**Lab 2 Report**

**Introduction**

The purpose of this lab was to implement linked lists and use them to find the 20 most common passwords inside the 10-million-combos.txt file that was provided. This was to be done in python3 and had to have 2 solutions.

**Solution**

To complete this lab, we had to come up with two solutions: In solution A we had to check using a for loop when reading the file if the password had already been added to the linked list by traversing through the list and use bubble sort to create a list in descending order. In solution B the same was done but instead we used a dictionary and mergeSort to create a list in descending order of the passwords. Some of the methods being implemented in the program are described below:

Def solutionA – This method will keep track of the passwords that have been read and added to the list. It will add passwords that have been checked to the list if they have not already been added to the list.

Def solution B – Same as method A but uses a dictionary to keep track of what passwords have been checked. The method will then sort the list in descending order using mergeSort.

def cleanerA – Cleans up formatting for solutionA so that it is more readable. Displays the 20 most common passwords from the 10-million-combos.txt file.

Def cleanerB – Does the same as def cleanerA does but for def solutionB instead. Cleans up the formatting so that it is easier to read.

def mergeSort – This method uses the mergeSort algorithm to sort the passwords in descending order. The arrays are divided and sorted individually and then in the end combined into one sorted array.

**Experimental Results**

Since the given file 10-million-combos.txt was so large the program would be left running for too long if left with a range of 10,000,000 in the program. In order to avoid this, I did 3 separate tests with 500, 5000 and 15,000 respectively. The results of the 3 tests can be seen below:

(base) Pedros-MacBook-Air:python pedrosantana$ python OptionA2.py

Work started on solution A.

Nothing will show up until finished to keep everything optimized.

Done

This took 0.00 Seconds

I found 306 duplicates, and I searched 694 passwords

The top 20 passwords are:

20.Password "000006" was found 2 times

19.Password "0000001" was found 2 times

18.Password "000455555" was found 3 times

17.Password "qwerty" was found 3 times

16.Password "1111" was found 3 times

15.Password "2PY000" was found 3 times

14.Password "12345" was found 3 times

13.Password "000001" was found 3 times

12.Password "99999" was found 3 times

11.Password "1000" was found 4 times

10.Password "000007" was found 4 times

9.Password "0004" was found 5 times

8.Password "0005" was found 6 times

7.Password "00001" was found 7 times

6.Password "0000000" was found 8 times

5.Password "0001" was found 13 times

4.Password "00000" was found 22 times

3.Password "00000" was found 31 times

2.Password "000000" was found 61 times

1.Password "0000" was found 61 times

(base) Pedros-MacBook-Air:python pedrosantana$

(base) Pedros-MacBook-Air:python pedrosantana$ python OptionA2.py

Work started on solution A.

Nothing will show up until finished to keep everything optimized.

Done

This took 0.51 Seconds

I found 2115 duplicates, and I searched 7885 passwords

The top 20 passwords are:

20.Password "013" was found 10 times

19.Password "0101" was found 10 times

18.Password "006" was found 10 times

17.Password "002" was found 10 times

16.Password "password" was found 10 times

15.Password "01213" was found 12 times

14.Password "008" was found 12 times

13.Password "007james" was found 12 times

12.Password "01213" was found 13 times

11.Password "0001" was found 13 times

10.Password "pass" was found 16 times

9.Password "0007" was found 17 times

8.Password "007007" was found 20 times

7.Password "00000" was found 22 times

6.Password "0123" was found 36 times

5.Password "0123" was found 42 times

4.Password "00000" was found 48 times

3.Password "007" was found 55 times

2.Password "000000" was found 61 times

1.Password "0000" was found 61 times

(base) Pedros-MacBook-Air:python pedrosantana$

(base) Pedros-MacBook-Air:python pedrosantana$ python OptionA2.py

Work started on solution A.

Nothing will show up until finished to keep everything optimized.

Done

This took 4.62 Seconds

I found 6050 duplicates, and I searched 23950 passwords

The top 20 passwords are:

20.Password "1234" was found 14 times

19.Password "0555" was found 15 times

18.Password "04975756" was found 15 times

17.Password "0320" was found 15 times

16.Password "0007" was found 17 times

15.Password "12345" was found 17 times

14.Password "12345678" was found 18 times

13.Password "007007" was found 20 times

12.Password "password" was found 22 times

11.Password "00000" was found 22 times

10.Password "043aaa" was found 29 times

9.Password "063" was found 35 times

8.Password "0123" was found 38 times

7.Password "finish" was found 40 times

6.Password "0123" was found 42 times

5.Password "00000" was found 49 times

4.Password "007" was found 55 times

3.Password "000000" was found 61 times

2.Password "0000" was found 61 times

1.Password "pass" was found 74 times

(base) Pedros-MacBook-Air:python pedrosantana$

**Conclusion**

This lab was very challenging. I had to research how to use a dictionary and then use the syntax properly in python. Once I got it up and running I was able to effectively get solution B up and running with mergesort and the dictionary. It was nice to see how this lab was applied to the 10-million-combos.txt file and very interesting to see how the size of the txt file passwords used changed the running time of the two different solutions. Seeing the running time and different big O notations in the solutions was very cool in seeing what we are learning in class being applied to a program.

**Appendix**

import time

import collections

from collections import OrderedDict

class Node(object):

# Initializes the lists

def \_\_init\_\_(self):

self.passwords = []

self.duplicates = []

self.duplicateAmount = 0

self.BdictioDuplicates = {}

# Solution A will use a for loop to increase the counter if a word is a duplciate. If the word is not a duplicate then it will be added to the linked list

def SolutionA(self, password):

for i in range(len(password)):

if password[i] in self.passwords:

self.duplicateAmount += 1

self.duplicates.append(password[i])

else:

self.passwords.append(password[i])

# Solution B will do the same as solution A but use a dictionary instead.

def SolutionB(self, password):

for i in range(len(password)):

if password[i] in self.passwords:

self.duplicateAmount += 1

self.duplicates.append(password[i])

else:

self.passwords.append(password[i])

def CleanerA(self):

print("I found {} duplicates, and I searched {} passwords".format(self.duplicateAmount, len(self.passwords)))

c = collections.Counter(self.duplicates)

print("The top 20 passwords are: \n")

i = 0

mostCommon = []

for name, score in c.most\_common(20):

i+=1

mostCommon.append("{}.Password \"{}\" was found {} times\n".format(i, name.rstrip(), score))

for i in range(len(mostCommon)):

print(mostCommon[len(mostCommon) - (i + 1)])

def CleanerB(self):

for item in self.duplicates:

if item in self.BdictioDuplicates:

self.BdictioDuplicates[item] = self.BdictioDuplicates[item] + 1

else:

self.BdictioDuplicates[item] = 1

arr = []

for item in self.BdictioDuplicates:

arr.append(self.BdictioDuplicates[item])

return arr

def DisplayResultsB(self, results):

i = 0

arr = []

for result in results:

i += 1

if i > 20:

break

for name, number in self.BdictioDuplicates.items():

if result == number:

arr.append("{}.Password \"{}\" was found {} times\n".format(i, name.rstrip(), number))

break

for i in range(len(arr)):

print(arr[len(arr) - (i + 1)])

def mergeSort(alist):

if len(alist)>1:

mid = len(alist)//2

lefthalf = alist[:mid]

righthalf = alist[mid:]

mergeSort(lefthalf)

mergeSort(righthalf)

i=0

j=0

k=0

while i < len(lefthalf) and j < len(righthalf):

if lefthalf[i] < righthalf[j]:

alist[k]=lefthalf[i]

i=i+1

else:

alist[k]=righthalf[j]

j=j+1

k=k+1

while i < len(lefthalf):

alist[k]=lefthalf[i]

i=i+1

k=k+1

while j < len(righthalf):

alist[k]=righthalf[j]

j=j+1

k=k+1

return alist

def bubbleSort(arr):

n = len(arr)

# Traverse through all elements in array

for i in range(n):

# Last i elements are already in place

for j in range(0, n-i-1):

# traverse the array from 0 to n-i-1

# Swap if element found is greater than the next

if arr[j] > arr[j+1] :

arr[j], arr[j+1] = arr[j+1], arr[j]

return arr

with open("10-million-combos.txt", 'r', encoding="utf-8", errors='ignore') as txt:

# This code is for the Solution B

# print("Work started on solution B.\nNothing will show up until finished to keep everything optimized.\n")

# start\_time = time.time()

# engine = Node()

# for i in range(10000):

# engine.SolutionB(txt.readline().split('\t'))

# results\_to\_merge = engine.CleanerB()

# results = list(OrderedDict.fromkeys(mergeSort(results\_to\_merge)))

# correct\_results = []

# for i in range(len(results)):

# correct\_results.append(results[len(results) - (i + 1)])

# print("These are the top 20 passwords\n")

# engine.DisplayResultsB(correct\_results)

# print("Done")

# print("This took ", "{0:.2f} Seconds".format(time.time() - start\_time))

# Solution A code

start\_time = time.time()

print("Work started on solution A.\nNothing will show up until finished to keep everything optimized.\n")

engine = Node()

# Change range to test other cases that are not 10 million. 10 million will take forever to run

for i in range(15000):

# .split splits the passwords into two categories: first and last

engine.SolutionA(txt.readline().split('\t'))

print("Done")

print("This took ", "{0:.2f} Seconds".format(time.time() - start\_time))

engine.CleanerA()