Project 5: HashMaps

Due: Friday Mar 30, 11:59 pm

1 Assignment Overview

For this project you will be implementing a hash map. Maps are associative data structures that pair key and value items together; by knowing the key it is possible to find the value item associated with the key. Keys within the map are unique, but values are not; multiple keys can map to the same value. Being a hash-based map, the expected time for most operations is essentially constant. This class is very similar to the Python `dict` class.

Once completed, you will use your hash map to find the number of occurrences of each word in a large document (the script to *Romeo and Juliet*). This is an important first step for word frequency analysis, an important tool in the field of data mining. Frequency analysis is also used in cryptography to try to break ciphers.

2 Assignment Deliverables

You must turn in completed versions of the following files:

- HashMap.py

Be sure to use the specified file name and to submit your files for grading via **D2L Dropbox** before the project deadline.

3 Assignment Specifications

Your task will be to complete the methods listed below:

- `__len__`
- `load`
- `__contains__`
- `__getitem__`
- `__setitem__`
- `__delitem__`
- `clear`
- `__iter__`
- `keys`

- `word_frequency`

Your hash map is an associative data structure with both keys and values. Most of the parameters are key objects, but `setitem` takes both a key and a value object. Each of the parameters makes it clear which parameters are keys and which are values.

`setitem` adds associations to your map while `delitem` removes associations. `contains` determines whether an association with the given key exists while `getitem` determines which value item is associated with a given key. `len` counts the number of associations (that is the number of key-value pairs); do not count the key and the value separately. `clear` removes all associations from the map. `iter` enumerates all of the *key-value pairs*, not just the keys. Instead, the `keys` function returns a set of the keys contained in map. The order of the items for both functions is undefined.

Hash tables have a limited number of spaces available to store items. Your `load` function computes the load factor: the average number of items per slot. You are required to keep the load factor below 1 at all times. You may choose to impose a stricter maximum load factor. For your convenience, the constructor has an optional parameter that sets the maximum load factor. You *are* allowed to change the default value of this parameter.

Most of your functions should have an expected amortized constant runtime. The `clear`, `iter`, and `keys` may have a $O(n)$ runtime. You must use $O(n)$ space.

The final method, `word_frequency` is not a member of your hash map. Instead, this function takes in a sequence of words and returns a hash map that maps each unique word to the number of times that word appears in the document. You may assume that the sequence has already been sanitized for your convenience; you do not have to worry about removing punctuation or converting cases (you may match the strings as they are given to you). This function should run in $O(n)$ time.

You should include comments where appropriate. In particular, you should describe the method used for handling hash collisions. You must also add documentation for each of the above methods (as well as for any helper functions created) in the same style as for the pre-defined methods.

4 Assignment Notes

- Points will be deducted if your solution has warnings of any type.
- You have been supplied with stubs for the required methods. You must complete these methods to complete the project. You may add more functions than what was provided, but **you may not modify the signatures of the provided methods**. You may, however, change the default maximum load factor provided to your constructor.
- You do not have to worry about error checking for valid input. You may assume that the supplied reader function correctly reads the input file.
- You **do** have to worry about accessing elements from an empty map.
- You must be able to handle duplicate elements. Duplicate keys are not allowed (but duplicate values are). Newer associations for a given key replace the older ones.
- Implementations for `bool`, `is_empty`, and `repr` have been provided. Note that these rely on the `len` and `iter` functions, so you may want to complete these functions early.
- You have been provided with some unit tests that can be used to assess your solution. There are additional test cases that will be kept hidden.
- It is your responsibility to check for potential edge cases. The supplied unit tests do not account for all possible valid inputs
- For the `iter` method, you may want to use the `yield` keyword. Remember that the `iter` function enumerates key-value pairs, not just the keys.

- There are 26 words in the phonetic alphabet: one for each letter. Each word is mapped to the number of characters in the string.
- 'x-ray' has 5 characters.
- 'romeo' appears in the play 293 times and 'juliet' appears 176 times. There are 3741 unique words.
- You may not use the `dict` class (the built in dictionary type). You *are* allowed to use the `list` class and all of its member functions.
- You may assume that the key objects are hashable. You can use the built-in hash function, which returns an integer from such an object (note that these integers are completely unrelated to the size of your map). The value types are not guaranteed to have any kind of special property.