Cheatsheets / **CS102: Data Structures and Algorithms**

Doubly Linked Lists

Adding to the Tail

A Python `DoublyLinkedList` class can implement an `.add_to_tail()` instance method for adding new data to the tail of the list. `.add_to_tail()` takes a single `new_value` argument. It uses `new_value` to create a new `Node` which it adds to the tail of the list.

```
def add_to_tail(self, new_value):
  new_tail = Node(new_value)
  current_tail = self.tail_node

  if current_tail != None:
    current_tail.set_next_node(new_tail)
    new_tail.set_prev_node(current_tail)

  self.tail_node = new_tail

  if self.head_node == None:
    self.head_node = new_tail
```

Adding to the Head

A Python `DoublyLinkedList` class can implement an `.add_to_head()` instance method for adding new data to the head of the list. `.add_to_head()` takes a single `new_value` argument. It uses `new_value` to create a new `Node` which it adds to the head of the list.

```
def add_to_head(self, new_value):
  new_head = Node(new_value)
  current_head = self.head_node

  if current_head != None:
    current_head.set_prev_node(new_head)
    new_head.set_next_node(current_head)

  self.head_node = new_head

  if self.tail_node == None:
    self.tail_node = new_head
```

Removing the Tail

A Python `DoublyLinkedList` class can implement a `.remove_tail()` instance method for removing the head of the list. `.remove_tail()` takes no arguments. It removes and returns the tail of the list, and sets the tail's previous node as the new tail.

```
  def remove_tail(self):
    removed_tail = self.tail_node

    if removed_tail == None:
      return None

    self.tail_node
= removed_tail.get_prev_node()

    if self.tail_node != None:
      self.tail_node.set_next_node(None
)

    if removed_tail == self.head_node:
      self.remove_head()

    return removed_tail.get_value()
```

Removing the Head

A Python `DoublyLinkedList` class can implement a `.remove_head()` instance method for removing the head of the list. `.remove_head()` takes no arguments. It removes and returns the head of the list, and sets the head's next node as the new head.

```
  def remove_head(self):
    removed_head = self.head_node

    if removed_head == None:
      return None

    self.head_node
= removed_head.get_next_node()

    if self.head_node != None:
      self.head_node.set_prev_node(None
)

    if removed_head == self.tail_node:
      self.remove_tail()

    return removed_head.get_value()
```

Removing by Value

A Python `DoublyLinkedList` class can implement a `.remove_by_value()` instance method that takes `value_to_remove` as an argument and returns the node that matches `value_to_remove`, or `None` if no match exists. If the node exists, `.remove_by_value()` removes it from the list and correctly resets the pointers of its surrounding nodes.

```
  def remove_by_value(self, value_to_remove):
    node_to_remove = None
    current_node = self.head_node

    while current_node != None:
      if current_node.get_value() == value_to_remove:
        node_to_remove = current_node
        break

      current_node = current_node.get_next_node()

    if node_to_remove == None:
      return None

    if node_to_remove == self.head_node:
      self.remove_head()
    elif node_to_remove == self.tail_node:
      self.remove_tail()
    else:
      next_node = node_to_remove.get_next_node()
      prev_node = node_to_remove.get_prev_node()
      next_node.set_prev_node(prev_node)
      prev_node.set_next_node(next_node)

    return node_to_remove
```

Constructor

A Python `DoublyLinkedList` class constructor should store:

- A `head_node` property to store the head of the list
- A `tail_node` property to store the tail of the list

The `head_node` and `tail_node` are set to `None` as their defaults.

```
class DoublyLinkedList:
  def __init__(self):
    self.head_node = None
    self.tail_node = None
```

Updated Node Class

Doubly linked lists in Python utilize an updated `Node` class that has a pointer to the previous node. This comes with additional setter and getter methods for accessing and updating the previous node.

```python
class Node:
  def __init__(self, value, next_node=None, prev_node=None):
    self.value = value
    self.next_node = next_node
    self.prev_node = prev_node

  def set_next_node(self, next_node):
    self.next_node = next_node

  def get_next_node(self):
    return self.next_node

  def set_prev_node(self, prev_node):
    self.prev_node = prev_node

  def get_prev_node(self):
    return self.prev_node

  def get_value(self):
    return self.value
```

Doubly Linked List Overview

A `DoublyLinkedList` class in Python has the following functionality:

- A constructor with `head_node` and `tail_node` properties
- An `.add_to_head()` method to add new nodes to the head
- An `.add_to_tail()` method to add new nodes to the tail
- A `.remove_head()` method to remove the head node
- A `.remove_tail()` method to remove the tail node
- A `.remove_by_value()` method to remove a node that matches the `value_to_remove` passed in

```
class DoublyLinkedList:
  def __init__(self):
    self.head_node = None
    self.tail_node = None


  def add_to_head(self, new_value):
    new_head = Node(new_value)
    current_head = self.head_node

    if current_head != None:
      current_head.set_prev_node(new_head)
      new_head.set_next_node(current_head)

    self.head_node = new_head

    if self.tail_node == None:
      self.tail_node = new_head


  def add_to_tail(self, new_value):
    new_tail = Node(new_value)
    current_tail = self.tail_node

    if current_tail != None:
      current_tail.set_next_node(new_tail)
      new_tail.set_prev_node(current_tail)

    self.tail_node = new_tail

    if self.head_node == None:
      self.head_node = new_tail


  def remove_head(self):
    removed_head = self.head_node

    if removed_head == None:
      return None
    self.head_node
= removed_head.get_next_node()

    if self.head_node != None:
      self.head_node.set_prev_node(None
)

    if removed_head == self.tail_node:
      self.remove_tail()

    return removed_head.get_value()


  def remove_tail(self):
    removed_tail = self.tail_node

    if removed_tail == None:
      return None

    self.tail_node
= removed_tail.get_prev_node()

    if self.tail_node != None:
      self.tail_node.set_next_node(None
)

    if removed_tail == self.head_node:
      self.remove_head()

    return removed_tail.get_value()


  def remove_by_value(self,
value_to_remove):
    node_to_remove = None
    current_node = self.head_node

    while current_node != None:
      if current_node.get_value() ==
value_to_remove:
        node_to_remove = current_node
        break

      current_node
= current_node.get_next_node()

    if node_to_remove == None:
      return None

    if node_to_remove ==
self.head_node:
      self.remove_head()
    elif node_to_remove ==
self.tail_node:
      self.remove_tail()
    else:
      next_node
= node_to_remove.get_next_node()
      prev_node
= node_to_remove.get_prev_node()
      next_node.set_prev_node(prev_node
)
      prev_node.set_next_node(next_node
)

    return node_to_remove
```