Operation	Singly Linked List	Doubly Linked List
Search	O(n)	O(n)
Insertion (at head)	O(1)	O(1)
Insertion (at tail)	O(1)	O(1)
Remove (at head)	O(1)	O(1)
Remove (at tail)	O(n)	O(1)
Remove (at middle)	O(n)	O(n)

> Insertion/Removal at tail at O(1) requires a tail pointer

Definition

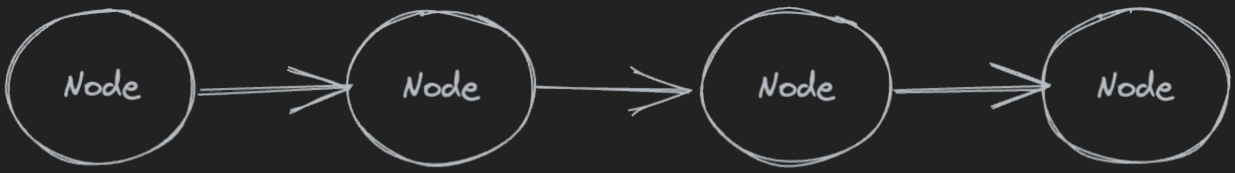
A linked list is a sequential list of nodes that hold data which points to other nodes also containing data. The last node points to null, there will be no more nodes after the null node

Terminology

Head	The first node in a linked list
Tail	The last node in a linked list
Pointer	Reference to another node
Node	An object containing data and pointer(s)

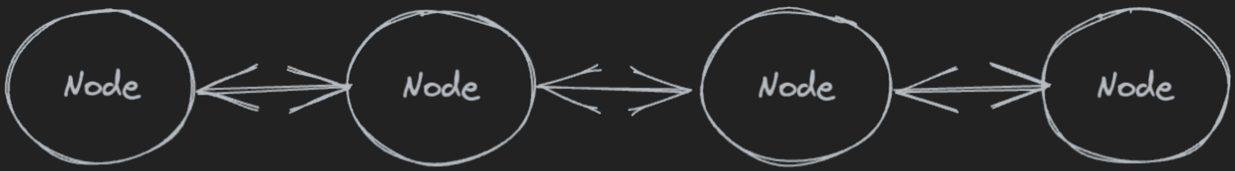
Singly Linked List

Only holds a reference to the next node



Doubly Linked List

Holds a reference to the previous and next node



Benefits and Losses

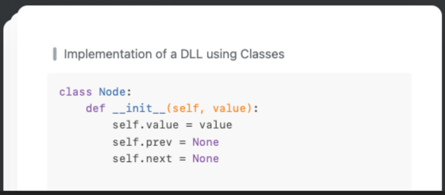
Singly Linked	Uses less memory Simpler implementation	Cannot easily access previous elements
Doubly Linked	Can be traversed backwards	Takes double the memory

Use Cases

- Many list, queue, stack, and adjacency list implementations
- Circular lists
- Can model real world objects such as trains
- Used in separate chaining to deal with hashing collision

Doubly Linked List Implementation

> Implementation of a DLL using Classes



> Implementation of a DLL using Classes

```
class Node:
    def __init__(self, value):
        self.value = value
        self.prev = None
        self.next = None
```

↑ Linked List

Doubly Linked List Implementation

> Implementation of a DLL using Classes

```
class Node:
    def __init__(self, value):
        self.value = value
        self.prev = None
        self.next = None


class DoublyLinkedList:
    def __init__(self):
        self.length = 0
        self.head = None
        self.tail = None

    def prepend(self, item):
        node = Node(item)

        self.length += 1
        if not self.head:
            self.head = self.tail = node
            return

        node.next = self.head
        self.head.prev = node
        self.head = node

    def insert_at(self, item, idx):
        if idx > self.length:
            raise IndexError('Linked List is not long enough')
        elif idx == self.length:
            self.append(item)
            return
        elif idx == 0:
            self.prepend(item)
            return

        self.length += 1
        curr = self.get_at(idx)
        node = Node(item)

        node.next = curr
        node.prev = curr.prev
        curr.prev = node

        if node.prev:
            node.prev.next = curr

    def append(self, item):
        node = Node(item)

        self.length += 1
        if not self.tail:
            self.head = self.tail = node
            return

        node.prev = self.tail
        self.tail.next = node
        self.tail = node

    def remove(self, item):
        curr = self.head
```