L13: Recursion

Alex Steiger
CompSci 201: Spring 2024
2/26/2024
Person in CS: Ellen Ochoa

- BS physics (‘75), PhD Elec. Eng. (‘85)
- Starting working on software for optical recognition systems (computer vision)
- Applied to be an astronaut in...
  - ‘85...rejected
  - ‘87...rejected
  - ‘90...accepted!!!
- Worked on flight software, computer hardware, and robotics
- First Hispanic woman in space ‘93
- Director of NASA Johnson Space Flight Center (Houston) ‘13
Announcements, Coming up

• Today, Monday 2/26
  • Nothing due

• Wednesday 2/28
  • APT 5 (linked list problems) due

• Next Monday 3/4
  • Project P3: DNA (linked list project) due
Today’s outline

• Introducing Recursion
  • Counting ListNodes
  • Reversing a LinkedList

• Live Coding w/ Recursion
  • (Time permitting)
Toward Recursion by counting nodes: Iterative vs. Recursive

• Standard linked list iteration
  • Advance local pointer, do something at each node

    ```java
    public int countIter(ListNode list) {
        int total = 0;
        while (list != null) {
            total += 1;
            list = list.next;
        }
        return total;
    }
    ```

• Recursion?
  • Base Case?
  • General case?
  • Define size in terms of size?
Key ideas in recursion

1. Base case: Solve for answer when instance is “small”

2. General case:
   1. Get answer on smaller instance(s) of the same problem using recursive call(s)
   2. Do something with the result of the recursive call(s) and then return

• Note: Methods/calls stacked, like all methods
Thinking recursively

1. When is the input small enough that the answer is trivial? Base case.

2. Otherwise, suppose a magical fairy (the Recursion Fairy!) could solve the **exact same problem** on a **smaller** input

3. Could you solve the larger problem given what the fairy told you?

Base case

Recurse

Use result
The call stack: How recursion works on a machine

• Each method call gets its own call frame (local variables, etc.)

• Eager evaluation: Invoking method does not resume until invoked method returns.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Eager evaluation and substitution

- Return value will be substituted into the expression calling the method.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```
Eager evaluation and substitution

- Return value will be substituted into the expression calling the method.
Eager evaluation and substitution

• Return value will be substituted into the expression calling the method.
Eager evaluation and substitution

- Return value will be substituted into the expression calling the method.
Counting Nodes

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}

int result = size(ptr);
System.out.println(result);
```

Recursive runtime

- Concept is the same: Count the number of constant time operations...across all recursive calls!

- **Ensure each recursive call gets closer to the base case**, else code may run forever.

```java
public int size(ListNode list) {
    if (list == null) return 0;
    return 1 + size(list.next);
}
```

- Moves one node toward the base case at each step.
- List of N nodes, makes O(N) total recursive calls, each takes O(1) time
- Overall O(N) runtime complexity.
Recall the reverse problem

- How do we reverse nodes in a linked list
  - Go from A->B->C to C->B->A
  - Typical interview style question
  - [https://leetcode.com/problems/reverse-linked-list/](https://leetcode.com/problems/reverse-linked-list/)
  - [https://www.hackerrank.com/challenges/reverse-a-linked-list](https://www.hackerrank.com/challenges/reverse-a-linked-list)
Base case, words and code

• Base case: When is there nothing to do?
  • A list with 0 or 1 nodes is its own reverse

```java
3 public static ListNode reverse(ListNode list) {
4     if (list == null || list.next == null) {
5         return list;
6     }
```
Recursive step in words

• Suppose the Recursion Fairy (a recursive call) reverses the list after the first node.

• How to use? Just put the first node at the end!

• Restated: The reverse of a list is the reverse of all but the first element, with the first element added to the end.
Recursive step in pictures

```
list -> A -> B -> C -> ||

Returned by recursive call on list.next

list -> A -> C -> B -> ||

reversedFirst -> reversedLast

Make reversedLast point to what list points to

Return reversedFirst
```
Recursive step in code

7  ListNode reversedLast = list.next;
8  ListNode reversedFirst = reverse(list.next);

Note that list.next still refers to reversedLast
Recursive step in code (continued)

9  \[\text{reversedLast.next} = \text{list};\]  Make \(B\) point to \(A\)
10 \[\text{list.next} = \text{null};\]  Make \(A\) point to \text{null}
11  \[\text{return reversedFirst};\]  Return overall reversed list
public static ListNode reverse(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode reversedLast = list.next;
    ListNode reversedFirst = reverse(list.next);
    reversedLast.next = list;
    list.next = null;
    return reversedFirst;
}
Revisiting the call stack: How it really works

reverse(list) → A → B → C

reverse(list) → reverse(list)
Revisiting the call stack: How it really works

reverse(list) → A → B → C

reverse(list) → reversedFirst
Revisiting the call stack: How it really works

Back to the case we considered first
Consider the rec method. If the input list is ['A', 'B', 'C'], what will be returned by rec(list)?

```java
public static ListNode rec(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = rec(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    return after;
}
```

Answer: ['A', 'B', 'C']
If the input list is ['C', 'B', 'A'], what will be returned by rec(list)?

```
public static ListNode rec(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = rec(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    return after;
}
```

Answer: ['A']
For an input list with \( N \) nodes, the best characterization of the runtime complexity of \( \text{rec}(\text{list}) \) is...

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

    ListNode after = rec(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }

    return after;
}
```

Answer: \( O(N) \)
Consider the mystery method. Note that it is the same as rec except for lines 24-29. If the input list is ['C', 'B', 'A'], what will be returned by mystery(list)?

```java
public static ListNode mystery(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = mystery(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    ListNode current = after;
    while (current.next != null && list.info > current.next.info) {
        current = current.next;
    }
    list.next = current.next;
    current.next = list;
    return after;
}
```

Answer: ['A', 'B', 'C']
Same mystery method. For an input list with N nodes, the best characterization of the runtime complexity of mystery(list) is...

```java
public static ListNode mystery(ListNode list) {
    if (list == null || list.next == null) {
        return list;
    }
    ListNode after = mystery(list.next);
    if (list.info <= after.info) {
        list.next = after;
        return list;
    }
    ListNode current = after;
    while (current.next != null && list.info > current.next.info) {
        current = current.next;
    }
    list.next = current.next;
    current.next = list;
    return after;
}
```

Answer: $O(N^2)$