COMP 250

Lecture 5

singly linked lists

Sept. 18, 2017
Recall last lecture: Java array

array of int

<table>
<thead>
<tr>
<th>34</th>
<th>657</th>
<th>-232</th>
<th>-823</th>
<th>23</th>
<th>1192</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

array of Shape objects

array (unspecified type)

I have drawn each of these as array lists.
Java ArrayList class

[https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html)

It uses an array as the underlying data structure.

It grows the array (by 50%, not 100%) when the array is full and a new element is added.

You don’t use the usual array notation `a[ ]`. Instead, use `get()` and `set()` and other methods.
Java generic type

An array of what? ArrayList<T>

Example:

```java
ArrayList<Shape> shape = new ArrayList<Shape>();
// initializes the array length (capacity) to 10

ArrayList<Shape> shape = new ArrayList<Shape>(23);
// initializes the array length to 23
```
Java ArrayList object

Has private field that holds the number of elements in the list (size).

Has a private field that references an array object.

These Shape objects do not belong to the ArrayList object.

Rather they are referenced by it.
Lists

• array list

• singly linked list  (today)

• doubly linked list  (next lecture)
array list

linked list

size = 4
array list

Array slots are in consecutive locations (addresses) in memory, but objects can be anywhere.

linked list

Linked list “nodes” and objects can be anywhere in memory.
Singly linked list node ("S" for singly)

class SNode<E> {
    SNode<E> next;
    E element;
}

e.g. E might be Shape
A linked list consists of a sequence of nodes, along with a reference to the first (head) and last (tail) node.
class SLinkedList<E> {

    SNode<E> head;
    SNode<E> tail;
    int size;

    :

    private class SNode<E> {  // inner class

        SNode<E> next;
        E element;
    :

    }

}
Linked list operations

• addFirst ( e )

• removeFirst( )

• addLast ( e )

• removeLast( )

• ....... many other list operations
addFirst ( △ )
addFirst ( e )  pseudocode

construct newNode
newNode.element = e 
newNode.next  = head
addFirst ( e ) pseudocode

construct newNode
newNode.element = e
newNode.next = head

// edge case
if head == null
tail = newNode

head = newNode
size = size+1
removeFirst ( )

BEFORE

AFTER
removeFirst( )  pseudocode

tmp = head
removeFirst() pseudocode

tmp = head
head = head.next
tmp.next = null

size = size – 1
removeFirst() edge cases (size is 0 or 1)

tmp = head

if (size == 0)
    throw exception
head = head.next
tmp.next = null
size = size – 1

if (size == 0) // size was 1
tail = null
<table>
<thead>
<tr>
<th>Operation</th>
<th>Array List Time Complexity</th>
<th>Linked List Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeFirst</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
Worse Case Time Complexity \((N = \text{size})\)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Array List</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst</td>
<td>(O(N))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>removeFirst</td>
<td>(O(N))</td>
<td>(O(1))</td>
</tr>
<tr>
<td>addLast</td>
<td>(O(1))*</td>
<td>?</td>
</tr>
<tr>
<td>removeLast</td>
<td>(O(1))</td>
<td>?</td>
</tr>
</tbody>
</table>

*if array is not full*
addLast ( ▲ )

BEFORE

AFTER

head

head

tail

tail
addLast ( △ )

ewNode = construct a new node
newNode.element = the new list element
tail.next = newNode

// ... and then after what
// figure shows we do:

tail = tail.next
size = size+1
Problem: we have no *direct* way to access the node before tail.
```java
removeLast()

if (head == tail) {
    head = null
    tail = null
} else {
    tmp = head
    while (tmp.next != tail) {
        tmp = tmp.next
    }
    tail = tmp
    tail.next = null
}
size = size - 1
// to return the element, you need to do a bit more
```
# Time Complexity

\( (N = \text{list size}) \)

<table>
<thead>
<tr>
<th>Method</th>
<th>Array List</th>
<th>Linked List</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFirst</td>
<td>( O(N) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>removeFirst</td>
<td>( O(N) )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>addLast</td>
<td>( O(1)* )</td>
<td>( O(1) )</td>
</tr>
<tr>
<td>removeLast</td>
<td>( O(1) )</td>
<td>( O(N) )</td>
</tr>
</tbody>
</table>

* if array is not full
class SLinkedList<E> {

    SNode<E> head;
    SNode<E> tail;
    int size;

    // various methods

private class SNode<E> { // inner class

    SNode<E> next;
    E element;

    //
    
}

//

}
class SLinkedList<E> {

  SNode<E> head;
  SNode<E> tail;
  int size;

  :

}
How many objects?

SLinkedList object

head

tail

size
4
How many objects?

1 \text{ SLinkedList} + 4 \text{ SNode} + 4 \text{ Shape} = 9
Announcements

• When I make mistakes on slides/lecture notes/exercises, please email me rather posting on discussion board. (However, compare the date on your version with the one on the public web page. I may have already corrected it.)

• Assignment 1 should be posted tomorrow (due in 2 weeks)

• Quiz 1 on Monday, Sept 25  (lectures 1-2, 4-6). Online.

• Coding tutorials on lists (coming soon)