Lab 2 Option A

Introduction

We're given two separate lists of IDs, our goal is to use four separate approaches in finding duplicate IDS. First things first is to build a linked list, then add every ID as its own node in the linked list. The first solution is considered a brute force option, get one ID check against every other ID, if there is a duplicate take note move to the next ID. The second solution requires you to sort the list using Bubble sort then check for duplicates that way. Third solution requires you to sort the list again, this time using merge sort. Fourth solution is unique to me at least, this solution has you build a boolean array, the size of the array is the value of the largest ID in the given lists. For example, if one ID is 12345, and that's the largest ID then the value of that ID is the size of the boolean array.

Design

Node Class

My Node class is standard, with the class containing two attributes. "next" pointing to the next node, and "data" pointing to ID, for this use case at least.

LinkedList Class

My LinkedList class is not your standard LinkedList class that'll you'll find online. If you take a look at previous commits in my GitHub, you'll see that it did start off quite basic with class attributes only being "head" and "size". However, when implementing solutions 3 and 4, I made the design choice to add "tail" attribute, and an "largest_ID" attribute. As IDs are loaded into my LinkedList I have an if statement that checks for the largest ID. This way it's a bit more efficient for solution 4.

My linked list contains two types of inserts, insert at the head and at the tail. When going through the first solution I had my IDS being inserted at the head, however, when implementing merge sort this became a problem. Will get into this in more detail later, other than that my linked list is simple.

Utility Functions

Read_files

This function takes a filename and a given list. The filename should be a file with IDs and adds to the given list.

Print lists function

Two functions that will print out a standard list and my custom LinkedList.

Solution 1

The design of this function is how it is described in the introduction. This function one linked list as a parameter, this list should contain all IDS. I felt this would be the most readable way to go about writing and understanding the code. I used a double for loop to check every ID with every other ID, to account for the fact that every ID would be seen at least once I created a variable to count how many times the same ID was seen. If it was seen more than once then the ID was added to a separate list. This will be known as the duplicate_list.

Solution 2

This solution requires us to first sort the list and then check for duplicates. Sort the list using bubble sort, for more info on bubble sort click here. I used a variable to count the amount of swaps made during an iteration, if the amount of swaps was 0 then the loops would cease. A while loop and for loop was used, while so iterations don't have to be tracked, and a for loop to go through the entire list and actually do the bubble sort. I did have trouble swapping nodes and it was becoming complicated so instead i kept the nodes where they were and just swapped the data. I feel this is what most programmers would do as its much more readable.

I implemented a function that would check for duplicates, it receives a sorted LinkedList and checks for duplicates.

Solution 3

This solution requires us to sort the LinkedList using merge sort. Merge sort requires splitting the given list, to do this I created two separate lists. One list would be the first half with the head being the given lists head, the second list's head would be the middle node plus one. When it comes to merging its standard, a new list is created, then both left and right lists compare to and add to the newly created list. To check for duplicates the same function used in solution 2 is used here.

Solution 4

As mentioned before to make this faster I had every linked list created track the largest ID being passed into the list. So this function takes that ID builds a boolean list based on the size of the largest ID. Each item in the boolean list is set to false and to actually check for duplicates I used the hint given in the lab document. I loop through the entire list, if an ID is seen in the boolean list then that index is set to true. If an ID is seen and the index is already true then we have a duplicate.

Results

Test_file_1: This test was the first test made, it tested that duplicates were being collected. All solutions passed
Test_file_2: This test was mainly aimed at the sort functions, tested that lists weres being sorted properly. No duplicates in this file and no duplicates were collected.
Test_file_3: This was test was meant to give the function a random collection of numbers, no pattern whatsoever. All solutions sorted and collected the correct amount of duplicates.
Test_file_4: This tested that my program would not crash if a bad file was passed.

Time Complexity

Time was taken for every solution, the IDs being analyzed are the IDs given by the lab.
Solution 1:

- N^2
- 20.06064304895699

Solution 2:

- N^2
- 42.70197809697129

Solution 3:

- $NlogN$
- 0.2834856740082614

Solution 4:

- N + M (M being the largest ID in the list)
- 0.004791056038811803

Conclusions

Based on the results the clear winner is solution 4, this is because its a linear progression. Merge sort is also wicked fast and both in sorting, what gives solution 4 the advantage it doesnt need to sort it just immediately checks for duplicates. I solidified my understanding of merge sort and bubble sort, was a challenge using lists rather than arrays. I also learned that boolean arrays are pretty cool. I also learned how to build classes in python, I still need more practice I think but learned a lot.

Appendix