Final Report: Reverse Linked List with Constant Space Complexity

Student: Carlos Dip

Description

The goal is to reverse a linked list, such as: $$A\rightarrow B\rightarrow C\rightarrow D \Longrightarrow D\rightarrow C\rightarrow B\rightarrow A$$ Without allocating any non-linear amount of additional memory (such as arrays, lists, sets, etc.). Since this list is only singly-linked, the challenge comes from doing the necessary changes to pointers without losing track of elements inside the list.

Solution and Complexity

The proposed solution requires the use of three variables to store pointers into the input list (meaning it is of constant memory use, as required by the problem), as well $O(n)$ time complexity, as the list is scanned through only once.

The goal is to reverse the linked list. To do so, it would suffice to simply reverse the pointers on every node, such that children nodes point to their parents. Of course, a single linked list does not have a parent attribute, so the search through the list is a little more in-depth.

$$A\rightarrow B\rightarrow C\rightarrow D \Longrightarrow A\leftarrow B\leftarrow C\leftarrow D$$

First, you must keep track of three nodes at any given moment, call them **previous**, **current** and **next**. The algorithm is as follows: Initially, **previous** and **next** are set to null, and **current** is set to the head node of the list (as received via input to the function). We then store the **current** node's next value in the **next** pointer. Then we update our **current** node's next pointer to the **previous** value, set **previous** to be our **current** pointer and update **current** to match **next**. We repeat this untill **current** has no next node (end of list).

Hints:

1. Do you need to manipulate the values inside the nodes?
2. Maybe you can keep track of previous node pointers as you traverse the list?
3. How far back do you need to keep track of the node pointers? 1, 2, 3 nodes back?

Assessed skills:

In this interview, I was able to assess several things, but mainly how familiar interviewees were with linked lists, pointers, and also quite importantly, how organized they were in their thought process. This problem requires constant manipulation of at least **3** node pointers, so it can be very easy to lose track and get confused about which variable is pointing to which node.

Common Mistakes:

Most commonly, subjects attempted to manipulate values initially, trying to move them to new nodes at the end of the list, or swapping them around.

Also pretty commonly, subjects attempted to understand solutions that began from the end of the list, which is very impractical and often caused more confusion since the list is only singly-linked, meaning that solutions like this can only ever be of $O(n^2)$ time complexity (or worse).

General Problems:

No general problems, all the interviewees were very different, and achieved different amounts of completion of the proposed problem.

Identification of Pros and Cons:

It is much easier to identify the strengths and weaknesses of candidates when they all work on the same problem. You start to accumulate experience in different ways of thinking to solve the same problem, and you learn which paths work, and which don't. It is very interesting to observe a candidate come up with an idea that you know won't work, and try and measure how long they take to realize it won't work. Some candidates might never give up on their bad ideas, while some abandon ideas too quickly.

Examples

There was really only one example that drew a lot of attention, which was one of the interviewees who came up with a very nice recursive solution initially, which altough is linear in space complexity (not inplace), was very intuitive, and in a good direction. I had not though of any recursive solutions before this, and really liked the idea. It would have been better if they had noticed it themselves that it was not inplace, however, it was still a very good starting point. Then, the following interviewee started to develop another recursive solution also, which led to me to ponder whether or not a recursive solution is valid.

My conclusion was that it actually meets the requirements for the proposed problem, since the only restriction was to alter the list inplace, which the recursion did. This caused me to reformulate the question into "Reverse a linked list with constant space complexity".