CS150 Lab 4

Oanh Doan

September 24, 2017

1 Introduction

The goal of this lab is to understand implementation of a generic linked list. We want to create a linked list class with *insert*, *remove* methods and an iterator. The iterator is used to make it easier to access/search for list elements, which in turn may be helpful to implement other methods if necessary.

2 Approach

2.1 Design of program

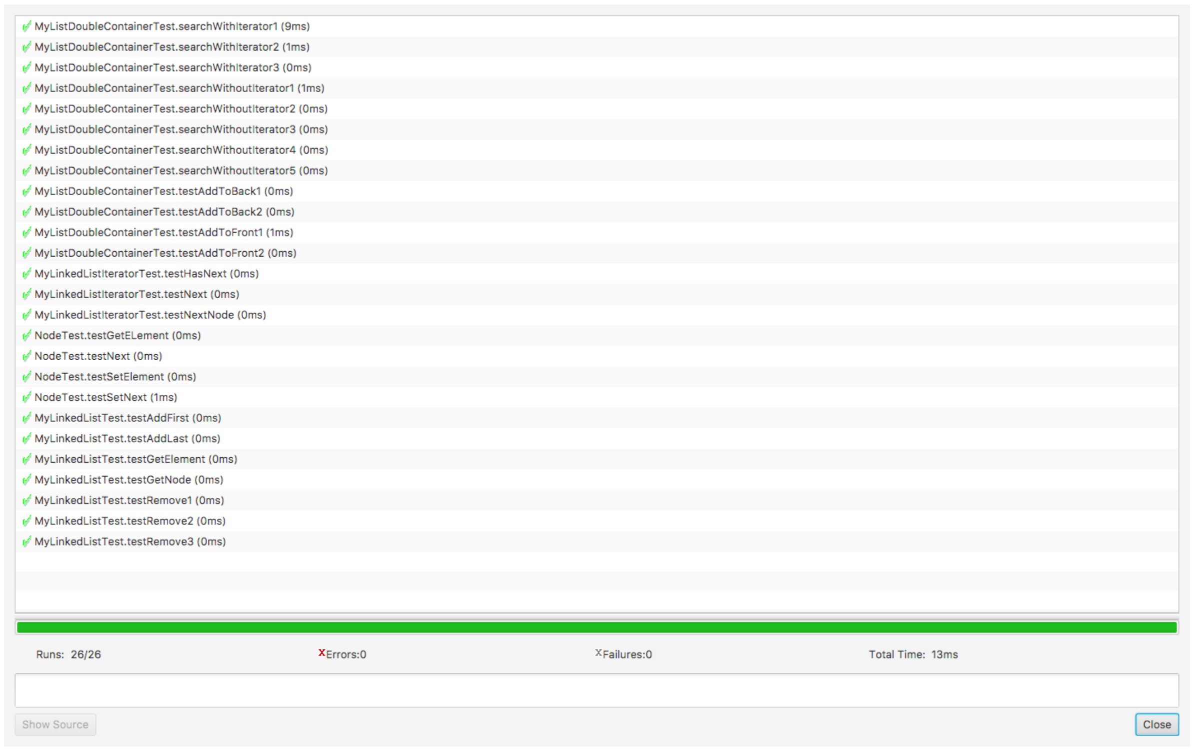
The program contains the following classes:

- Node - contains element and pointer to next node
- MyLinkedList - contains add and remove methods
- MyLinkedListIterator
- ExercisingLinkedList - exercise methods in myLinkedList to test the program
- MyListDoubleContainer - contains a list, add and search methods
- ExperimentController

The first three classes construct a linked list of a generic data type. The *node* (*Linked Lists*, n.d.) class contains information on the value of the link and a pointer to the next link. A linked list is constructed by using the *add* and *remove* methods in *MyLinkedList.* All methods in the first three classes are tested using different input size in *ExcersingLinkedList.* The fifth class stores a list and two search methods. The ExperimentController class records and compares search time of these two methods using the same input list and searching values.
We also have unit test classes to test if the methods perform as expected:

- NodeTest

- MyLinkedListTest
- MyLinkedListIteratorTest
- MyListDoubleContainerTest



```
MyListDoubleContainerTest.searchWithIterator1 (9ms)
MyListDoubleContainerTest.searchWithIterator2 (1ms)
MyListDoubleContainerTest.searchWithIterator3 (0ms)
MyListDoubleContainerTest.searchWithoutIterator1 (1ms)
MyListDoubleContainerTest.searchWithoutIterator2 (0ms)
MyListDoubleContainerTest.searchWithoutIterator3 (0ms)
MyListDoubleContainerTest.searchWithoutIterator4 (0ms)
MyListDoubleContainerTest.searchWithoutIterator5 (0ms)
MyListDoubleContainerTest.testAddToBack1 (0ms)
MyListDoubleContainerTest.testAddToBack2 (0ms)
MyListDoubleContainerTest.testAddToFront1 (1ms)
MyListDoubleContainerTest.testAddToFront2 (0ms)
MyLinkedListIteratorTest.testHasNext (0ms)
MyLinkedListIteratorTest.testNext (0ms)
MyLinkedListIteratorTest.testNextNode (0ms)
NodeTest.testGetELement (0ms)
NodeTest.testNext (0ms)
NodeTest.testSetElement (0ms)
NodeTest.testSetNext (1ms)
MyLinkedListTest.testAddFirst (0ms)
MyLinkedListTest.testAddLast (0ms)
MyLinkedListTest.testGetElement (0ms)
MyLinkedListTest.testGetNode (0ms)
MyLinkedListTest.testRemove1 (0ms)
MyLinkedListTest.testRemove2 (0ms)
MyLinkedListTest.testRemove3 (0ms)

Runs:  26/26          Errors:0          Failures:0          Total Time:  13ms
```

Figure 1. Results of unit tests

2.2 Choice of algorithms and data structures

- Generic linked list (*Linked List Implementation*, n.d.): We implement our own version of linked list using three classes as described earlier. These classes construct each node of the list, provide *insert* and *remove* methods to put the nodes together to form a list, and give faster access to the nodes through an iterator.
- Iterable interface (*Iterable (Java Platform SE 8)*, n.d.): Implementing Iterable interface allows us to ”loop through” elements in the list.
- Iterator interface (*Iterator (Java Platform SE 7)*, n.d.): Implemenitng Iterator interface provides access to list elements. Unlike ArrayList, Linked list is not easy to access elements by position. Thus, using iterators makes it easier to control and write programs.

2.3 Description of Node Design (*Linked Lists*, n.d.)

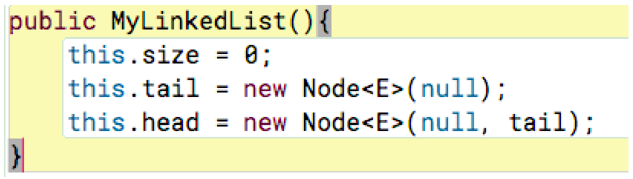
The node class contains the following information:

- Element, i.e value of node
- Pointer to next node

There are 2 types of nodes that can be created:

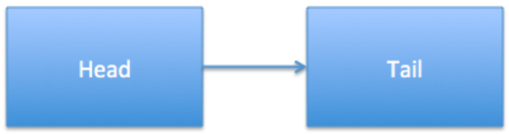
- A node that takes information on both elements and pointer, which can be set to *null* if necessary.
- A node that takes information on the pointer only. The default element is *null*.

In MyLinkedList, heads and tails are nodes created by the following way:



```
public MyLinkedList(){
    this.size = 0;
    this.tail = new Node<E>(null);
    this.head = new Node<E>(null, tail);
}
```

We first create a tail node, using the second type of node. By default, the tail element is *null*, and its pointer is also null. We then create the head node using the first type of node. Its element is *null*, but its next pointer is the tail. The head and tail are now connected.



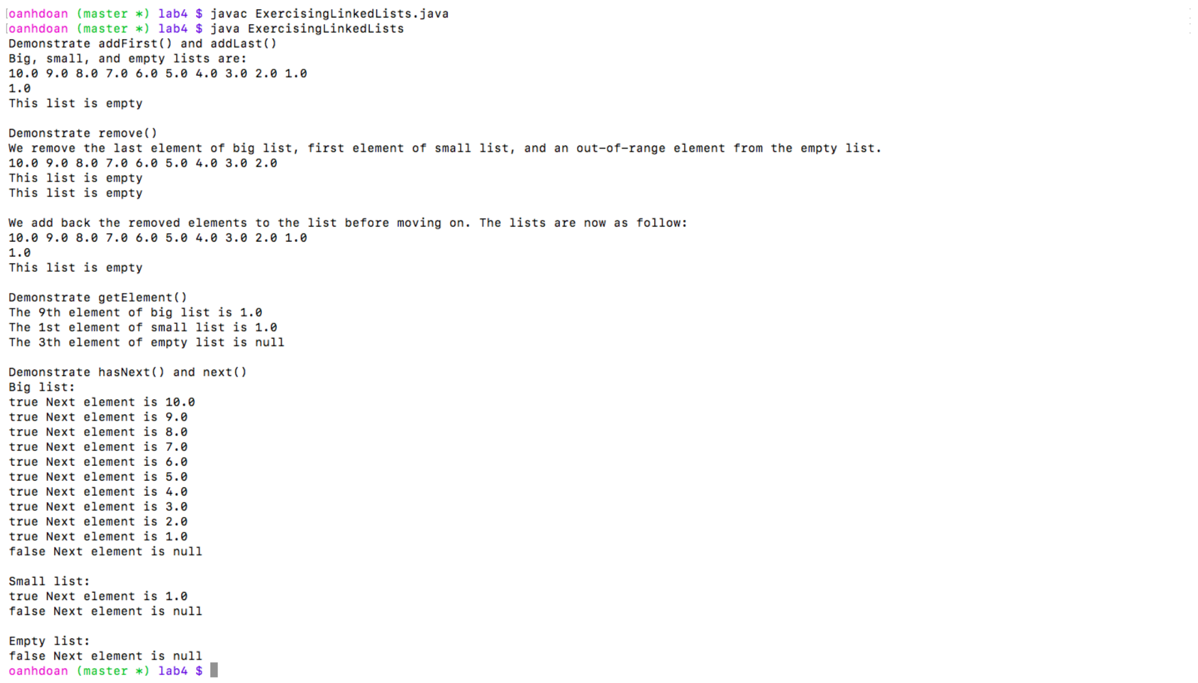
All methods in this class are related to the node's elemet and pointer.

- getElement()
- next()
- setElement()
- setNext()

The first two methods are used to return information of the current node. The next two are used to change information of the node (if necessary) after initialization.

3 Methods

- We test if our linked list works properly by testing all functionalities in Node, MyLinkedList, and MyLinkedList iterator within the ExercisingLinkedList class. This is necessary because it is difficult to control for the connection between the nodes in unit testing. We tests the methods using a 10−node list, a 1−node list, and an empty list to control for boundary cases.



```
oanhdoan (master *) lab4 $ javac ExercisingLinkedLists.java
oanhdoan (master *) lab4 $ java ExercisingLinkedLists
Demonstrate addFirst() and addLast()
Big, small, and empty lists are:
10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
1.0
This list is empty

Demonstrate remove()
We remove the last element of big list, first element of small list, and an out-of-range element from the empty list.
10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0
This list is empty
This list is empty

We add back the removed elements to the list before moving on. The lists are now as follow:
10.0 9.0 8.0 7.0 6.0 5.0 4.0 3.0 2.0 1.0
1.0
This list is empty

Demonstrate getElement()
The 9th element of big list is 1.0
The 1st element of small list is 1.0
The 3th element of empty list is null

Demonstrate hasNext() and next()
Big list:
true Next element is 10.0
true Next element is 9.0
true Next element is 8.0
true Next element is 7.0
true Next element is 6.0
true Next element is 5.0
true Next element is 4.0
true Next element is 3.0
true Next element is 2.0
true Next element is 1.0
false Next element is null

Small list:
true Next element is 1.0
false Next element is null

Empty list:
false Next element is null
oanhdoan (master *) lab4 $
```

Figure 2. Run ExercisingLinkedList

- To measure search time by value of 2 search methods (with and withou iterator), we use the same input list and intentionally search for an element at the beginning, one in the middle, and one at the end of the list. For each search value, we run the method 5 times and take the average. Since this is linear search, we hypothize that the search time increases as the position of the searching value increases.

- To measure search time by index using 2 different data structures (ArrayList and linked list), we use 3 input sizes. For each input size, we also search for the start, the middle, and the end of the list. For each index, we run 5 times and take the average.

4 Data and Analysis

4.1 Search by value

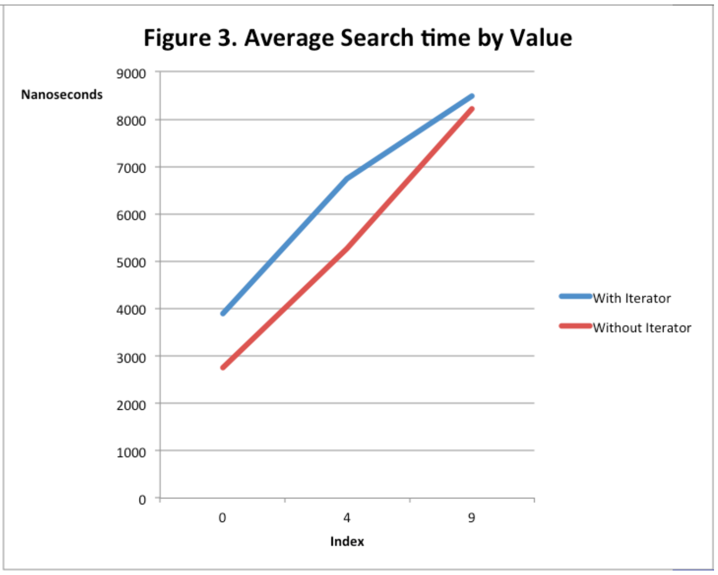
Table 1 presents the run time of 2 search methods using iterator and not using iterator. Table 2 displays the average search time for each input value.

Index	WithIterator	WithoutIterator
0	33352	2721
0	4941	2511
0	4071	2879
0	3323	2535
0	3365	2809
0	3754	3041
4	8457	7529
4	6680	3598
4	5149	4998
4	6612	4946
4	6820	5228
9	10345	9510
9	8219	7095
9	8010	4929
9	7113	12305
9	8749	7232

Table 1. search time of 2 methods

Index	Average With Iterator	Average Without Iterator
0	3890.8	2755
4	6743.6	5259.8
9	8487.2	8214.2

Table 2. Average search time of 2 methods



We notice that for the first time a search method is called, the search time, regardless of the actual position of the search value, is significantly large compared to the remaining times. This happens because when the program first starts, the computer may need to do initialize something before it actually runs the method. Therefore, when calculating the average, we ignore the first run time in Table 1 (highlighted in red). From Figure 3, we observe that the search time using list iterator is slightly greater than the search time not using iterator.

4.2 Search by index

input size	index	array	linked list
100000	0	59	1200
	49999	130	116154
	99999	97	228760
5000	0	73	1520
	2499	120	5566
	4999	86	10949
1	0	50	1526
	3	46	15684

Table 3. Average search time by index using array and linked list

Table 3 shows the average search time by index using array and linked list. Note that similar to the previous experiment (highlighted in red in output file), we exclude the run time of the first run because of the same reason. From the table, we can see that the search time using array does not vary greatly regardless of searching index (recall that time is measured in nanoseconds). However, search time using linked list increases as searching index increases. This makes sense because linked list does not provide instant access to elements by position. What happens in the back is that the iterator starts from head list and moves towards the given position. Therefore,the larger the index is, the more time it will take to search.

5 Conclusion

- Implementation of a generic linked list is not always straightforward. However, it is important to understand the implementation to know how the nodes will shuffle and change the list after a method is called (i.e understand the list behavior).
- Even though iterator is a powerful tool to process elements and is helpful for the implementation of various methods. However, it does not always guarantee efficiency, as in the example of the search time by value, where using and not using iterators give approximately the same results.

References

Iterable (java platform se 8). (n.d.). January 12, 2016. Retrieved from https://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

Iterator (java platform se 7). (n.d.). January 11, 2016. Retrieved from https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html

Linked list implementation. (n.d.). Retrieved from https://canvas.instructure.com/courses/1171720/assignments/6432392?module_item_id=12945299

Linked lists. (n.d.). Retrieved from https://canvas.instructure.com/courses/1171720/modules/items/12945203