

Problem 82: Remove Duplicates from Sorted List II

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given the

head

of a sorted linked list,

delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list

. Return

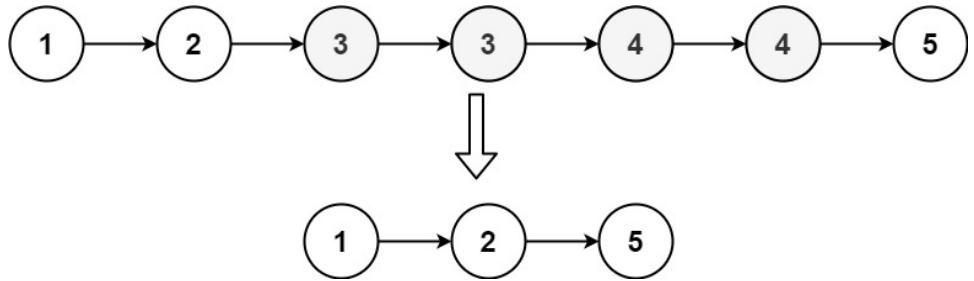
the linked list

sorted

as well

.

Example 1:



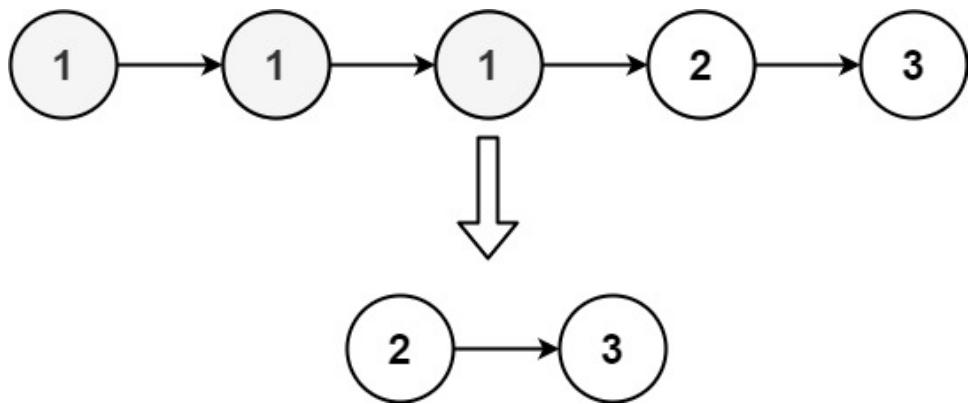
Input:

head = [1,2,3,3,4,4,5]

Output:

[1,2,5]

Example 2:



Input:

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

Output:

[2,3]

Constraints:

The number of nodes in the list is in the range

[0, 300]

$-100 \leq \text{Node.val} \leq 100$

The list is guaranteed to be

sorted

in ascending order.

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* deleteDuplicates(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 deleteDuplicates(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 deleteDuplicates(self, head: Optional[ListNode]) -> 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 deleteDuplicates(self, head):
        """
        :type head: Optional[ListNode]
        :rtype: 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
 * @return {ListNode}
 */
var deleteDuplicates = 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 deleteDuplicates(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 DeleteDuplicates(ListNode head) {

```

```
}
```

```
}
```

C:

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

Go:

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

```
}
```

```
}
```

Dart:

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

end
end

```

Erlang:

```

%% Definition for singly-linked list.

%% -record(list_node, {val = 0 :: integer(),
%% next = null :: 'null' | #list_node{}}).

-spec delete_duplicates(Head :: #list_node{} | null) -> #list_node{} | null.
delete_duplicates(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 (delete-duplicates head)
  (-> (or/c list-node? #f) (or/c list-node? #f)))
)
```

Solutions

C++ Solution:

```
/*
 * Problem: Remove Duplicates from Sorted List II
 * Difficulty: Medium
 * Tags: array, sort, 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:
    ListNode* deleteDuplicates(ListNode* head) {
        }

    };
}
```

Java Solution:

```
/**
 * Problem: Remove Duplicates from Sorted List II
 * Difficulty: Medium
 * Tags: array, sort, 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() {
 * // 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 deleteDuplicates(ListNode head) {
}
}

```

Python3 Solution:

```

"""
Problem: Remove Duplicates from Sorted List II
Difficulty: Medium
Tags: array, sort, 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 deleteDuplicates(self, head: Optional[ListNode]) -> 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 deleteDuplicates(self, head):
        """
:type head: Optional[ListNode]
:rtype: Optional[ListNode]
"""

```

JavaScript Solution:

```
/**
 * Problem: Remove Duplicates from Sorted List II
 * Difficulty: Medium
 * Tags: array, sort, 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 {ListNode}
 */
var deleteDuplicates = function(head) {
```

```
};
```

TypeScript Solution:

```
/**  
 * Problem: Remove Duplicates from Sorted List II  
 * Difficulty: Medium  
 * Tags: array, sort, 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 deleteDuplicates(head: ListNode | null): ListNode | null {  
};
```

C# Solution:

```
/*  
 * Problem: Remove Duplicates from Sorted List II  
 * Difficulty: Medium  
 * Tags: array, sort, 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 DeleteDuplicates(ListNode head) {
        }
    }
}

```

C Solution:

```

/*
 * Problem: Remove Duplicates from Sorted List II
 * Difficulty: Medium
 * Tags: array, sort, 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;
 * };
 */
struct ListNode* deleteDuplicates(struct ListNode* head) {
    }

```

Go Solution:

```
// Problem: Remove Duplicates from Sorted List II
// Difficulty: Medium
// Tags: array, sort, 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 deleteDuplicates(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 deleteDuplicates(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 deleteDuplicates(_ head: ListNode?) -> ListNode? {
}
}

```

Rust Solution:

```

// Problem: Remove Duplicates from Sorted List II
// Difficulty: Medium
// Tags: array, sort, 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 delete_duplicates(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(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 deleteDuplicates($head) {  
}  
}  
}
```

Dart Solution:

```
/**  
 * Definition for singly-linked list.  
 * class ListNode {  
 * int val;  
 * ListNode? next;  
 * ListNode([this.val = 0, this.next]);  
 * }  
 */  
class Solution {  
ListNode? deleteDuplicates(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 deleteDuplicates(head: ListNode): 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 delete_duplicates(head :: ListNode.t | nil) :: ListNode.t | nil
def delete_duplicates(head) do
  end
end

```

Erlang Solution:

```

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

-spec delete_duplicates(Head :: #list_node{} | null) -> #list_node{} | null.
delete_duplicates(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 (delete-duplicates head)
  (-> (or/c list-node? #f) (or/c list-node? #f)))
  )
```