L11: Linked List and Pointer Problems

Alex Steiger
CompSci 201: Spring 2024
2/19/2024

Logistics, Coming up

• Monday, 2/19 (today)
  • Project 2: Markov Due
  • Project 3: DNA out by tomorrow

• Thursday, 2/22
  • APT Quiz 1 due

• Next Monday 2/26
  • Nothing due 🎉
  • Work on Project 3: DNA, due the following week

Today’s Outline

• Part 1: LinkedList review + low-level details

• Part 2: Implementing DIYLinkedList

• Part 3 / Wed: Working directly with List Node objects, algorithmic problem-solving
  1. Get to index'th node
  2. Append one list to another
  3. Reverse a list in place
What is the runtime complexity of the reverseCopy method as a function of n where n is the size of myList?

```java
public static list<Integer> reverseCopy(Linkedlist<Integer> mylist) {
    list<Integer> reversed = new linkedList();
    for (Integer val : mylist) {
        // add val to front of list
        reversed.add(0, val);
    }
    return reversed;
}
```

What is the runtime complexity of the removeZeros method as a function of n, the number of elements in the list? Answer in the worst case / without making any assumptions about the elements of the input myList.

```java
public static void removeZeros(Linkedlist<Integer> mylist) {
    for (int i = 0; i < mylist.size(); i++) {
        if (mylist.get(i) == 0) {
            mylist.remove(i);
        }
    }
}
```

What is the runtime complexity of the removeZeros method as a function of n, the number of elements in the list? Answer in the worst case / without making any assumptions about the elements of the input myList.

The Java API documentation clarifies that the remove() method on an Iterator “Removes from the underlying collection the last element returned by this iterator.”

```java
public static void removeZeros(Linkedlist<Integer> mylist) {
    Iterator<Integer> listIter = mylist.iterator();
    while (listIter.hasNext()) {
        if (listIter.next() == 0) {
            listIter.remove();
        }
    }
}
```
Linked List, Low-level DIY perspective

Contrasting how things look to your computer / in memory

**Array/ArrayList**
- Elements laid out sequentially, one at a time, in order, in memory.
- `myArray`
- 5 11 6 7

**LinkedList**
- Elements at arbitrary locations in memory, connected only by references to the next element.
- `myLinkedList`
- 5 6 11 7

Memory and references

- In Java, **variables for reference types** (anything that is an object/not a primitive) really **store the location of the object in memory**.
- Can have **multiple references** to the same object in memory!

```java
6 7 8 9 10
list<String> words = new LinkedList<String>();
words.add("CS");
otherWords = words;
otherWords.add("201");
System.out.println(words);
```

Prints "CS", "201", only one actual List in memory!
Multiple objects or multiple references

Java creates a reference type object in memory only when the code calls the `new` operator.

First example create 2 distinct empty lists, but...

Second example creates one list in memory with two references / variable names.

---

Pass by value of reference

- Java does NOT copy all of `words` when we call this method.
- Copies the reference (memory address) and passes that, $O(1)$ time [memory addresses are 64 bits].
- Changes relevant outside of method.

More Pass by value of reference

- Why does it matter that Java passes a copy of the reference to methods?
- Cannot "lose" a reference inside a method.
Null reference/pointer

- The default value for an uninitialized (no memory allocated by a call to new) object is null.

- Can check if an object == null.
  - We will use to denote the end of a linked list, the node with no more nodes following.

- If you try to call any methods on a null object, will get a NullPointerException error.

Linked list is a list implemented by linked nodes. What is a node?

- Just a Java object of a class we write, like any other!
- We want to "link" them together, so each node has a pointer (really a reference = a memory location) to another node.

Creating and traversing a linked list

- ListNode class used in APTs, etc.
  - The variable for the "linked list itself" is just a reference to the first ListNode
Creating Nodes, constructing lists

1. Calling `new Node(…)` always creates a Node in memory that did not exist before

2. Writing `node.next = otherNode;` makes `node` point to `otherNode`

3. `node.next` or `node.info` gives an error (null pointer exception) if `node` is null
Again suppose we run the following code.

```java
ListNode myList = new ListNode(2, new ListNode(0, new ListNode(1)));
```

What is `myList.next.next.next.next`?

- 1
- The third ListNode object
- null
- error, null pointer exception

myList.next.next.next.next causes a NullPointerException

Consider the following code, assume the printList method prints the values in a list (meaning everything from a given starting ListNode and following next references until reaching null). What would be printed by line 19, which prints `null`?

```java
public static ListNode foo(ListNode list) {
    list = list.next;
    return list;
}
```

- nothing
- 2
- 2.0

Same code: What would be printed by line 19, which prints `list`?

```java
public static ListNode foo(ListNode list) {
    list = list.next;
    return list;
}
```

- nothing
- 0
- 2.0
WOTO Answers

What would line 18 print? 0

What would line 19 print? 2, 0

```java
public static ListNode foo(ListNode list) {
    list = list.next;
    list.next = null;
    return list;
}

public static void main(String[] args) {

    ListNode list = new ListNode(0, new ListNode(1, new ListNode(2, null));

    ListNode ret = foo(list);

    printList(ret);
    printList(list);
}
```

DIYLinkedList

Live Coding 🤖