Abiola Gabriel Olofin

CS 150-02

Professor Falconi

03/06/2020

Lab 6 Report

Introduction

In this lab, we are asked to create our own version of the data types, queue and stacks using a linkedlist. We are to create a singly linked, non-circular list where each node contains only one link to the next node and the overall list has a head and a tail. In addition, we have to create an iterator for the linkedlist. We also had to implement the linkedlist as a generic class in order to have different types of object use the linkedlist. We then add to implement the linkedlist as stacks and queues. Stacks are ADT where elements are first in, last out while queues are ADT where elements are first in, first out.

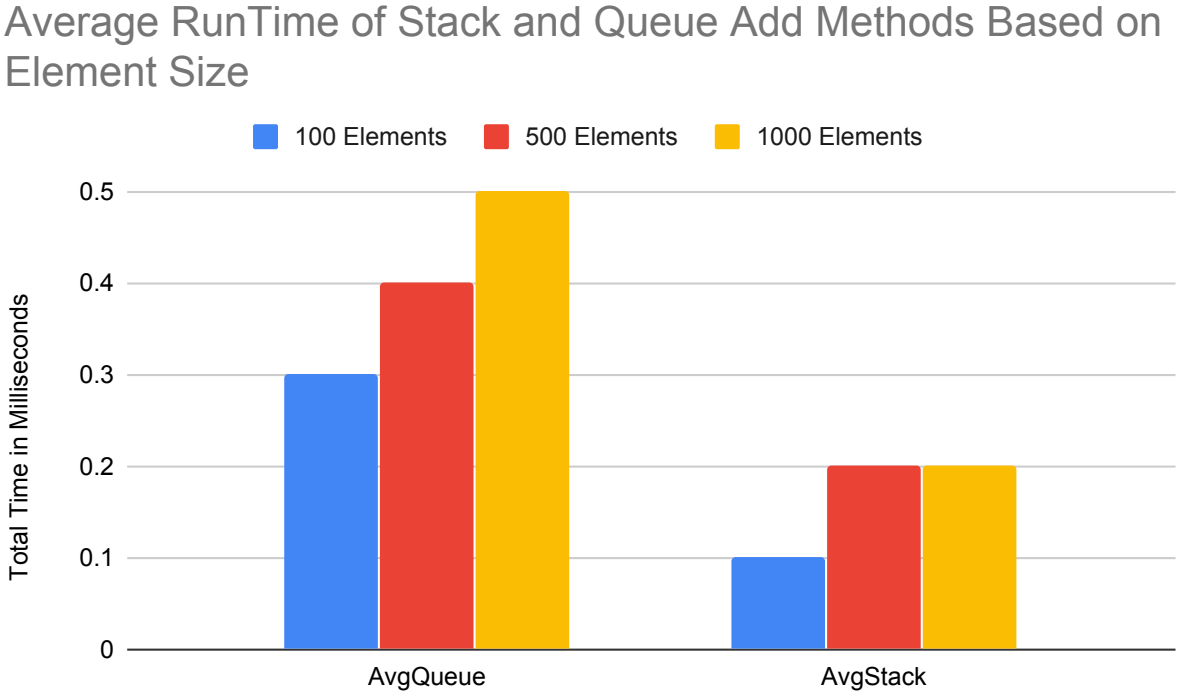
Approach

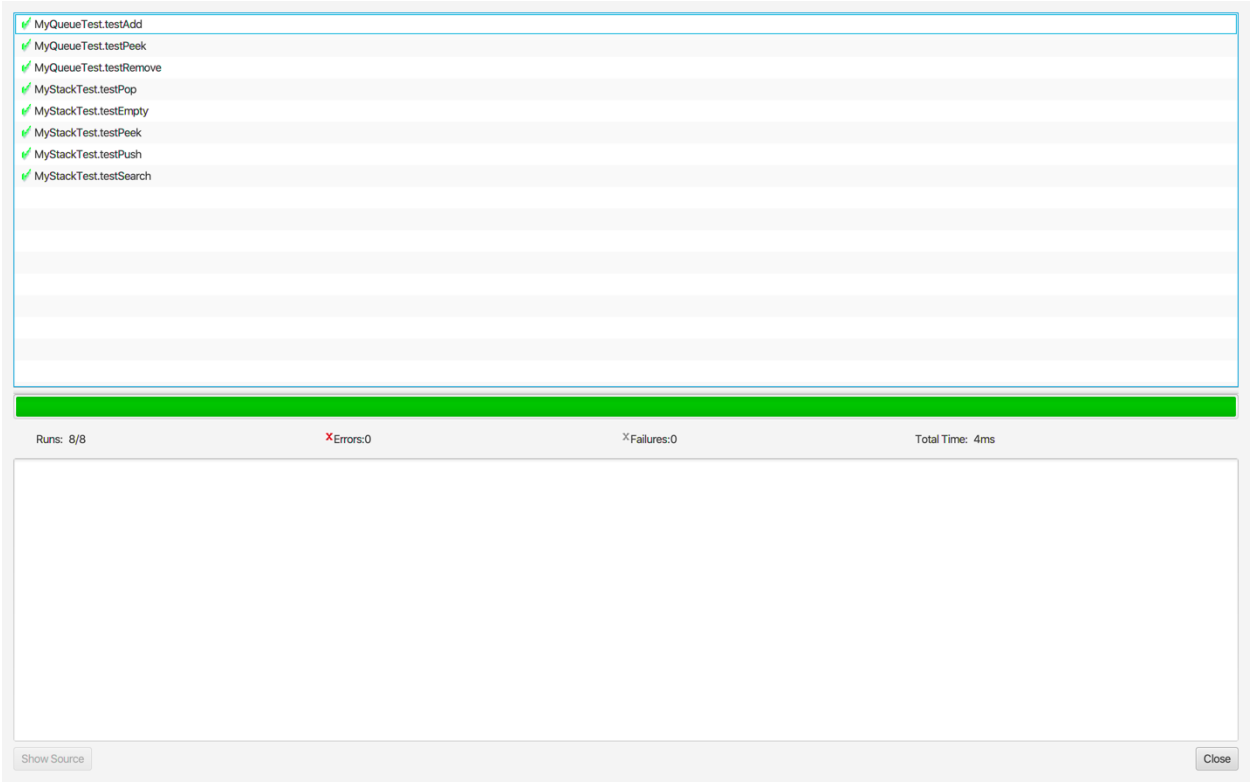
When I was designing the program, I had to create a linkedList class which implemented the iterable interface. The methods in the linkedlist class were iterator(), addFirst(<E>value), addEnd(<E> value), and a getElement(int x). The iterator() method returns an instance of the MyLinkedListIterator class, the addFirst method adds the value that was put in as a parameter to the front of the list, the addEnd method adds the value that was put in as a parameter to the end of the list, and the getElement method takes in a integer to represent the index you want to retrieve the element from and it returns the element at that index. The next classes I created were the Node<E> class and MyLinkedListIterator class which implements the iterator interface.

For MyLinkedListIterator class, I made methods hasNext() which checks to see if the node is pointing to a next node to iterate through the list and next() which move to the next node in the list and keeps the iterator going. Before I created the MyStack and MyQueue classes, I had to create the MyStackInterface and MyQueueInterface which are then implemented in the MyStack and MyQueue classes respectively. The methods in the MyStack class that needed to be implemented are empty() which checks whether the Stack is empty or not, peek() which returns the element at the top of the stack, pop() which removes the object at the top of the stack, push() which pushes an item to the top of the stack, and search(object o) which returns the distance to the first occurrence of the element you are looking for. The methods in the MyQueue class that needed to be implemented are add() which inserts an element into the queue, peek() which returns the first element in the queue, and remove() which removes the head of the queue.

Data

I collected data to graph by running 10 trials of the timed add methods to see how long it takes the computer on average to add elements to either a stack or queue and which data structure took more time to add methods to due to the size of the stack or queue.


Average RunTime of Stack and Queue Add Methods Based on Element Size
100 Elements
500 Elements
1000 Elements
Total Time in Milliseconds
0.5
0.4
0.3
0.2
0.1
0
AvgQueue
AvgStack


MyQueueTest.testAdd
MyQueueTest.testPeek
MyQueueTest.testRemove
MyStackTest.testPop
MyStackTest.testEmpty
MyStackTest.testPeek
MyStackTest.testPush
MyStackTest.testSearch
Runs: 8/8
Errors:0
Failures:0
Total Time: 4ms
Show Source
Close

Conclusion

Overall, this program was successful, and it taught me how to implement a LinkedList data structure as either a Stack or queue. It also showed me that stacks usually take shorter time to add elements to than queues.

References

ArrayList API

Random API

MyLinkedList API

file:///Users/abiolaolofin/Desktop/CS150Labs/Lab4/doc/MyLinkedList.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab4/doc/Node.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab4/doc/MyLinkedListIterator.html

file:///Users/abiolaolofin/Desktop/CS150Labs/Lab6/doc/StackInterface.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab6/doc/MyStack.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab6/doc/MyQueue.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab6/doc/QueueInteface.html
file:///Users/abiolaolofin/Desktop/CS150Labs/Lab6/doc/ExperimentController.html