Problem 92: Reverse Linked List II

Problem Information

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

Problem Description

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

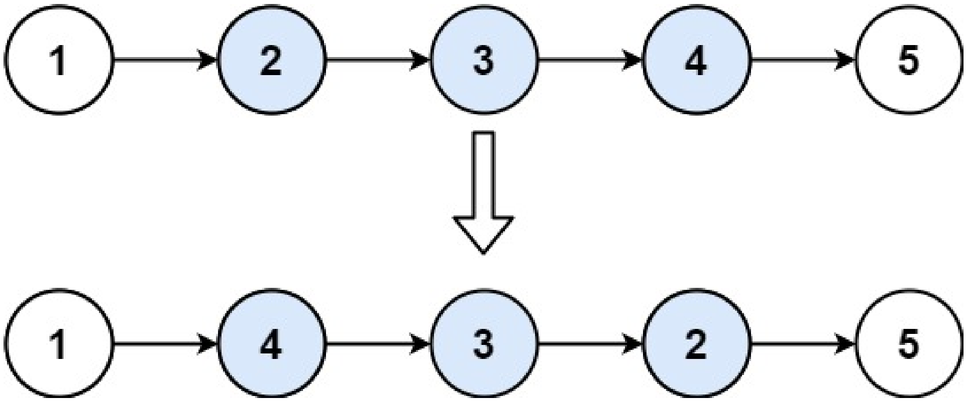
right

, and return

the reversed list

.

Example 1:



Input:

head = [1,2,3,4,5], left = 2, right = 4

Output:

[1,4,3,2,5]

Example 2:

Input:

head = [5], left = 1, right = 1

Output:

[5]

Constraints:

The number of nodes in the list is

n

.

1 <= n <= 500

-500 <= Node.val <= 500

1 <= left <= right <= n

Follow up:

Could you do it in one pass?

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* reverseBetween(ListNode* head, int left, int right) {

}
};
```

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 reverseBetween(ListNode head, int left, int right) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def reverseBetween(self, head: Optional[ListNode], left: int, right: int) ->
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 reverseBetween(self, head, left, right):
"""
:type head: Optional[ListNode]
:type left: int
:type right: int
: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
* @param {number} left
* @param {number} right
* @return {ListNode}
*/
var reverseBetween = function(head, left, right) {

};
```

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 reverseBetween(head: ListNode | null, left: number, right: number):
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 ReverseBetween(ListNode head, int left, int right) {

}
}
```

C:

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

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func reverseBetween(head *ListNode, left int, right int) *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 reverseBetween(head: ListNode?, left: Int, right: Int): 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 reverseBetween(_ head: ListNode?, _ left: Int, _ right: Int) ->
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 reverse_between(head: Option<Box<ListNode>>, left: i32, right: i32) ->
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
# @param {Integer} left
# @param {Integer} right
# @return {ListNode}
def reverse_between(head, left, right)

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
* @param Integer $left
* @param Integer $right
* @return ListNode
*/
function reverseBetween($head, $left, $right) {

}
}
```

Dart:

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

}
}
```

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 reverseBetween(head: ListNode, left: Int, right: Int): 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 reverse_between(head :: ListNode.t | nil, left :: integer, right ::
integer) :: ListNode.t | nil
def reverse_between(head, left, right) do

end
end
```

Erlang:

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

-spec reverse_between(Head :: #list_node{} | null, Left :: integer(), Right
:: integer()) -> #list_node{} | null.
reverse_between(Head, Left, Right) ->
.
```

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 (reverse-between head left right)
(-> (or/c list-node? #f) exact-integer? exact-integer? (or/c list-node? #f))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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* reverseBetween(ListNode* head, int left, int right) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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 reverseBetween(ListNode head, int left, int right) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Linked List II
Difficulty: Medium
Tags: linked_list

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

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

JavaScript Solution:

```
/**
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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 reverseBetween(head: ListNode | null, left: number, right: number):
ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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 ReverseBetween(ListNode head, int left, int right) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Linked List II
* Difficulty: Medium
* Tags: linked_list
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
```

Go Solution:

```
// Problem: Reverse Linked List II
// Difficulty: Medium
// Tags: linked_list
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func reverseBetween(head *ListNode, left int, right int) *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 reverseBetween(head: ListNode?, left: Int, right: Int): 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 reverseBetween(_ head: ListNode?, _ left: Int, _ right: Int) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Reverse Linked List II
// Difficulty: Medium
// Tags: linked_list
```

```
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

// 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 reverse_between(head: Option<Box<ListNode>>, left: i32, right: i32) ->
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
# @param {Integer} left
# @param {Integer} right
# @return {ListNode}
def reverse_between(head, left, right)

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
* @param Integer $left
* @param Integer $right
* @return ListNode
*/
function reverseBetween($head, $left, $right) {

}
}
```

Dart Solution:

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

```
class Solution {
ListNode? reverseBetween(ListNode? head, int left, int right) {

}
}
```

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 reverseBetween(head: ListNode, left: Int, right: Int): 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 reverse_between(head :: ListNode.t | nil, left :: integer, right ::
integer) :: ListNode.t | nil
def reverse_between(head, left, right) do

end
end
```

Erlang Solution:

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

-spec reverse_between(Head :: #list_node{} | null, Left :: integer(), Right
:: integer()) -> #list_node{} | null.
reverse_between(Head, Left, Right) ->
.
```

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 (reverse-between head left right)
(-> (or/c list-node? #f) exact-integer? exact-integer? (or/c list-node? #f))
)
```