ÇUKUROVA UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT CEN556 Data Structures Lab.
Final Homework Report

23.01.2021

Buğra Sarıkaya
2016556506
İ.Ö.

Introduction:
The final project's purpose is measuring student's ability in handling big files by using lecture methods. I achieved success on every objective, including 1 million passwords, by using lecture methods and extra methods. For completing these objectives properly, minimum memory usage and time were considered.

Menu:
The program opens with a menu on console screen. The homework's objectives and extra implementations can be started on the menu. To control memory usage, used arrays is dynamically allocated in beginning of every option implementations and they are deallocated at end of the option implementations. For determining minimum size, text file's line count and maximum password length were detected before main implementations.

Search Objective:
First time I inserted text data into a dynamical allocated 2D char array to prepare text data for searching. But then I realize when I was inserting data into the array, I scanned data from beginning to the end of file. Also for searching last element of the array, I scanned the array from beginning to the end. And also for minimizing dynamical allocated memory of 2D char array, I detected text file's line count and maximum password length by scanning text file entirely again. The program scanned data 3 times entirely and I decided to remove this method of saving data into array before searching. The program implements searching process in the related file without creating any 2D array. This method decreases total time and amount of used memory from 10 MB to 2 MB. Time to search last password which is worst case, is 0.04 seconds.

Sort Objective:
The program used Quick Sort for sorting text data. The reason of selecting Quick Sort is avoiding to exceed 5 second limit. It was on the warnings of the homework's announcement. I thought the best way is choosing the quickest sorting algorithm which is Quick Sort. The program dynamically allocates a 2D char array in minimum size, inserts text data into the array and implements Quick Sort in ascending order. For sorting in descending order, the program creates another array and inserts the ascending sorted array's element's reversely into the new array. Result was displayed on the screen in 0.03 seconds. The amount of used memory is 3 MB.

Hash Objective:
The get minimum collision count in hash table, the program uses 3 methods. Creating a bigger sized array method's formula is taken in this forum page [1]. The program chooses nearest prime number to "(element count * 100) / 40" formula's result for hash table's size. Jenkins's one_at_a_time hash function [2] was chosen for the hash objective's hash function. It modifies strings' characters in bit dimension and grants better distribution on inserting them into hash table. Thanks to these 3 methods, the program inserts text file's data into a hash table sized 2503 in 0.01 seconds and its maximum collision count is 7. Total collision count is 347. When searching a password, the program doesn't implement linear probing more than 7. The benefit of hash table which is searching quicker, is noticed when comparing the search objective's time and the hash objective's search time for given same password. Searching in hash table is quicker and its time is smaller than the search objective's time because of less linear probing. In this objective, the amount of used memory is 2 MB.

Linked List Objective:
In this objective, status of seeing slower accessing speed's reason is traversing in linked list's nodes is slower than moving in array's element. Because array's elements' addresses are adjacent. Reason of the high time is linked list's node's addresses are not adjacent even though they are attached with "next" or "prev" pointers. Fortunately the program avoids traversing, except in displaying result. Instead of traversing, the program saves the last node address with a pointer after inserting an element into linked list and updates this pointer for every inserting. Inserting last node's next node is faster than finding last node by using a loop which uses traversing from head node to last node. For easier displaying last ten nodes' data, doubly linked list is used in the program because doubly linked list is quicker at accessing previous nodes with its "prev" pointers.

In scoring passwords, many methods was used. The program examines passwords according to their various statuses. Factors of scoring are password length, characters' consecutiveness, characters' consecutive iterations, characters' iterations in password, characters' transitions to another character type (symbol, number, upper-case letter, lower-case letter), character's types, password's availability in dictionary and passwords' occurrence from common keyboard combination (wasd, qwerty…). For creating a dictionary, the program is used Linux machine words [3] and nsfw words [4]. Linked nsfw words text file is edited for easier searching. Their text files are available in "dictionary" folder. The program merges these text files by discarding duplicate words and changing Linux machine word's first characters to lower-case letters. Quick Sort is used in the program for quicker sorting linked list's data according to their password scores in ascending order. The only reason of swapping nodes without changing their data, is that changing node's data without changing their "next" and "prev" pointers' pointed address was easy for me. I wanted to choose hard way to implement swapping. Both two methods spend nearly same time but function of swapping nodes' data has shorter code lines. The program completes Quick Sort and inserting data into linked list in 0.8 seconds and the amount of used memory is 10 MB.

1 Million Passwords:
The program's previous version completed first 3 objectives successfully but last one which is linked list objective composed stack overflow error whose one of reason is calling a function extremely a lot of times. Because of succession at hash objective which is used 2D char array, I tried to store passwords' score in short int and string type. After failures of many tries, I decided to store passwords' data in a struct array and implement Iterative Quick Sort in this struct array. After implementing Iterative Quick Sort, the program inserts sorted passwords data into linked list without changing their order. For 1 million passwords, the search objective's maximum memory usage is 6 MB and searching time in worst case is 0.4 seconds, the sort objective's maximum memory usage is 202 MB and time is 5.6 seconds, the hash objective's memory usage is 265 MB and creating hash table time is 2.6 seconds, the linked list objective's maximum memory usage is 180 MB and time is 582 seconds which is approximately 9.30 minutes.

Conclusion:
Just for searching, saving text file into an array is useless. Searching can be quicker if implementation of searching is in text file. For quicker sorting Quick Sort is the best but for extremely high sized files like 1 million passwords list, the program will crashes because of stack overflow. For avoiding stack overflow, using Quick Sort in iterative way was determined. Searching in hash table is quicker than searching in file or array. Traversing in a linked list consume more times than consumed time of moving in an array's elements. Saving last node's address and updating every insertion was determined for quicker moving between nodes. Common keyboard combinations, consecutive characters and dictionary words are worst passwords. More complex character combinations can be used for powerful passwords.

References:
[1] https://stackoverflow.com/questions/22741966/how-to-choose-size-of-hash-table, last visited at 23.01.2021.
[2] https://en.wikipedia.org/wiki/Jenkins_hash_function, last visited at 23.01.2021.
[3] https://users.cs.duke.edu/~ola/ap/linuxwords, last visited at 23.01.2021.
[4] https://github.com/LDNOOBW/List-of-Dirty-Naughty-Obscene-and-Otherwise-Bad-Words/blob/master/en, last visited at 23.01.2021.