Problem 1019: Next Greater Node In Linked List

Problem Information

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

Problem Description

You are given the

head

of a linked list with

n

nodes.

For each node in the list, find the value of the

next greater node

. That is, for each node, find the value of the first node that is next to it and has a

strictly larger

value than it.

Return an integer array

answer

where

answer[i]

is the value of the next greater node of the

i

th

node (

1-indexed

). If the

i

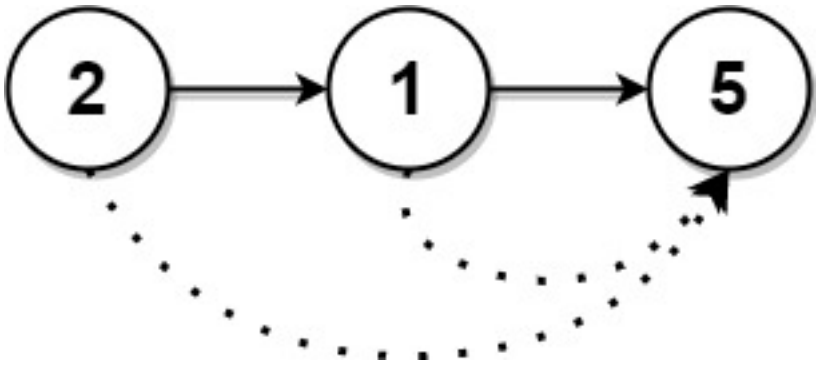
th

node does not have a next greater node, set

answer[i] = 0

.

Example 1:



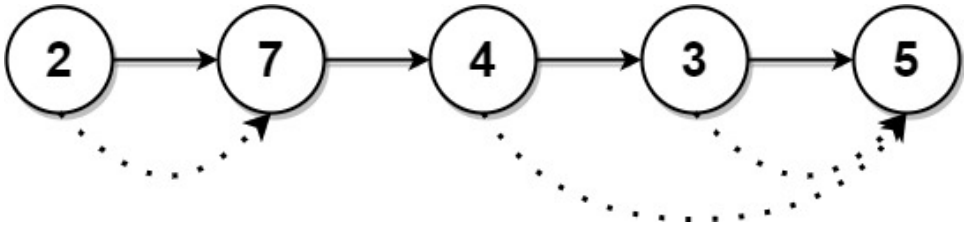
Input:

head = [2,1,5]

Output:

[5,5,0]

Example 2:



Input:

head = [2,7,4,3,5]

Output:

[7,0,5,5,0]

Constraints:

The number of nodes in the list is

n

.

1 <= n <= 10

4

1 <= Node.val <= 10

9

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> nextLargerNodes(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[] nextLargerNodes(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 nextLargerNodes(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 nextLargerNodes(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 nextLargerNodes = 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 nextLargerNodes(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[] NextLargerNodes(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* nextLargerNodes(struct ListNode* head, int* returnSize) {

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func nextLargerNodes(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 nextLargerNodes(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 nextLargerNodes(_ 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 next_larger_nodes(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 next_larger_nodes(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 $head
* @return Integer[]
*/
function nextLargerNodes($head) {

}
}
```

Dart:

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

```
*/
class Solution {
List<int> nextLargerNodes(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 nextLargerNodes(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 next_larger_nodes(head :: ListNode.t | nil) :: [integer]
def next_larger_nodes(head) do

end
end
```

Erlang:

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

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

Solutions

C++ Solution:

```
/*
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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> nextLargerNodes(ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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[] nextLargerNodes(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Next Greater Node In Linked List
Difficulty: Medium
Tags: array, linked_list, stack

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

JavaScript Solution:

```
/**
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 nextLargerNodes = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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 nextLargerNodes(head: ListNode | null): number[] {

};
```

C# Solution:

```
/*
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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[] NextLargerNodes(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Next Greater Node In Linked List
* Difficulty: Medium
* Tags: array, linked_list, stack
*
* 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* nextLargerNodes(struct ListNode* head, int* returnSize) {

}
```

Go Solution:

```
// Problem: Next Greater Node In Linked List
// Difficulty: Medium
// Tags: array, linked_list, stack
//
// 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 nextLargerNodes(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 nextLargerNodes(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 nextLargerNodes(_ head: ListNode?) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Next Greater Node In Linked List
// Difficulty: Medium
// Tags: array, linked_list, stack
//
// 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 next_larger_nodes(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 next_larger_nodes(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 $head
* @return Integer[]
*/
function nextLargerNodes($head) {

}
}
```

Dart Solution:

```
/**
* Definition for singly-linked list.
* class ListNode {
* int val;
* ListNode? next;
* ListNode([this.val = 0, this.next]);
* }
*/
class Solution {
List<int> nextLargerNodes(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 nextLargerNodes(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 next_larger_nodes(head :: ListNode.t | nil) :: [integer]
def next_larger_nodes(head) do

end
end
```

Erlang Solution:

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

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