Isaac Acosta | CS2302 | October 18, 2018

Lab 2

Option B

# Introduction

The objective of this project is to create a linked list of passwords from a text file. List must not have any duplicates and the following list must be sorted in some sorting algorithm.

Each node of the list will contain

1. A unique password stored as a string.
2. The number of times read password appears in the text file.
3. A pointer to the next node.

As for the sorting procedure. There will be two sorting algorithms used in this case, merge and bubble sorting. In overall, this project will be a 2 part process, as there will be 2 solutions applied for this instance.

# Design and Implementation

In overall, the Python file proceeds as follows when executed; it reads a text file (text file is expected to have a username on the left and a password on the right, having a tab spacing in between each other for every line. in a manner that it splits the username and passwords. It then creates a new text file with just the passwords from the previous text file. And it is this new text file that will be used for the remainder of the program.

Afterwards, it proceeds to execute Solution A, which is reading the text file, creating the linked list, and then bubble sort the list that will be the output of Solution A.

Secondly, Solution B follows similarly, only this time, the linked list is merge sorted. Also, the creation of the linked list is followed by a distinct procedure than Solution A.

Lastly, both solutions are to be printed out. To be more specific, the top 20 passwords being used is to be outputted on console after executing both A and B solutions.

**Linked list and Node**

To illustrate the Node class, it stores the password string, count times that the password appears as an integer, and a pointer to the next node. And for good practice, it has four more functions. And in these five functions, the purpose of two of them is to return the password string and the count times of the password’s appearance. A third function is to modify the count using an integer as a reference, and a fourth function is to increment the count by 1. The fourth functions will be useful to record the number of appearances every time the file reader

The Linked List class follows as is in general, it two Nodes. One is for the head and the other is for the Tail. The class is composed of eight functions. Four being for the general functionality the class, and the second four to make the linked list class more essential for the program as whole. The purpose of the first four functions are to insert into the linked list, get the length, print the list, and to search inside the linked list to add to the count in consequence of the Node’s string item to be found again in the text file.

* The insertion process uses a head and tail in so that when inserting into the list, it is from first to last, so that the linked list follows the same order as the passwords in the text file and not the reverse.
* The output list is also received an integer value x, if that integer is not -1, then the functions will just output the top x passwords used. Otherwise, if x is -1, then the whole list will be outputted.
* The search function return either a 0 or 1 as from of Boolean. If item was found, it returns 1, else it returns zero.

As for the second four functions, three are to merge sort the list, and one is to bubble sort the list. For the three that perform the merge sort, function mergeSortLinkedList() is to call on a second function, split\_list(), to split the list. And it continues to split every half recursively, until nothing can be split. And for every recursive call, split\_list() returns the two lists from its split process back to mergeSortLinkedList().Which in turn, it calls a third function, split\_list(), is called in order to perform a merge on every piece or half of the list until the first function finally comes through with its recursive calls that modify the list.

The last function of the class simply bubble sorts the list without using other functions, as opposed to the merge sort. The inserts on itself every time a bubble sort between two linked Nodes is performed.

**Implementation of Solution A**

In the manner that Solution A is undertaken by the program is by calling a method that returns a linked list of passwords that were merge sorted. And that method follows is designed as follows; It starts by reading the text file and initializing a Linked List. And for every line being read, it calls the search function, search(item) from the linked list class, if it returns false, a new Node is created in terms of the line in the text file and then added to the list, aside from this condition, no Node is created or added into the list. After the list is created and text file reaches its end, the solution functions then finally calls bubbleSortLinkedList() from the linked list class and returns the sorted list back to the Main.

**Implementation of Solution B**

In contrast to solution A, solution B uses a dictionary to check if the password string has already been read from the text file before. As far as the format of the dictionary is laid upon, it uses the password string as a key to represent a Node containing that password. So, every time a new password is found a new dictionary key is created, and for every password found for a second time, the line string is used as a key to find the node to increment the count by calling search(item). After the text file reaches its end, the contents in the dictionary are then inserted into a Linked list through for loop for every item in the Dictionary. Lastly, after the Linked list is populated, the function calls mergeSortedLinkedList() from the Linked List class is called to bubble sort the list. The sorted list is then returned back to the Main.

**Printing the top 20 passwords from for both Solutions**

In the Main, we have two lists, list\_A and list\_B, and both are finally defined once Solution A and Solution B functions are called. Now, for list\_A and list\_B to print out the top 20 used passwords, we just call list\_A.output\_list(20) and list\_B.output\_list(20).

**Reading the passwords from the initial text file.**

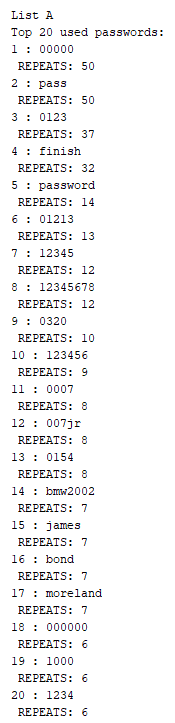
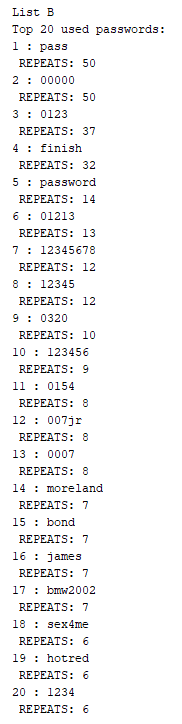
Because the format of the initial text file features a format where the usernames and passwords are in each line, and separated by a tab space, text\_file() manages to split each line into two strings, the tab spacing being the splitting factor. In the course that the function reads and splits each line, since an array of strings is created, in this case lim[“username”, “password”], the function only grabs the item in lim[1] and writes it in a new text file. The program has a faster running time when splitting the lines in a text file is only done once and just output into a separate text file with just the desired contents.

# Experimental Results

As it can be illustrated in the chart above, Solution A takes longer as the input value rises and the running time rises dramatically as compared to Solution B. It almost seems that Solution A’s running time rises exponentially compared to Solution B. This is mainly due to the sorting nature that each Solution has distinctively. Bubble Sort seems to take longer than merge sorting. And add merge sorting to traversing the linked list for every line in the text. It makes up for a long lasting process for large inputs.

**Console Outputs:**

(Next Page)

**8000 Passwords**

As it can be seen, these are sample outputs for an input of 8000 passwords. On the right, the list outputs two list for solutions A and B.

List A is the output of linked list, list\_A, and the list is output after it is populated from the text file and bubble sorted. The list then calls the output\_list(20) from the Linked List Node and uses an integer as a reference as how much to traverse the linked list, since the linked list is expected to be in descending order.

List B is the same, it is output after the Linked List is populated and sorted. Only this time, the linked list is merge sorted. The name of the linked list for solution B is list\_A, and it is output by calling the outout\_list(20) from linked list class.

# Conclusion

What I learned in this project was that sorting algorithms can vary in terms of running times. Also, that the format in which a linked list inserts a new node is crucial when sorting the list. Traversing the list vs using a dictionary hash to store values vary and impact on the running of a program in this nature. Not every algorithm has a linear growth in running time as the input size increases. It could be exponential, as we saw in solution A.

# Academic Honesty

I certify that this project is

entirely my own work. I wrote, debugged, and tested the code being presented, performed the

experiments, and wrote the report. I also certify that I did not share my code or report or

provided inappropriate assistance to any student in the class.