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? *

- O(1)
- O(n) [Correct]
- O(n^2)
- O(n^3)
What is the runtime complexity of the `removeZeros` method be 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);
        }
    }
```

- O(1)
- O(n)
- O(n^2)
- O(n^3)
What is the runtime complexity of the removeZeros method be 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 \( \text{myList} \).

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();
        }
    }
}
```

- \( O(1) \)
- \( O(n) \) ✔
- \( O(n^2) \)
- \( O(n^3) \)
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 List<String> words = new LinkedList<>();
7 words.add("CS");
8 List<String> otherWords = words;
9 otherWords.add("201");
10 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.

```
11 List<String> listA = new LinkedList<>();
12 List<String> listB = new LinkedList<>();
```

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

```
11 List<String> listA = new LinkedList<>();
12 List<String> listB = listA;
```

Second example creates *one* list in memory with two references / variable names.
Pass by value of reference

```java
12   public static void removeFront(List<String> words) {
13       words.remove(0);
14   }
```

- 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.

```java
6       List<String> words = new LinkedList<>();
7       words.add("CS");
8       removeFront(words);
9       System.out.println(words);
```

Prints `[]` (empty), change to words in method changes the only List in memory. Different for primitive types.
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.

```java
public static void tryBreakReference(List<String> words) {
    words = new LinkedList<>();
}
```

```java
List<String> words = new LinkedList<>();
words.add("CS");
tryBreakReference(words);
System.out.println(words);
```

Even though this reassigns `words` in the method...

Still prints [“CS”], only the copy of the reference was reassigned.
Null reference/pointer

• The default value for an uninitialized (no memory allocated by a call to new) object is \texttt{null}.

• Can check if an object \texttt{== 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 \texttt{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.

```java
public class ListNode {
    int info;
    ListNode next;
    ListNode(int x){
        info = x;
    }
    ListNode(int x, ListNode node){
        info = x;
        next = node;
    }
}
```

```java
ListNode first = new ListNode(5);
ListNode second = new ListNode(3);
first.next = second;

info = 5;
next = null;
next = x012;
```

```java
info = 3;
next = null;
```

Address x001
Address x012
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**

```java
ListNode list = new ListNode(5);
list.next = new ListNode(7);
list.next.next = new ListNode(9);
print(list);
```

```java
public static void printList(ListNode list) { 
    while(list != null) {
        System.out.println(list.info);
        list = list.next;
    }
}
```
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
This and following questions reference the ListNode class shown. Suppose we run the following code:

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

What is `myList.next.next`? *

- 0
- The second ListNode object
- The third ListNode object
- null
Again suppose we run the following code.

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

What is `myList.next.info`?

- 0
- The second ListNode object
- 1
- The third ListNode object
- 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`? *

- 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 18, which prints ret? *

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

```java
public static void main(String[] args) {
    ListNode list = new ListNode(info: 2, new ListNode(info: 0, new ListNode(info: 1)));
    ListNode ret = foo(list);
    printList(ret);
    printList(list);
}
```

- nothing
- 0
- 2, 0
- 2, 0, 1
Same code. What would be printed by line 19, which prints list? *  

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

```java
public static void main(String[] args) {
    ListNode list = new ListNode(info: 2, new ListNode(info: 0, new ListNode(info: 1)));
    ListNode ret = foo(list);
    printList(ret);
    printList(list);
}
```

- nothing
- 0
- 2, 0
- 2, 0, 1

**Correct Answer:** 2, 0
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(info: 2, new ListNode(info: 0, new ListNode(info: 1)));
    ListNode ret = foo(list);
    printList(ret);
    printList(list);
}
```
DIYLinkedList

Live Coding