**Assignment12**

**1.**

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function deleteMiddle(head) {

if (head === null || head.next === null) {

return null;

}

let slow = head;

let fast = head;

let prev = null;

while (fast !== null && fast.next !== null) {

fast = fast.next.next;

prev = slow;

slow = slow.next;

}

prev.next = slow.next;

return head;

}

const head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

head.next.next.next = new ListNode(4);

head.next.next.next.next = new ListNode(5);

const newHead = deleteMiddle(head);

let current = newHead;

while (current !== null) {

console.log(current.val);

current = current.next;

}

**Output:** 1

2

4

5

**2.**

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function hasLoop(head) {

let slow = head;

let fast = head;

while (fast !== null && fast.next !== null) {

slow = slow.next;

fast = fast.next.next;

if (slow === fast) {

return true;

}

}

return false;

}

const head = new ListNode(1);

head.next = new ListNode(3);

head.next.next = new ListNode(4);

head.next.next.next.next = head.next;

const hasLoopResult = hasLoop(head);

console.log(hasLoopResult);

**Output:** true

const head = new ListNode(1);

head.next = new ListNode(8);

head.next.next = new ListNode(3);

head.next.next.next = new ListNode(4);

const hasLoopResult = hasLoop(head);

console.log(hasLoopResult);

**Output**: false

**3.**

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function findNthFromEnd(head, n) {

let mainPtr = head;

let refPtr = head;

for (let i = 0; i < n; i++) {

if (refPtr === null) {

return -1;

}

refPtr = refPtr.next;

}

while (refPtr !== null) {

mainPtr = mainPtr.next;

refPtr = refPtr.next;

}

return mainPtr.val;

}

**Example1:**

const head = new ListNode(1);

head.next = new ListNode(2);

head.next.next = new ListNode(3);

head.next.next.next = new ListNode(4);

head.next.next.next.next = new ListNode(5);

head.next.next.next.next.next = new ListNode(6);

head.next.next.next.next.next.next = new ListNode(7);

head.next.next.next.next.next.next.next = new ListNode(8);

head.next.next.next.next.next.next.next.next = new ListNode(9);

const nthFromEnd = findNthFromEnd(head, 2);

console.log(nthFromEnd);

**Output:** 8

**Example2:**

const head = new ListNode(10);

head.next = new ListNode(5);

head.next.next = new ListNode(100);

head.next.next.next = new ListNode(5);

const nthFromEnd = findNthFromEnd(head, 5);

console.log(nthFromEnd);

**Output:** -1

**4.**

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function isPalindrome(head) {

if (head === null || head.next === null) {

return true;

}

let slow = head;

let fast = head;

while (fast !== null && fast.next !== null) {

fast = fast.next.next;

slow = slow.next;

}

let secondHalf = reverseList(slow.next);

let firstHalf = head;

while (secondHalf !== null) {

if (firstHalf.val !== secondHalf.val) {

return false;

}

firstHalf = firstHalf.next;

secondHalf = secondHalf.next;

}

return true;

}

function reverseList(head) {

let prev = null;

let current = head;

while (current !== null) {

let next = current.next;

current.next = prev;

prev = current;

current = next;

}

return prev;

}

**Example1**

const head1 = new ListNode('R');

head1.next = new ListNode('A');

head1.next.next = new ListNode('D');

head1.next.next.next = new ListNode('A');

head1.next.next.next.next = new ListNode('R');

const isPalindrome1 = isPalindrome(head1);

console.log(isPalindrome1);

**Output:** true

**Example 2:**

const head2 = new ListNode('C');

head2.next = new ListNode('O');

head2.next.next = new ListNode('D');

head2.next.next.next = new ListNode('E');

const isPalindrome2 = isPalindrome(head2);

console.log(isPalindrome2);

**Output:** false

5.

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function detectAndRemoveLoop(head) {

let slow = head;

let fast = head;

while (fast !== null && fast.next !== null) {

slow = slow.next;

fast = fast.next.next;

if (slow === fast) {

break;

}

}

if (fast === null || fast.next === null) {

return head;

}

slow = head;

while (slow.next !== fast.next) {

slow = slow.next;

fast = fast.next;

}

fast.next = null;

return head;

}

**Example 1:** N = 3, value[] = {1,3,4}, X = 2

const head1 = new ListNode(1);

head1.next = new ListNode(3);

head1.next.next = new ListNode(4);

head1.next.next.next = head1.next; // Create a loop

const newList1 = detectAndRemoveLoop(head1);

console.log(newList1);

**Output**: 1

// Example 2: N = 4, value[] = {1,8,3,4}, X = 0

const head2 = new ListNode(1);

head2.next = new ListNode(8);

head2.next.next = new ListNode(3);

head2.next.next.next = new ListNode(4);

const newList2 = detectAndRemoveLoop(head2);

console.log(newList2);

Output: 1

// Example 3: N = 4, value[] = {1,2,3,4}, X = 1

const head3 = new ListNode(1);

head3.next = new ListNode(2);

head3.next.next = new ListNode(3);

head3.next.next.next = new ListNode(4);

head3.next.next.next.next = head3.next; // Create a loop

const newList3 = detectAndRemoveLoop(head3);

console.log(newList3);

Output: 1

6.

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function deleteNodes(head, M, N) {

if (head === null || M <= 0 || N <= 0) {

return head;

}

let current = head;

let prev = null;

let mCount = 0;

let nCount = 0;

while (current !== null) {

mCount++;

nCount++;

if (mCount < M) {

prev = current;

current = current.next;

} else if (mCount === M) {

if (nCount <= N) {

prev.next = current.next;

current = prev.next;

} else {

mCount = 0;

nCount = 0;

prev = current;

current = current.next;

}

}

}

return head;

}

// Example 1: M = 2, N = 2, Linked List: 1->2->3->4->5->6->7->8

const head1 = new ListNode(1);

head1.next = new ListNode(2);

head1.next.next = new ListNode(3);

head1.next.next.next = new ListNode(4);

head1.next.next.next.next = new ListNode(5);

head1.next.next.next.next.next = new ListNode(6);

head1.next.next.next.next.next.next = new ListNode(7);

head1.next.next.next.next.next.next.next = new ListNode(8);

const modifiedList1 = deleteNodes(head1, 2, 2);

console.log(modifiedList1);

Output: 1->2->5->6

// Example 2: M = 3, N = 2, Linked List: 1->2->3->4->5->6->7->8->9->10

const head2 = new ListNode(1);

head2.next = new ListNode(2);

head2.next.next = new ListNode(3);

head2.next.next.next = new ListNode(4);

head2.next.next.next.next = new ListNode(5);

head2.next.next.next.next.next = new ListNode(6);

head2.next.next.next.next.next.next = new ListNode(7);

head2.next.next.next.next.next.next.next = new ListNode(8);

head2.next.next.next.next.next.next.next.next = new ListNode(9);

head2.next.next.next.next.next.next.next.next.next = new ListNode(10);

const modifiedList2 = deleteNodes(head2, 3, 2);

console.log(modifiedList2); // Output: 1->2->3->6->7->8

// Example 3: M = 1, N = 1, Linked List: 1->2->3->4->5->6->7->8->9->10

const head3 = new ListNode(1);

head3.next = new ListNode(2);

head3.next.next = new ListNode(3);

head3.next.next.next = new ListNode(4);

head3.next.next.next.next = new ListNode(5);

head3.next.next.next.next.next = new ListNode(6);

head3.next.next.next.next.next.next = new ListNode(7);

head3.next.next.next.next.next.next.next = new ListNode(8);

head3.next.next.next.next.next.next.next.next = new ListNode(9);

head3.next.next.next.next.next.next.next.next.next = new ListNode(10);

const modifiedList3 = deleteNodes(head3, 1, 1);

console.log(modifiedList3); // Output: 1->3->5->7->9

7.

class ListNode {

constructor(val, next = null) {

this.val = val;

this.next = next;

}

}

function insertAlternate(firstHead, secondHead) {

if (!firstHead || !secondHead) {

return firstHead;

}

let firstCurr = firstHead;

let secondCurr = secondHead;

while (firstCurr && secondCurr) {

let firstNext = firstCurr.next;

let secondNext = secondCurr.next;

firstCurr.next = secondCurr;

secondCurr.next = firstNext;

firstCurr = firstNext;

secondCurr = secondNext;

}

return firstHead;

}

// Example: First List: 5->7->17->13->11, Second List: 12->10->2->4->6

const firstHead = new ListNode(5);

firstHead.next = new ListNode(7);

firstHead.next.next = new ListNode(17);

firstHead.next.next.next = new ListNode(13);

firstHead.next.next.next.next = new ListNode(11);

const secondHead = new ListNode(12);

secondHead.next = new ListNode(10);

secondHead.next.next = new ListNode(2);

secondHead.next.next.next = new ListNode(4);

secondHead.next.next.next.next = new ListNode(6);

const mergedList = insertAlternate(firstHead, secondHead);

console.log(mergedList);

// Output: 5->12->7->10->17->2->13->4->11->6

console.log(secondHead);

// Output: null (second list becomes empty)

8.

class ListNode {

constructor(val) {

this.val = val;

this.next = null;

}

}

function isCircular(head) {

if (!head) {

return false;

}

let slow = head;

let fast = head;

while (fast && fast.next) {

slow = slow.next;

fast = fast.next.next;

if (slow === fast) {

return true;

}

}

return false;

}

// Example: Circular Linked List: 10->12->14->16

const head = new ListNode(10);

head.next = new ListNode(12);

head.next.next = new ListNode(14);

head.next.next.next = new ListNode(16);

head.next.next.next.next.next = head.next;

const isCircularList = isCircular(head);

console.log(isCircularList);

// Output: true