COP 3330

Programming Assignment 4: Shrinking Linked List

Program Overview

This assignment is about linked lists in Java. You will implement a linked list and do operations on that. Your program will take the name of the file as user input. You will clean the data and process it using a linked list. You will be modifying the linked list while you are traversing.

Your linked list Node implementation will **not** have a *setItem* method. Therefore changing the element inside the linked list node after it has been created, will NOT be possible. You will lose points if you add a method to modify the linked list node content. *You can not store the elements in an array/Arraylist!* You also cannot use doubly linked list for this assignment.

You will need to implement a *Node* class which does not have setItem method
You will need to implement a *LinkedList* class.
You will need to implement a *LinkedListTester* class which will have a main method inside.

You can implement any LinkedList method as you prefer, however, the three methods below are necessary.

`public void construct(String fname)`
This method will construct a linked list by reading the file. The constructed linked list will have no negative elements inside.

`public void process()`
This method will process the linked list based on the description below. It will modify the linked list which will be ready to be printed to file.

`public void printToFile(String fileName)`
This method will print the linked list to the file whose filename is given as *fileName*.

Clarification

In this assignment you will ask the user to enter the name of the file as from keyboard. The given data file will not be empty and it will exist/readable. Let us assume data file is written as data.txt from now on. There will be a single space between numbers. All the numbers are on the same line. This data file contains integers with a space between them. You will add the numbers which are **greater than zero** to a linked list. Then you will traverse this linked list from the beginning and delete some nodes and add a new one. Then you will create a file *processeddata.txt* and write the new linked list data. When createing output file you will just need to concatenate "processed" before the input file name. The rules are below.

When start traversing the linked list, you will need to store *the count* number to some variable. *count* will be initialized to -100. While traversing If you see this condition in the list,

X[i] = X[i-1] x 2 + 7

You will delete X[i] and X[i-1] and add a node that contains the item *the count*. You will increase *the count* number as you make insertions to the list. After you have done this, you can continue traversing. You won't need to check the previous element after adding a new node since the added element will be negative. In other words, once a shrink has happened, it cannot happen again with the next element. When you reach at the end of the list you will add a new node with *the count* number as the item.

An example *data.txt* and *processeddata.txt* is added to this document
Lets assume we have removed the negative numbers from the data (maybe we in fact cleaned the data) and we got the numbers below to our linked list. You may assume this is test case 1.

If your linked list contain those numbers:

1 4 3 9 25 6 7 21 8 9 19
|
current

here 9 x2 + 7 =25
When we reach the element 9
You will delete 9 and 25 and add -100

After insertion our current node will be -100
previous will be 3
next one will be 6

Note that this shrinking will not happen again once a shrink has taken place.

So the linked list will become like this:
1 4 3 -100 6 7 21 8 9 19

At the next iteration:

1 4 3 -100 6 7 21 8 9 19
|
current

when we continue and reach 7 we see that 7 x 2 + 7 = 21. You will delete 7 and 21 and add a node that contains -99.

After insertion our current node will be -99
previous will be 6
next one will be 8

The linked list becomes
1 4 3 -100 6 -99 8 9 19

Until the end we won't see the condition again. Therefore at the end of the list we will add -98. This addition will make sure that the data is processed.

the last linked list becomes

1 4 3 -100 6 -99 8 9 19 -98

You will need to write a text file called *processeddata.txt* which contains the linked list elements. The items will have a single space between them. There is no space/newline character at the end of the output in the *processeddata.txt* file.

Test case 2

data.txt

-1 5 1 -5 4 3 -4 9 -7 -88 25 17 7 -25 21 7 9 19 -99 -87

processeddata.txt

5 1 4 3 -100 17 -99 7 9 19 -98

Test case 3

data.txt

15 37 -20 7 -3 60 127 31 69 11 9 -5 20 74 155 22 21 69 145 9 6 4 -16 14 -12 36 79 8 14 17 41 -23 11 5 22 -18 -17 82 171 18 27 61 20 -11 49 105 -7 24 -4 80 167 35 77 23 4 4 15 -20 -13 21 8 -7 -15 -23 -22 14 -18 -7 5 0 20 -7 -4 9 25 -20

processeddata.txt

-100 7 -99 -98 11 9 20 -97 22 21 -96 9 6 4 14 -95 8 14 -94 11 5 22 -93 18 -92 20 -91 24 -90 -89 23 4 -88 21 8 14 5 20 -87

Note that if the last two elements are processed and a negative number is added, we don't add another negative number at the end.
It is also possible that the first/last two elements should shrink.

Standard input/output example: (the user input is written in bold)

Enter the data file name: **data.txt**
processeddata.txt is created.

Submission Guidelines

- Your Java source file must begin with the following package declaration:

```
package assignment4;
```

- Do not use the default package.
- Include a main method in *LinkedListTester* file only. If another file has also a main method, you will lose 20 pts.
- Do not use ArrayList or Array to store numbers in this assignment. You need to use LinkedList to store numbers.You cannot use any other 3rd party library.
- Do not use static imports. You will lose 10 pts if you do so.
- All instance variables should be private and all methods should be public unless otherwise stated.
- Do not provide absolute file path anywhere in your code. If your path includes some folders which is specific to your local machine, the output will not be created. You will lose considerable amount of points if you assignment does not create output in our system. **We won't fix anything in your submission code to run it!**
- Your code should compile fine. Non compiling submissions will get 0 from test cases.
- Do not submit class files. Please submit a zip file which contains java files. You will receive 0 if you do not follow the submission guidelines with no exceptions.
- The file names will be as specified above. If you use a different file name, autograder will have issues with your assignment. As stated before, we will NOT fix anything in your submission.
- Note that the codes shared in the webcourses may **NOT** be used as is since the assignment needs to remove/replace nodes. Copy pasting the code in the sample file may not be useful.
- The filename acquired by the user input will not have spaces.
- Assume the given input file exists and it is readable. Anyways, it is better if you stop the program if there is a file input output exception.

Evaluation Criteria

- Test case 1: 20 pts
- Test case 2: 20 pts
- Test case 3: 20 pts
- Code Comments: 15 pts
    - if each file contains comments explaining the code. For each file which does not have a header comment, you will lose 5 pts. Your submission files will also need comments besides the header one.
- Code Efficiency and Structure: 15 pts
    - 5 points off if there is no method to read the file to LinkedList class (construct())
    - 5 points off if there is no method to process the LinkedList class (process())
    - 5 points off if there is no method to print the LinkedList class (printToFile())
- Submission guideline follow: 10 pts
    - If SetItem method exists (not private), you will lose 10 pts. (no matter if you used or not)
    - If the any instance variable of any class is public, you will lose 10 pts. (no matter if you used or not)