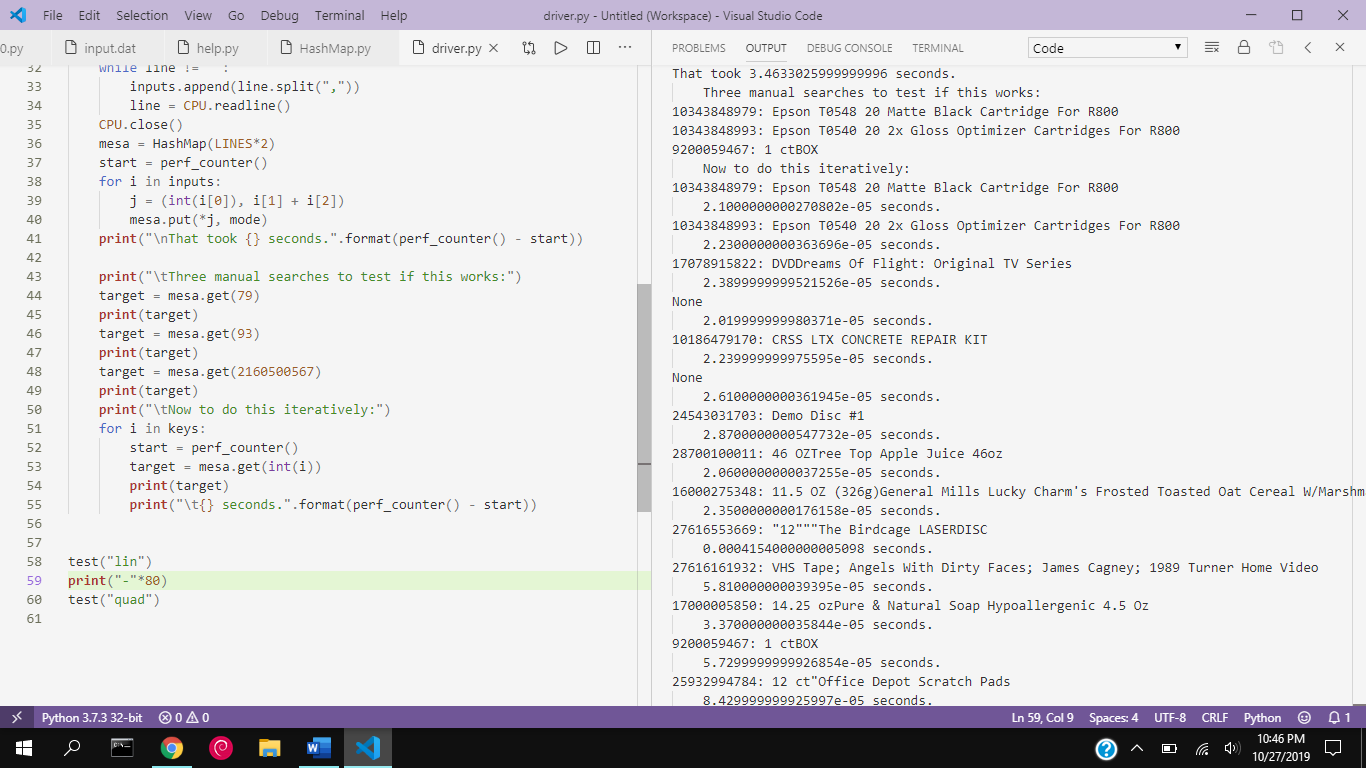
The HashMap class was tested by testing the effectiveness of giving it one value, using linear\_probe to insert, using linear search to get, using quadratic probe to insert, using quadratic search to get, using the search calculation from the problem text, using the same calculation to get, and finally testing for arbitrary manual inputs.

4. **Test Cases**

HashMap.py tests each kind of probe and get method on a hard-coded list and then allows arbitrary testing input for later. driver.py tests these same methods on input.dat before testing a subset of that on the shuffled UPC file. It tests three manual searches on keys from input.dat to get the items from UPC.csv before testing every key from input.dat. There was no time to finally fix up get for custom, nor to fix the bugs in returning None or the wrong value.

5. **Analysis and Conclusions**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Searched Key | Expected Linear (s) | Expected Quadratic (s) | Linear Time (s) | Quadratic Time (s) | Done as expected? |
| 79 | 0.001 | 0.0005 | 2.1e-05 | 1.589e-05 | No and no (both faster) |
| 93 | 0.001 | 0.0005 | 2.23e-05 | 1.66e-05 | No and no (both faster) |
| 123 | 0.001 | 0.0005 | 2.389e-05 | 1.49e-05 | No and no (both faster) |
| 161 | 0.001 | 0.0005 | 2.019e-05 | 1.38e-05 | No and no (both faster) |
| 2140000070 | 0.001 | 0.0005 | 2.239e-05 | 1.609e-05 | No and no (both faster) |
| 2140118461 | 0.001 | 0.0005 | 2.61e-05 | 1.49e-05 | No and no (both faster) |
| 2144209103 | 0.001 | 0.0005 | 2.87e-05 | 1.849e-05 | No and no (both faster) |
| 2144622711 | 0.001 | 0.0005 | 2.06e-05 | 8.129e-05 | No and no (both faster) |
| 2147483647 | 0.001 | 0.0005 | 2.35e-05 | 2.609e-05 | No and no (both faster) |
| 2158242769 | 0.001 | 0.0005 | 0.0004154 | 2.54e-05 | No and no (both faster) |
| 2158561631 | 0.001 | 0.0005 | 5.81e-05 | 6.759e-05 | No and no (both faster) |
| 2158769549 | 0.001 | 0.0005 | 3.37e-05 | 0.0001186 | No and yes (former faster than expected) |
| 2160500567 | 0.001 | 0.0005 | 5.729e-05 | 2.25e-05 | No and no (both faster) |
| 2172307284 | 0.001 | 0.0005 | 8.429e-05 | 1.879e-05 | No and no (both faster) |
| 2177000074 | 0.001 | 0.0005 | 2.939e-05 | 6.1e-05 | No and no (both faster) |
| 2184000098 | 0.001 | 0.0005 | 2.119e-05 | 2.22e-05 | No and no (both faster) |
| 2187682888 | 0.001 | 0.0005 | 3.77e-05 | 6.91e-05 | No and no (both faster) |



The above data is likely wrong. For some reason, there are more outputs than there are keys per search, a bug which does not seem to show up in the HashMap.py tests. As I do not know how to implement the corresponding get for the custom hash function without an infinite loop, it remains unfinished. At the very least, I can conclude that a hash table in its half-baked implementation is of similar speed to the binary search tree.

6. **References**

Dr. Unan's slides

https: // stackoverflow.com/questions/21581085/how-to-allocate-array-size-in-python/21581165

<https://www.programiz.com/python-programming/anonymous-function>

My report for last week served as the direct basis for this one.