Problem 2058: Find the Minimum and Maximum Number of Nodes Between Critical Points

Problem Information

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

Problem Description

A

critical point

in a linked list is defined as

either

a

local maxima

or a

local minima

.

A node is a

local maxima

if the current node has a value

strictly greater

than the previous node and the next node.

A node is a

local minima

if the current node has a value

strictly smaller

than the previous node and the next node.

Note that a node can only be a local maxima/minima if there exists

both

a previous node and a next node.

Given a linked list

head

, return

an array of length 2 containing

[minDistance, maxDistance]

where

minDistance

is the

minimum distance

between

any two distinct

critical points and

maxDistance

is the

maximum distance

between

any two distinct

critical points. If there are

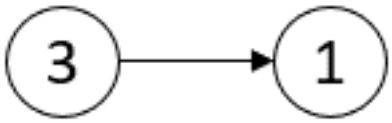
fewer

than two critical points, return

[-1, -1]

.

Example 1:



Input:

head = [3,1]

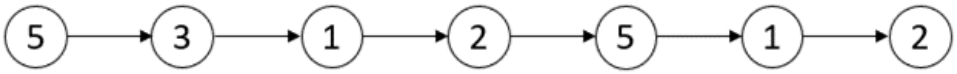
Output:

[-1,-1]

Explanation:

There are no critical points in [3,1].

Example 2:



Input:

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

Output:

[1,3]

Explanation:

There are three critical points: - [5,3,

1

,2,5,1,2]: The third node is a local minima because 1 is less than 3 and 2. - [5,3,1,2,

5

,1,2]: The fifth node is a local maxima because 5 is greater than 2 and 1. - [5,3,1,2,5,

1

,2]: The sixth node is a local minima because 1 is less than 5 and 2. The minimum distance is between the fifth and the sixth node. minDistance = 6 - 5 = 1. The maximum distance is between the third and the sixth node. maxDistance = 6 - 3 = 3.

Example 3:

Input:

head = [1,3,2,2,3,2,2,2,7]

Output:

[3,3]

Explanation:

There are two critical points: - [1,

3

,2,2,3,2,2,2,7]: The second node is a local maxima because 3 is greater than 1 and 2. - [1,3,2,2,

3

,2,2,2,7]: The fifth node is a local maxima because 3 is greater than 2 and 2. Both the minimum and maximum distances are between the second and the fifth node. Thus, minDistance and maxDistance is 5 - 2 = 3. Note that the last node is not considered a local maxima because it does not have a next node.

Constraints:

The number of nodes in the list is in the range

[2, 10

5

]

.

1 <= Node.val <= 10

5

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<int> nodesBetweenCriticalPoints(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 int[] nodesBetweenCriticalPoints(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 nodesBetweenCriticalPoints(self, head: Optional[ListNode]) -> List[int]:
```

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 nodesBetweenCriticalPoints(self, head):
"""
:type head: Optional[ListNode]
:rtype: List[int]
"""
```

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 {number[]}
*/
var nodesBetweenCriticalPoints = 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 nodesBetweenCriticalPoints(head: ListNode | null): number[] {

};
```

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 int[] NodesBetweenCriticalPoints(ListNode head) {

}
}
```

C:

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

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* nodesBetweenCriticalPoints(struct ListNode* head, int* returnSize) {

}
```

Go:

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

}
```

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 nodesBetweenCriticalPoints(head: ListNode?): IntArray {

}
}
```

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 nodesBetweenCriticalPoints(_ head: ListNode?) -> [Int] {

}
}
```

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 nodes_between_critical_points(head: Option<Box<ListNode>>) -> Vec<i32>
{

}
}
```

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 {Integer[]}
def nodes_between_critical_points(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 $head
* @return Integer[]
*/
function nodesBetweenCriticalPoints($head) {

}
}
```

Dart:

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

```
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
List<int> nodesBetweenCriticalPoints(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 nodesBetweenCriticalPoints(head: ListNode): Array[Int] = {

}
}
```

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 nodes_between_critical_points(head :: ListNode.t | nil) :: [integer]
def nodes_between_critical_points(head) do

end
```

```
end
```

Erlang:

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

-spec nodes_between_critical_points(Head :: #list_node{} | null) ->
[integer()].
nodes_between_critical_points(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 (nodes-between-critical-points head)
(-> (or/c list-node? #f) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
```

```
Points
* 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<int> nodesBetweenCriticalPoints(ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
* 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 int[] nodesBetweenCriticalPoints(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Find the Minimum and Maximum Number of Nodes Between Critical Points
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 nodesBetweenCriticalPoints(self, head: Optional[ListNode]) -> List[int]:
# 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 nodesBetweenCriticalPoints(self, head):
"""
:type head: Optional[ListNode]
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
* 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} head
* @return {number[]}
*/
var nodesBetweenCriticalPoints = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
```

```
* 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 nodesBetweenCriticalPoints(head: ListNode | null): number[] {

};
```

C# Solution:

```
/*
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
* 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 int[] NodesBetweenCriticalPoints(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
* 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().
*/
int* nodesBetweenCriticalPoints(struct ListNode* head, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
// 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 nodesBetweenCriticalPoints(head *ListNode) []int {

}
```

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 nodesBetweenCriticalPoints(head: ListNode?): IntArray {

}
}
```

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 nodesBetweenCriticalPoints(_ head: ListNode?) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find the Minimum and Maximum Number of Nodes Between Critical
Points
// 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.
// #[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 nodes_between_critical_points(head: Option<Box<ListNode>>) -> Vec<i32>
{

}
}
```

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 {Integer[]}
def nodes_between_critical_points(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 Integer[]
```

```
*/
function nodesBetweenCriticalPoints($head) {

}
}
```

Dart Solution:

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

}
}
```

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 nodes_between_critical_points(head :: ListNode.t | nil) :: [integer]
def nodes_between_critical_points(head) do

end
end
```

Erlang Solution:

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

-spec nodes_between_critical_points(Head :: #list_node{} | null) ->
[integer()].
nodes_between_critical_points(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 (nodes-between-critical-points head)
(-> (or/c list-node? #f) (listof exact-integer?))
)
```