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) 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

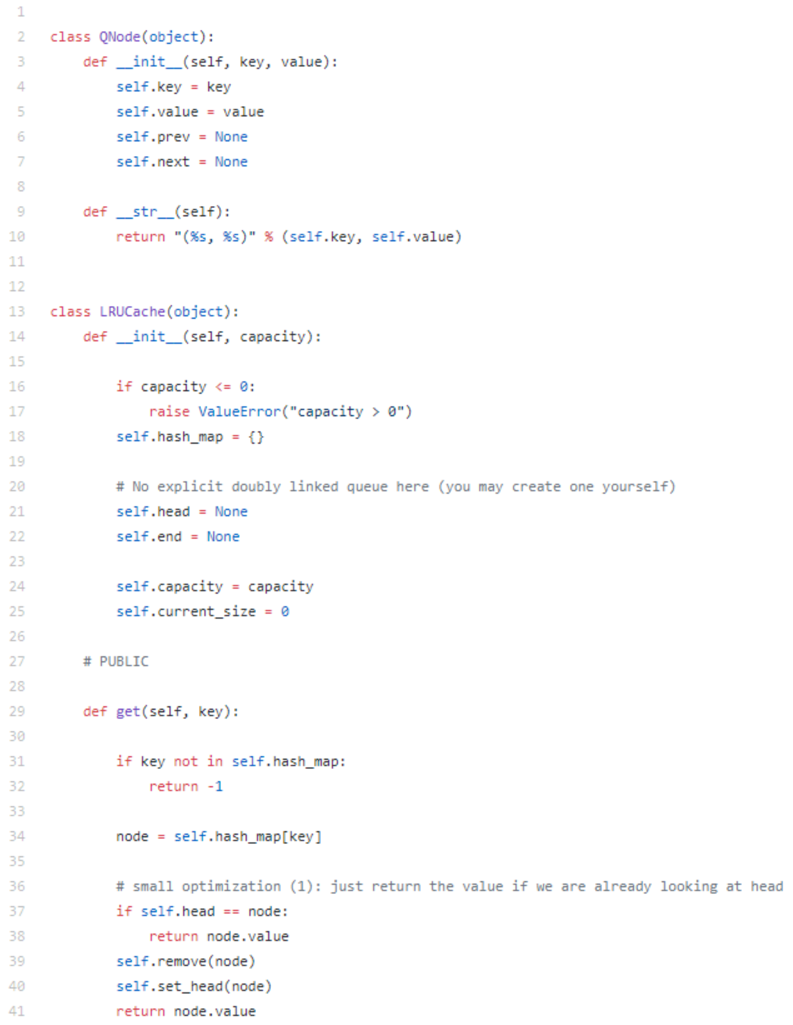
All operations MUST run in O(1) time complexity. You are free to uses Python's set and/or dictionary data structures. If you need to use a doubly linked list (hint), you need to code it yourself.

- 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.

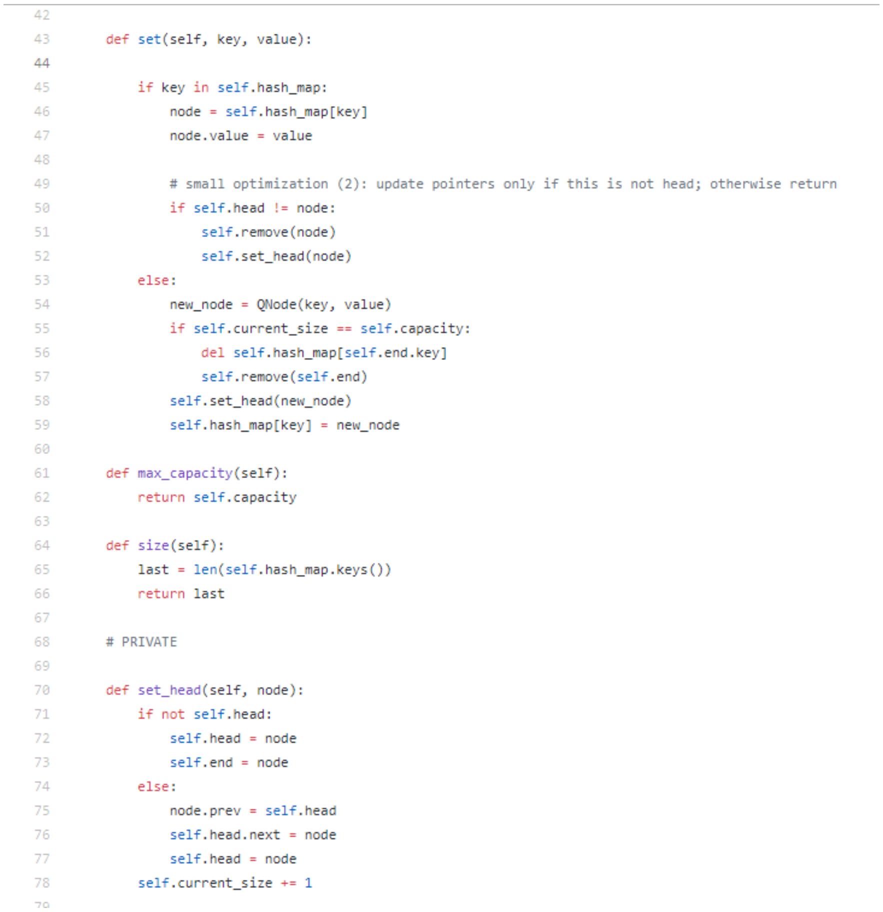
- Process:
    - Investigate purpose, uses and process of the Least recently used data structure.
    - Create a doubly linked list with nodes containing next, previous pointers and the respective data of the node.
    - Create a LRU class with the methods get, put, size, and max capacity.
    - For 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

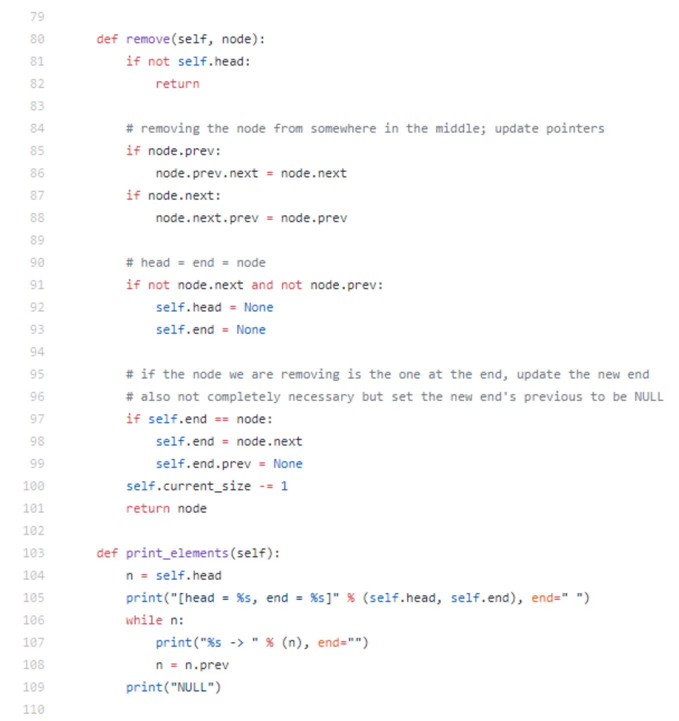
- Lab_5 program codes
  - **Problem A**



```

class QNode(object):
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.prev = None
        self.next = None

    def __str__(self):
        return "(%s, %s)" % (self.key, self.value)


class LRUCache(object):
    def __init__(self, capacity):

        if capacity <= 0:
            raise ValueError("capacity > 0")
        self.hash_map = {}

        # No explicit doubly linked queue here (you may create one yourself)
        self.head = None
        self.end = None

        self.capacity = capacity
        self.current_size = 0

    # PUBLIC

    def get(self, key):

        if key not in self.hash_map:
            return -1

        node = self.hash_map[key]

        # small optimization (1): just return the value if we are already looking at head
        if self.head == node:
            return node.value
        self.remove(node)
        self.set_head(node)
        return node.value
```

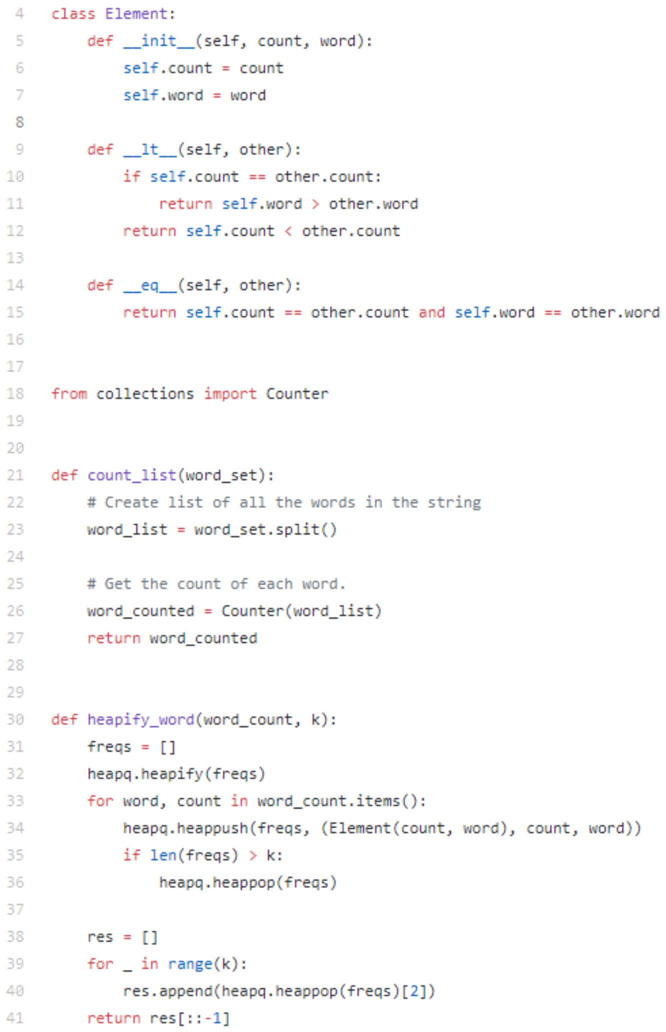


```python

    def set(self, key, value):

        if key in self.hash_map:
            node = self.hash_map[key]
            node.value = value

            # small optimization (2): update pointers only if this is not head; otherwise return
            if self.head != node:
                self.remove(node)
                self.set_head(node)
        else:
            new_node = QNode(key, value)
            if self.current_size == self.capacity:
                del self.hash_map[self.end.key]
                self.remove(self.end)
            self.set_head(new_node)
            self.hash_map[key] = new_node

    def max_capacity(self):
        return self.capacity

    def size(self):
        last = len(self.hash_map.keys())
        return last

    # PRIVATE

    def set_head(self, node):
        if not self.head:
            self.head = node
            self.end = node
        else:
            node.prev = self.head
            self.head.next = node
            self.head = node
        self.current_size += 1
```

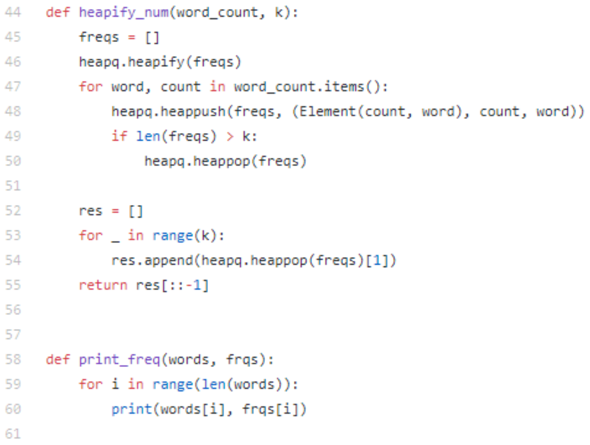


```python

    def remove(self, node):
        if not self.head:
            return

        # removing the node from somewhere in the middle; update pointers
        if node.prev:
            node.prev.next = node.next
        if node.next:
            node.next.prev = node.prev

        # head = end = node
        if not node.next and not node.prev:
            self.head = None
            self.end = None

        # if the node we are removing is the one at the end, update the new end
        # also not completely necessary but set the new end's previous to be NULL
        if self.end == node:
            self.end = node.next
            self.end.prev = None
        self.current_size -= 1
        return node

    def print_elements(self):
        n = self.head
        print("[head = %s, end = %s]" % (self.head, self.end), end=" ")
        while n:
            print("%s -> " % (n), end="")
            n = n.prev
        print("NULL")

```

- ## Problem B



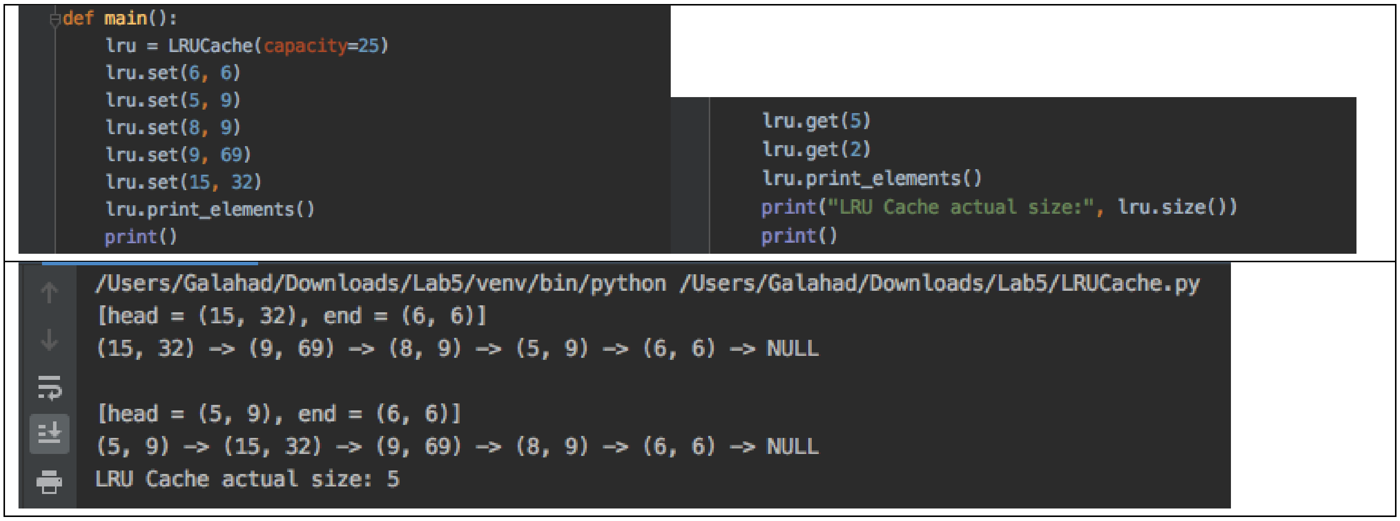
```python
class Element:
    def __init__(self, count, word):
        self.count = count
        self.word = word

    def __lt__(self, other):
        if self.count == other.count:
            return self.word > other.word
        return self.count < other.count

    def __eq__(self, other):
        return self.count == other.count and self.word == other.word


from collections import Counter


def count_list(word_set):
    # Create list of all the words in the string
    word_list = word_set.split()

    # Get the count of each word.
    word_counted = Counter(word_list)
    return word_counted


def heapify_word(word_count, k):
    freqs = []
    heapq.heapify(freqs)
    for word, count in word_count.items():
        heapq.heappush(freqs, (Element(count, word), count, word))
        if len(freqs) > k:
            heapq.heappop(freqs)

    res = []
    for _ in range(k):
        res.append(heapq.heappop(freqs)[2])
    return res[::-1]
```



```python
def heapify_num(word_count, k):
    freqs = []
    heapq.heapify(freqs)
    for word, count in word_count.items():
        heapq.heappush(freqs, (Element(count, word), count, word))
        if len(freqs) > k:
            heapq.heappop(freqs)

    res = []
    for _ in range(k):
        res.append(heapq.heappop(freqs)[1])
    return res[::-1]


def print_freq(words, frqs):
    for i in range(len(words)):
        print(words[i], frqs[i])

```

- Test Cases
  - Problem A

For problem A, I made a test case of making the cache capacity of 25. Then I set some numbers inside the cache. Then get two of them to see the changes as this would become the Least Recently Used nodes In addition, print the size of the cache as of now

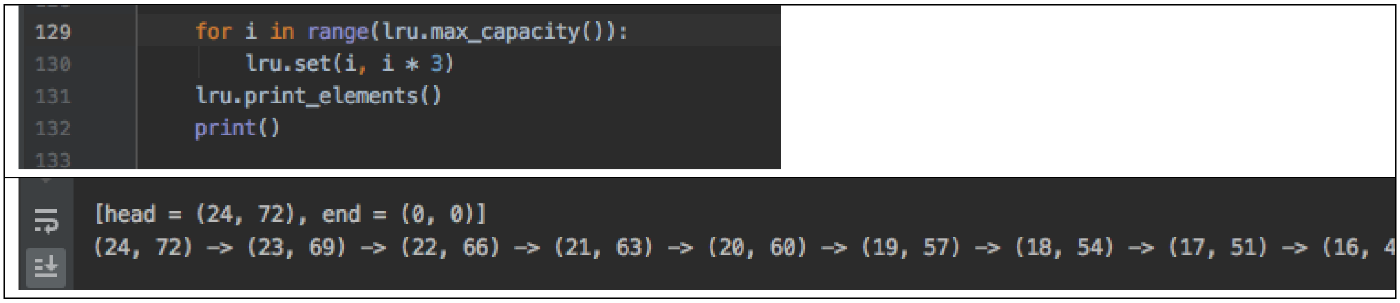


```
def main():
    lru = LRUCache(capacity=25)
    lru.set(6, 6)
    lru.set(5, 9)
    lru.set(8, 9)
    lru.set(9, 69)
    lru.set(15, 32)
    lru.print_elements()
    print()
```

```
    lru.get(5)
    lru.get(2)
    lru.print_elements()
    print("LRU Cache actual size:", lru.size())
    print()
```

```
/Users/Galahad/Downloads/Lab5/venv/bin/python /Users/Galahad/Downloads/Lab5/LRUCache.py
[head = (15, 32), end = (6, 6)]
(15, 32) -> (9, 69) -> (8, 9) -> (5, 9) -> (6, 6) -> NULL

[head = (5, 9), end = (6, 6)]
(5, 9) -> (15, 32) -> (9, 69) -> (8, 9) -> (6, 6) -> NULL
LRU Cache actual size: 5
```

Then I proceed to fill the whole cache using a for loop until it is full.

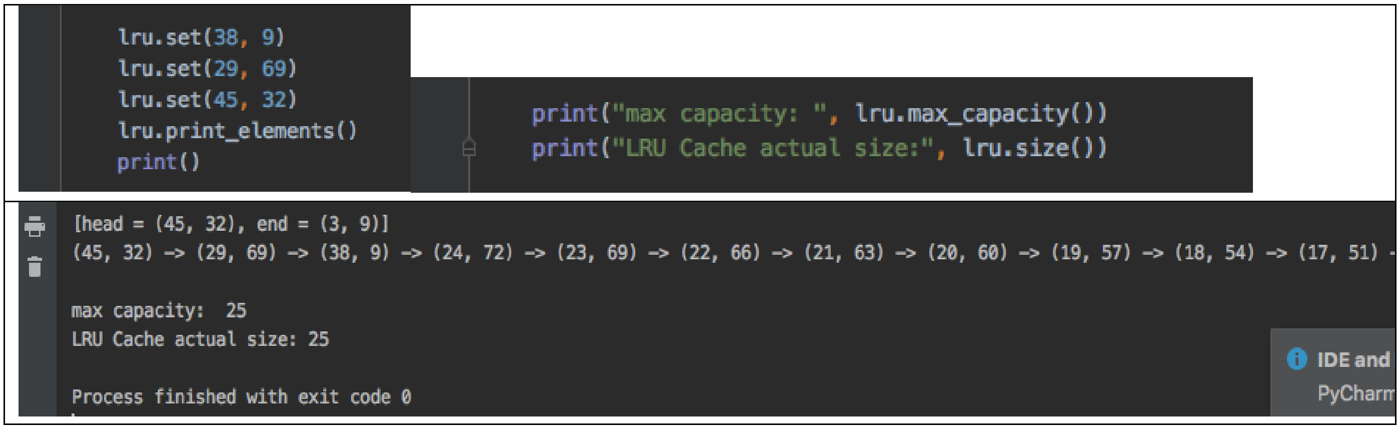


```
    for i in range(lru.max_capacity()):
        lru.set(i, i * 3)
    lru.print_elements()
    print()
```

```
[head = (24, 72), end = (0, 0)]
(24, 72) -> (23, 69) -> (22, 66) -> (21, 63) -> (20, 60) -> (19, 57) -> (18, 54) -> (17, 51) -> (16, 4
```

After it'ss done, some new elements are set so it replace the oldest ones inside the LRU

Print the newly edited cache and the max size as well as the current size of the cache



```
    lru.set(38, 9)
    lru.set(29, 69)
    lru.set(45, 32)
    lru.print_elements()
    print()
```

```
    print("max capacity: ", lru.max_capacity())
    print("LRU Cache actual size:", lru.size())
```

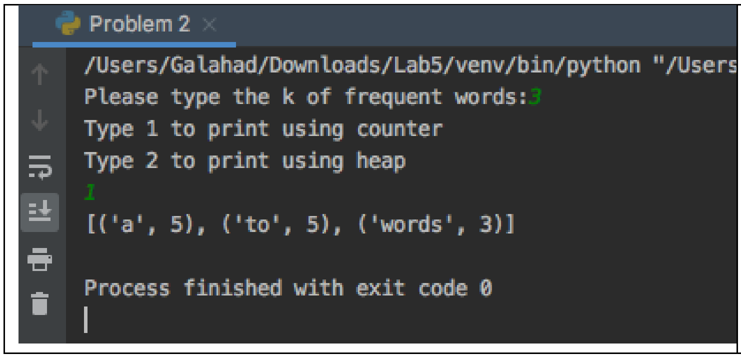
```
[head = (45, 32), end = (3, 9)]
(45, 32) -> (29, 69) -> (38, 9) -> (24, 72) -> (23, 69) -> (22, 66) -> (21, 63) -> (20, 60) -> (19, 57) -> (18, 54) -> (17, 51) -

max capacity:  25
LRU Cache actual size: 25

Process finished with exit code 0
```

- Problem B

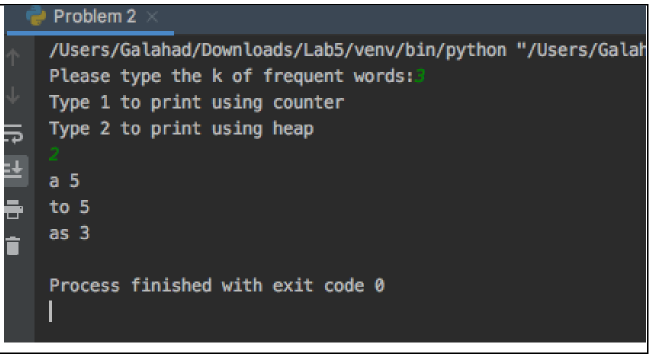
Word set:

This is a test to find the most repeated words in a series of strings to count
it will test many words, as those that can be found in a book.
They sometime hurt and words sometime inspire
Also sometime fewer words convey more meaning than a bag of words.
And books are to the mind as flint is to a sword, as it to keep its edge.

- K: 3

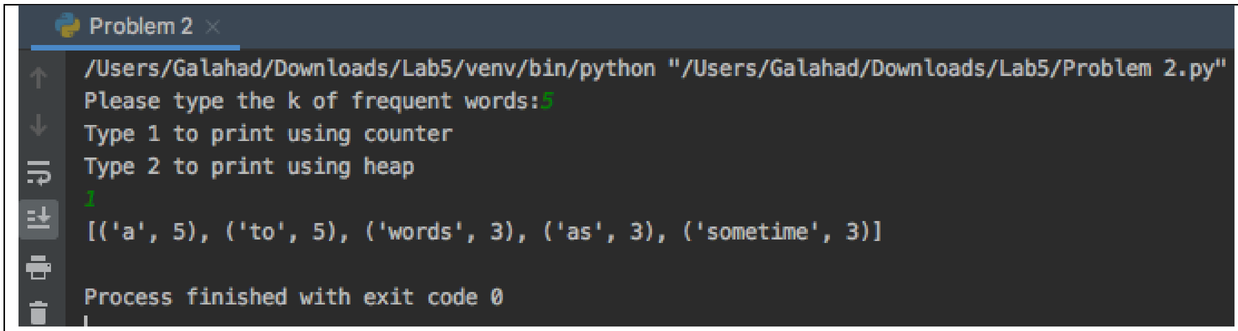


```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users
Please type the k of frequent words:3
Type 1 to print using counter
Type 2 to print using heap
1
[('a', 5), ('to', 5), ('words', 3)]

Process finished with exit code 0
```

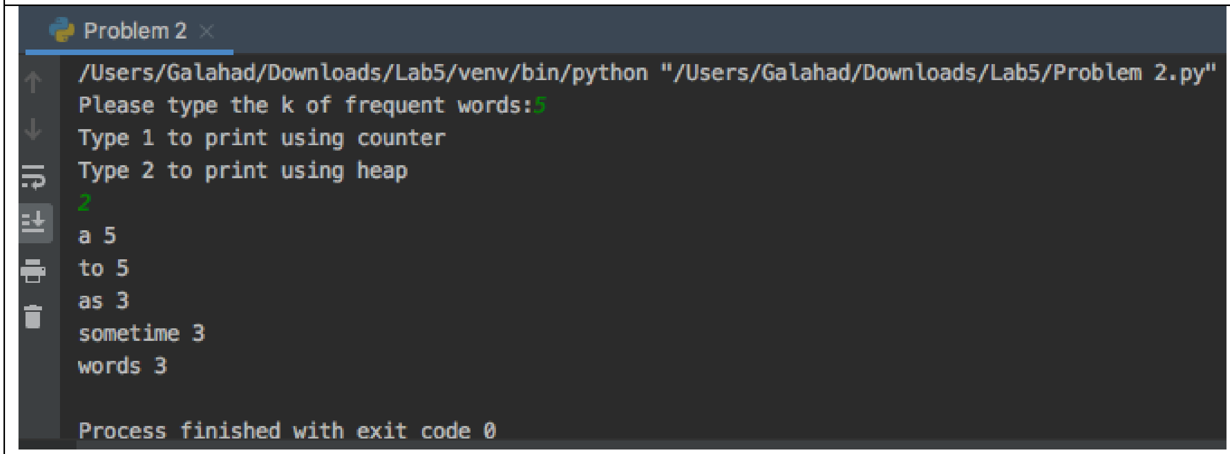


```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users/Galah
Please type the k of frequent words:3
Type 1 to print using counter
Type 2 to print using heap
2
a 5
to 5
as 3

Process finished with exit code 0
```

- K: 5

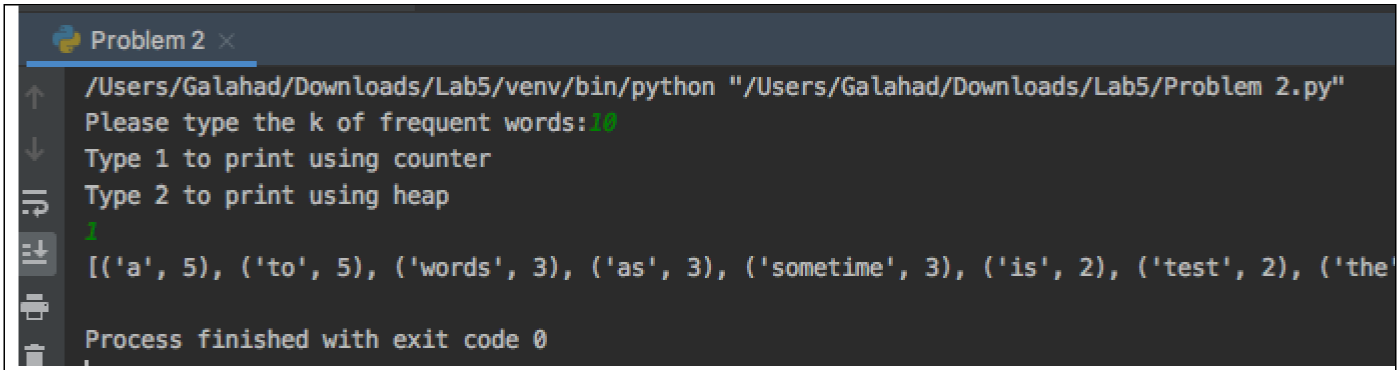


```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users/Galahad/Downloads/Lab5/Problem 2.py"
Please type the k of frequent words:5
Type 1 to print using counter
Type 2 to print using heap
1
[('a', 5), ('to', 5), ('words', 3), ('as', 3), ('sometime', 3)]

Process finished with exit code 0
```

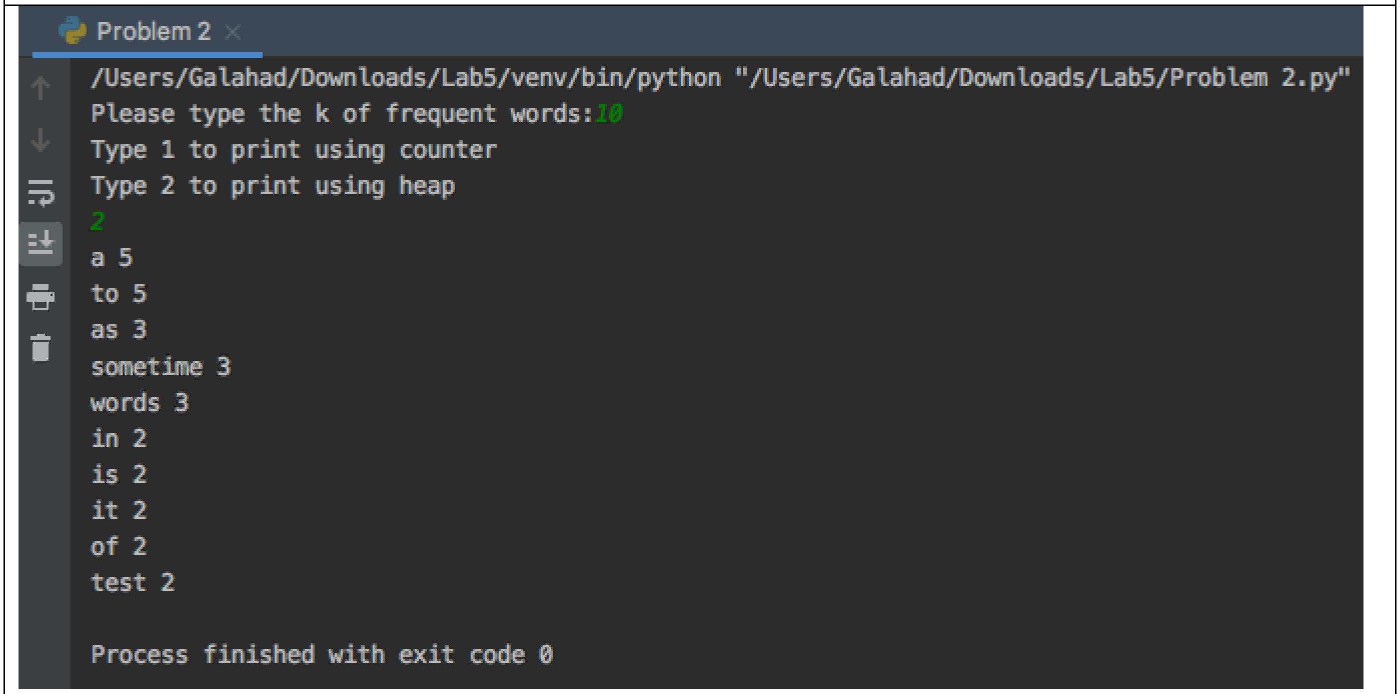


```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users/Galahad/Downloads/Lab5/Problem 2.py"
Please type the k of frequent words:5
Type 1 to print using counter
Type 2 to print using heap
2
a 5
to 5
as 3
sometime 3
words 3

Process finished with exit code 0
```

- K: 10



```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users/Galahad/Downloads/Lab5/Problem 2.py"
Please type the k of frequent words:10
Type 1 to print using counter
Type 2 to print using heap
1
[('a', 5), ('to', 5), ('words', 3), ('as', 3), ('sometime', 3), ('is', 2), ('test', 2), ('the'

Process finished with exit code 0
```



```
/Users/Galahad/Downloads/Lab5/venv/bin/python "/Users/Galahad/Downloads/Lab5/Problem 2.py"
Please type the k of frequent words:10
Type 1 to print using counter
Type 2 to print using heap
2
a 5
to 5
as 3
sometime 3
words 3
in 2
is 2
it 2
of 2
test 2

Process finished with exit code 0
```

- 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.