Problem 725: Split Linked List in Parts

Problem Information

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

Problem Description

Given the

head

of a singly linked list and an integer

k

, split the linked list into

k

consecutive linked list parts.

The length of each part should be as equal as possible: no two parts should have a size differing by more than one. This may lead to some parts being null.

The parts should be in the order of occurrence in the input list, and parts occurring earlier should always have a size greater than or equal to parts occurring later.

Return

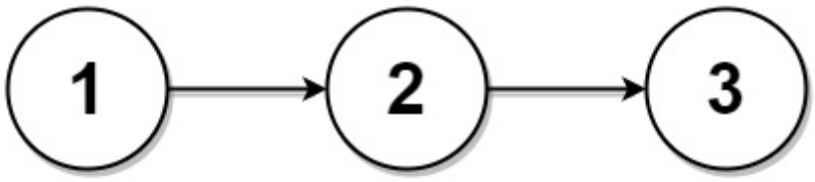
an array of the

k

parts

.

Example 1:



Input:

head = [1,2,3], k = 5

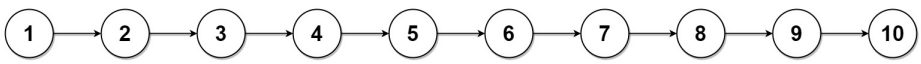
Output:

[[1],[2],[3],[],[]]

Explanation:

The first element output[0] has output[0].val = 1, output[0].next = null. The last element output[4] is null, but its string representation as a ListNode is [].

Example 2:



Input:

head = [1,2,3,4,5,6,7,8,9,10], k = 3

Output:

[[1,2,3,4],[5,6,7],[8,9,10]]

Explanation:

The input has been split into consecutive parts with size difference at most 1, and earlier parts are a larger size than the later parts.

Constraints:

The number of nodes in the list is in the range

[0, 1000]

.

0 <= Node.val <= 1000

1 <= k <= 50

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*> splitListToParts(ListNode* head, int k) {

}
};
```

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[] splitListToParts(ListNode head, int k) {

}
}
```

Python3:

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

};
```

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 splitListToParts(head: ListNode | null, k: number): 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[] SplitListToParts(ListNode head, int k) {

}
}
```

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** splitListToParts(struct ListNode* head, int k, int*
returnSize) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func splitListToParts(head *ListNode, k 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 splitListToParts(head: ListNode?, k: Int): 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 splitListToParts(_ head: ListNode?, _ k: 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 split_list_to_parts(head: Option<Box<ListNode>>, k: i32) ->
Vec<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} k
# @return {ListNode[]}
def split_list_to_parts(head, k)

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 $k
* @return ListNode[]
*/
function splitListToParts($head, $k) {

}
}
```

Dart:

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

}
}
```

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 splitListToParts(head: ListNode, k: Int): Array[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 split_list_to_parts(head :: ListNode.t | nil, k :: integer) ::
[ListNode.t | nil]
def split_list_to_parts(head, k) do

end
end
```

Erlang:

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

-spec split_list_to_parts(Head :: #list_node{} | null, K :: integer()) ->
[#list_node{} | null].
split_list_to_parts(Head, K) ->
.
```

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 (split-list-to-parts head k)
(-> (or/c list-node? #f) exact-integer? (listof (or/c list-node? #f)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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*> splitListToParts(ListNode* head, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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[] splitListToParts(ListNode head, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Split Linked List in Parts
Difficulty: Medium
Tags: array, string, 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 splitListToParts(self, head: Optional[ListNode], k: int) ->
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 splitListToParts(self, head, k):
"""
:type head: Optional[ListNode]
:type k: int
:rtype: List[Optional[ListNode]]
"""
```

JavaScript Solution:

```
/**
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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} head
* @param {number} k
* @return {ListNode[]}
*/
var splitListToParts = function(head, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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 splitListToParts(head: ListNode | null, k: number): Array<ListNode |
null> {

};
```

C# Solution:

```
/*
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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[] SplitListToParts(ListNode head, int k) {

}
}
```

C Solution:

```
/*
* Problem: Split Linked List in Parts
* Difficulty: Medium
* Tags: array, string, 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** splitListToParts(struct ListNode* head, int k, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Split Linked List in Parts
// Difficulty: Medium
// Tags: array, string, 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 splitListToParts(head *ListNode, k 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 splitListToParts(head: ListNode?, k: Int): 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 splitListToParts(_ head: ListNode?, _ k: Int) -> [ListNode?] {

}
}
```

Rust Solution:

```
// Problem: Split Linked List in Parts
// Difficulty: Medium
// Tags: array, string, 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.
// #[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 split_list_to_parts(head: Option<Box<ListNode>>, k: i32) ->
Vec<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} k
# @return {ListNode[]}
def split_list_to_parts(head, k)
```

```
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 $k
* @return ListNode[]
*/
function splitListToParts($head, $k) {

}
}
```

Dart Solution:

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

```
}
}
```

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 splitListToParts(head: ListNode, k: Int): Array[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 split_list_to_parts(head :: ListNode.t | nil, k :: integer) ::
[ListNode.t | nil]
def split_list_to_parts(head, k) do

end
end
```

Erlang Solution:

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

-spec split_list_to_parts(Head :: #list_node{} | null, K :: integer()) ->
[#list_node{} | null].
split_list_to_parts(Head, K) ->
.
```

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 (split-list-to-parts head k)
(-> (or/c list-node? #f) exact-integer? (listof (or/c list-node? #f)))
)
```