Linked Lists

Overview

A linked list is a data structure made up of link objects or nodes. Each node contains a value and a pointer to the next node in the list or null if it's the last node in the list.

Example class for a Linked List of ints also known as an IntList.

```
class Node {
    public int val;
    public Node next;

    Node(int val) {
        this.val = val;
        this.next = null;
    }
}
```

Functionalities

Inserting

1. Traverse linked list to desired place
2. Create or insert node at desired place
3. Point 'next' variable of new node to the rest of list if necessary

Inserting into a linked list is $\Theta(1)$. The insertion of the new node takes $\Theta(1)$, but often times we need to account for how much work was done to reach the place of insertion, which is $O(n)$, so overall we say inserting is $O(n)$.
Example code:

```
/* Inserts the value v to the end of the lst Linked List
*/
public void insert(Node lst, int v) {
while (lst != null) {
    lst = lst.next;
}
lst.next = new Node(v);
}
```

Retrieving

1. Traverse linked list to desired place
2. Return/use node or node.val

Just like inserting, the actual retrieving takes $\Theta(1)$, but we need to account for how much work it took to get to the specific place, so overall it is $O(n)$.
Example code:

```
/* Returns the value of the Node at the index'th index of the lst Linked List.
*/
public int get(Node lst, int index) {
for (int i = 0; i < index; i++) {
    if (lst == null) {
        return -1;
```

```
        }
        lst = lst.next;
    }
    return lst.val;
    }
```

Containing

1. Traverse linked list to end
2. Check if element exists while traversing

Traversing the linked list means possibly visiting every node, so this is $O(n)$.
Example code:

```
/* Returns true if the value v is in the lst Linked List, otherwise returns false.
*/
public boolean contains(Node lst, int v) {
    while (lst != null) {
        if (lst.val == v) {
            return true;
        }
        lst = lst.next;
    }
    return false;
}
```

General Advice

1. To be able to solve **any** linked list question, there's really only one thing you need to do *consistently...*

ALWAYS DRAW BOX AND POINTER DIAGRAMS

Trust me on this one. It might take a little time to draw the pictures, but if done correctly, I assure you that you will be able to solve any *"what would this code output"* and have the pieces to make most code writing questions less daunting. To consistently draw correct box and pointer diagrams, you will need to have a good understanding of objects and object references.

2. When writing code using linked lists, I stick by one golden rule to help me solve the problem.

One variable (reference) can only serve one purpose

When solving problems with linked lists, often times you need to do multiple things to achieve a functionality/solution. These include traversing the linked list, reassigning a pointer, and, most importantly, keeping a reference to a part of the list. Let's say I want to add an element to a linked list. I could write something like this.

```
Node lst; // assume 'lst' is an already populated linked list
while (lst != null) {
    lst = lst.next;
}
lst.next = new Node(v); // where 'v' is the value to be added
```

This seems like it would work perfectly fine, but actually the reference to the beginning of the list was lost as a result. ***This is different from function version because of pass by reference***. Drawing a box and pointer diagram will make this very clear. So before solving coding questions with linked lists, figure out what what functionalities you will need and use the appropriate amount of variables to solve the problem.

TL;DR

1. Each node contains a value and a pointer to the next node or null
2. Getting to a specific index takes as much work as the number of nodes traversed, or $O(n)$
3. Act of inserting takes $O(1)$, but inserting to the end may take as much as $O(n)$
4. Each pointer/reference can only serve one purpose at a time
5. Understand pass by reference for objects when working with functions
6. Get good at drawing box and pointer diagrams and **ACTAULLY DRAW THEM**