Problem 2674: Split a Circular Linked List

Problem Information

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

Problem Description

Given a

circular linked list

list

of positive integers, your task is to split it into 2

circular linked lists

so that the first one contains the

first half

of the nodes in

list

(exactly

ceil(list.length / 2)

nodes) in the same order they appeared in

list

, and the second one contains

the rest

of the nodes in

list

in the same order they appeared in

list

.

Return

an array answer of length 2 in which the first element is a

circular linked list

representing the

first half

and the second element is a

circular linked list

representing the

second half

.

A

circular linked list

is a normal linked list with the only difference being that the last node's next node, is the first node.

Example 1:

Input:

nums = [1,5,7]

Output:

[[1,5],[7]]

Explanation:

The initial list has 3 nodes so the first half would be the first 2 elements since ceil(3 / 2) = 2 and the rest which is 1 node is in the second half.

Example 2:

Input:

nums = [2,6,1,5]

Output:

[[2,6],[1,5]]

Explanation:

The initial list has 4 nodes so the first half would be the first 2 elements since ceil(4 / 2) = 2 and the rest which is 2 nodes are in the second half.

Constraints:

The number of nodes in

list

is in the range

[2, 10

5

]

0 <= Node.val <= 10

9

LastNode.next = FirstNode

where

LastNode

is the last node of the list and

FirstNode

is the first one

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:
vector<ListNode*> splitCircularLinkedList(ListNode* list) {

}
};
```

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[] splitCircularLinkedList(ListNode list) {

}
}
```

Python3:

```
# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def splitCircularLinkedList(self, list: Optional[ListNode]) ->
List[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 splitCircularLinkedList(self, list):
"""
:type list: Optional[ListNode]
:rtype: List[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} list
* @return {ListNode[]}
*/
var splitCircularLinkedList = function(list) {

};
```

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 splitCircularLinkedList(list: ListNode | null): Array<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[] SplitCircularLinkedList(ListNode list) {

}
}
```

C:

```
/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct ListNode** splitCircularLinkedList(struct ListNode* list){

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
```

```
*/
func splitCircularLinkedList(list *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 splitCircularLinkedList(list: ListNode?): Array<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 splitCircularLinkedList(_ list: ListNode?) -> [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} list
# @return {ListNode[]}
def split_circular_linked_list(list)

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 $list
* @return ListNode[]
*/
function splitCircularLinkedList($list) {

}
}
```

Dart:

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

```
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/

import 'dart:collection'; // DO NOT REMOVE THIS LINE

class Solution {
List<ListNode?> splitCircularLinkedList(ListNode? list) {

}
}
```

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 splitCircularLinkedList(list: ListNode): Array[ListNode] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* 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:
vector<ListNode*> splitCircularLinkedList(ListNode* list) {

}
};
```

Java Solution:

```
/**
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* 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[] splitCircularLinkedList(ListNode list) {

}
}
```

Python3 Solution:

```
"""
Problem: Split a Circular Linked List
Difficulty: Medium
Tags: array, linked_list

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
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 splitCircularLinkedList(self, list: Optional[ListNode]) ->
List[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 splitCircularLinkedList(self, list):
"""
:type list: Optional[ListNode]
```

```
:rtype: List[Optional[ListNode]]
"""
```

JavaScript Solution:

```
/**
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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} list
* @return {ListNode[]}
*/
var splitCircularLinkedList = function(list) {

};
```

TypeScript Solution:

```
/**
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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 splitCircularLinkedList(list: ListNode | null): Array<ListNode |
null> {

};
```

C# Solution:

```
/*
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* 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[] SplitCircularLinkedList(ListNode list) {

}
}
```

C Solution:

```
/*
* Problem: Split a Circular Linked List
* Difficulty: Medium
* Tags: array, linked_list
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Definition for singly-linked list.
* struct ListNode {
* int val;
* struct ListNode *next;
* };
*/
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
struct ListNode** splitCircularLinkedList(struct ListNode* list){

}
```

Go Solution:

```
// Problem: Split a Circular Linked List
// Difficulty: Medium
// Tags: array, linked_list
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach
```

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func splitCircularLinkedList(list *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 splitCircularLinkedList(list: ListNode?): Array<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 splitCircularLinkedList(_ list: ListNode?) -> [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} list
# @return {ListNode[]}
def split_circular_linked_list(list)

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 $list
* @return ListNode[]
```

```
*/
function splitCircularLinkedList($list) {

}
}
```

Dart Solution:

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

import 'dart:collection'; // DO NOT REMOVE THIS LINE

class Solution {
List<ListNode?> splitCircularLinkedList(ListNode? list) {

}
}
```

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 splitCircularLinkedList(list: ListNode): Array[ListNode] = {

}
}
```