Linked List II

Pre-requisites

1. Linked list

Problem I

Return the n-th last element of a singly linked list.

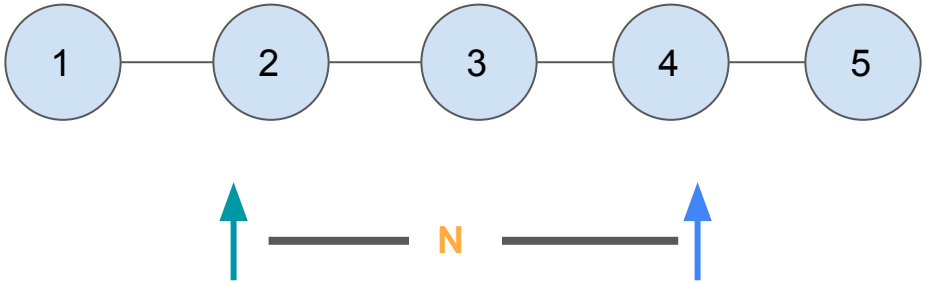
Approach I

- Count number of nodes (K) in one pass
- Find (K - n)th node on second pass

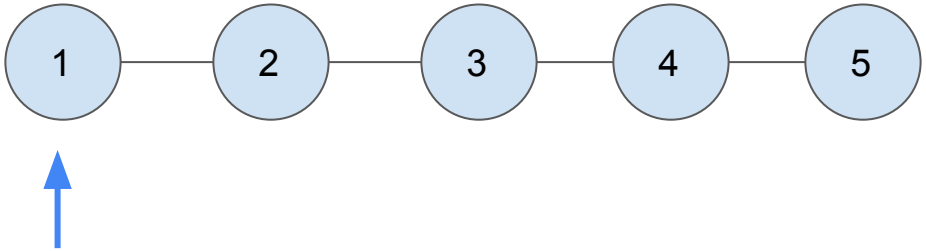
Two Pointer Technique in Linked List

Approach II

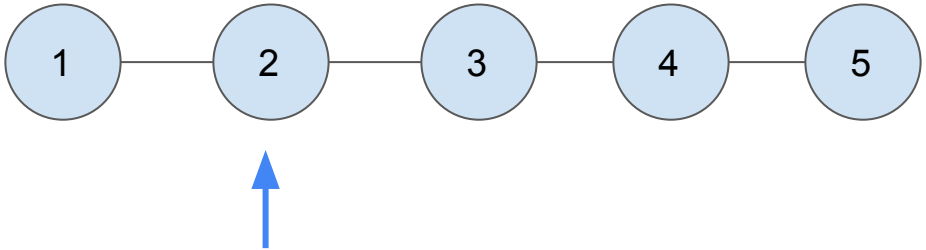
- iterate with two pointers
- one seeking the tail and the other lagging by the nth amount.



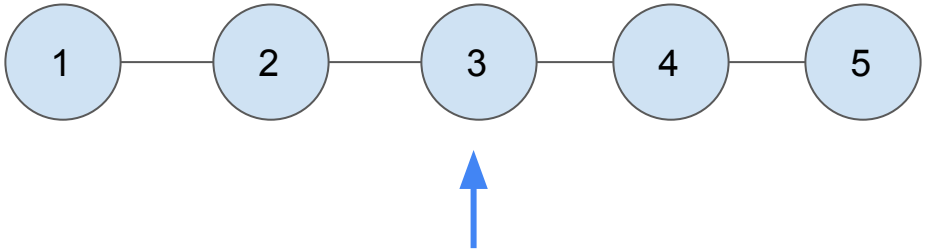
Simulation



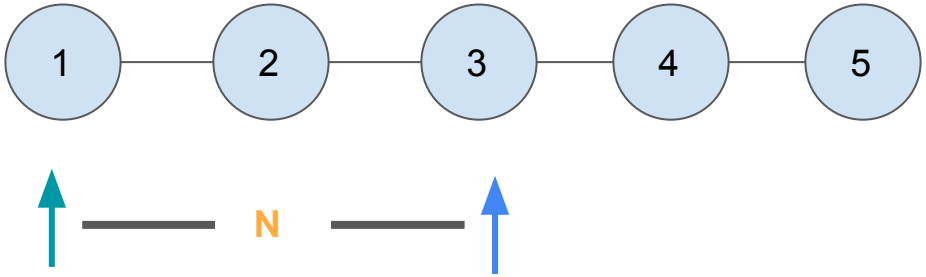
Simulation



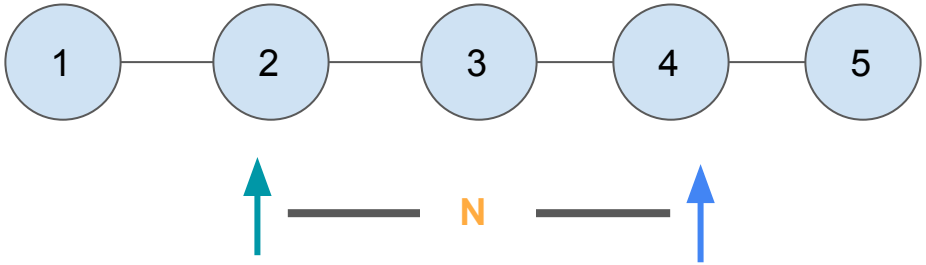
Simulation



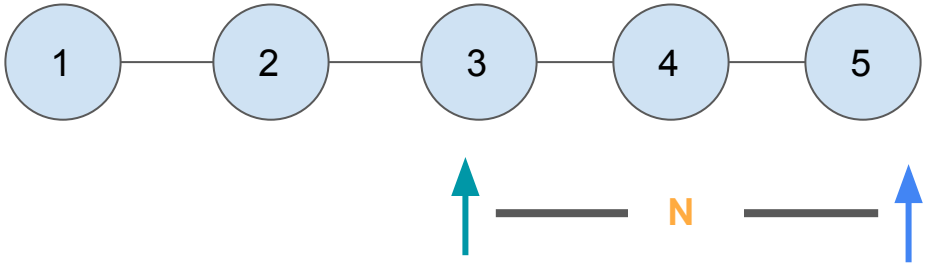
Simulation



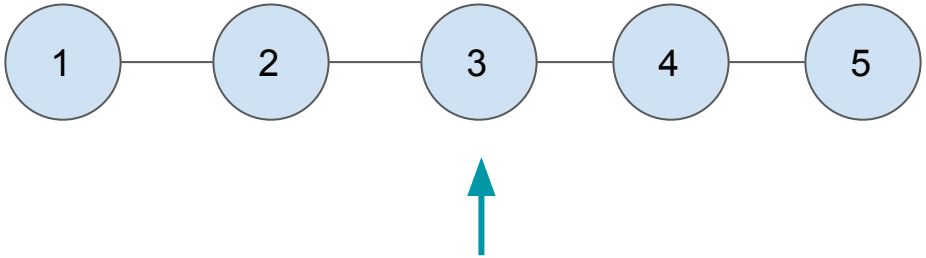
Simulation



Simulation



Simulation



Nth element from right

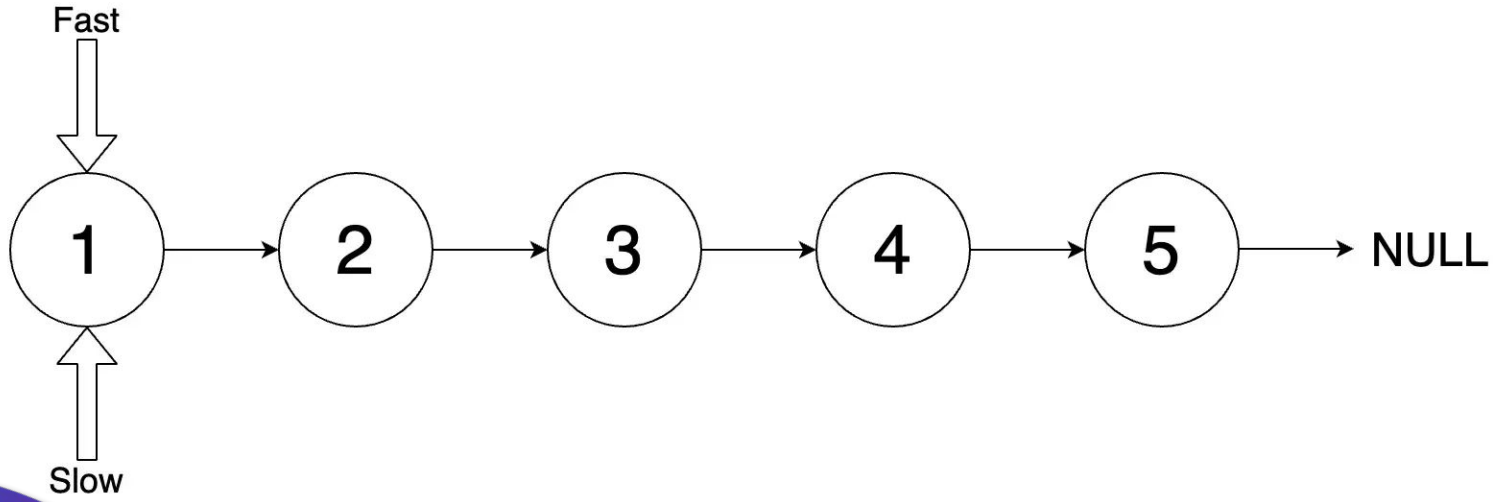
Problem II

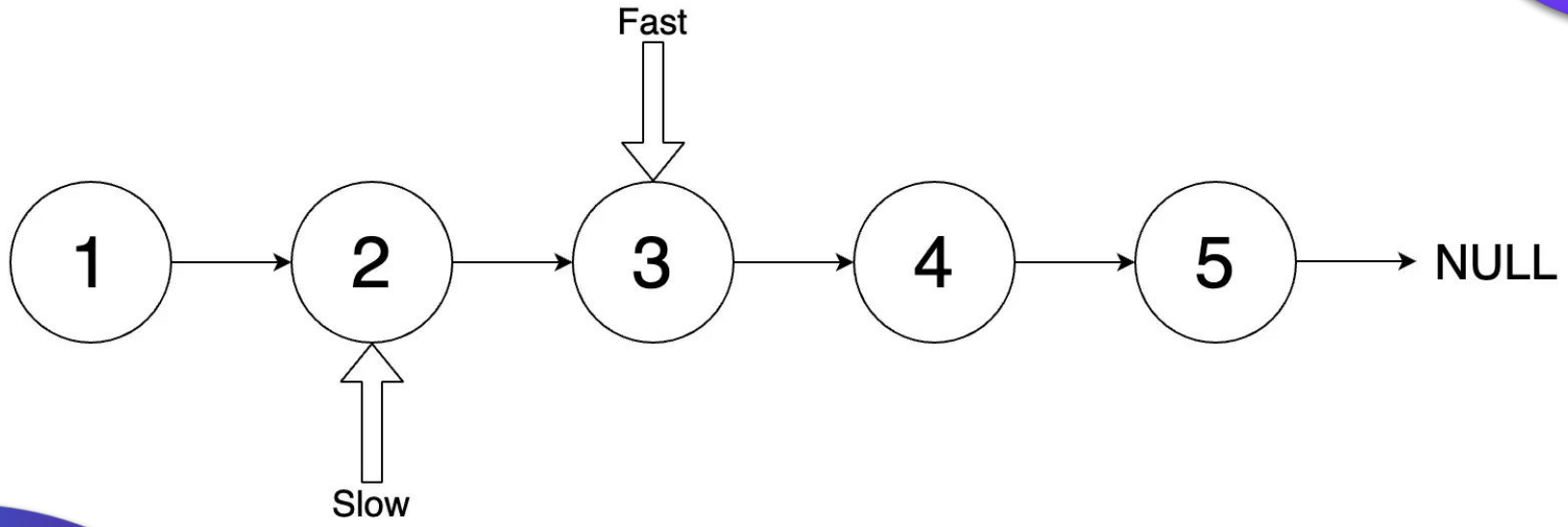
Return the middle element of a linked List

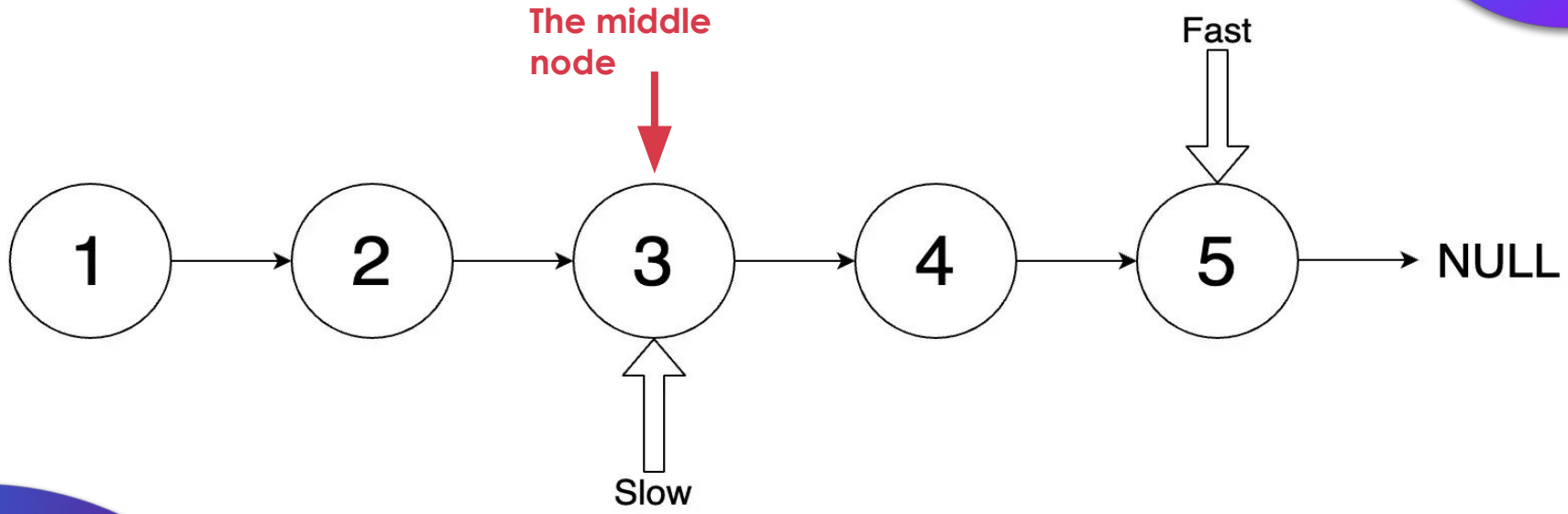
Approach : Fast and slow Pointers

- Iterate with two pointers: Fast and slow
- Slow pointer goes one step at a time
- Fast goes two steps at a time
- When the fast pointer moves to the very end of the Linked List, the slow pointer is going to point to the middle element of the linked list.

Simulation




Fast
1
2
3
4
5
NULL
Slow


The middle node
Fast
1
2
3
4
5
NULL
Slow

Pair Programming

[Middle element of a linked list]

Check Point

Link

Floyd’s Cycle Finding Algorithm

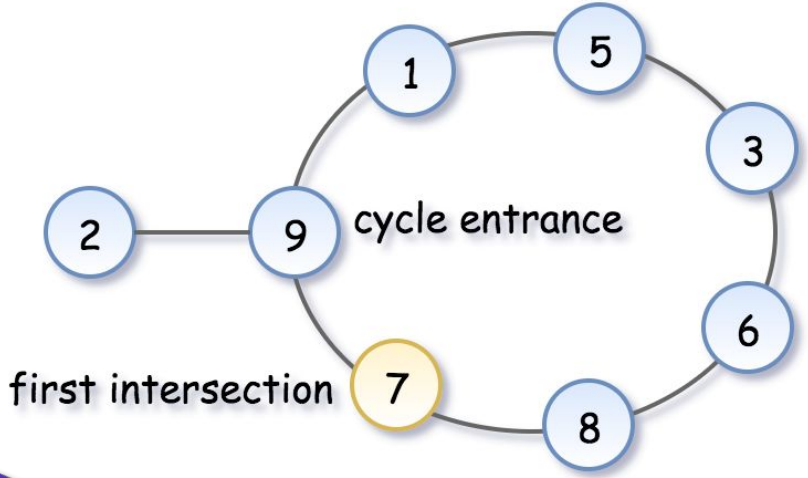
Initial Problem

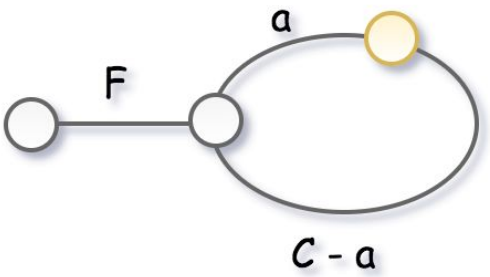
Given a linked list, return the node where the cycle begins?

Floyd's algorithm consists of two phases and uses two pointers, usually called tortoise and rabbit.

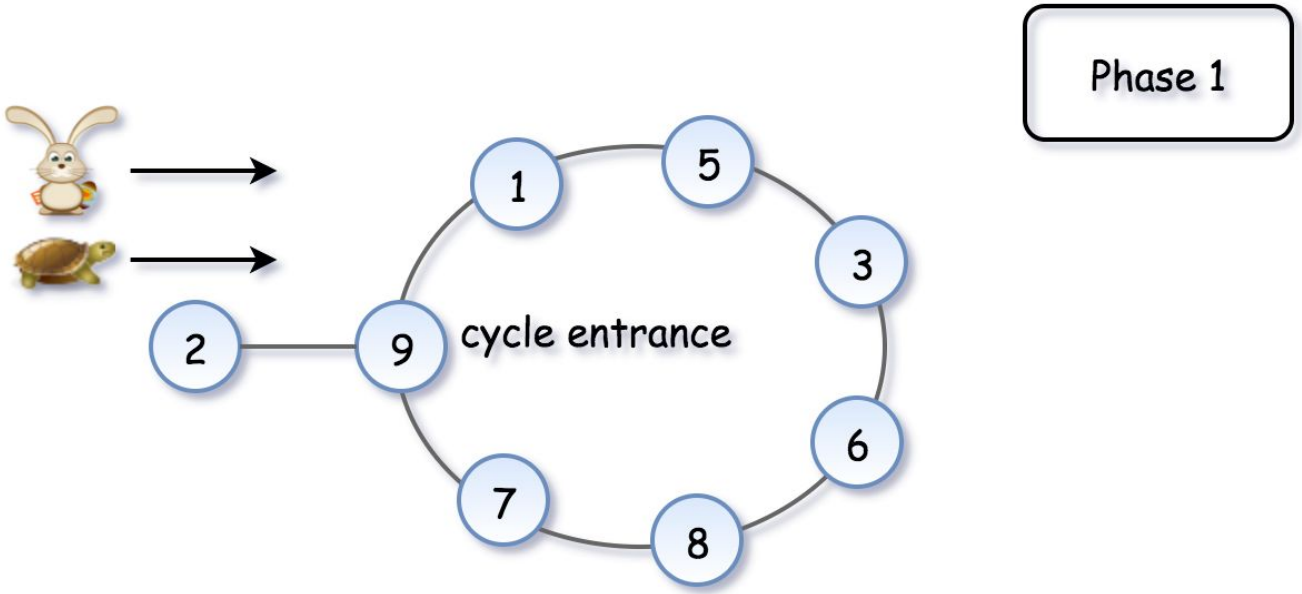
Phase I - Cycle Detection

- Rabbit = cur.next.next goes twice as fast as tortoise = cur.next
- Since tortoise is moving slower, the rabbit catches up to the tortoise at some *intersection* point
- Note that the intersection point is not the cycle entrance in the general case.

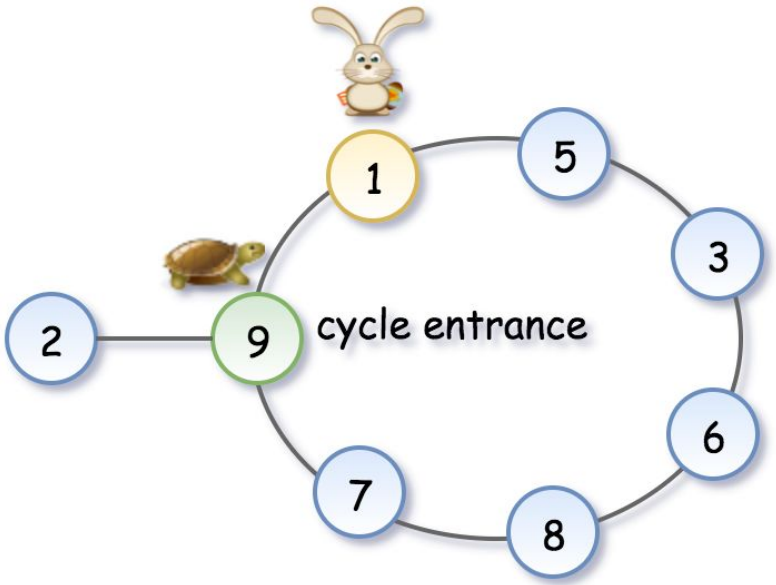

1
5
3
2
9
cycle entrance
6
first intersection
7
8


a
F
C - a

Phase 1

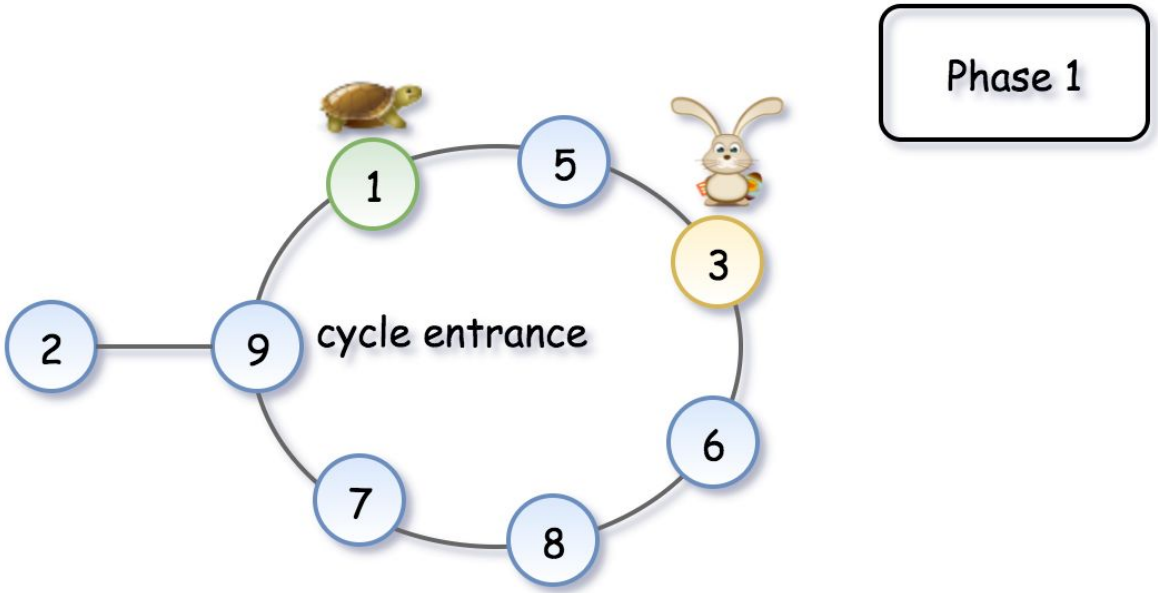


Phase 1

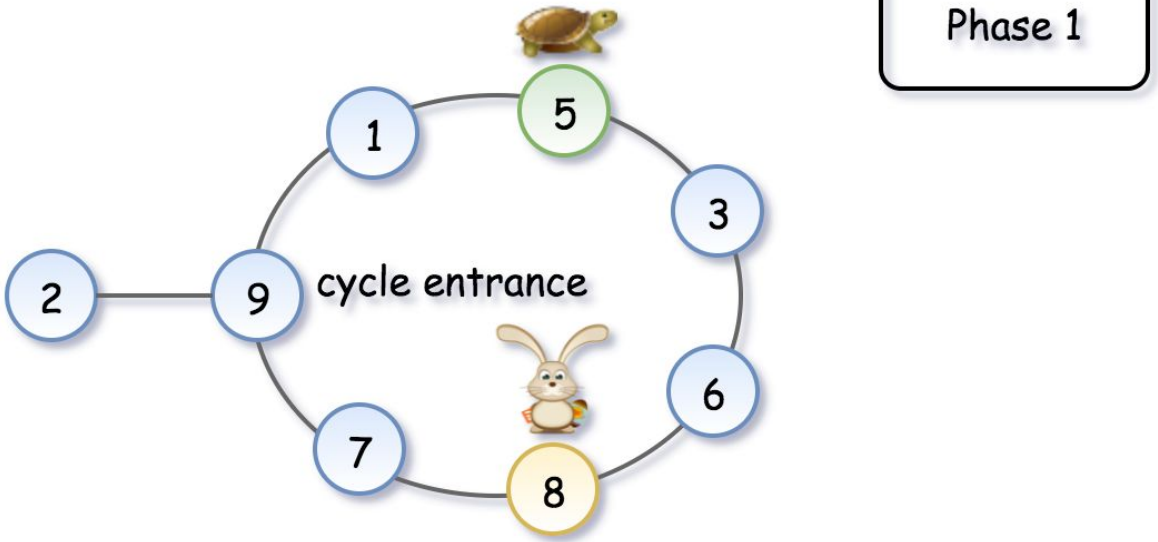


Phase 1

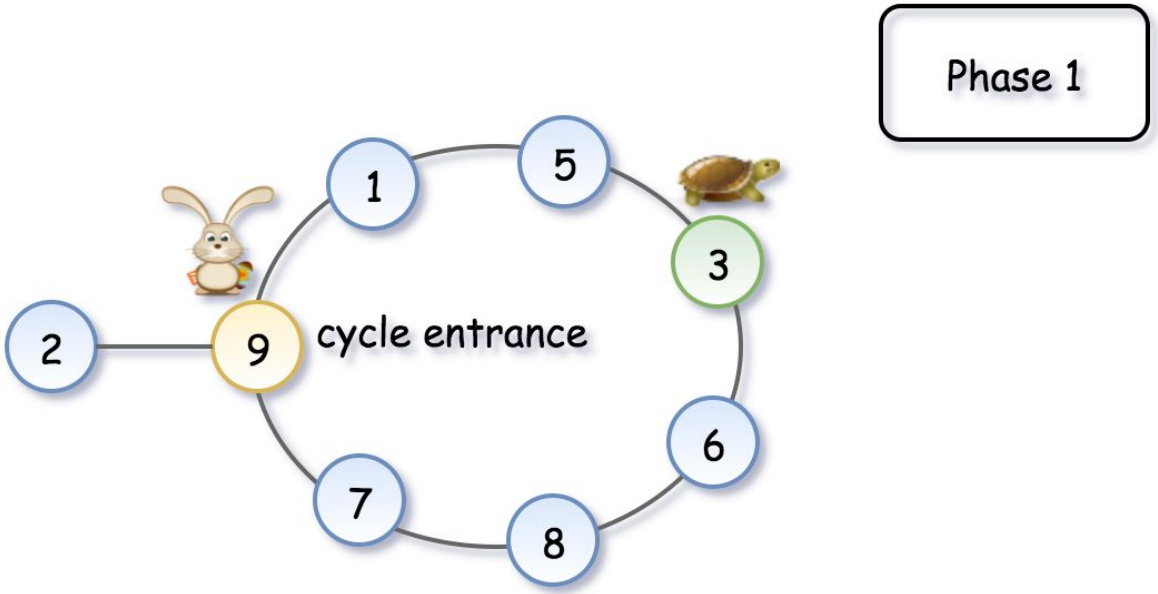
Phase 1



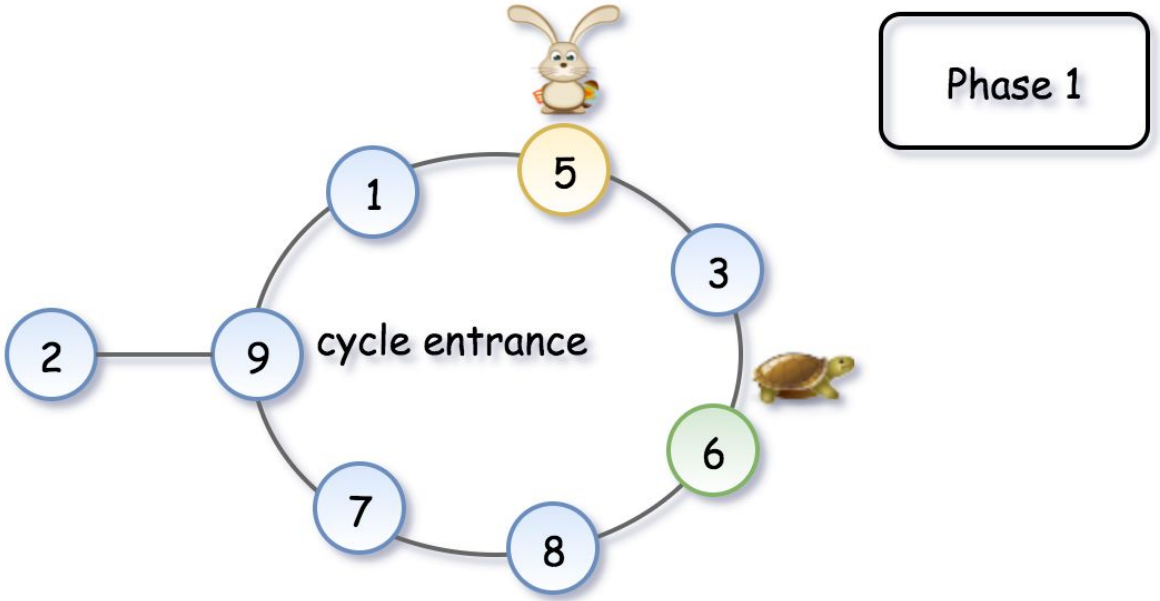
Phase 1



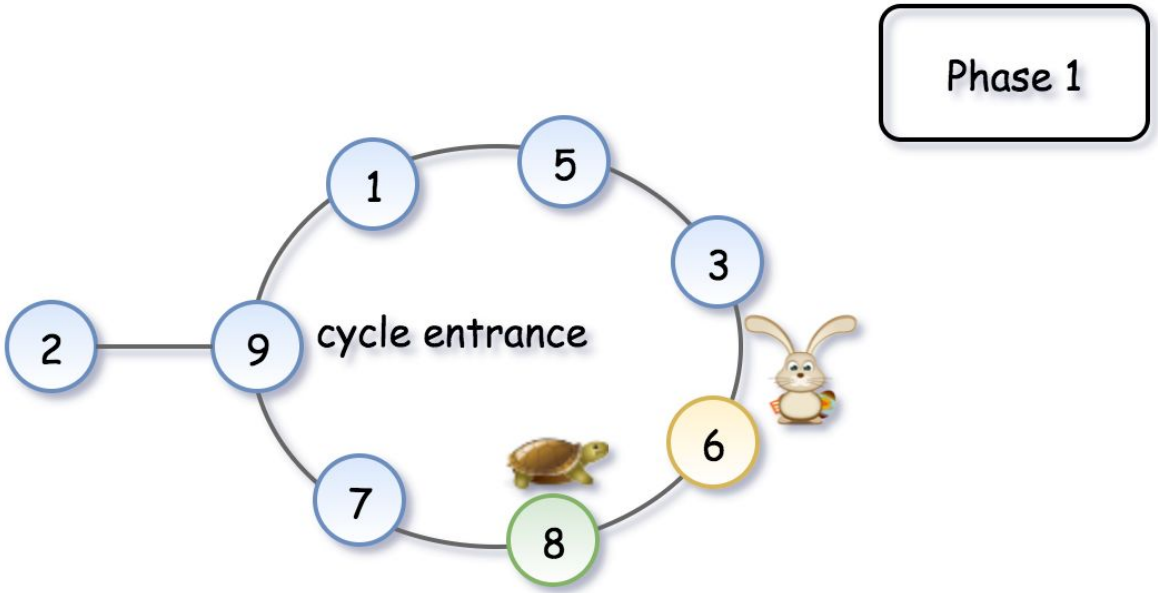
Phase 1



Phase 1



Phase 1



Phase II - Determine Cycle Entrance

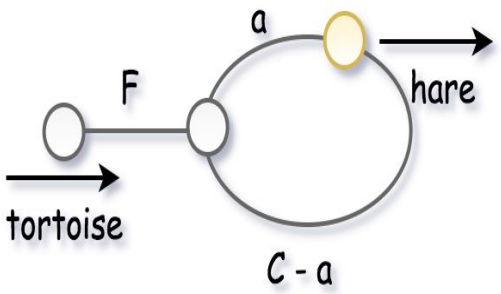
To compute the cycle entrance, let's note that the rabbit has traversed twice as many nodes as the tortoise, *i.e.*

$2 \cdot d_{tortoise} = d_{rabbit}$, implying:

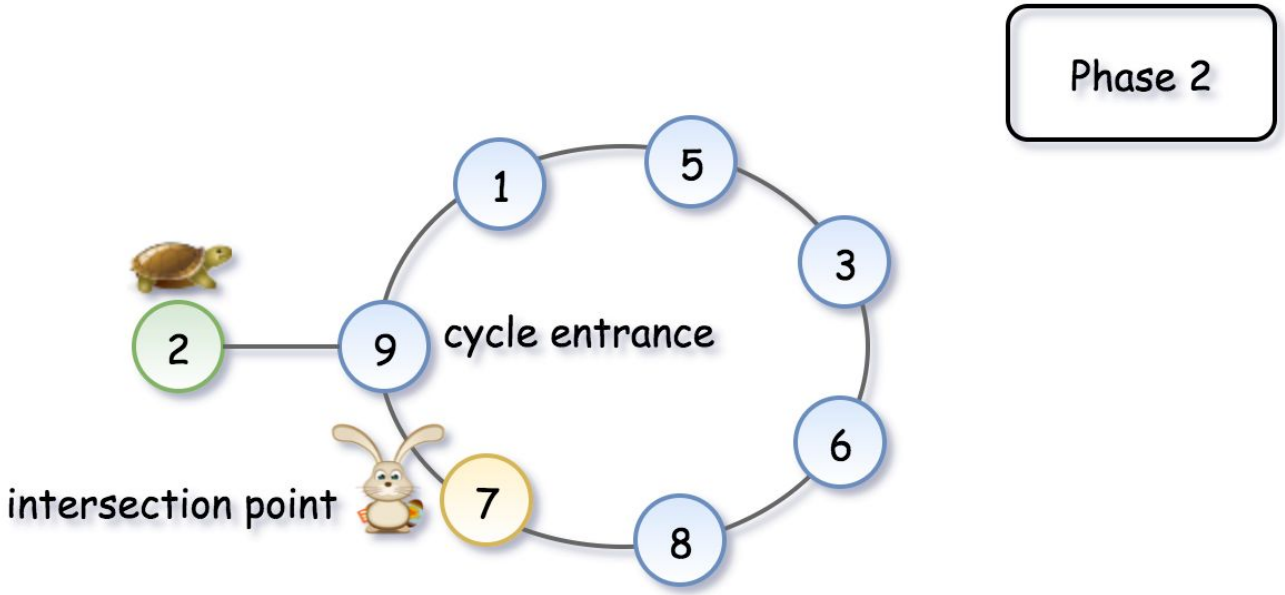
$2 \cdot (F + m \cdot C + a) = F + n \cdot C + a$, where n, m is some positive integer.

Hence the coordinate of the intersection point is $F = (n - 2 \cdot m) \cdot C - a$

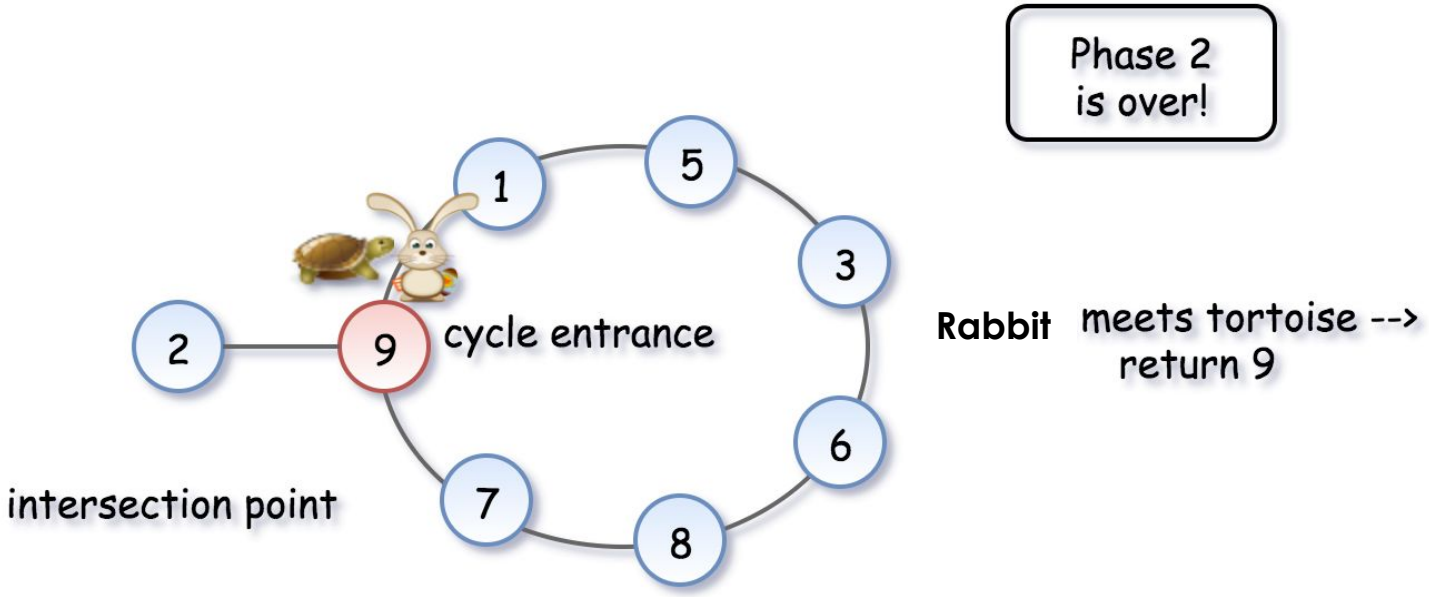
- we give the tortoise a second chance by slowing down the rabbit,
  - tortoise = tortoise.next
  - rabbit = rabbit.next
- The tortoise is back at the starting position, and the rabbit star



Phase 2



Phase 2 is over!



Back to Initial Problem
Pair Programming

[Linked List Cycle]

Common Pitfalls

Can you spot the error?

```
def containsTarget(head, target):
    while head.val != target:
        head = head.next

    return head.val == target
```

Null Pointer Example

```
def containsTarget(head, target):
    while head.val != target:
        head = head.next

    return head.val == target
```

What if head is null?

Solution

```
def containsTarget(head, target):
    while head and head.val != target:
        head = head.next

    return head.val == target
```

Can you spot the error?

```
def middleNode(self, head):
    slow = head
    fast = head

    while fast:
        slow = slow.next
        fast = fast.next.next
    return slow
```

Solution

```
def middleNode(self, head):
    slow = head
    fast = head

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

- Null pointer
  - head.next <> check existence of head
  - head.next.next <> check existence of head and head.next

Pitfalls

```
def deleteNodeAtGivenPosition(position,head):
    if head is None:
        return
    index = 0
    previous = None
    while head.next and index < position:
        previous = head
        head = head.next
        index += 1
    previous.next = head.next
    return head
```

Losing the reference to head of the linked list Example

```
def deleteNodeAtGivenPosition(position,head):
    if head is None:
        return
    index = 0
    previous = None
    while head.next and index < position:
        previous = head
        head = head.next
        index += 1
    previous.next = head.next
    return head
```

What if head is null?

Is head the head of the modified linked list?

Pitfalls - Continued

- Losing the reference to head of the linked list
  - Put the head in a variable before any operation to return the head at the end

Practice Questions

- Reverse Linked List
- Palindrome Linked List
- Merge Two Sorted Lists
- Remove duplicates from sorted list
- Partition List
- Linked List Cycle II
- LRU Cache
- Delete Node in a Linked List

Resources

- Leetcode Explore Card: has excellent track path with good explanations
- Leetcode Solution (Find the Duplicate Number) : has good explanation about Floyd's cycle detection algorithm with good simulation
- Elements of Programming Interview book: has a very good Linked List Problems set