Assignment 2

Please read this document very carefully. Follow instructions exactly. If you have any questions please post them to MS Teams or ask during office hours.

This assignment is due June 4th, by 11:59pm

I have created an Assignment 2 channel in Teams. If you have questions about the assignment, please post them there. Thank you.

Overview

In this assignment you will implement a custom data-structure called a LinkedArray. A LinkedArray behaves similarly to a linked list with the exception that at every node instead of storing a single value, the node will store several values using an array. All the nodes of a linked-array have equal length underlying arrays. See below for further details to how the data-structure operates.

You are responsible for implementing four methods within the LinkedArray class. I have partially implemented the class to get you started. That is, I have implemented a print method for you (which should aid in simple testing), as well as an append method. You should study and understand the append method. It will give you insight to cases you will need to consider as well as potential implementation details for the remaining methods.

You will note that the append method adds a value to the end of the LinkedArray. If the array in the last node of the LinkedArray is full, a new node is create, linked to the data structure, and the value is added to start of the array in the new node. Play around with print and append to see how the LinkedArray behaves. Here is some code as an example:

```
LinkedArray L = new LinkedArray(k);
for(int i = 0; i < 10; i++) {
    L.append(i);
}
L.print();

/* Outputs:
When k = 1: [0 ] -> [1 ] -> [2 ] -> [3 ] -> [4 ] -> [5 ] -> [6 ] -> [7 ] -> [8 ] -> [9 ] ->
When k = 2: [0 1 ] -> [2 3 ] -> [4 5 ] -> [6 7 ] -> [8 9 ] ->
When k = 3: [0 1 2 ] -> [3 4 5 ] -> [6 7 8 ] -> [9 ] ->
When k = 5: [0 1 2 3 4 ] -> [5 6 7 8 9 ] ->
When k = 10: [0 1 2 3 4 5 6 7 8 9 ] ->
*/
```

Methods

1. `getValueAt(int index)` Returns the integer value located at the index of the LinkedArray. For examples as to what indices correspond to which value see the examples below. You may assume index is always less than length and never negative.

```
If L.print() outputs:
[7 1 9 ] -> [10 1 5 ] -> [2 0 ] ->

Then your getValueAt(int index) should return:
L.getValueAt(0) -> 7
L.getValueAt(4) -> 1
```

```
L.getValueAt(7) -> 0
L.getValueAt(3) -> 10

If L.print() outputs:
[7 8 ] -> [1 5 ] -> [2 ] ->

Then your getValueAt(int index) should return:
L.getValueAt(0) -> 7
L.getValueAt(4) -> 2
L.getValueAt(2) -> 1
```

2. `appendFront(int value)` Adds a value to the front of the LinkedArray. This may cause a new node to be create and linked to the back. See below for examples.

```
L.print(); //Outputs [7 1 9 ] -> [10 1 5 ] -> [2 0 ] ->
L.appendFront(12)
L.print(); //Outputs [12 7 1 ] -> [9 10 1 ] -> [5 2 0 ] ->
L.appendFront(13)
L.print(); //Outputs [13 12 7 ] -> [1 9 10 ] -> [1 5 2 ] -> [0 ] ->
```

3. `resetSize(int newSize)` Resizes the underlying length of the arrays in each node. The order and values are maintained. See below for examples:

```
L.print(); //Outputs [7 1 9 ] -> [10 1 5 ] -> [2 0 ] ->
L.resetSize(2)
L.print(); //Outputs [7 1 ] -> [9 10 ] -> [1 5 ] -> [2 0 ] ->
L.resetSize(4)
L.print(); //Outputs [7 1 9 10 ] -> [1 5 2 0 ] ->
L.resetSize(6)
L.print(); //Outputs [7 1 9 10 1 5 ] -> [2 0 ] ->
L.resetSize(1)
L.print(); //Outputs [7 ] -> [1 ] -> [9 ] -> [10 ] -> [1 ] -> [5 ] -> [2 ] -> [0 ] ->
```

4. `deleteFirstOccurence(int value)` Removes the first occurrence of value in the LinkedArray. If the value is not in the LinkedArray, nothing is changed. Deleting a value may result in the last node being removed from the LinkedArray. See below for examples.

```
L.print(); //Outputs [7 1 9 ] -> [10 1 5 ] -> [2 0 ] ->
L.deleteFirstOccurence(2)
L.print(); //Outputs [7 1 9 ] -> [10 1 5 ] -> [0 ] ->
L.deleteFirstOccurence(8)
L.print(); //Outputs [7 1 9 ] -> [10 1 5 ] -> [0 ] ->
L.deleteFirstOccurence(1)
L.print(); //Outputs [7 9 10 ] -> [1 5 0 ] ->
L.deleteFirstOccurence(1)
L.print(); //Outputs [7 9 10 ] -> [5 0 ] ->
```

Additional Comments

- Only modify the TODO methods in LinkedArray.java. Changing the implementation such that LinkedArray is no longer a linked list where nodes store arrays will result in a grade of 0.

- Remember to appropriately update the length variable!
- Define and use helper methods.
- Understand how why append is using `length%size` and why it is useful.
- When testing, consider empty LinkedArrays.

Submitting and Grading

This assignment will be submitted electronically via Avenue. You should submit one file to Avenue named `LinkedArray.java` This assignment is worth 15% of your final grade. Your methods are grading 100% on correctness. No weight is put on efficiency, elegance of solution, etc. Grading is done automatically. That is, a program calls your method, passes it certain arguments, and checks to see if it returns the expected output. The assignment is broken down as follows:

- `getValueAt()`: 20%
- `appendFront()`: 25%
- `resetSize()`: 25%
- `deleteFirstOccurence()`: 30%

Good luck!

Academic Dishonesty Disclaimer

All of the work you submit must be done by you, and your work must not be submitted by someone else. Plagiarism is academic fraud and is taken very seriously. The department uses software that compares programs for evidence of similar code.

Please don't copy. The TAs and I want you to succeed and are here to help. Here are a couple of general guidelines to help you avoid plagiarism:

Never look at another assignment solution, whether it is on paper or on the computer screen. Never show another student your assignment solution. This applies to all drafts of a solution and to incomplete solutions. If you find code on the web that solves part or all of an assignment, do not use or submit any part of it! A large percentage of the academic offenses in involve students who have never met, and who just happened to find the same solution online. If you find a solution, someone else will too.