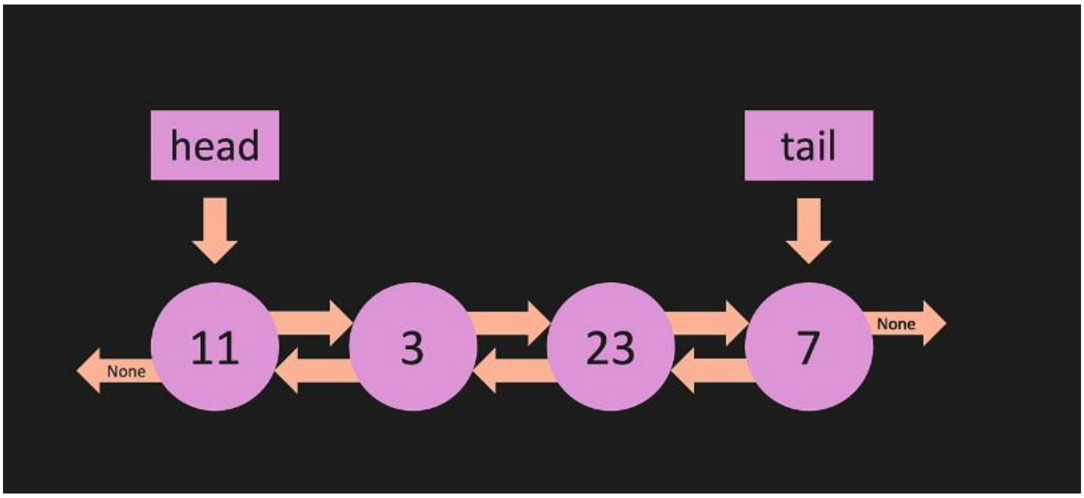
Doubly Linked List



Constructor:

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


class DoublyLinkedList:
    def __init__(self, value):
        new_node = Node(value)
        self.head = new_node
        self.tail = new_node
        self.length = 1
```

This code defines a **Node** class and a **DoublyLinkedList** class.

1. The **Node** class is defined with an **__init__** method that takes a value as its input. It sets the **value** property of the **Node** instance to the given value and initializes the **next** and **prev** properties to **None**. These **next** and **prev** properties will be used to reference the next and previous nodes in the doubly linked list.

2. The **DoublyLinkedList** class is defined with an **__init__** method that takes a value as its input. It creates a new **Node** instance called **new_node** with the given value. The **head**, **tail**, and **length** properties of the **DoublyLinkedList** instance are initialized as follows:
    - The **head** property is set to **new_node**, indicating the beginning of the list.
    - The **tail** property is set to **new_node**, indicating the end of the list.
    - The **length** property is set to 1, as there is only one node in the list at this point.

Append to DLL

```
def append(self, value):
    new_node = Node(value)
    if self.head is None:
        self.head = new_node
        self.tail = new_node
    else:
        self.tail.next = new_node
        new_node.prev = self.tail
        self.tail = new_node
    self.length += 1
    return True
```

The **append** method is to add a new node with the given value to the end of the doubly linked list.

1. Create a new **Node** instance called **new_node** with the given value.
2. Check if the head of the doubly linked list is **None**, which means the list is empty. a. If the list is empty, set both the **head** and **tail** of the list to **new_node**, because it's now the only node in the list.
3. If the list is not empty, perform the following steps: a. Set the **next** property of the current **tail** (last node) to **new_node**. This connects the new node to the end of the list. b. Set the **prev** property of **new_node** to the current **tail**. This connects the new node to the previous node in the list. c. Update the **tail** property of the list to point to **new_node**, as it's now the new last node in the list.
4. Increment the **length** property of the list by 1, as we've added a new node to the list.
5. Return **True** to indicate that the operation was successful.

Pop from DLL

```
def pop(self):
    if self.length == 0:
        return None
    temp = self.tail
    if self.length == 1:
        self.head = None
        self.tail = None
    else:
        self.tail = self.tail.prev
        self.tail.next = None
        temp.prev = None
    self.length -= 1
    return temp
```

1. **def pop(self):**: This line defines a method called **pop** which will be a part of a class. The **self** parameter refers to the instance of the class itself.
2. **if self.length == 0:**: This is a check to see if the linked list is empty. The attribute **length** keeps track of the number of nodes in the list.
3. **return None**: If the list is empty, the method returns **None**.
4. **temp = self.tail**: Here, we save the current tail node of the linked list in a temporary variable called **temp**.
5. **if self.length == 1:**: This checks if the list has only one node.
6. **self.head = None; self.tail = None**: If there's only one node, both the head and tail pointers are set to **None**, effectively emptying the list.
7. **else:**: This is the case where the list has more than one node.
8. **self.tail = self.tail.prev**: The tail pointer is updated to point to the node just before the current tail.
9. **self.tail.next = None; temp.prev = None**: The next pointer of the new tail is set to **None**. Also, the previous pointer of the node that we are going to return is set to **None**.
10. **self.length -= 1**: Decrease the length of the list by 1, as we are removing a node.
11. **return temp**: Finally, the method returns the node that was removed from the list.