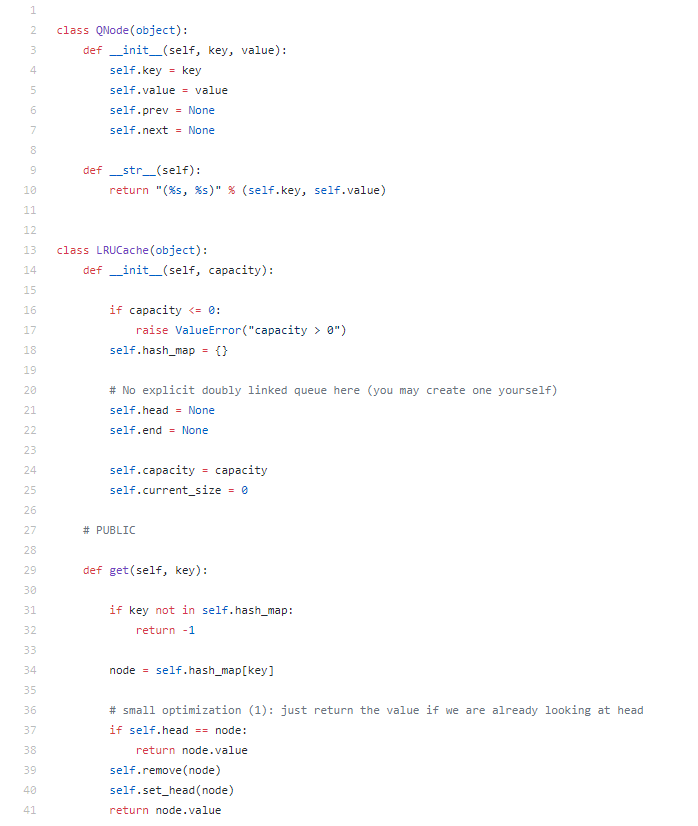
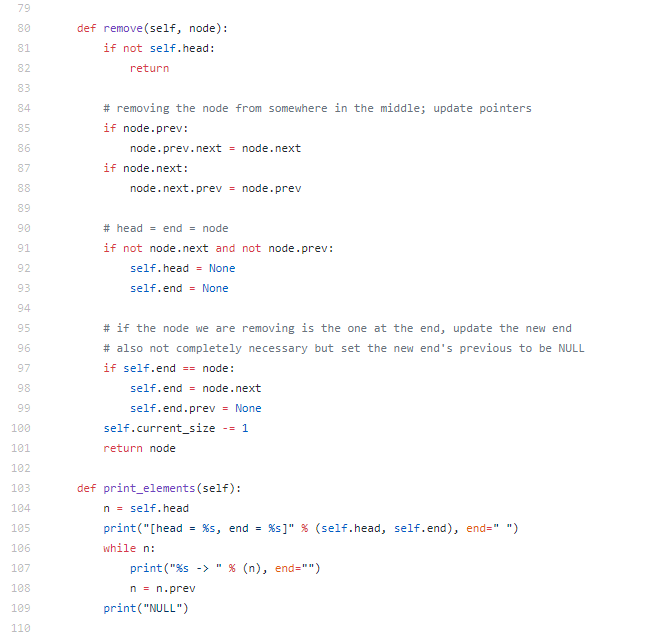
|  |  |
| --- | --- |
| Lab 5 Report | Antonio Zavala Anaya  UTEP ID 80622587 |

|  |
| --- |
| Project Name: Lab\_5 (GitHub Repo)November 2019 |
| * Purpose Problem A: Write a Python 3 program that:   + Design and implement a data structure called [Least Recently Used (LRU)](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU) cache. This data structure supports the following operations,     - get(key) - Gets the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.     - put(key, value) - Insert or replace the value if the given key is not already in the cache. When the cache reaches its maximum capacity, it should invalidate the **least recently used item** before inserting a new item.     - size() – Returns the number of key/value pairs currently stored in the cache.     - max\_capacity() – Returns the maximum capacity of the cache   All operations MUST run in O(1) time complexity. You are free to uses Python’s set and/or dictionary data structures. If you need to use a doubly linked list (hint), you need to code it yourself.    * Purpose Problem B: Write a Python 3 program that:   + Given a list of words (strings), print the *k* most frequent elements in descending order. When you print, you have to print the word and its number of occurrences in the list.   + If two words have the same frequency, the word with the lower alphabetical order comes first. Use a heap to receive credit. |
| * Process:   + Investigate purpose, uses and process of the Least recently used data structure.   + Create a doubly linked list with nodes containing next, previous pointers and the respective data of the node.   + Create a LRU class with the methods get, put, size, and max capacity.   + For the get method, return the value of the item and then move that item as the most recent used item.   + For the put method, check if the key is not in the hash map, if not, the item is added in the hash map and set the item as the head of the doubly linked list and remove the tail if full. If the key is in the hash map, the value is changed. |
| * Files used that will be used: * LRUCache.py * Problem 2.py |

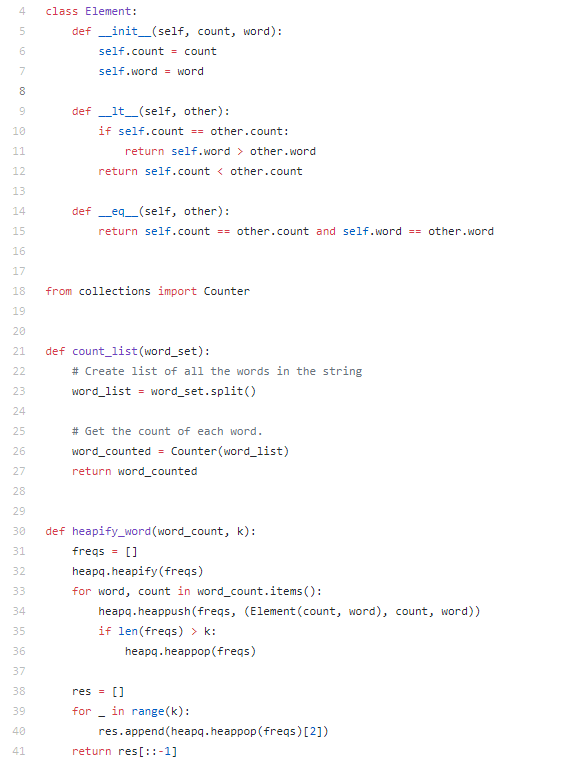
* Lab\_5 program Test cases
* **Problem A**

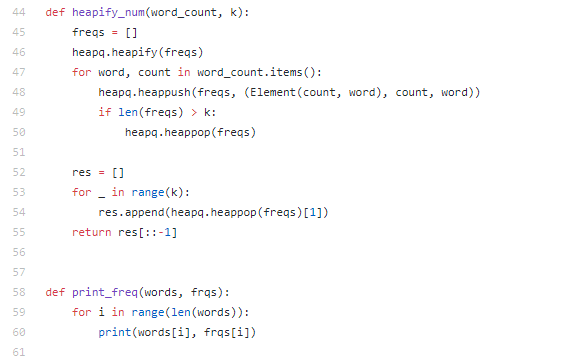






* **Problem B**





* Test Cases
* Conclusion

With this lab I learned first the use of maps and hash tables in a different way such as in LRU Caches and how to make them as well as how to implement them at constant time by making the LRU Cache.

This helped me understand better the efficiency in time complexity. From problem B the use of dictionaries and counters helped me to find easier the repeated words in the text file and organize them seeing which repeated the most as well as to implement libraries that can help such as by making the heap.