CPE 202 Winter 2020

Lab 8: Implementing Hash Tables with different Collision Resolutions

For this lab you will explore implementations of hash tables with the following three collision resolution strategies:

1. Separate Chaining
    a. You will use the linked_list.py you have implemented (Make modifications if necessary) to implement the separate chaining. **You can not use the python list as the chain.**
2. Linear Probing
    a. h + s
3. Quadratic Probing
    a. $h + f(i), f(i) = i^2$

In this lab, you are going to store stop words in your hash tables as a test. Stop words are high-frequency words like the, to and also that we sometimes want to filter out of a document before further processing. Stopwords usually have little lexical content, and their presence in a text fails to distinguish it from other texts.

You are going to import stop words from supplied **stop_words.txt** which contains stop words separated by a space. You are required to read the stop words contained in the file and create a hash table of the stop words so that you can use the table to determine if a word is a stop word or not with ‘in’ operator: e.g. is_stopword = ‘the’ in stop_words. You are going to use each stop word as a key and 0 as its value and store the word - 0 pairs as key - value pairs.

Use the hash function for hashing string presented in a lecture:

```
def hash_string(string, size):
    hash = 0
    for c in string:
        hash = (hash * 31 + ord(c)) % size
    return hash
```

Define classes **HashTableSepchain** that implements the separate chaining (use linked list as the list), **HashTableLinear** that implements the linear probing, and **HashTableQuadratic** that implements the quadratic probing.

- **__init__(self, table_size=11):** this function takes no parameters and returns a hash table object, having initialized an empty hash table ([None] * size). The table_size parameter (default value of 11) is the starting size of the table (number of “slots” in the table). This function should initialize the hash table (use a Python list) and any other instance variables used in your hash table class (e.g. num_items, num_collisions).

- **put(self, key, data):** this function takes a key, and an item. **Keys are string values.** The function will insert the key-item pair into the hash table based on the hash value of the key (use the hash_string function shown above). If the key-data pair being inserted into the hash table is a duplicate key, the old key-item pair will be replaced by the new key-item pair. If the insert would cause the load factor of the hash table to become greater than a predetermined threshold value (**1.5 for the separate chaining**, **0.75 for the linear and quadratic probing**), the number of slots in the hash table should be increased to twice the old number of slots, plus 1 (new_size = 2*old_size + 1). After creating the new hash table, the items in the old hash table need to be rehashed into the new table. In this lab, the key and the data are the same string, which is a stop word. Your linked list implementation needs to store a string as its payload.
- **get(self, key):** this function takes a key and returns the item (key, item) pair from the hash table associated with the key. If no key-item pair is associated with the key, the function raises a **LookupError** exception.
- **contains(self, key)**: this function returns True if the key exists in the table, otherwise returns False.
- **remove(self, key):** this function takes a key, removes the key-item pair from the hash table and returns the key-item pair. If no key-item pair is associated with the key, the function raises a **LookupError** exception.
- **size(self):** this function returns the number of key-item pairs currently stored in the hash table.
- **load_factor(self):** this function returns the current load factor of the hash table.
- **collisions(self):** this function returns the number of collisions that have occurred during insertions into the hash table. A collision occurs when an item is inserted into the hash table at a location where one or more key-item pairs has already been inserted. When the table is resized, do not increment the number of collisions unless a collision occurs when the new key-item pair is being inserted into the resized hash table.
- Overload following two builtin methods to enable ‘in’ and ‘[]’ operators:

```
def __getitem__(self,key):

   return self.get(key)



def __setitem__(self,key,data):

   self.put(key,data)
```

```
def __contains__(self, key):
    return self.contains(key)
```

In addition to classes described above, implement a function **import_stopwords(filename, hashtable)** outside of your hash table classes. The argument filename is the name of a file containing the stop words. The argument hashtable is an object of HashTable classes which can be either an implementation of the separate chaining, linear probing, or quadratic probing. The function should return a hash table object even though the object is mutable.

You are free to create helper functions. Try to reuse code by creating common helper functions that can be commonly used by the three implementations of hash tables.

DO NOT FORGET TO IMPLEMENT ALL THREE BOILERPLATE METHODS IN ALL CLASSES YOU USE.

Submit a file **hashtables.py** containing the above, **linked_list.py** containing your linked list implementation and a file **hashtables_tests.py** containing your set of test cases. Zip all three files as one zip file, and submit it to the grader. After getting your work graded by the grader, submit your work to Polylearn as well.