Singly Linked List Deletion

After insertion and search, we'll be tackling the concept of deletion in a linked list.

We'll cover the following

Introduction

The **deletion** operation combines principles from both **insertion** and **search**. It uses the search functionality to find the value in the list.

Deletion is one of the instances where linked lists are more efficient than arrays. In an array, you have to shift all the elements backward if one element is deleted. Even then, the end of the array is empty and it takes up unnecessary memory.

In the case of linked lists, the node can simply be removed in *constant time*.

Let’s take a look at the different types of deletion operations we can perform in singly linked lists.

Types of Deletion

There are three basic delete operations for linked lists:

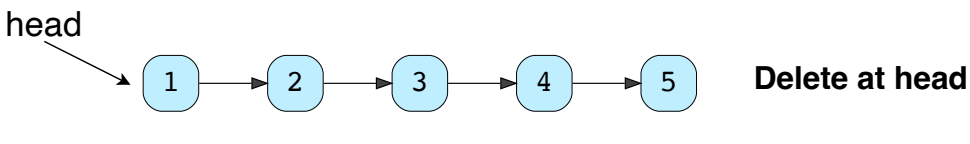
1. Deletion at the head
2. Deletion by value
3. Deletion at the tail

In this lesson, we will look at the implementation of the **deletion at head** algorithm. The rest will be covered in the following lessons.

Delete at Head

This operation simply deletes the first node from a list. If the list is empty, the function does nothing.

Here's an illustration of how this type of deletion works:



1 of 4

Implementation

Now that we've seen it work in theory, let's shape the function in the form of Python code.

main.py

LinkedList.py

Node.py

```
from LinkedList import LinkedList
from Node import Node


def delete_at_head(lst):
    # Get Head and firstElement of List
    first_element = lst.get_head()

    # if List is not empty then link head to the
    # nextElement of firstElement.
    if first_element is not None:
        lst.head_node = first_element.next_element
        first_element.next_element = None
    return


lst = LinkedList()
for i in range(11):
    lst.insert_at_head(i)

lst.print_list()

delete_at_head(lst)
delete_at_head(lst)

lst.print_list()
```

Explanation

Time Complexity: O(1)

There is nothing too complicated going on here. We access the first element of the list

```
first_element = lst.get_head()
```

`first_element` can either be a node (the list is not empty) or not intialized (if the list is empty).

If a node is found, its `next_element` becomes the `head`.

Now, `first_element` has been removed from the linked list [illegible] from memory will be handled by Python since we haven't specified a destructor.

In the next lesson, we will discuss the second deletion strategy, **deletion by value**.