Doubly Linked List

This section explores operations on a doubly linked list

We ended the section on linked list with a question on the complexity of an operation that seeks to find the parent of a given linked list node. The answer is O(n) since if you give me a linked list node, I still need to start from the head and traverse down the linked list matching each node with the one given to me until I find an exact match. At that point, I return the just previously seen node that I can keep track of using an additional variable. Worst case, it takes me to the end of the linked list for a total of n operations.

Doubly Linked List

Meet doubly linked list. It solves the above described problem by keeping a link to the previous node. Sure, we now have double the number of nodes, but the space complexity doesn't change.

> $$n + 2n * (pointer\ variable\ size)$$

Since the pointer variable size would be a constant number of bytes in any language, we'll be left with,

> $$n + 2n * (constant)$$
>
> $$n + n * 2c$$
>
> $$(2c + 1)n$$
>
> $$O(n)$$

However, by using the additional back pointer node, we are now able to determine the predecessor for a node in constant time or O(1). The complexity for the rest of the operations remains the same as the linked list.

One may wonder what the benefit is of finding the predecessor in constant time. Usually, a doubly linked list is combined with a hash table to create a more advanced data structures called a least recently used cache or LRU cache.

Pop Quiz

Q Given a doubly linked list node, what is the complexity of deleting the node?

○ A) O(n^2)

○ B) O(n)

○ C) O(1)

CHECK ANSWERS