Implementing a LinkedList and a Priority Queue
Report #06

By
Eduardo Castro

CS 303 Algorithms and Data Structures

October 8, 2014

1. Problem Specification

The goal of this assignment was to create a linked list, implement its basics methods for proper functioning(like adding, deleting, searching, etc) and to implement a priority queue.

The Linked List is basically a structure that each element has a node and its value. The node always points for the the next node, if it is pointing for a null value so it means that this node is the last one. We always need to store the position from the position node: the head.

The priority queue is an structure that have only two methods: insert and delete. It inserts an element at the end of its structure and delete the smallest value. Always and only the smallest.

2. Program Design

The first part of this lab requires the creation of three classes: one for the Node, one for the LinkedList and the driver.

The following steps were required to develop this program:

a) write the Node class
b) write the LinkedList class
c) write the LinkedListTest class
d) use the driver class to display and measure the results

The following methods were defined within the Node, LinkedList and LinkedListTest classes:

a) Node(int value, Node next)
    a. Constructor
a) Node(Object value)
    a. Constructor
b) getValue()
    a. Returns the value from the node
c) setValue(Object Value)
    a. Sets the value from this node
d) getNext()
    a. Returns the next node
e) setNext(Node Next)
    a. Sets the next node
f) LinkedList()
    a. Constructor
g) getSizeOfTheList()
    a. Returns the size of the Linked List
h) isTheListEmpty()
    a. Returns false if the list is populated or true if it is empty
i) add(Object value)
    a. Adds a node at the end of the list
j) add(Object value, int position)

    a. Adds a node at specific position

k) removeElement()
    a. Removes the last element from the linked list
l) removeElement(int position)
    a. Removes an element at specific position
m) getValue(int position)
    a. Returns the value from a node at specific position
n) containsElement(int value)
    a. Checks if a specific value it is inside the Linked List

The second part requires us to build a Priority Queue using a min heap algorithm. For this part I created two classes: PriorityQueue and PriorityQueueTest.

The following steps were required to develop this program:

a) write the PriorityQueue class
b) write the PriorityQueueTest class
c) use the driver class to display and measure the results

The following methods were defined within the PriorityQueue and PriorityQueueTest classes:

a) PriorityQueue(int size)
    a. Constructor
a) PriorityQueue()
    a. Constructor
a) getElement(int position)
    a. Returns the element at specific position
a) insert(int value)
    a. Insert an element to the priority queue
a) delete(int value)
    a. Returns false if the delete was not successful(like if the value was not found or if it is not the smallest) and true if the deletion was fine
a) resizeArray(int newSize)
    a. Doubles the size of the array
a) swap(int i, int j)
    a. Swap the position from both elements

The Scanner class provided by Java was used to read in the necessary values within the provided driver program. The println method of the System.out object was used to display the inputs and results for the provided driver program.

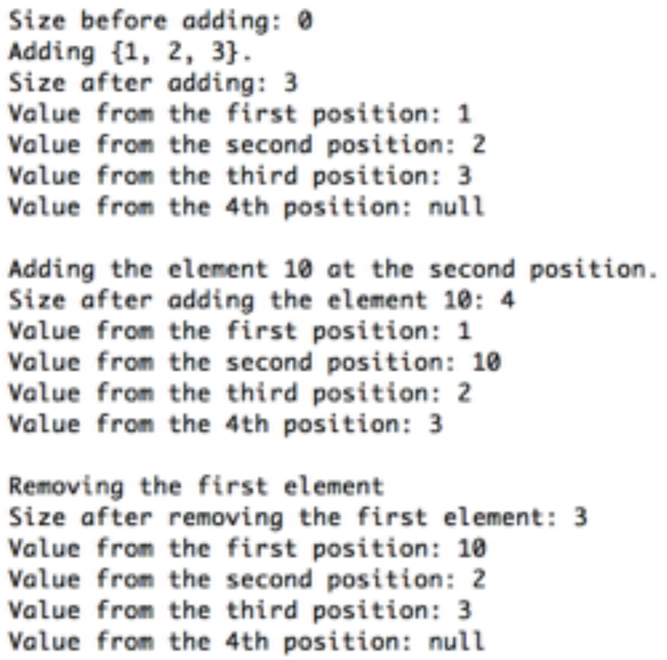
3. Testing Plan

For the first part of the lab, the LinkedList part, I added three elements, got the new size of the LinkedList, tried to add a new element at specific position, got the new size again and printed the values, tried to remove an element, got the new size and printed the elements again.

For the second part, the PriorityQueue part, I did something similar. I added 5 elements, checked the return of the add method(that returns the position that the element was added) and tried to delete some numbers.

4. Test Cases

The test cases from the first part of the Lab, the Linked List part:

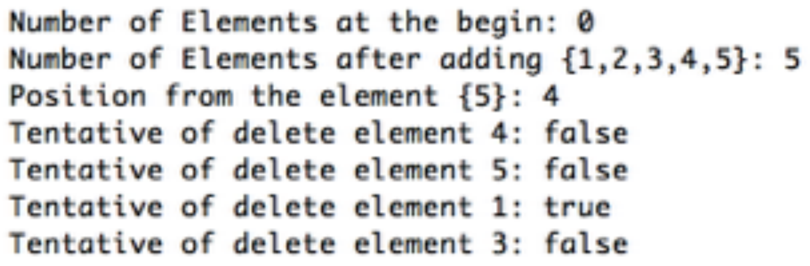


```
Size before adding: 0
Adding {1, 2, 3}.
Size after adding: 3
Value from the first position: 1
Value from the second position: 2
Value from the third position: 3
Value from the 4th position: null

Adding the element 10 at the second position.
Size after adding the element 10: 4
Value from the first position: 1
Value from the second position: 10
Value from the third position: 2
Value from the 4th position: 3

Removing the first element
Size after removing the first element: 3
Value from the first position: 10
Value from the second position: 2
Value from the third position: 3
Value from the 4th position: null
```

The test cases from the second part of the Lab, the Priority Queue part:



```
Number of Elements at the begin: 0
Number of Elements after adding {1,2,3,4,5}: 5
Position from the element {5}: 4
Tentative of delete element 4: false
Tentative of delete element 5: false
Tentative of delete element 1: true
Tentative of delete element 3: false
```

5. Analysis and Conclusions

Working with LinkedList was very interesting, it doesn't seem to be so "practical" and it is strange how it works and keeps track of the elements because I'm used to work with arrays but it's interesting. When we are working with arrays trying to exclude/add elements or resizing it is very painful but LinkedLists makes it very easy and simple, we just ignore what we don't want and points the node for what it is interesting.

The Priority Queue it is very interesting because it takes O(log n) to insert or remove elements(in some cases we can have O(1)) and it is easy to implement but it is a good approach only in very specific cases. Sometimes you don't want to loose data and Priority Queue only excludes the smallest element from the queue, which is good but also can be dangerous.

6. References

The parameters used was from the homework assignment provided in class and I used as reference for learning how to work with Priority Queue the Stack Overflow(http://bit.ly/1gS00Ne) and another references from an Algorithms course from Princeton.