JavaScript Reverse Linked List II

Challenge

Given the `head` of a singly linked list and two integers `left` and `right` where `left <= right`, reverse the nodes of the list from position `left` to position `right`, and return the reversed list.

1st Example

```
Input: head = [1,2,3,4,5], left = 2, right = 4
Output: [1,4,3,2,5]
```

2nd Example

```
Input: head = [5], left = 1, right = 1
Output: [5]
```

Constraints

- `1 <= n <= 500`
- `-500 <= Node.val <= 500`
- `1 <= left <= right <= n`
- The number of nodes in the list is `n`.

Solution

```
const reverseBetween = (head, m, n) => {
    let start = head,
        cur   = head,
        i     = 1;

    while (i < m) {
        start = cur;
        cur   = cur.next;
        i++;
    }

    let prev = null,
        tail = cur;

    while (i <= n) {
        let next = cur.next;

        cur.next = prev;
        prev     = cur;
        cur      = next;
        i++;
    }

    start.next = prev;

    tail.next = cur;

    return m == 1 ? prev : head;
};
```

Explanation

I've written a function called `reverseBetween` that takes three

parameters: `head`, `m`, and `n`. The purpose of this function is to reverse a portion of a linked list, starting from the `m` th node and ending at the `n` th node.

Inside the function, three variables are initialized: `start`, `cur`, and `i`. These variables keep track of the starting node, current node, and the index, respectively. They are all initially set to the `head` node.

A `while` loop is used to find the starting node of the reversed portion. The loop runs until the index `i` is less than `m`. In each iteration, the `start` variable is updated to the current node, and the `cur` variable is moved to the next node. The index `i` is incremented by `1`.

After exiting the first loop, the `start` variable will be pointing to the node before the reversed portion, and the `cur` variable will be pointing to the `m` th node.

Two more variables, `prev` and `tail`, are initialized. The `prev` variable will be used to reverse the nodes, and the `tail` variable will be the last node of the reversed portion. They are initially set to `null` and `cur`, respectively.

Another `while` loop is used to reverse the nodes from `m` to `n`. The loop runs until the index `i` is less than or equal to `n`. In each iteration, the next node of the `cur` variable is stored in a variable called `next`. This is done to keep track of the next node before reversing the current node's `next` pointer. The current node's `next` pointer is then reversed to point to the previous node (`prev`). The `prev` variable is updated to the current node, and the

`cur` variable is moved to the next node (`next`). The index `i` is incremented by `1`.

After exiting the second loop, the `prev` variable will be pointing to the last node of the reversed portion, and the `cur` variable will be pointing to the node after the reversed portion.

The `next` pointer of the `start` node is set to `prev` to connect the reversed portion to the rest of the list. The `next` pointer of the `tail` node is set to `cur` to connect the end of the reversed portion to the remaining nodes.

Finally, the function returns `prev` if `m` is equal to `1`, indicating that the reversal starts from the beginning of the list. Otherwise, it returns `head`, which is the original head of the list.

In summary, this function reverses a portion of a linked list by modifying the `next` pointers of the nodes. It iterates through the list to find the starting and ending nodes of the portion to be reversed, and then reverses the `next` pointers of the nodes within that portion.

Author: Trevor Morin