Description | Editorial | Solutions | Submissions

876. Middle of the Linked List

Easy | Topics | Companies

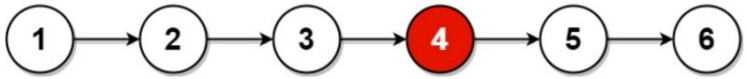
Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

Example 1:


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:




```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

11.9K | 152

Code

Python3 | Auto

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:

```

Saved | Ln 8, Col 9

Testcase | Test Result

Case 1 | Case 2 | +

head =

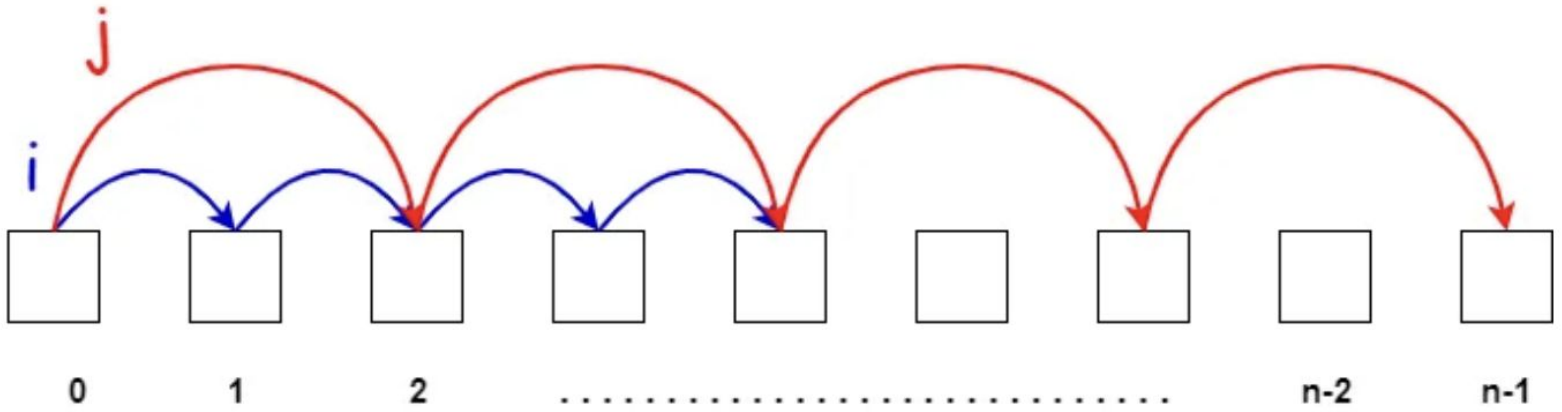
```
[1,2,3,4,5]
```

Source

Description | Editorial | Solutions | Submissions

876. Middle of the Linked List

Easy | Topics | Companies



Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

Description | Accepted | Editorial | Solutions | Submissions

876. Middle of the Linked List

Solved

Easy | Topics | Companies

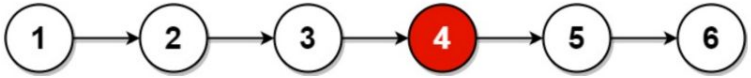
Given the `head` of a singly linked list, return *the middle node of the linked list*.

If there are two middle nodes, return **the second middle** node.

Example 1:


```
Input: head = [1,2,3,4,5]
Output: [3,4,5]
Explanation: The middle node of the list is node 3.
```

Example 2:




```
Input: head = [1,2,3,4,5,6]
Output: [4,5,6]
Explanation: Since the list has two middle nodes with values 3 and 4, we return the second one.
```

Constraints:

- The number of nodes in the list is in the range `[1, 100]`.
- `1 <= Node.val <= 100`

11.9K | 152

Code

Python3 | Auto

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def middleNode(self, head):
        fast = head
        slow = head
        while fast and fast.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow
```

Saved | Ln 7, Col 32

Testcase | Test Result

Accepted Runtime: 62 ms

• Case 1 • Case 2

Input

head =

[1,2,3,4,5]

Description | Editorial | Solutions | Submissions

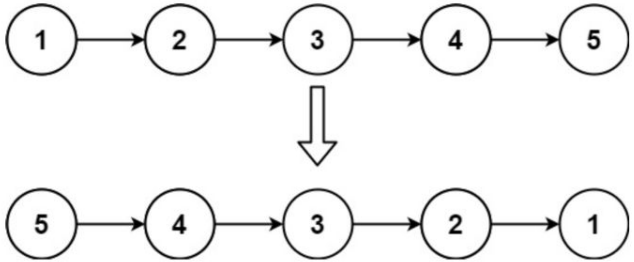
206. Reverse Linked List

Solved

Easy | Topics | Companies

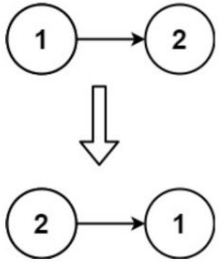
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




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

Example 2:



Code

Python3 | Auto

```python
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:

```

Saved

Testcase | Test Result

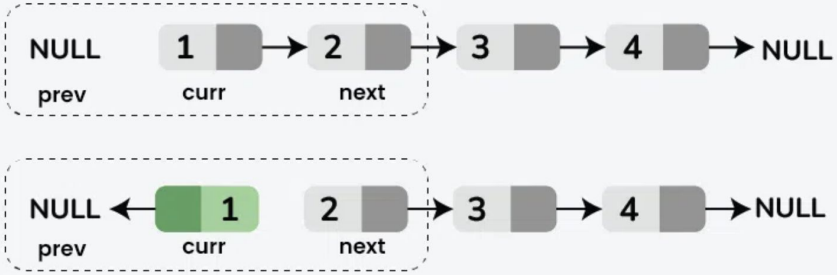
Case 1 | Case 2 | Case 3

head =

```
[1,2,3,4,5]
```

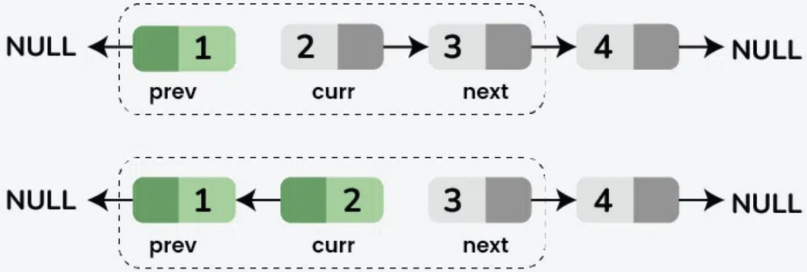
02 Step

Initialize prev pointer = NULL
Store next = curr.next
Update curr.next = prev



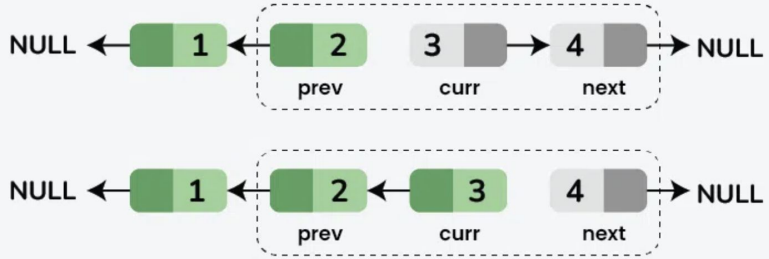
03 Step

Update prev = curr and curr = next
Store next node 3 as next
Update next pointer of current node 2 to previous node 1.



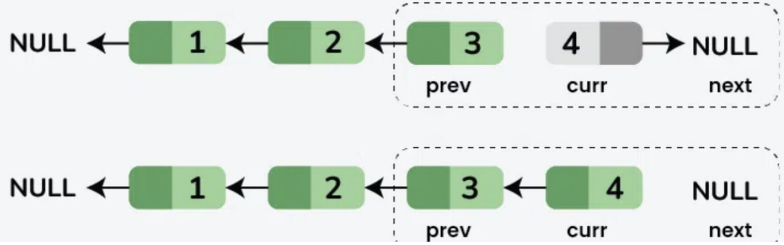
04 Step

Update prev = curr and curr = next
Store next node 4 as next
Update next pointer of current node 3 to previous node 2.



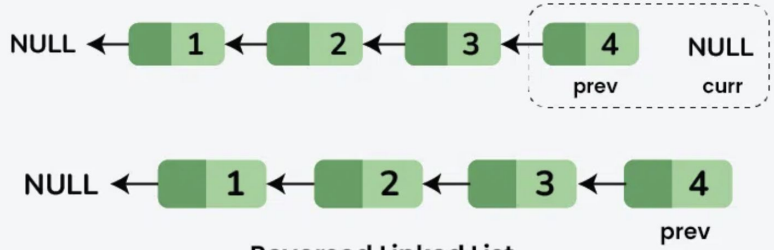
05 Step

Update prev = curr and curr = next
next pointer points to NULL
Update next pointer of current node 4 to previous node 3.



06 Step

Update prev = curr and curr = next
Finally, curr becomes NULL and prev stores the head of the reversed linked list



Reversed Linked List

Description | Editorial | Solutions | Submissions

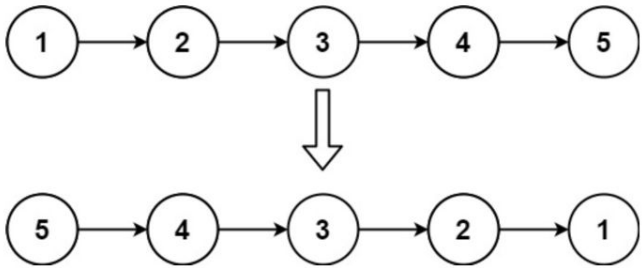
206. Reverse Linked List

Solved

Easy | Topics | Companies

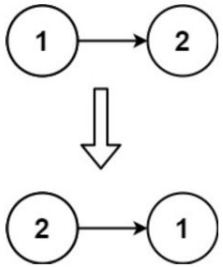
Given the `head` of a singly linked list, reverse the list, and return *the reversed list*.

Example 1:




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

Example 2:



21.9K | 231

Code

Python3 | Auto

```python
class Solution:
    def reverseList(self, head: Optional[ListNode]) -> Optional[ListNode]:
        # Initialize three pointers: curr, prev and next
        curr = head
        prev = None

        # Traverse all the nodes of Linked List
        while curr is not None:
            # Store next
            next_node = curr.next

            # Reverse current node's next pointer
            curr.next = prev

            # Move pointers one position ahead
            prev = curr
            curr = next_node

        # Return the head of reversed linked list
        return prev
```

Saved

Testcase | Test Result

Accepted Runtime: 43 ms

• Case 1 • Case 2 • Case 3

Input

head =

[1,2,3,4,5]

Description | Editorial | Solutions | Submissions

143. Reorder List

Solved

Medium | Topics | Companies

You are given the head of a singly linked-list. The list can be represented as:

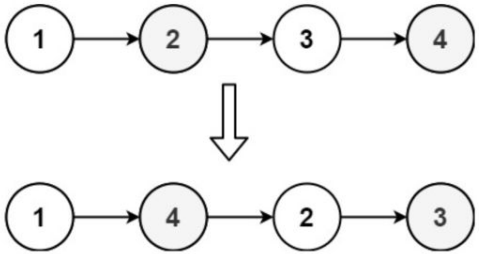
$L_0 \rightarrow L_1 \rightarrow \ldots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \ldots$

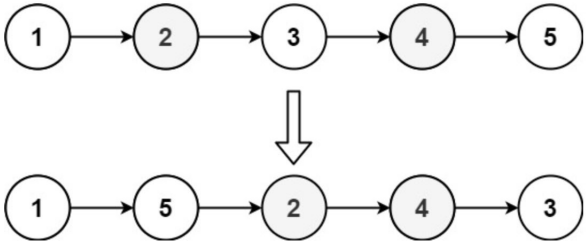
You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:




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

Example 2:



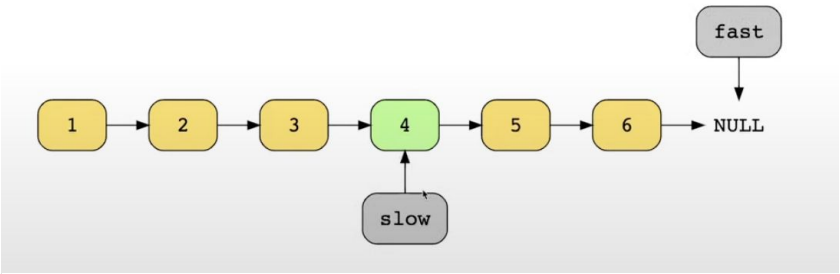

```
Input: head = [1,2,3,4,5]
Output: [1,5,2,4,3]
```

Saved

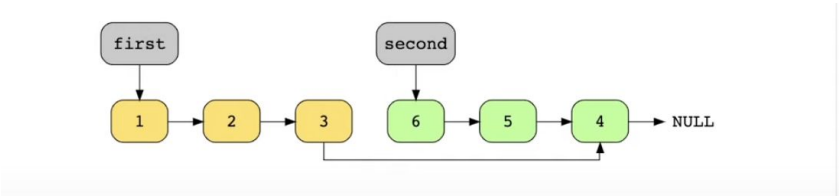
Ln 11, Col 12

Testcase | Test Result

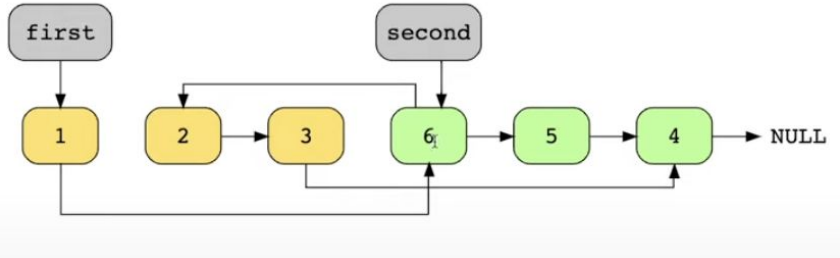
1. Find the middle of the linked list using the fast and slow pointers



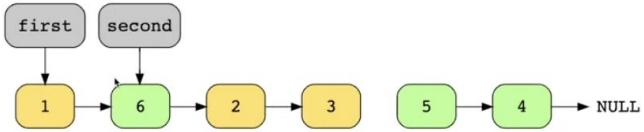
2. Reverse the second half of the linked list using in-place reversal



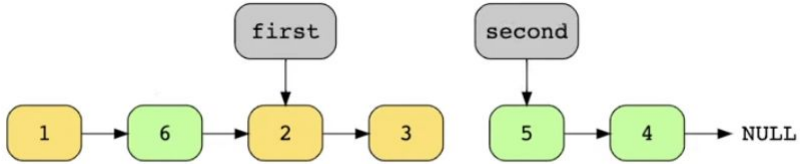
3. Point first.next at second and second.next at first.next

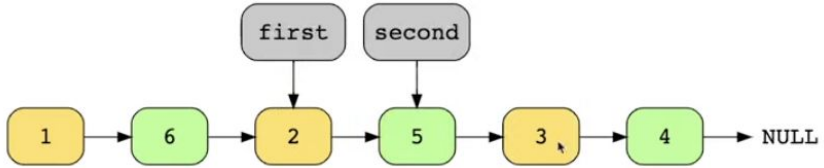


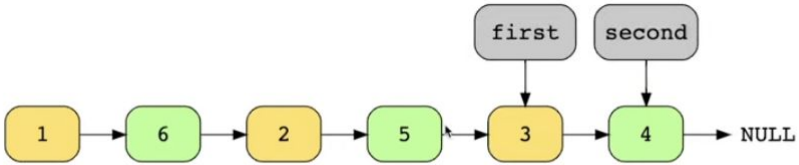
4. After first iteration



5. Move first and second forward







Description | Editorial | Solutions | Submissions

143. Reorder List

Solved

Medium | Topics | Companies

You are given the head of a singly linked-list. The list can be represented as:

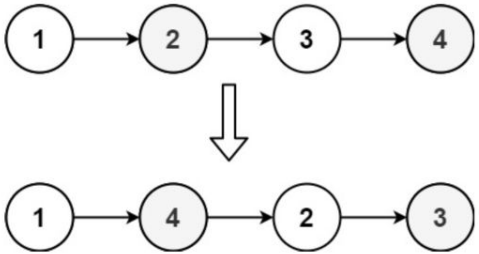
$L_0 \rightarrow L_1 \rightarrow \ldots \rightarrow L_{n-1} \rightarrow L_n$

Reorder the list to be on the following form:

$L_0 \rightarrow L_n \rightarrow L_1 \rightarrow L_{n-1} \rightarrow L_2 \rightarrow L_{n-2} \rightarrow \ldots$

You may not modify the values in the list's nodes. Only nodes themselves may be changed.

Example 1:




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

Example 2:

Code

Python3 | Auto

```python
class Solution:
    def reorderList(self, head: Optional[ListNode]) -> None:
        """
        Do not return anything, modify head in-place instead.
        """
        if not head:
            return
        #find the middle node
        slow = fast = head

        while fast and fast.next:
            slow = slow.next
            fast = fast.next.next

        #reverse the second part of the list
        prev, curr = None, slow
        while curr:
            curr.next, prev, curr = prev, curr, curr.next
        #merge the two sorted lists
        first, second = head, prev
        while second.next:
            first.next, first = second, first.next
            second.next, second = first, second.next
```

Saved | Ln 28, Col 53

Testcase | Test Result

Accepted Runtime: 39 ms

• Case 1 • Case 2

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

36. Valid Sudoku

Solved

Medium | Topics | Companies

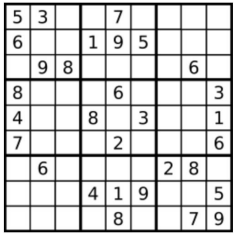
Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:



```
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".",".","2",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: true
```

Example 2:

```
Input: board =
[["8","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".",".","2",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: false
Explanation: Same as Example 1, except with the 5 in the top left corner being modified to 8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.
```

Code

Python | Auto

```python
class Solution(object):
    def isValidSudoku(self, board):
        """
        :type board: List[List[str]]
        :rtype: bool
        """

```

Problem List

Run Submit Premium

Description | Editorial | Solutions | Submissions

36. Valid Sudoku

Medium Topics Companies

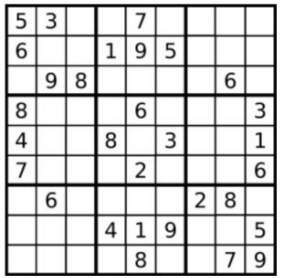
Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:



```
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
```

10.9K 153

`r // 3`: This part divides the row index `r` by 3 using integer division. It determines which of the three rows of 3x3 boxes the cell is in.

`c // 3`: determines which of the three columns of 3x3 boxes the cell is in

Problem List | Run | Submit | Premium

Description | Editorial | Solutions | Submissions

36. Valid Sudoku

Solved

Medium | Topics | Companies

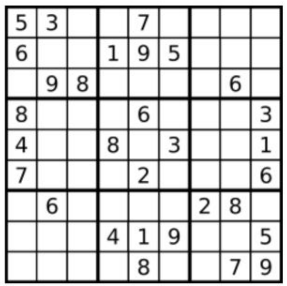
Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:




```
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
```

10.9K | 153

Code

Python | Auto

```python
class Solution(object):
    def isValidSudoku(self, board):
        cols = collections.defaultdict(set)
        rows = collections.defaultdict(set)
        squares = collections.defaultdict(set)

        for r in range(9):
            for c in range(9):
                if board[r][c] == ".":
                    continue
                if (board[r][c] in rows[r] or
                    board[r][c] in cols[c] or
                    board[r][c] in squares[(r//3,c//3)]):
                    return False
                cols[c].add(board[r][c])
                rows[r].add(board[r][c])
                squares[(r//3,c//3)].add(board[r][c])
        return True
```

Saved | Ln 18, Col 20

Testcase | Test Result

Accepted Runtime: 11 ms

• Case 1 • Case 2

Input

board =

Description | Editorial | Solutions | Submissions

48. Rotate Image

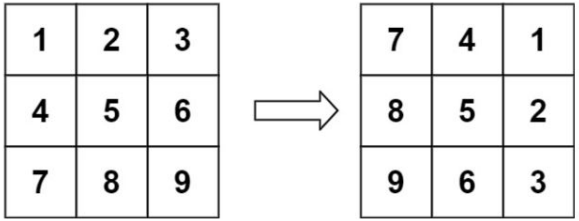
Solved

Medium | Topics | Companies

You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

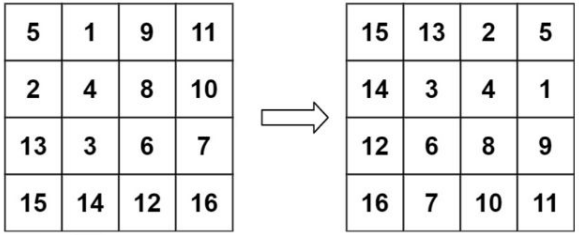
You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:



17.9K | 168

Code

Python | Auto

```python
class Solution(object):
    def rotate(self, matrix):

```

Saved | Ln 3, Col 1

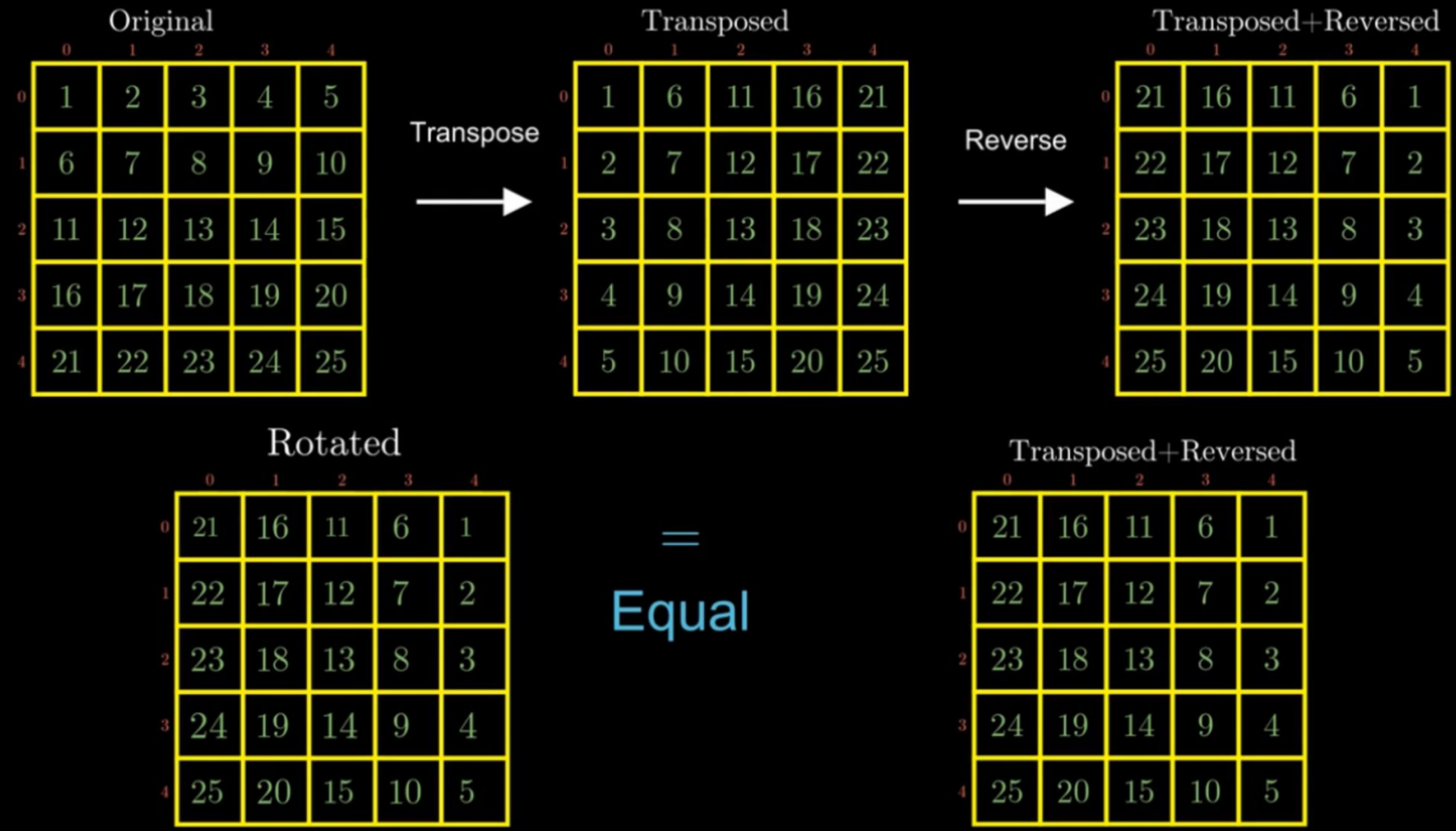
Testcase | Test Result

Accepted Runtime: 12 ms

• Case 1 • Case 2

Input

matrix =


Original
Transpose
Transposed
Reverse
Transposed+Reversed
Rotated
=
Equal
Transposed+Reversed

Description | Editorial | Solutions | Submissions

48. Rotate Image

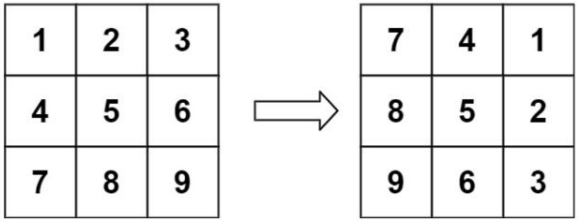
Solved

Medium | Topics | Companies

You are given an `n x n` 2D `matrix` representing an image, rotate the image by **90** degrees (clockwise).

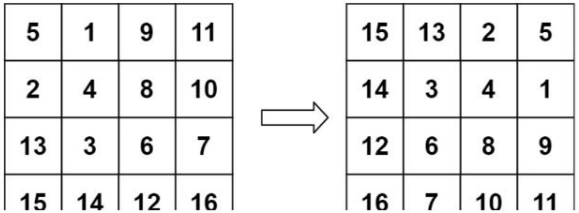
You have to rotate the image **in-place**, which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:



Code

Python | Auto

```python
class Solution(object):
    def rotate(self, matrix):
        self.transpose(matrix)
        self.reflect(matrix)

    def transpose(self, matrix):
        n = len(matrix)
        for i in range(n):
            for j in range(i + 1, n):
                matrix[j][i], matrix[i][j] = matrix[i][j], matrix[j][i]

    def reflect(self, matrix):
        n = len(matrix)
        for i in range(n):
            for j in range(n // 2):
                matrix[i][j], matrix[i][-j - 1] = (
                    matrix[i][-j - 1],
                    matrix[i][j],
                )
```

Saved | Ln 19, Col 18

Testcase | Test Result

Accepted Runtime: 12 ms

• Case 1 • Case 2

Input