LAB 2 REPORT

In lab 2, I was given two options; The first option was to concatenate two data files that contain at least 4k each and abide a set of instructions.

* + Solution A: Every time you read a password from the file, check (using a loop) if that password has already been added to the linked list. That is, you need to traverse the linked list to see if that password has been added already. If the password is already in your linked list, update the number of times the password has been seen in the file. Otherwise, add a the password to the linked list.
  + Solution B: This is a variation of Solution A. Instead of traversing the linked list to check if a password has been seen before, we will be using what is called [a dictionary](https://docs.python.org/3/tutorial/datastructures.html#dictionaries). Read the following code snippet to understand how to use a dictionary in a similar context:

Once the Linked List has been created, I must implement the solutions to find 20 passwords that repeat among the list; there’s a few ways to do this:

* + Solution A: Sort the list (in descending order) using bubble sort, and print the 20 most used passwords along with the number of times they appear in the file.
  + Solution B: Sort the list (in descending order) using merge sort, and print the 20 most used passwords along with the number of times they appear in the file.

Then, I must calculate the time complexity of the running program using big O notation. In order to do all of this, I must create a Node class, which is given.

**class** Node(object):

password = **""**

count = -1

next = **None**

**def** \_\_init\_\_(self, password, count, next):

self.password = password

self.count = count

self.next = next

This was my assigned task, below is how I completed it:

class Node(object):

    password = ""

    count = -1

    next = None

    def \_\_init\_\_(self, password, count, next): #Sets Count, password and next to be in Node class

        self.count = count

        self.password = password

        self.next = next

class SLL(object):

    head = None

    def \_\_init\_\_(self, head=None):

        self.head = None

    def append(self, x):

        # Inserts x at end of list L

        if is\_empty(self):

            self.head = Node(x)

            self.tail = self.head

        else:

            self.tail.next = Node(x)

            self.tail = self.tail.next

    # checks if is empty

    def is\_empty(self):

        return self.head is None

# method checks for duplicate passwords and checks to see if it is empty

def BoolReplace(string, link):

    temp = link.head

    while temp is not None:

        if temp.password == string: #loop will traverse all the list and add 1 to count through each iteration

            temp.count += 1

            return True

        else:

            temp = temp.next

    return False

# reads file as well as creates all elements into a linked list

def Create():

    linkedlist = SLL()

    FileR = open("10-million-combos.txt", "r")

    line = FileR.readline() #Reads the file

    for line in FileR:

        line = line.strip().split(" ")

        password = line[-1]

        # If password was not found:

        if not BoolReplace(password, linkedlist):

            linkedlist.head = Node(password, 1, linkedlist.head)

    FileR.close()

    tmp = linkedlist.head

    return linkedlist

# reads file as well as creates all elements into a dictionary

def CreateDic():

    passwordD = {}

    dictionary\_llist = SLL()

    FileR = open("10-million-combos.txt", "r")

    line = FileR.readline()

    for line in FileR:

        line = line.strip().split(" ") #Will read all the file and split each when a space bar is read

        password = line[-1]

        if password in passwordD:

            passwordD[password].count += 1

        else:

            dictionary\_llist.head = Node(password, 1, dictionary\_llist.head) #Creates new Node

            passwordD[password] = dictionary\_llist.head

    FileR.close()

    tmp = dictionary\_llist.head

    while tmp is not None:

        print(tmp.password, tmp.count)

        tmp = tmp.next

# method allows for only the top 20 passwords to be printed, will be used in merge and bubble sort

def print20(LList):

    temp = LList.head

    for i in range(20): #Will print count for top 20 times, since method BoolReplace already got # the counts, It will simply print top 20.

        print("Password: ", temp.password, "occurred ",

              temp.count, " amount of times")

        # i + 1

        temp = temp.next

# Splist instance of the list into two halves

def splitTheList(sourceList):

    if sourceList == None or sourceList.next == None:

        leftHalf = sourceList

        rightHalf = None

        return leftHalf, rightHalf

    else:

        midPointer = sourceList

        frontRunner = sourceList.next

        # totalLength += 1        - This is unnecessary

        while frontRunner != None:

            frontRunner = frontRunner.next

            if frontRunner != None:

                frontRunner = frontRunner.next

                midPointer = midPointer.next

    leftHalf = sourceList

    rightHalf = midPointer.next

    midPointer.next = None

    return leftHalf, rightHalf

# Merges both linked lists passed

def mergeTheLists(leftHalf, rightHalf):

    fake\_head = Node(None)

    curr = fake\_head

    while leftHalf and rightHalf:

        if leftHalf.val < rightHalf.val:

            curr.next = leftHalf

            leftHalf = leftHalf.next

        else:

            curr.next = rightHalf

            rightHalf = rightHalf.next

        curr = curr.next

    if leftHalf == None:

        curr.next = rightHalf

    elif rightHalf == None:

        curr.next = leftHalf

    return fake\_head.next

# Merge sort to head done by recursivly callig itself and split, then printing top 20

def mergeSortLinkedList(head):

    if head is None or head.next is None:

        return head

    leftHalf, rightHalf = splitTheList(head)

    left = mergeSortLinkedList(leftHalf)

    right = mergeSortLinkedList(rightHalf)

    print20(mergeTheLists(left, right))

# bubble sort would organize passwords in descending order with only top 20 printed

def bubble\_sort(head):

    swap = True

    while swap:

        temp = head.head

        swap = False

        while temp.next is not None:

            if temp.count < temp.next.count:

                temp2 = temp.count

                temp.count = temp.next.count

                temp.next.count = temp2

                swap = True

            temp = temp.next

    print20(head)

# main

def main():

    x = SLL()

    x = Create()

    print("Linked list: ")

    Create()

    print("")

    print("Dictionary: ")

    CreateDic()

    print("")

    print("Bubble Sort: ")

    bubble\_sort(x)

    print("")

    print("Merge Sort: ")

    mergeSortLinkedList(x)

main()