

Double-Linked List

A doubly linked list is a data structure which allows reading, writing and deletion at any node in linear time.

I. Language Agnostic Overview

Linked lists are fundamentally simple data structures which were developed in the 1950's by workers at the RAND Corporation. Normal (singly-linked) lists allow traversal of the list in only one direction, however the implementation discussed here allows for traversal in both a backwards and forwards fashion. Errors in lists can be subtle and unnoticed since pointers can get mishandled or assigned incorrectly. The impact of these errors can be significant, however, and require much debugging time and effort. The difficulty of detection and impact these errors can have justifies further analysis.

The source code for the Doubly-Linked List class was developed by Sun as part of version 1.4 of the Java framework.

Implementations of doubly-linked lists are typically required to satisfy the following constraints:

1. Any node A that is linked to by node B should also have a link back to B.
2. A node should never have links whose target is unknown – nodes at the beginning and end of the list should point to sentinel nodes, null nodes, or each other.

II. Java Specific Overview

getFirst

- | |
|--|
| <p>a. This returns the first element in the list.</p> <p>b. The precondition calls the repOK function (explained below) which checks for a number of common problems. It also verifies that there is at least one element in the list. The postcondition again calls the repOK function.</p> |
|--|

removeLast

- | |
|--|
| <p>a. This method removes the last element in the list.</p> <p>b. The precondition and postcondition both call the repOK function (explained below) which checks to make sure that basic assumptions about the list still hold true.</p> <pre> return last;
 }</pre> |
|--|

remove

- | |
|--|
| <p>a. This method deletes the first occurrence of the specified element in the list.</p> |
|--|

b. The preconditions check all the invariants coded into repOK (explained below) as well. The postcondition again calls repOK.

clear

a. This method deletes everything in the list.

b. The precondition checks to make sure the repOK function (explained below) returns true. The postcondition rechecks repOK and makes sure that all the elements were successfully deleted.

indexOf

a. This method returns the index of a given element. The index is an integer representing the element's position such that were the list an array, you could retrieve the element with the returned index. This method finds the first occurrence of the given element, searching forwards from the beginning.

b. The precondition for this method checks to make sure all of the conditions in the repOK function hold. The postcondition then verifies that the indexOf method didn't break any of those conditions.

lastIndexOf

a. This method returns the index of a given element. The index is an integer representing the element's position such that were the list an array, you could retrieve the element with the returned index. This method finds the last occurrence of the given element, searching backwards from the end.

b. The precondition for this method checks to make sure all of the conditions in the repOK function hold. The postcondition then rechecks the repOK method.

addBefore

a. This method adds the given object before the given entry. It should be noted that this is an internal method, and not available in the public API.

b. The preconditions first check the repOK function (explained below) and then make sure the referenced element is actually in the list. The postcondition calls the repOK function.

toArray

a. This returns an array containing all of the objects in the list in the correct order.

b. The preconditions check to make sure that all the conditions in repOK (explained below) hold. The postcondition makes sure a non-null array is returned, and that its size matches that of the list.

repOK

- a. This is the repOK method, used in most of the invariants in the DoubleLinkedList class. It performs a number of checks, and returns false if any of them fail and true only if they all pass:
- The header cannot be null.
 - No element (except the header) can contain any null pointers, and the node after the current node must point back to the current element.
 - The list cannot be empty.
- b. This method has no invariants, as it is an internal method used for checking other function's invariants.

Metrics:

Lines of code for implementation: 277

Methods for implementation: 32

Lines of code for invariants: 95

Methods for invariants: 10

References:

Wikipedia's entry on doubly-linked lists: http://en.wikipedia.org/wiki/Linked_list#Doubly-linked_list

Data Structures, Algorithms and Applications in Java by Sartaj Sahni. Section 6.3: “Doubly Linked Lists.” pg 212.