**CS 2302 - Data Structures**

**Fall 2019**

**Project 5**

**Overview**

For this lab, you need to solve the following problems:

**Problem A**

Your job is do 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.

**Problem B**

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.

**Part 1 - Due Thursday, November 14, 2019**

Upload the progress you have made. You need to have at least 50% of the lab done by this date.

**Part 2 - Due Tuesday, November 19, 2019**

Final due date (everything finished - code).

**Part 3- Due Friday, November 22, 2019**

Final due date (report)

**Rubric**

|  |  |  |  |
| --- | --- | --- | --- |
| **Criteria** | **Proficient** | **Neutral** | **Unsatisfactory** |
| **Correctness** | The code compiles, runs, and solves the problem. | The code compiles, runs, but does not solve the problem (partial implementation). | The code does not compile/run, or little progress was made. |
| **Space and Time complexity** | Appropriate for the problem. | Can be greatly improved. | Space and time complexity not analyzed |
| **Problem Decomposition** | Operations are broken down into loosely coupled, highly cohesive methods | Operations are broken down into methods, but they are not loosely coupled/highly cohesive | Most of the logic is inside a couple of big methods |
| **Style** | Variables and methods have meaningful/appropriate names | Only a subset of the variables and methods have meaningful/appropriate names | Few or none of the variables and methods have meaningful/appropriate names |
| **Robustness** | Program handles erroneous or unexpected input gracefully | Program handles some erroneous or unexpected input gracefully | Program does not handle erroneous or unexpected input gracefully |
| **Documentation** | Non-obvious code segments are well documented | Some non-obvious code segments are documented | Few or none non-obvious segments are documented |
| **Report** | Covers all required material in a concise and clear way with proper grammar and spelling. | Covers a subset of the required material in a concise and clear way with proper grammar and spelling. | Does not cover enough material and/or the material is not presented in a concise and clear way with proper grammar and spelling. |