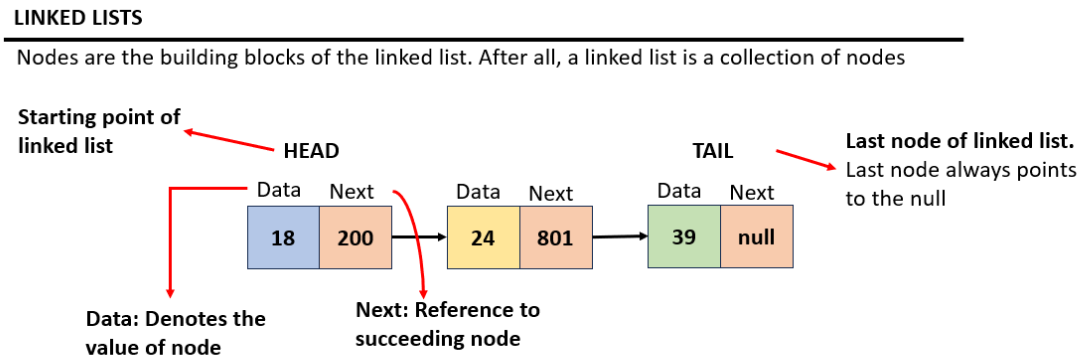
DAILY DSA | DAY-10 | Linked lists – Part 1|
-GOPALKRISHNA A

Some of the data structures we have gone through so far are arrays, lists, Dictionaries/maps, and sets. They all do a good job storing and accessing data.

But, **Linked lists** come with their own advantages and also serve as a **building block for advanced data structures** like stacks, queues, and graphs.

Analogy:
Just like a garland is made with flowers, a linked list is made up of nodes.
Every flower in the garland is a node
like how each flower points to the next, each node points to the next node in this list by holding the current value & address of the next node (sometimes including the previous)

Linked list: Linked list is a linear data structure used for storing a collection of elements. Unlike arrays, linked lists use nodes to store elements where each node contains data as well as the memory address of the next node in the list.



Here, as you can see that the addresses of the nodes are not necessarily immediate sequential. The first node has an address of **200** and the second node has an address of **801,** instead of 201 as you might expect

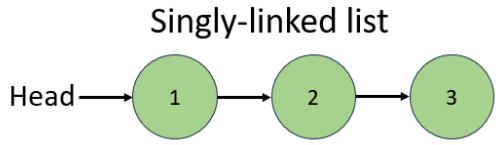
How are nodes stored linearly?

Even though the nodes are not in a contiguous memory, the nodes are stored linearly through links. Every node has the address of its succeeding node. That is how each node can access its succeeding node.

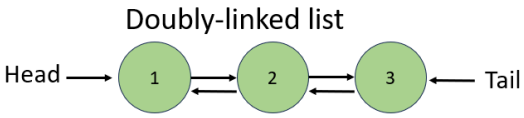
Linked lists allow to insertion and removal of arbitrary elements from a list without reallocating or moving elements, preventing the worst performance case of an array.

Types of linked lists:

1. **Singly-linked list:** Consists of nodes that store references to the next node in the list. Operations that can be performed on singly linked lists are **insertion**, **deletion**, and **traversal**



2. **Doubly-linked list:** Consists of nodes that store references to the previous & next node in the list.

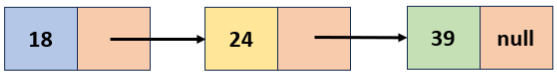


3. **Circular linked list**: Singly linked list in which the last node, **next** field points to the first node in the sequence

Singly-linked list:

SINGLY-LINKED LIST

Items are connected in order via pointers, making adding/removing elements flexibly to one end of the list easy to do, but also making lookup difficult



	Avg & Worst case
Space	O(N)
Index (Lookup)	O(N)
Search	O(N)
Append	O(1)
Insert	O(N)
Delete	O(N)

Pros

- Very simple and easy to implement with no pre-defined size
- Easy to add or remove elements from one end of the linked list

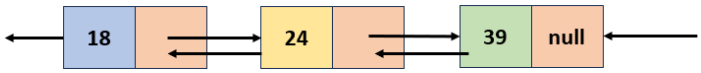
Cons

- Inefficient at look up. Accessing elements in the middle of the list requires traversal through the list since you can only start at the beginning of the linked list
- Not cache friendly. Elements are not stored in adjacent pieces of memory

Doubly-linked list

DOUBLY-LINKED LIST

Items are connected in order via pointers, making adding/removing elements flexibly to one end of the list easy to do, but also making lookup difficult



	Avg & Worst case
Space	O(N)
Index (Lookup)	O(1)
Search	O(N)
Append	O(1)
Insert	O(N)
Delete	O(N)

Pros

- When at a node, the previous and the next node can be accessed
- Allows for easy manipulation at the front and the back of the list

Cons

- Takes more memory than a singly linked list since it requires storing previous pointers as well
- Indexing still requires traversal through the list to reach a specific element in the list
- Not cache friendly, elements are not stored in adjacent pieces of memory

Note: In the Last post I missed attaching operations on dictionaries, and lists. Please find the attachment.