Singly Linked Lists vs. Doubly Linked Lists

Let's see how the two renditions of the linked list structure fare against each other.

We'll cover the following

Which is Better?

DLLs have a few advantages over **SLLs**, but these perks do not come without a cost:

- Doubly linked lists can be traversed in both directions, which makes them more compatible with complex algorithms.
- Nodes in doubly linked lists require extra memory to store the `previous_element` pointer.
- Deletion is more efficient in doubly linked lists as we do not need to keep track of the previous node. We already have a backwards pointer for it.

At this point, we've compared the two major types of linked lists. The minor memory load that comes with DLLs can be forgone because of the convenience they provide.

This doesn't mean that DLLs are perfect. There is always room for improvement!

Let's discuss a tweak which can improve the functionality of both types.

Tail Pointer in a Linked List

The `head` node is essential for any linked list, but what if we also kept account of the **tail** of the list? Now, you are aware of both ends of a linked list.

To add the `tail` functionality, all we have to do is add another member to our `LinkedList` class:

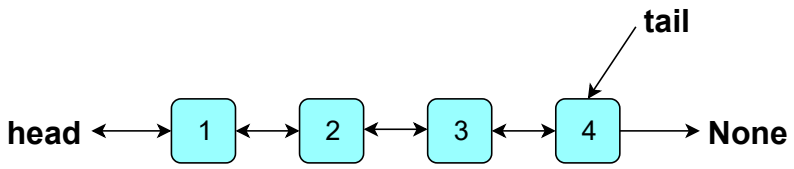
```
class Node:
    def __init__(self, data):
        self.data = data
        self.next_element = None
        self.previous_element = None


class LinkedList:
    def __init__(self):
        self.head_node = None
        self.tail_node = None  # Keep track of the last

```

In a singly linked list, `insert_at_tail` now works in **O(1)**. We can simply set the new node as the `next_element` of the previous end node and update the `tail_node`.

However, the `tail` really shines in doubly linked lists.

Apart from tail operations, insertion and deletion become twice as fast because we can traverse the list from both sides.

Here is an illustration of the modified doubly linked list:



The **tail** updates every time a new node is added at the end or a node is deleted from the end. The good news is that these operations are just as fast as `delete_at_head` and `insert_at_head`.

We've covered all there is to know about the mechanics of linked lists. In the following lessons, we'll take a look at several coding challenges which test your knowledge on linked lists. These exercises will also be explained in detail so don't shy away from them.

See you there!

Back

Doubly Linked Lists (DLL)

Next

Challenge 4: Find the Length of a Link...

Mark as Completed

Report an Issue

Ask a Question (https://discuss.educative.io/tag/singly-linked-lists-vs-doubly-linked-lists__introduction-to-linked-lists__data-structures-for-coding-interviews-in-python)