Problem 1171: Remove Zero Sum Consecutive Nodes from Linked List

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Given the

head

of a linked list, we repeatedly delete consecutive sequences of nodes that sum to

0

until there are no such sequences.

After doing so, return the head of the final linked list. You may return any such answer.

(Note that in the examples below, all sequences are serializations of

ListNode

objects.)

Example 1:

Input:

head = [1,2,-3,3,1]

Output:

[3,1]

Note:

The answer [1,2,1] would also be accepted.

Example 2:

Input:

head = [1,2,3,-3,4]

Output:

[1,2,4]

Example 3:

Input:

head = [1,2,3,-3,-2]

Output:

[1]

Constraints:

The given linked list will contain between

1

and

1000

nodes.

Each node in the linked list has

-1000 <= node.val <= 1000

.

Code Snippets

C++:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
ListNode* removeZeroSumSublists(ListNode* head) {

}
};
```

Java:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* int val;
* ListNode next;
* ListNode() {}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode removeZeroSumSublists(ListNode head) {
}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def removeZeroSumSublists(self, head: Optional[ListNode]) ->
Optional[ListNode]:
```

Python:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def removeZeroSumSublists(self, head):
"""
:type head: Optional[ListNode]
:rtype: Optional[ListNode]
"""
```

JavaScript:

```
/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {ListNode} head
* @return {ListNode}
*/
```

```
var removeZeroSumSublists = function(head) {

};
```

TypeScript:

```
/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function removeZeroSumSublists(head: ListNode | null): ListNode | null {

};
```

C#:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {
public ListNode RemoveZeroSumSublists(ListNode head) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* removeZeroSumSublists(struct ListNode* head) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func removeZeroSumSublists(head *ListNode) *ListNode {

}
```

Kotlin:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun removeZeroSumSublists(head: ListNode?): ListNode? {

}
}
```

Swift:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/
class Solution {
func removeZeroSumSublists(_ head: ListNode?) -> ListNode? {

}
}
```

Rust:

```
// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn remove_zero_sum_sublists(head: Option<Box<ListNode>>) ->
Option<Box<ListNode>> {

}
}
```

Ruby:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} head
# @return {ListNode}
def remove_zero_sum_sublists(head)

end
```

PHP:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param ListNode $head
* @return ListNode
*/
function removeZeroSumSublists($head) {

}
}
```

Dart:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? removeZeroSumSublists(ListNode? head) {

}
}
```

Scala:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
object Solution {
def removeZeroSumSublists(head: ListNode): ListNode = {

}
}
```

Elixir:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec remove_zero_sum_sublists(head :: ListNode.t | nil) :: ListNode.t | nil
```

```
def remove_zero_sum_sublists(head) do

end
end
```

Erlang:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec remove_zero_sum_sublists(Head :: #list_node{} | null) -> #list_node{} |
null.
remove_zero_sum_sublists(Head) ->
.
```

Racket:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (remove-zero-sum-sublists head)
(-> (or/c list-node? #f) (or/c list-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
class Solution {
public:
ListNode* removeZeroSumSublists(ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* public class ListNode {
```

```
* int val;
* ListNode next;
* ListNode() {
// TODO: Implement optimized solution
return 0;
}
* ListNode(int val) { this.val = val; }
* ListNode(int val, ListNode next) { this.val = val; this.next = next; }
* }
*/
class Solution {
public ListNode removeZeroSumSublists(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Zero Sum Consecutive Nodes from Linked List
Difficulty: Medium
Tags: hash, linked_list

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def removeZeroSumSublists(self, head: Optional[ListNode]) ->
Optional[ListNode]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# Definition for singly-linked list.
# class ListNode(object):
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution(object):
def removeZeroSumSublists(self, head):
"""
:type head: Optional[ListNode]
:rtype: Optional[ListNode]
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* function ListNode(val, next) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
*/
/**
* @param {ListNode} head
* @return {ListNode}
*/
var removeZeroSumSublists = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* class ListNode {
* val: number
* next: ListNode | null
* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function removeZeroSumSublists(head: ListNode | null): ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/
public class Solution {
public ListNode RemoveZeroSumSublists(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Remove Zero Sum Consecutive Nodes from Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
struct ListNode* removeZeroSumSublists(struct ListNode* head) {

}
```

Go Solution:

```
// Problem: Remove Zero Sum Consecutive Nodes from Linked List
// Difficulty: Medium
// Tags: hash, linked_list
```

```
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func removeZeroSumSublists(head *ListNode) *ListNode {

}
```

Kotlin Solution:

```
/**
* Example:
* var li = ListNode(5)
* var v = li.`val`
* Definition for singly-linked list.
* class ListNode(var `val`: Int) {
* var next: ListNode? = null
* }
*/
class Solution {
fun removeZeroSumSublists(head: ListNode?): ListNode? {

}
}
```

Swift Solution:

```
/**
* Definition for singly-linked list.
* public class ListNode {
* public var val: Int
* public var next: ListNode?
* public init() { self.val = 0; self.next = nil; }
* public init(_ val: Int) { self.val = val; self.next = nil; }
* public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
* }
*/
class Solution {
func removeZeroSumSublists(_ head: ListNode?) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Remove Zero Sum Consecutive Nodes from Linked List
// Difficulty: Medium
// Tags: hash, linked_list
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
// pub val: i32,
// pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
// #[inline]
// fn new(val: i32) -> Self {
// ListNode {
// next: None,
// val
// }
// }
// }
impl Solution {
pub fn remove_zero_sum_sublists(head: Option<Box<ListNode>>) ->
Option<Box<ListNode>> {
}
}
```

Ruby Solution:

```
# Definition for singly-linked list.
# class ListNode
# attr_accessor :val, :next
# def initialize(val = 0, _next = nil)
# @val = val
# @next = _next
# end
# end
# @param {ListNode} head
# @return {ListNode}
def remove_zero_sum_sublists(head)

end
```

PHP Solution:

```
/**
* Definition for a singly-linked list.
* class ListNode {
* public $val = 0;
* public $next = null;
* function __construct($val = 0, $next = null) {
* $this->val = $val;
* $this->next = $next;
* }
* }
*/
class Solution {

/**
* @param ListNode $head
* @return ListNode
*/
function removeZeroSumSublists($head) {

}
}
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
ListNode? removeZeroSumSublists(ListNode? head) {

}
}
```

Scala Solution:

```
/**
* Definition for singly-linked list.
* class ListNode(_x: Int = 0, _next: ListNode = null) {
* var next: ListNode = _next
* var x: Int = _x
* }
*/
object Solution {
def removeZeroSumSublists(head: ListNode): ListNode = {

}
}
```

Elixir Solution:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
# @type t :: %__MODULE__{
# val: integer,
# next: ListNode.t() | nil
# }
# defstruct val: 0, next: nil
# end

defmodule Solution do
@spec remove_zero_sum_sublists(head :: ListNode.t | nil) :: ListNode.t | nil
def remove_zero_sum_sublists(head) do

end
end
```

Erlang Solution:

```
%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec remove_zero_sum_sublists(Head :: #list_node{} | null) -> #list_node{} |
null.
remove_zero_sum_sublists(Head) ->
.
```

Racket Solution:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
(val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
(list-node val #f))

|#

(define/contract (remove-zero-sum-sublists head)
(-> (or/c list-node? #f) (or/c list-node? #f))
)
```