Problem 1836: Remove Duplicates From an Unsorted Linked List

Problem Information

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

Problem Description

Given the

head

of a linked list, find all the values that appear

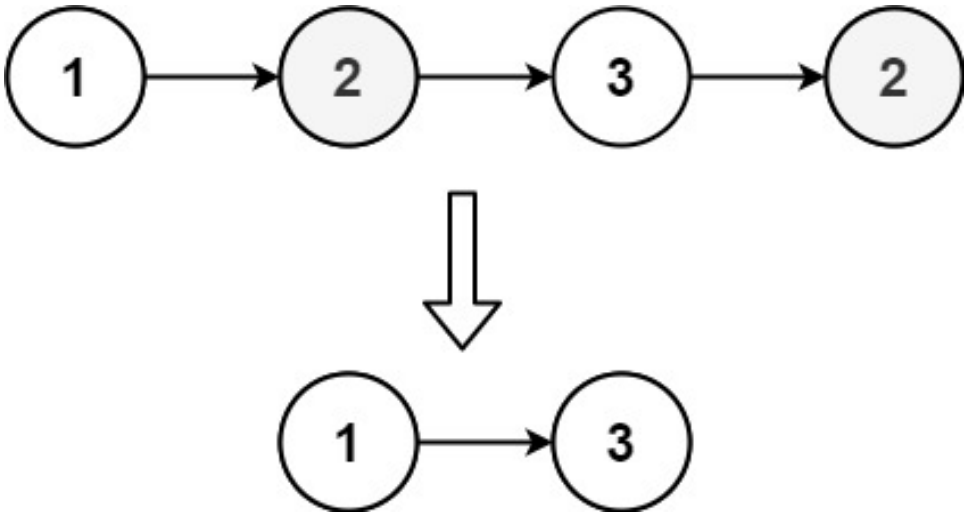
more than once

in the list and delete the nodes that have any of those values.

Return

the linked list after the deletions.

Example 1:



Input:

head = [1,2,3,2]

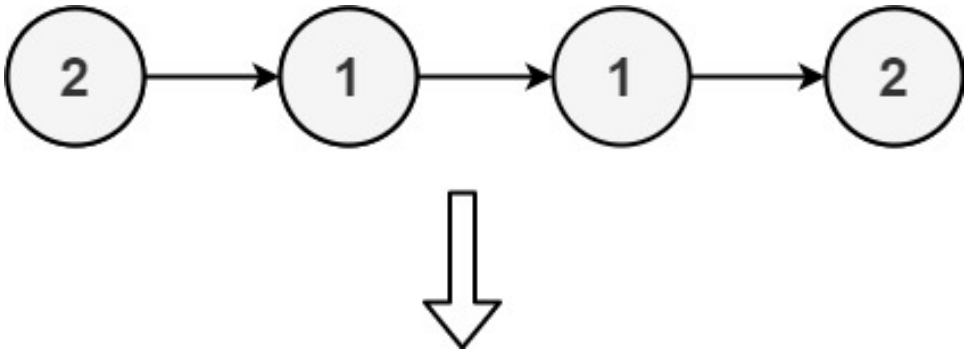
Output:

[1,3]

Explanation:

2 appears twice in the linked list, so all 2's should be deleted. After deleting all 2's, we are left with [1,3].

Example 2:



Input:

head = [2,1,1,2]

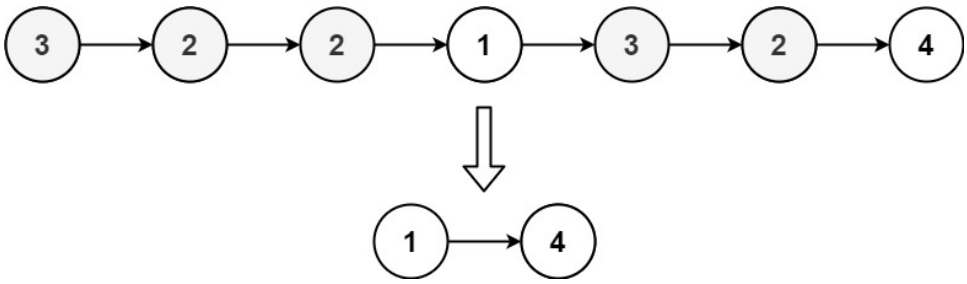
Output:

[]

Explanation:

2 and 1 both appear twice. All the elements should be deleted.

Example 3:



Input:

head = [3,2,2,1,3,2,4]

Output:

[1,4]

Explanation:

3 appears twice and 2 appears three times. After deleting all 3's and 2's, we are left with [1,4].

Constraints:

The number of nodes in the list is in the range

[1, 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:
ListNode* deleteDuplicatesUnsorted(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 ListNode deleteDuplicatesUnsorted(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 deleteDuplicatesUnsorted(self, head: 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 deleteDuplicatesUnsorted(self, head):
"""
:type head: 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} head
* @return {ListNode}
*/
var deleteDuplicatesUnsorted = 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 deleteDuplicatesUnsorted(head: 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 DeleteDuplicatesUnsorted(ListNode head) {

}
}
```

C:

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

```
* struct ListNode *next;
* };
*/


struct ListNode* deleteDuplicatesUnsorted(struct ListNode* head){

}
```

Go:

```
/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func deleteDuplicatesUnsorted(head *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 deleteDuplicatesUnsorted(head: 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 deleteDuplicatesUnsorted(_ head: 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 delete_duplicates_unsorted(head: 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} head
# @return {ListNode}
def delete_duplicates_unsorted(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 $head
* @return ListNode
*/
function deleteDuplicatesUnsorted($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 deleteDuplicatesUnsorted(head: ListNode): ListNode = {

}
}
```

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 (delete-duplicates-unsorted head)
(-> (or/c list-node? #f) (or/c list-node? #f))

)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
```

```
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* 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* deleteDuplicatesUnsorted(ListNode* head) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* 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 deleteDuplicatesUnsorted(ListNode head) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Duplicates From an Unsorted Linked List
Difficulty: Medium
Tags: hash, linked_list

Approach: Use hash map for O(1) lookups
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(n) for hash map
"""

# Definition for singly-linked list.
# class ListNode:
# def __init__(self, val=0, next=None):
# self.val = val
# self.next = next
class Solution:
def deleteDuplicatesUnsorted(self, head: 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 deleteDuplicatesUnsorted(self, head):
"""
:type head: ListNode
:rtype: ListNode
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* 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 {ListNode}
*/
var deleteDuplicatesUnsorted = function(head) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
```

```
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* 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 deleteDuplicatesUnsorted(head: ListNode | null): ListNode | null {

};
```

C# Solution:

```
/*
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

/**
* 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 DeleteDuplicatesUnsorted(ListNode head) {

}
}
```

C Solution:

```
/*
* Problem: Remove Duplicates From an Unsorted Linked List
* Difficulty: Medium
* Tags: hash, linked_list
*
* Approach: Use hash map for O(1) lookups
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(n) for hash map
*/

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


struct ListNode* deleteDuplicatesUnsorted(struct ListNode* head){

}
```

Go Solution:

```
// Problem: Remove Duplicates From an Unsorted Linked List
// Difficulty: Medium
// Tags: hash, linked_list
```

```
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

/**
* Definition for singly-linked list.
* type ListNode struct {
* Val int
* Next *ListNode
* }
*/
func deleteDuplicatesUnsorted(head *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 deleteDuplicatesUnsorted(head: 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 deleteDuplicatesUnsorted(_ head: ListNode?) -> ListNode? {

}
}
```

Rust Solution:

```
// Problem: Remove Duplicates From an Unsorted Linked List
// Difficulty: Medium
// Tags: hash, linked_list
//
// Approach: Use hash map for O(1) lookups
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(n) for hash map

// 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 delete_duplicates_unsorted(head: 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} head
# @return {ListNode}
def delete_duplicates_unsorted(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 $head
* @return ListNode
*/
function deleteDuplicatesUnsorted($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 deleteDuplicatesUnsorted(head: ListNode): ListNode = {

}
}
```

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 (delete-duplicates-unsorted head)
(-> (or/c list-node? #f) (or/c list-node? #f))

)
```