CS2302 Lab2 Report **Govinda KC**
Prof. Diego Aguirre Class 10:30 am
TA: Manoj Saha Date: 10/21/18

Problem Description

Owners of Blizzard Entertainment and Sierra Entertainment have decided to merge Activision and Vivendi Games into a single company. Instead of creating the new ID for every Activision Blizzard employee, they wanted to use the old IDs and decided to detect the possible collisions of the employee IDs.My Job is to tackle this problem using the linkedlist skills.

Proposed Solution

In order to resolve the collision between the employee IDs, I have come up with four different solutions. But before that I have stored all the ids in a single linked list. Four different purposed solutions are as follows:
Solution 1 - Comparing every element in the linked list with every other element in the list using nested loops.
Solution 2 - Linked list is first sorted using the bubble sort and then counted the possible duplicates in the list. They are in neighbours in the list if they are duplicate.
Solution 3 - Linked list is sorted using the merge sort recursive and counted if there are duplicates of each other.
Solution 4 - Highest number Id is identified in the list and boolean array of length m+1 is used, where m is the largest ID in the list in order to determine the duplicates by performing a single pass through the unsorted list.

Test of Algorithms

In order to test if the algorithm works perfectly, I ran the codes for each solution multiple times. I even created my own linked list with random numbers in order to further verify code if it is working good. The purposed solution 1 has different number of duplicates than other purposed solutions. This is because the given lists of employee ids themselves contain duplicates inside them. So while traversing through the linked list without sorting nested loop gives more number of duplicates than actual number. However, I got the same number of duplicates from rest three purposed solutions for each single linked list.

Big-O running Time

Nested Loop	Bubble Sort	Merge Sort	Boolean Array
$O(n^2)$	$O(n^2)$	O(n*logn)	O(n)

But since is n is fixed, the Big-O of all cases is

$$O(1).$$

Conclusion

Big-O running times of all the solutions are shown int he above table. From the implementation of the above four solutions, I found that Boolean array is good in terms of the running time than others since its running time is:

$$O(n).$$

From this lab, I learned many details about the linked list such as append, insert, sort, search, and many more. More important thing I realized in this lab is that linked list better than array such as it has dynamic size and ease of insertion and deletion. It saves memory. However, there are certain drawbacks with it. Random access is not allowed. We have to access elements sequentially starting from the first node when we want the data from a particular node. Binary search is not possible in linked list.

Appendix:

```
# Govinda KC
# CS 2302 lab2
**********************************************************

# import sys to increase the recursion limit.
import sys

sys.setrecursionlimit(6500)

# Define class to create nodes
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

# Creating the class for linked list
class LinkedList:
    def __init__(self):
        self.head = None

# method to print the linked list
    def printList(self,n=-1):
        temp = self.head
        count = 0
        while temp:
            if count == n:
                break
            print(temp.data)
            temp = temp.next
            count+=1
```

```
# Defining the method to create a node at the end of the linked list
    def append(self, new_data):
        new_node = Node(new_data)
        if self.head is None:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node
# reading files
def read_file(filename):
    text_file = open(filename, 'r')
    l_list = text_file.read().splitlines()
    return l_list

# Creating the linked list of the one single file
def get_linked_list(a_list):
    list1 = LinkedList()
    for i in range(len(a_list)):
        list1.append(int(a_list[i])) # use float if you have floating num
    return list1

# Defining function which will merge two linked lists
def mergeLists(list1, list2):
    temp = None
    if list1 is None:
        return list2
    if list2 is None:
        return list1
    if list1.data <= list2.data:
        temp = list1
        temp.next = mergeLists(list1.next, list2)
    else:
        temp = list2
        temp.next = mergeLists(list1, list2.next)
    return temp

# Solution 3
# Defining function which will sort the linked list using mergeSort
def mergeSort(head):
    if head is None or head.next is None:
        return head

    list1, list2 = divideLists(head)
    list1 = mergeSort(list1)
    list2 = mergeSort(list2)
    head = mergeLists(list1, list2)
```

```
    return head

# Defining function which will divide a linked list into two equal linked l
def divideLists(head):
    slow = head  # slow is a pointer to reach the mid of linked list
    fast = head  # fast is a pointer to reach the end of the linked list
    if fast:
        fast = fast.next
    while fast:
        fast = fast.next    # fast is increased by 2 times than slow.
        if fast:
            fast = fast.next
            slow = slow.next
    mid = slow.next
    slow.next = None
    return head, mid
# Solution 2
def bubbleSort(head):
    if head is None or head.next is None:
        return head
    else:
        sorted = False
        while not sorted:
            sorted = True
            prev = head
            current = head.next
            while current!=None:
                if prev.data > current.data:
                    sorted = False
                    temp = current.data
                    current.data = prev.data
                    prev.data = temp
                prev = current
                current = current.next
    return head

def count_sorted(head):
    if head is None:
        return 0
    elif head.next is None:
        return 1
    duplicates = 0
    temp = []
    current = head
    while current.next != None:
        next = current.next
        if next.data == current.data:
            temp.append(current.data)
```

```
            duplicates += 1
        current = current.next
    temp.sort()
    print("Number of duplicates = "+str(duplicates))

# Solution 1
def count_nested(head):
    if head is None:
        return 0
    elif head.next is None:
        return 1
    duplicates = 0
    temp = []
    current1 = head
    while current1 !=None:
        #data = head.data
        data = current1.data
        current2 = current1.next

        while current2!=None:
            if current2.data == data:
                temp.append(data)
                duplicates += 1
            current2 = current2.next
        current1 = current1.next

    temp.sort()
    print(temp)
    print("Number of duplicates = " + str(duplicates))

# Solution 4
def seen(head):
    numDuplicates = 0
    numComparisions = 0
    seenBefore = [False]*6001
    temp = head
    list1 =[]
    while temp is not None:
        numComparisions +=1
        if seenBefore[temp.data] ==True:
            numDuplicates +=1
            list1.append(True)
        else:
            list1.append(False)
            seenBefore[temp.data] = True
        temp = temp.next
    print(list1)
    return {numComparisions, numDuplicates}
```

```
# Main logic and all the commented lines can be
# uncommented as needed to get the results.
if __name__ == '__main__':
    a_list = read_file('activision.txt')
    v_list = read_file('vivendi.txt')

    a_list.extend(v_list)

    list = get_linked_list(a_list)

    #print("Linked list before sorting")
    #list.printList()
    #list.head = bubbleSort(list.head)

    #list.head = mergeSort(list.head)
    #print("Linked list after sorting")
    #list.printList()
    #print('----------')
    #count_sorted(list.head)
    count_nested(list.head)

    #out1, out2=seen(list.head)
    #print(out1, out2)
```

****

A signed academic honesty certification: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.