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


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

    def append(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.prev = None
            self.head = new_node
        else:
            cur = self.head
            new_node = Node(data)
            while cur.next:
                cur = cur.next
            cur.next = new_node
            new_node.prev = cur
            new_node.next = None

    def prepend(self, data):
        if self.head is None:
            new_node = Node(data)
            new_node.prev = None
            self.head = new_node
        else:
            new_node = Node(data)
            self.head.prev = new_node
            new_node.next = self.head
            self.head = new_node
            self.head.prev = None

    def print_list(self):
        cur = self.head
        while cur:
            print(cur.data)
            cur = cur.next

    def add_after_node(self, key, data):
        cur = self.head
        while cur:
            if cur.next is None and cur.data == key:
                self.append(data)
                return
            elif cur.data == key:
                new_node = Node(data)
                nxt = cur.next
                cur.next = new_node
                new_node.next = nxt
                new_node.prev = cur
                nxt.prev = new_node
            cur = cur.next

    def add_before_node(self, key, data):
        cur = self.head
        while cur:
            if cur.prev is None and cur.data == key:
                self.prepend(data)
                return
            elif cur.data == key:
                new_node = Node(data)
                prev = cur.prev
                prev.next = new_node
```

```
                cur.prev = new_node
                new_node.next = cur
                new_node.prev = prev
            cur = cur.next

    def delete(self, key):
        cur = self.head
        while cur:
            # Case 1   # Deleting head node
            if cur.data == key and cur == self.head:
                if not cur.next:
                    cur = None
                    self.head = None
                    return
                # Case - 2  # Deleting head node attached to a node
                else:
                    nxt = cur.next
                    cur.next = None
                    nxt.prev = None
                    cur = None
                    self.head = nxt
                    return
            # Case - 3 - Deleting middle node in a list
            elif cur.data == key:
                if cur.next:
                    nxt = cur.next
                    prev = cur.prev
                    prev.next = nxt
                    nxt.prev = prev
                    cur.next = None
                    cur.prev = Node
                    cur = Node
                    return
                else:
                    # Case - 4 Deleting the tail node
                    prev = cur.prev
                    prev.next = None
                    cur.prev = None
                    cur = None
                    return
            cur = cur.next

    def delete_node(self, node):
        cur = self.head
        while cur:
            # Case 1   # Deleting head node
            if cur == node and cur == self.head:
                if not cur.next:
                    cur = None
                    self.head = None
                    return
                # Case - 2  # Deleting head node attached to a node
                else:
                    nxt = cur.next
                    cur.next = None
                    nxt.prev = None
                    cur = None
                    self.head = nxt
                    return
            # Case - 3 - Deleting middle node in a list
            elif cur == node:
                if cur.next:
                    nxt = cur.next
                    prev = cur.prev
                    prev.next = nxt
                    nxt.prev = prev
                    cur.next = None
                    cur.prev = Node
```

```
                    cur = Node
                    return
                else:
                    # Case - 4 Deleting the tail node
                    prev = cur.prev
                    prev.next = None
                    cur.prev = None
                    cur = None
                    return
            cur = cur.next

    def reverse(self):
        tmp = None
        cur = self.head
        while cur:
            tmp = cur.prev
            cur.prev = cur.next
            cur.next = tmp
            cur = cur.prev
        if tmp:
            self.head = tmp.prev

    def remove_duplicates(self):
        cur = self.head
        seen = dict()
        while cur:
            if cur.data not in seen:
                seen[cur.data] = 1
                cur = cur.next
            else:
                nxt = cur.next
                self.delete_node(cur)
                cur = nxt

    def pairs_with_sums(self, sum_val):
        pairs = list()
        p = self.head
        q = None
        while p:
            q = p.next
            while q:
                if p.data + q.data == sum_val:
                    pairs.append("(" + str(p.data) + "," + str(q.data) + ")")
                q = q.next
            p = p.next
        return pairs
```