

LeetCode

206 – Reverse a linked list:

Problem List < > 🔍

Description | Accepted × | Editorial | Solutions | St | <

206. Reverse Linked List

Solved ✓

Easy | Topics | Companies

Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:

Input: `head = [1,2,3,4,5]`
Output: `[5,4,3,2,1]`

Problem List < > 🔍

Description | Accepted × | Editorial | Solutions | Submission

All Submissions

28 / 28 Testcases passed
Time taken: 8 hrs 57 m 4 s
Shreya... submitted at Jan 20, 2026 23:02

Editorial | Solution

Runtime
0 ms | Beats 100.00%
[Analyze Complexity](#)

Memory
13.37 MB | Beats 70.83%

Code

```
1 // Reverse a singly linked list
2
3 // Definition for singly-linked list.
4 struct ListNode {
5     int val;
6     ListNode *next;
7     ListNode() : val(0), next(nullptr) {}
8     ListNode(int x) : val(x), next(nullptr) {}
9     ListNode(int x, ListNode *next) : val(x), next(next) {}
10 };
11
12 //left case will be handle by recursion
13 ListNode* recursionAns = reverseUsingRecursion(prev, curr);
14 return recursionAns;
15
16 }
17
18 ListNode* reverseList(ListNode* head) {
19     ListNode* prev = NULL;
20     // ...
21     prev = curr;
22     curr = nextNode;
23
24     //left case will be handle by recursion
25     ListNode* recursionAns = reverseUsingRecursion(prev, curr);
26     return recursionAns;
27 }
28
29 ListNode* reverseList(ListNode* head) {
30     ListNode* prev = NULL;
31     // ...
32     prev = curr;
33     curr = nextNode;
34
35     //left case will be handle by recursion
36     ListNode* recursionAns = reverseUsingRecursion(prev, curr);
37     return recursionAns;
38 }
```

Saved | Ln 43, Col 3

Testcase | Test Result

Accepted | Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

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

Output

[5,4,3,2,1]

Expected


```
Code
C++ Auto
13 ListNode* reverseUsingRecursion(ListNode* prev, ListNode* curr){
14     //base case
15     if(curr == NULL){
16         return prev;
17     }
18     ListNode* nextNode = curr->next;
19     curr->next = prev;
20     prev = curr;
21     curr = nextNode;
22
23     //left case will be handle by recursion
24     ListNode* recursionAns = reverseUsingRecursion(prev, curr);
25     return recursionAns;
26 }
27
28 ListNode* reverseList(ListNode* head) {
29     ListNode* prev = NULL;
30     ListNode* curr = head;
31
32     return reverseUsingRecursion(prev, curr);
33
34     // while(curr != NULL){
35     //     ListNode* nextNode = curr->next; // so that LL track can't be lost.
36     //     curr->next = prev;
37     //     prev = curr;
38     //     curr = nextNode;
39     // }
40     // head = prev; //updating head '
41     // return prev;

```

Saved Ln 43, Col 3

Testcase Test Result

142. Linked List Cycle II:

 Problem List < > 🔍

Description | Editorial | Solutions | Submissions

142. Linked List Cycle II

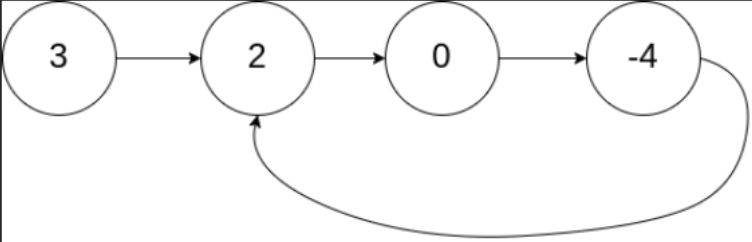
Medium Topics Companies

Given the `head` of a linked list, return *the node where the cycle begins*. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (0-indexed). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter.**

Do not modify the linked list.

Example 1:



```
graph LR
    3((3)) --> 2((2))
    2 --> 0((0))
    0 --> -4((-4))
    -4 --> 2
```

Input: `head = [3,2,0,-4], pos = 1`
Output: tail connects to node index 1

👍 15K 🗨 250 | ☆ 📄 ⓘ 143 Online

```

C++
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
*/
*/
class Solution {
public:
    ListNode* detectCycle(ListNode* head) {
        ListNode* slow = head; //take 1 step at a time
        ListNode* fast = head; //take 2 steps
        ListNode* pos = head;
        while(fast != NULL && fast->next != NULL){
            slow = slow->next;
            fast = fast->next->next;

            if(fast == slow)
            {
                while(slow != NULL && slow->next != NULL)
                {
                    if(pos == slow){
                        return pos;
                    }

                    slow = slow->next;
                    pos = pos->next;
                }
            }
        }
        return NULL;
    }
};

```

Problem List

DescriptionEditorialSolutionsSubmissions

142. Linked List Cycle II

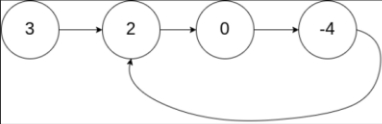
MediumTopicsCompanies

Given the `head` of a linked list, return the node where the cycle begins. If there is no cycle, return `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that `tail's next` pointer is connected to (0-indexed). It is `-1` if there is no cycle. **Note that `pos` is not passed as a parameter.**

Do not modify the linked list.

Example 1:



Input: `head = [3,2,0,-4]`, `pos = 1`
Output: tail connects to node index 1

15K250150 Online

Code

C++Auto

*/
*/
class Solution {
public:
 ListNode* detectCycle(ListNode* head) {
 ListNode* slow = head; //take 1 step at a time
 ListNode* fast = head; //take 2 steps
 ListNode* pos = head;
 while(fast != NULL && fast->next != NULL){
 slow = slow->next;
 fast = fast->next->next;

 if(fast == slow)
 {
 while(slow != NULL && slow->next != NULL)
 {
 if(pos == slow){
 return pos;
 }

 slow = slow->next;
 pos = pos->next;
 }
 }
 }
 return NULL;
 }
};

SavedLn 33, Col 6

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

tail connects to node index 1

Problem List

DescriptionAcceptedEditorialSolutionsSubmission

All Submissions

Accepted18 / 18 testcases passed

Shreya...submitted at Jan 20, 2026 23:15

EditorialSolution

Runtime

3 msBeats 98.27%

Analyze Complexity

Memory

11.42 MBBeats 23.63%

40%
30%
20%
10%
0%

2ms4ms6ms8ms10ms12ms14ms

CodeC++

Code

C++Auto

*/
*/
class Solution {
public:
 ListNode* detectCycle(ListNode* head) {
 ListNode* slow = head; //take 1 step at a time
 ListNode* fast = head; //take 2 steps
 ListNode* pos = head;
 while(fast != NULL && fast->next != NULL){
 slow = slow->next;
 fast = fast->next->next;

 if(fast == slow)
 {
 while(slow != NULL && slow->next != NULL)
 {
 if(pos == slow){
 return pos;
 }

 slow = slow->next;
 pos = pos->next;
 }
 }
 }
 return NULL;
 }
};

SavedLn 33, Col 6

TestcaseTest Result

AcceptedRuntime: 0 ms

Case 1Case 2Case 3

Input

head =
[3,2,0,-4]

pos =
1

Output

tail connects to node index 1

