Problem 1669: Merge In Between Linked Lists

Problem Information

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

Problem Description

You are given two linked lists:

list1

and

list2

of sizes

n

and

m

respectively.

Remove

list1

's nodes from the

a

th

node to the

b

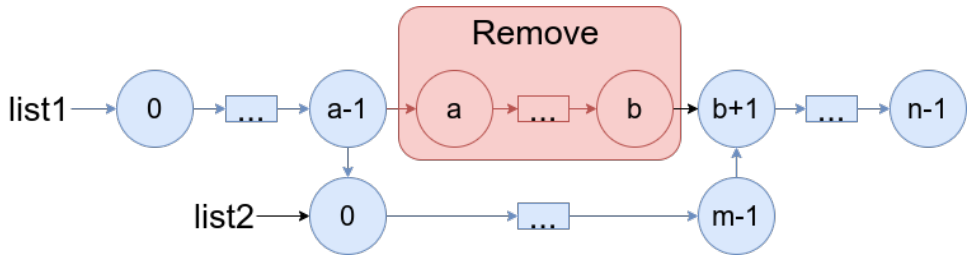
th

node, and put

list2

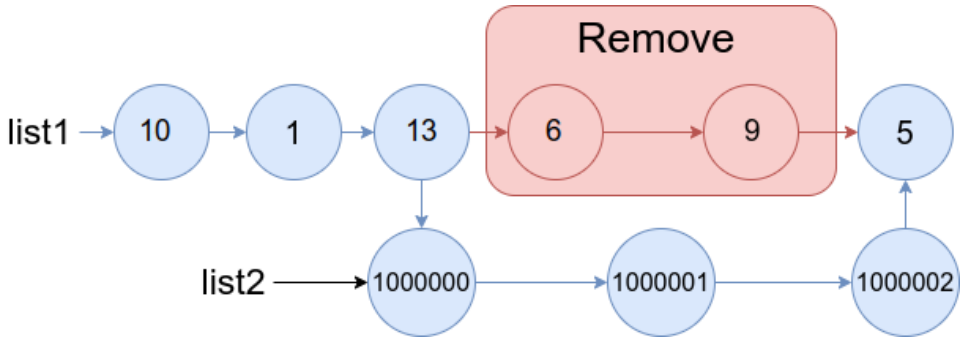
in their place.

The blue edges and nodes in the following figure indicate the result:



Build the result list and return its head.

Example 1:



Input:

list1 = [10,1,13,6,9,5], a = 3, b = 4, list2 = [1000000,1000001,1000002]

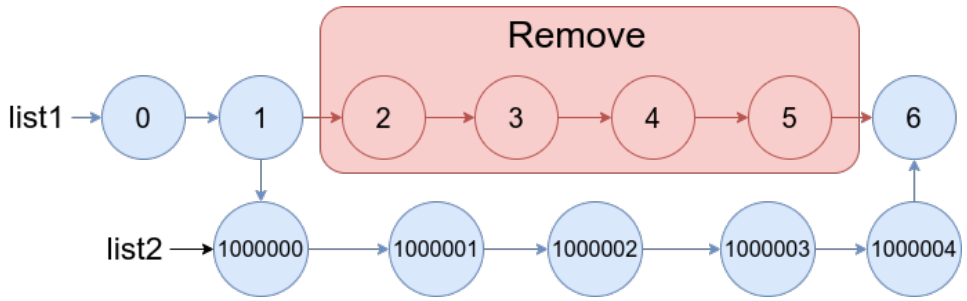
Output:

[10,1,13,1000000,1000001,1000002,5]

Explanation:

We remove the nodes 3 and 4 and put the entire list2 in their place. The blue edges and nodes in the above figure indicate the result.

Example 2:



Input:

list1 = [0,1,2,3,4,5,6], a = 2, b = 5, list2 = [1000000,1000001,1000002,1000003,1000004]

Output:

[0,1,1000000,1000001,1000002,1000003,1000004,6]

Explanation:

The blue edges and nodes in the above figure indicate the result.

Constraints:

3 <= list1.length <= 10

4

1 <= a <= b < list1.length - 1

1 <= list2.length <= 10

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* mergeInBetween(ListNode* list1, int a, int b, ListNode* list2) {

}
};
```

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 mergeInBetween(ListNode list1, int a, int b, ListNode list2)
{

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def mergeInBetween(self, list1: ListNode, a: int, b: int, list2: ListNode) ->
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 mergeInBetween(self, list1, a, b, list2):
"""
:type list1: ListNode
:type a: int
:type b: int
:type list2: ListNode
:rtype: 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} list1
* @param {number} a
* @param {number} b
* @param {ListNode} list2
* @return {ListNode}
```

```
*/
var mergeInBetween = function(list1, a, b, list2) {

};
```

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 mergeInBetween(list1: ListNode | null, a: number, b: number, list2:
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 MergeInBetween(ListNode list1, int a, int b, ListNode list2)
{

}
```

```
}
```

C:

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


struct ListNode* mergeInBetween(struct ListNode* list1, int a, int b, struct
ListNode* list2){

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func mergeInBetween(list1 *ListNode, a int, b int, list2 *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 mergeInBetween(list1: ListNode?, a: Int, b: Int, list2: 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 mergeInBetween(_ list1: ListNode?, _ a: Int, _ b: Int, _ list2:
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 merge_in_between(list1: Option<Box<ListNode>>, a: i32, b: i32, list2:
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} list1
# @param {Integer} a
# @param {Integer} b
# @param {ListNode} list2
# @return {ListNode}
def merge_in_between(list1, a, b, list2)

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 $list1
* @param Integer $a
* @param Integer $b
* @param ListNode $list2
* @return ListNode
*/
function mergeInBetween($list1, $a, $b, $list2) {

}
}
```

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 mergeInBetween(list1: ListNode, a: Int, b: Int, list2: ListNode):
ListNode = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Merge In Between Linked Lists
* 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* mergeInBetween(ListNode* list1, int a, int b, ListNode* list2) {

}
};
```

Java Solution:

```
/**
* Problem: Merge In Between Linked Lists
* 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 mergeInBetween(ListNode list1, int a, int b, ListNode list2)
{

}
}
```

Python3 Solution:

```
"""
Problem: Merge In Between Linked Lists
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 mergeInBetween(self, list1: ListNode, a: int, b: int, list2: ListNode) ->
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 mergeInBetween(self, list1, a, b, list2):
"""
:type list1: ListNode
:type a: int
:type b: int
:type list2: ListNode
:rtype: ListNode
"""
```

JavaScript Solution:

```
/**
* Problem: Merge In Between Linked Lists
* 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} list1
* @param {number} a
* @param {number} b
* @param {ListNode} list2
* @return {ListNode}
*/
var mergeInBetween = function(list1, a, b, list2) {

};
```

TypeScript Solution:

```
/**
* Problem: Merge In Between Linked Lists
* 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 mergeInBetween(list1: ListNode | null, a: number, b: number, list2:
ListNode | null): ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Merge In Between Linked Lists
* 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 MergeInBetween(ListNode list1, int a, int b, ListNode list2)
{

}
}
```

C Solution:

```
/*
* Problem: Merge In Between Linked Lists
* 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* mergeInBetween(struct ListNode* list1, int a, int b, struct
ListNode* list2){
```

```
}
```

Go Solution:

```
// Problem: Merge In Between Linked Lists
// 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 mergeInBetween(list1 *ListNode, a int, b int, list2 *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 mergeInBetween(list1: ListNode?, a: Int, b: Int, list2: 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 mergeInBetween(_ list1: ListNode?, _ a: Int, _ b: Int, _ list2:
ListNode?) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Merge In Between Linked Lists
// 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 merge_in_between(list1: Option<Box<ListNode>>, a: i32, b: i32, list2:
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} list1
# @param {Integer} a
# @param {Integer} b
# @param {ListNode} list2
# @return {ListNode}
def merge_in_between(list1, a, b, list2)

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 $list1
* @param Integer $a
* @param Integer $b
* @param ListNode $list2
* @return ListNode
*/
function mergeInBetween($list1, $a, $b, $list2) {

}
}
```

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 mergeInBetween(list1: ListNode, a: Int, b: Int, list2: ListNode):
ListNode = {

}
}
```