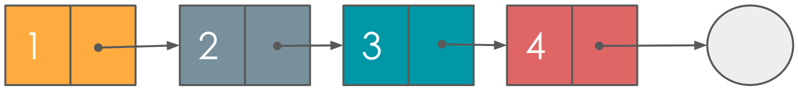
Linked List



Lecture Flow

1) Pre-requisites
2) Problem definitions and applications
3) Types of linked list
4) Different approaches
5) Checkpoint
6) Variants
7) Pair Programming
8) Recognizing in questions
9) Things to pay attention(common pitfalls)
10) Practice questions
11) Resources
12) Quote of the day

Pre-requisites

- Class
- Two Pointer Technique
- Linear Data Structure

A2SV
Africa To Silicon Valley

Definitions

Linked List is a linear data structure that stores value and grows dynamically

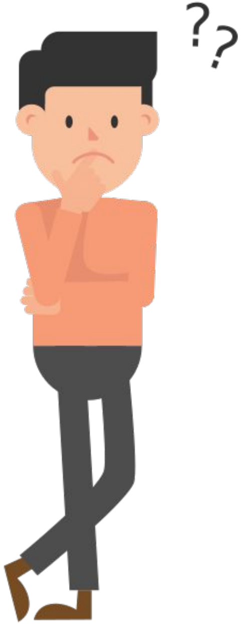
Node

- stores the **value** and the **reference to the next node**.
- is a collection of two or more sub-elements or parts.
- Is the simplest linked list example

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None # Type: Node
```

Singly Linked List is when nodes have only next’s node reference.

Why do we need **linked list** when we have **arrays**?



Africa To Silicon Valley

Why Linked List when you have Arrays?

- Arrays by default **don't grow** dynamically
- **Inserting** in the **middle** of an array is **costly**
- **Removing** elements from the **middle** of array is **costly**

Arrays vs Linked List

Array	Linked List
Fixed size	Dynamic size
Insertions and Deletions are inefficient	Insertions and Deletions are efficient
Random access	No random access
Possible waste of memory	No waste of memory
Sequential access is faster	Sequential access is slow

Problem Definition

A problem with requirement of O(1) deletion and insertion

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in `O(1)` average time complexity.

Constraints:

- `1 <= capacity <= 3000`
- $0 <= key <= 10^4$
- $0 <= value <= 10^5$
- At most $2 * 10^5$ calls will be made to `get` and `put`.

How would we approach the problem with Array?

Linked List **grows** and **shrinks** in **O(1) time** complexity. So, problems you observed from arrays won't apply here; If we have a **dictionary** to give us the **nodes in O(1)** time complexity.

Questions?

Pros and Cons?

Advantages of using linked list

- Dynamic data structure
- No memory wastage
- Insertion and Deletion Operations

Disadvantages

- Traversal
- Reverse Traversing
- Random Access

Implement Linked List with the given constraints(Link):

- addAtTail(val)
  - O(n)
  - Append elements
- print()
  - O(n)
  - Print all elements in their order
- main()
  - instantiate, add [1,2,3,4] and print

Dummy Node

- is **a node** that points to the head of a linked list which will be **discarded at the end**
- When to use a Dummy Node?
  - if you are potentially **modifying the head** of linked list, use dummy node

Check Point

Link

Useful Methods

Traversing a Linked List

- **Start with the head** of the list. Access the content of the head node if it is not null
- go to the **next node(if exists)** and access the node information
- **Continue until no more nodes** (that is, you have reached the last node)

Implementation

```python
def traverse(self, head) -> None:
    currentNode = head
    while currentNode:
        print(currentNode.val)
        currentNode = currentNode.next
```

Pair Programming

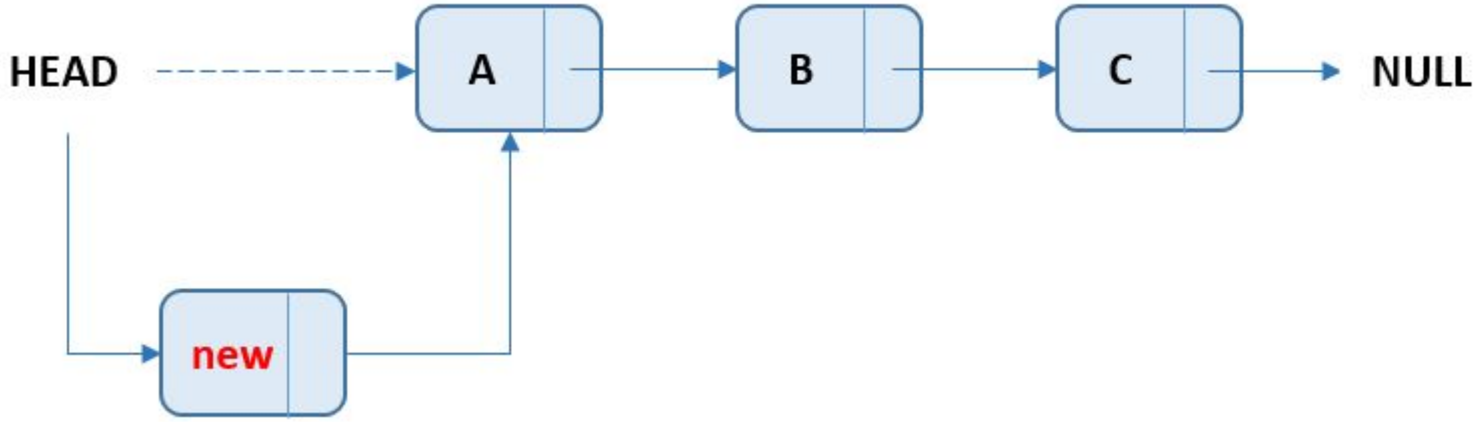
Design Linked List(implement **get**)

7:00

Inserting a node in linked list

Insert at the beginning

- If list is empty, **make new node the head** of the list
- Otherwise, **connect new node to the current head**
- **make new node the head** of the list.



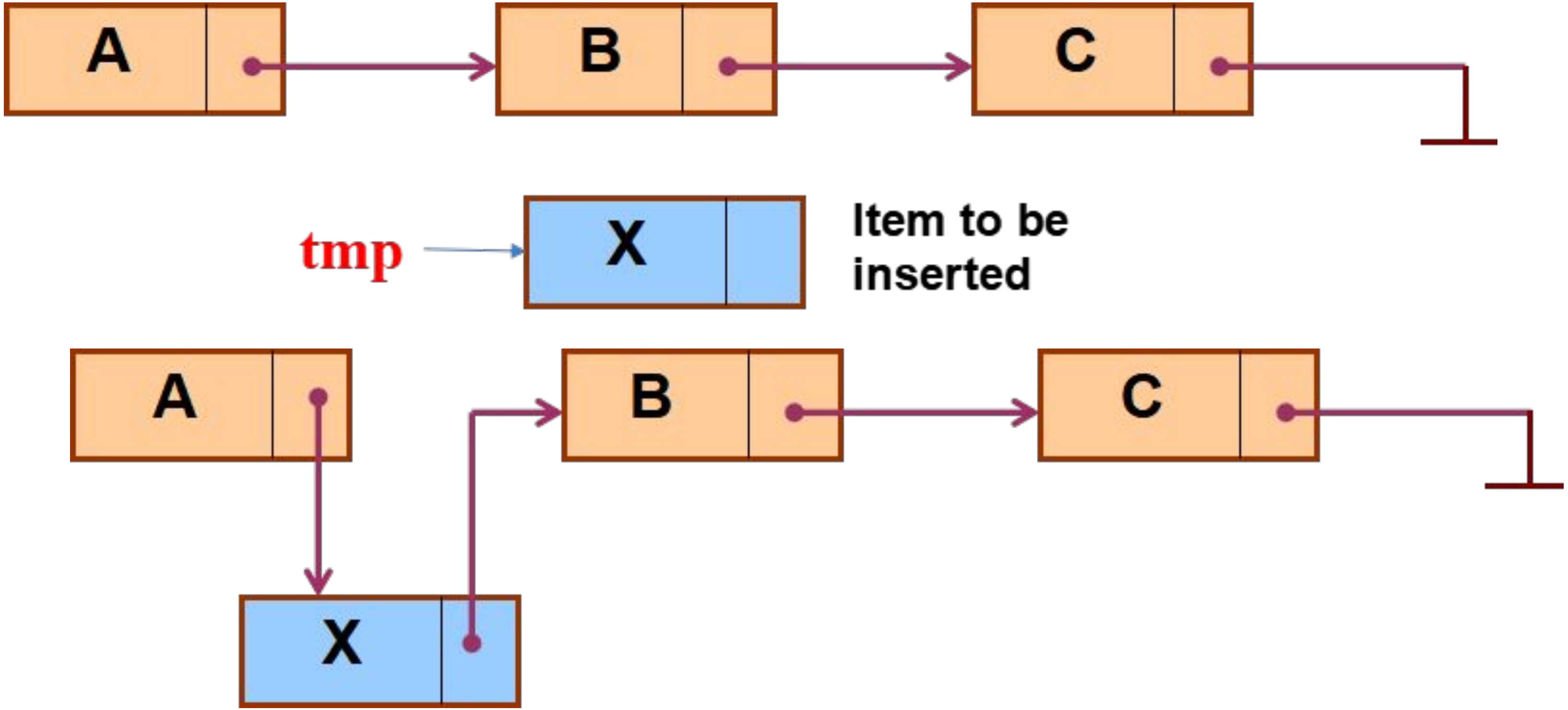
Implementation

```python
def insert(self,head, new_data):
    # if head is already None
    if not head:
        return Node(new_data)

    # Put in the data in a new node
    new_node = Node(new_data)

    # Make next of new Node as head
    new_node.next = head

    # Move the head to point to new Node
    head = new_node

    return head
```


Africa To Silicon Valley

Insert at any position

- find **the insert position** and **the previous node**
- And then **make the next of new node** as the next of previous node
- finally, **make the next of the previous node** the new node


A
B
C
tmp
X
Item to be inserted
A
B
C
X

Implementation

```
def insert(self,head, new_data,prev_node):
    # Put in the data
    new_node = Node(new_data)

    # Make next of new Node as next of prev_node
    new_node.next = prev_node.next

    # make next of prev_node as new_node
    prev_node.next = new_node
```

Can we avoid using two approaches when we are inserting(beginning & any place)? How?

Yes, we can use Dummy Node.

Pair Programming

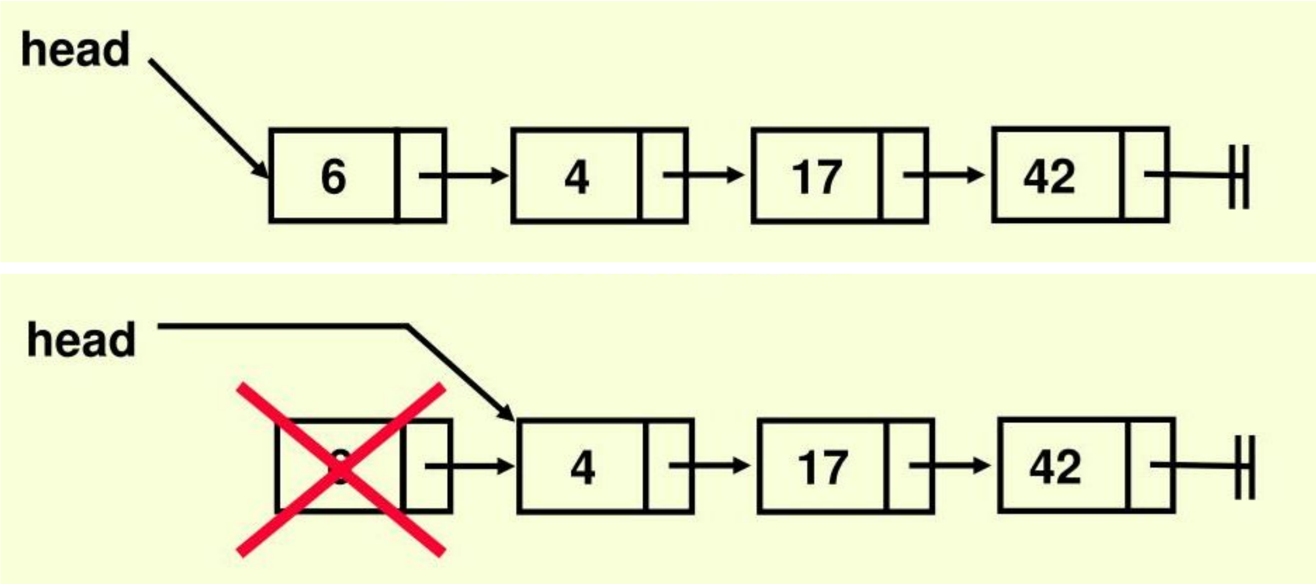
[Design Linked List](implement **addAtAnyIndex**)

7:00

Delete a node from the linked list

Delete a node at the beginning

- make the **second node as head**
- **Discard the memory allocated** for the first node.

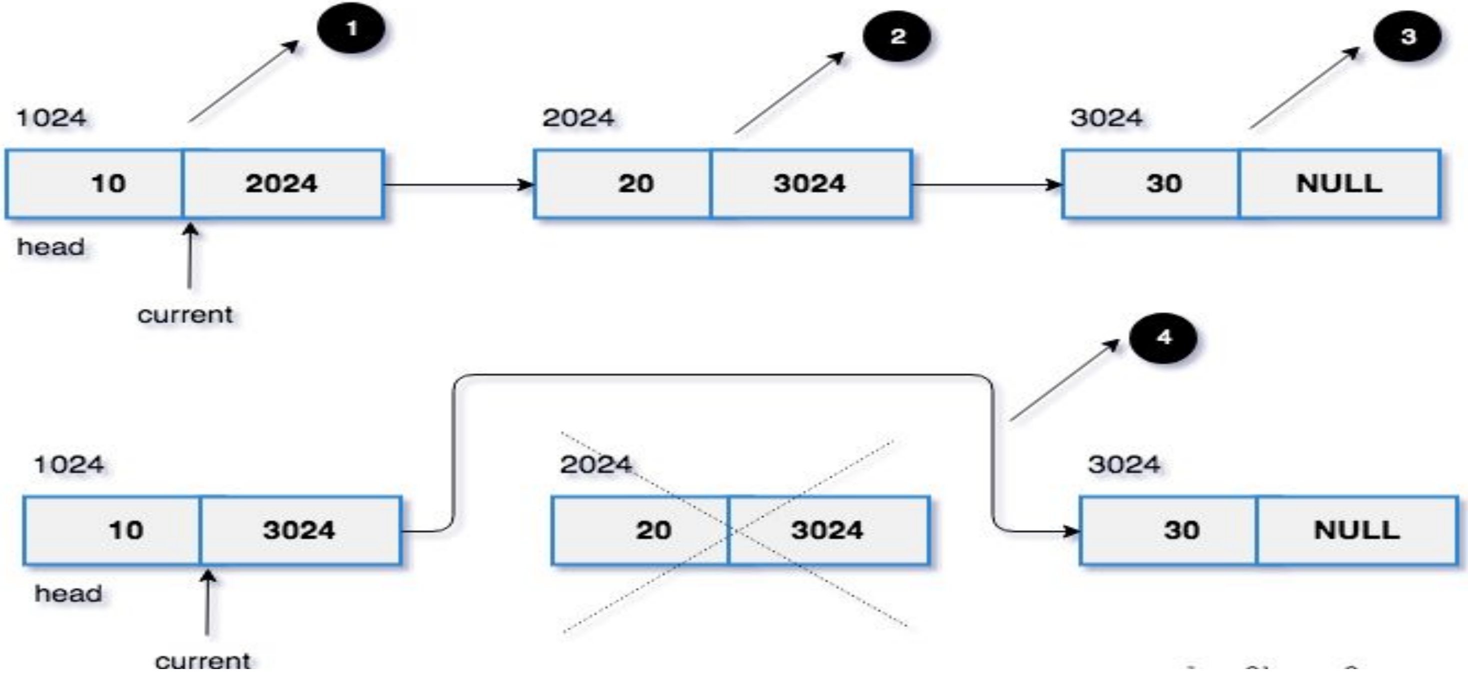


Implementation

```python
def removeFirstNode(head):
    if not head:
        return None

    # Move the head pointer to the next node
    head = head.next
    return head
```

Delete a node at any position

- **Find a match** the node to be deleted
- Get the **previous node**
- Make the **previous node next** point to the **next of the deleted node**


1
2
3
1024
10
2024
head
current
2024
20
3024
3024
30
NULL
4
1024
10
3024
head
current
2024
20
3024
3024
30
NULL

Implementation

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

Can we avoid using two approaches when we are deleting nodes? How?

Yes, we can use Dummy Node.

Pair Programming

Design Linked List(implement **deleteAtIndex**)


7:00

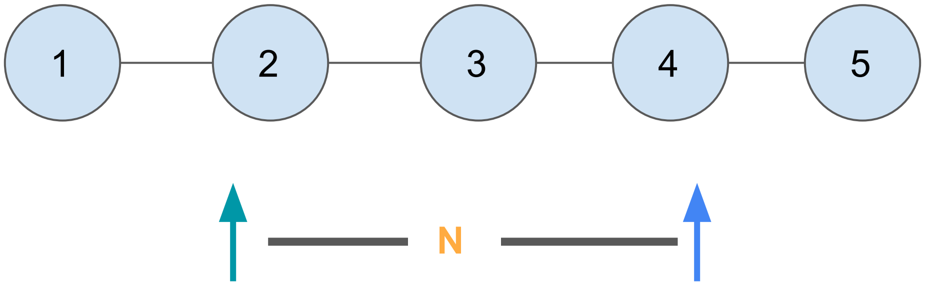
Two Pointer Technique in Linked List

Problem I

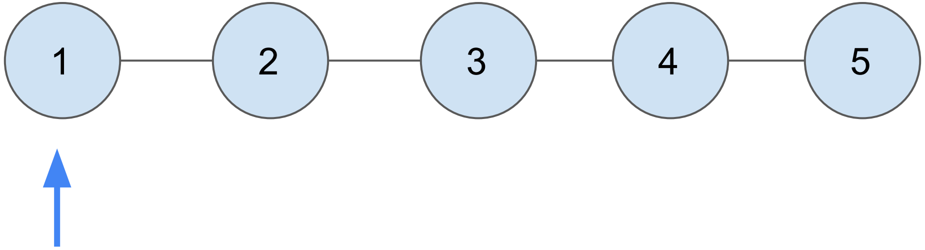
Return the nth last element of a singly linked list.

Approach I

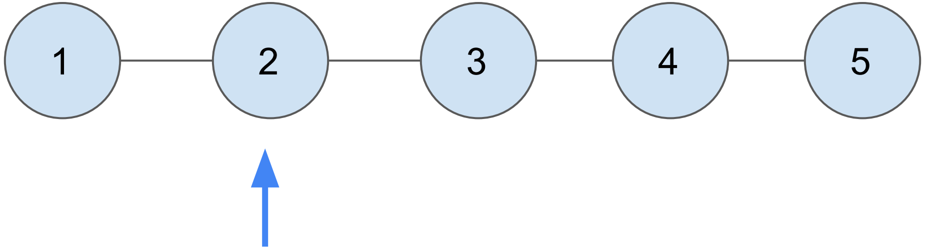
- iterate with two pointers
- one seeking the tail and the other lagging by the nth amount.



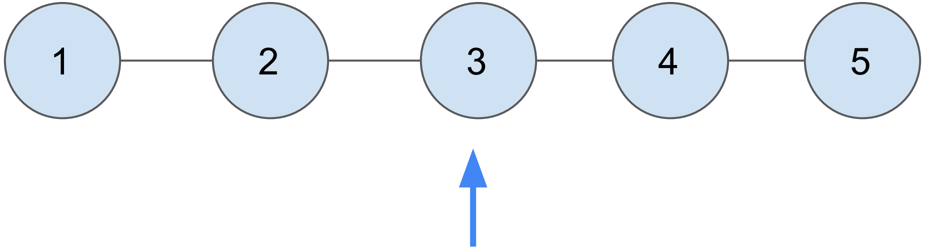
Simulation



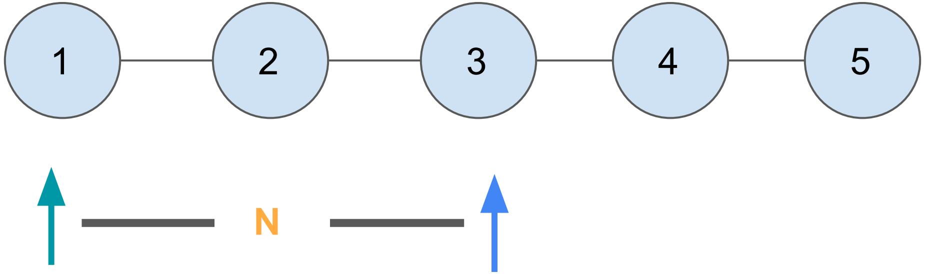
Simulation



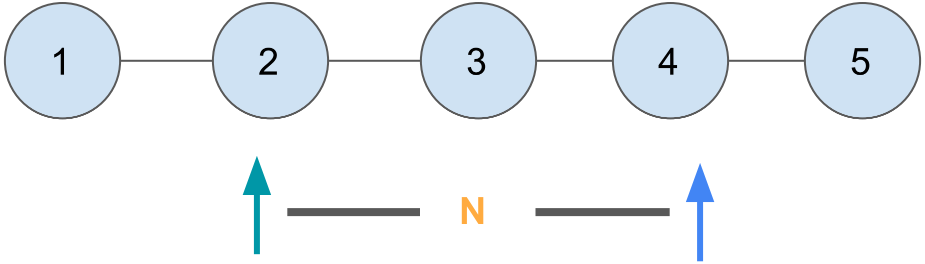
Simulation



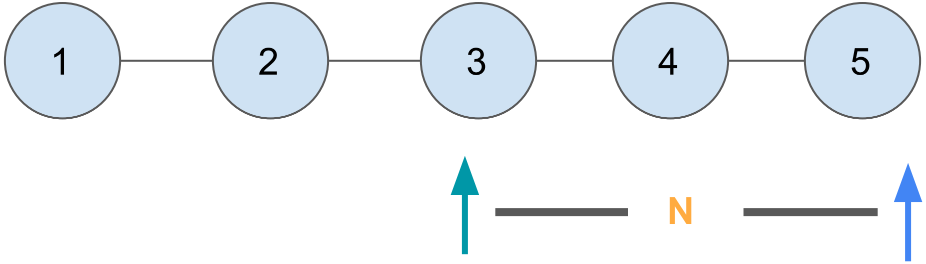
Simulation



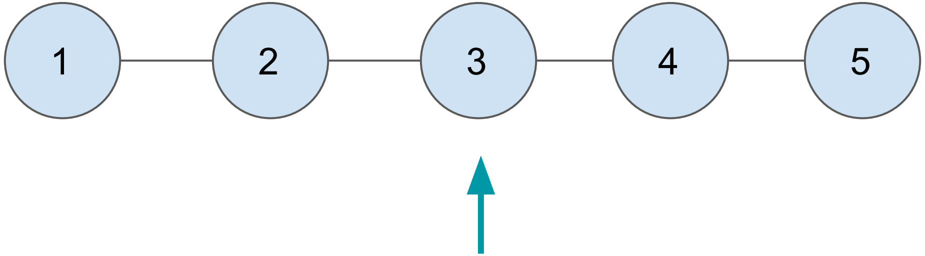
Simulation



Simulation



Simulation



Nth element from right

Implementation

```python
def nthElementFromRight(self, head, n):
    # phase I
    aheadPtr = head
    while n > 0:
        aheadPtr = aheadPtr.next
        n -= 1

    # phase II
    behindPtr = head
    while behindPtr:
        behindPtr = behindPtr.next
        aheadPtr = aheadPtr.next

    return behindPtr
```

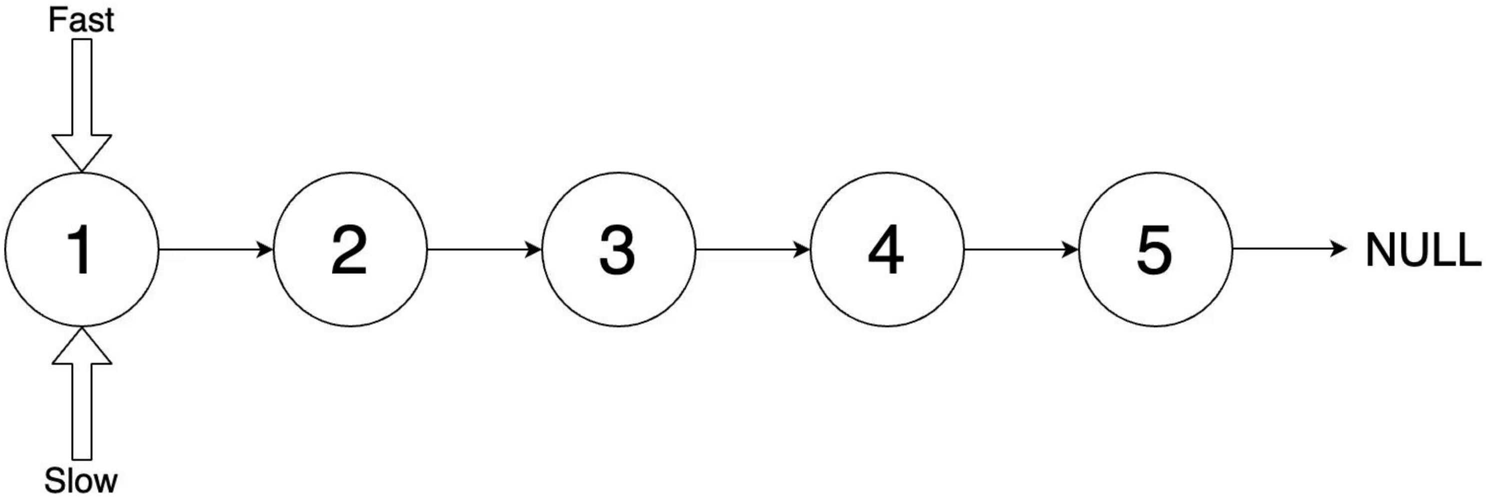
Problem II

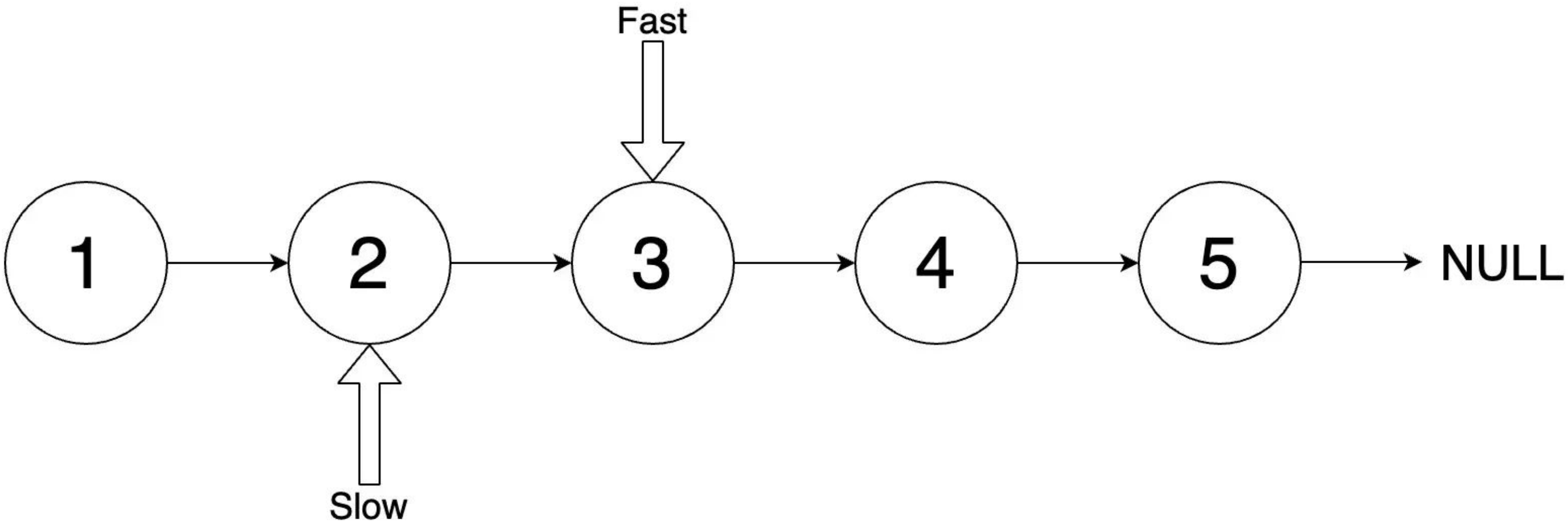
Return the middle element of a linked List.

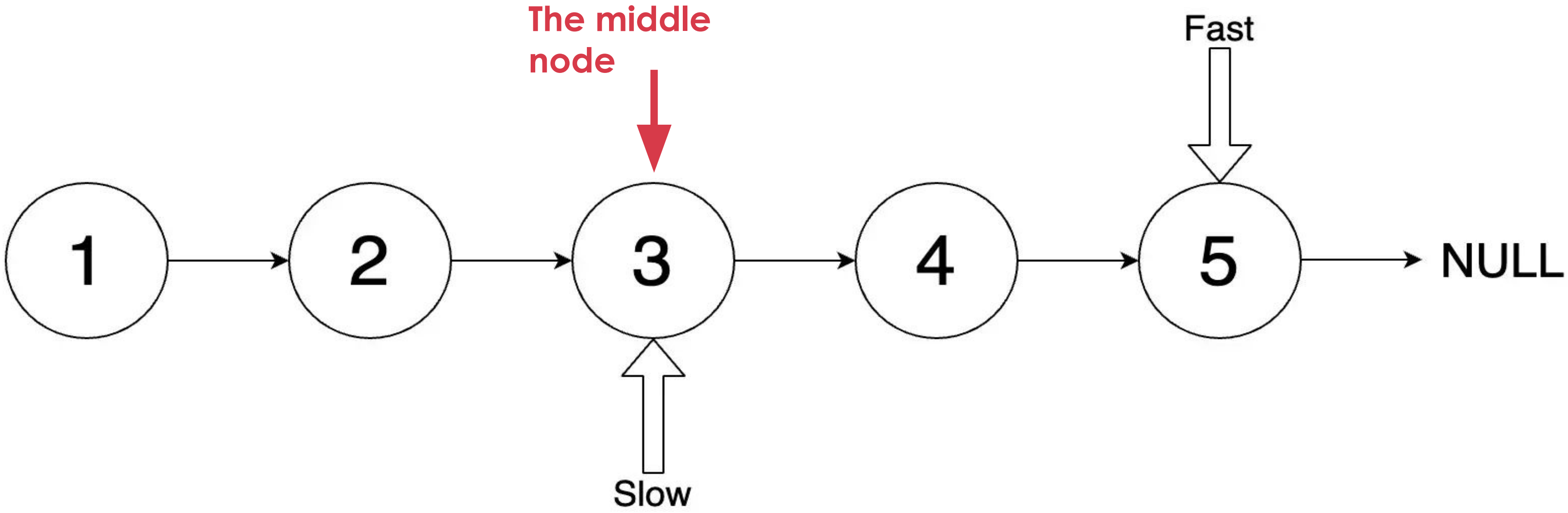
Approach : Fast and slow Pointers

- 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

Implementation

```python
def middleNode(self, head):
    slow = head
    fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
    return slow
```

Pair Programming

[Middle element of a linked list]

7:00

Floyd’s Cycle Finding Algorithm

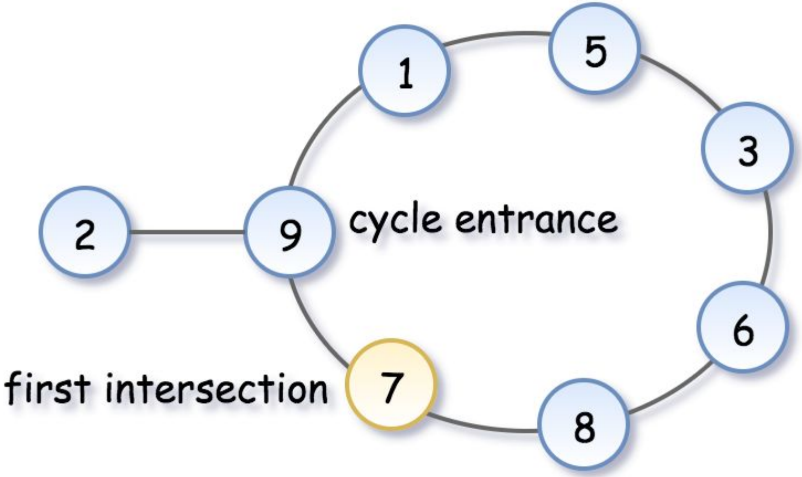
Problem

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

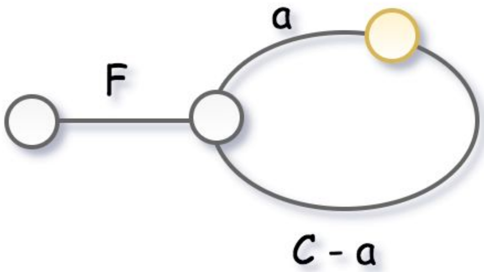
Floyd's algorithm consists of two phases and uses two pointers, usually called **tortoise** and **rabbit**.

Phase I

- **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.



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



F
a
C - a

Phase II

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

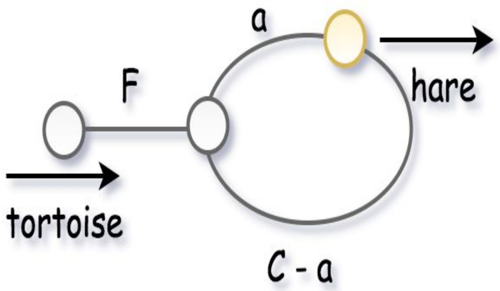
$2 \cdot d \cdot (\text{tortoise}) = d \cdot (\text{rabbit})$, implying:

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

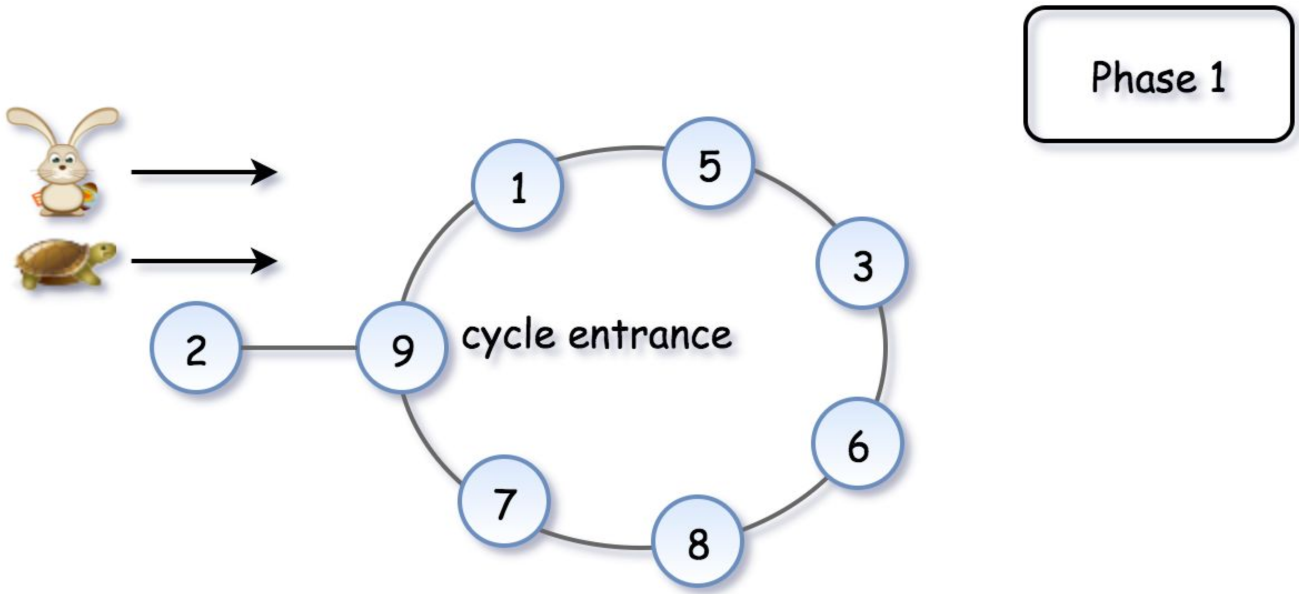
Hence the coordinate of the intersection point is

$$F + a = n \cdot C$$

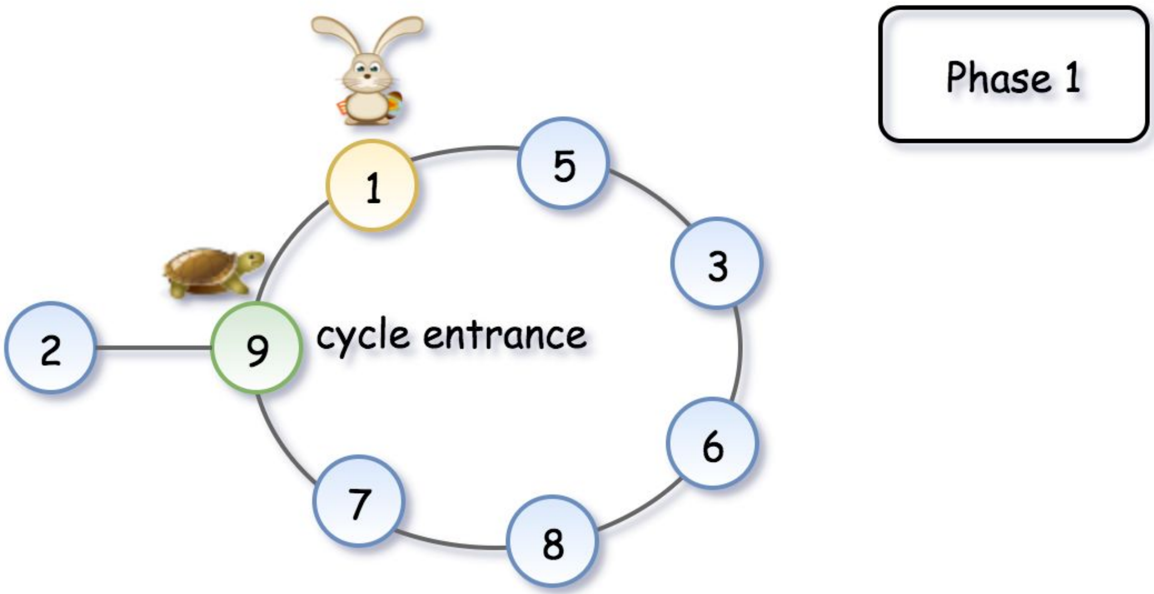
- we give the tortoise a second chance by slowing down the rabbit,
    - **tortoise = tortoise.next**
    - **rabbit = rabbit.next**
- The tortoise is back at the starting position, and the rabbit start



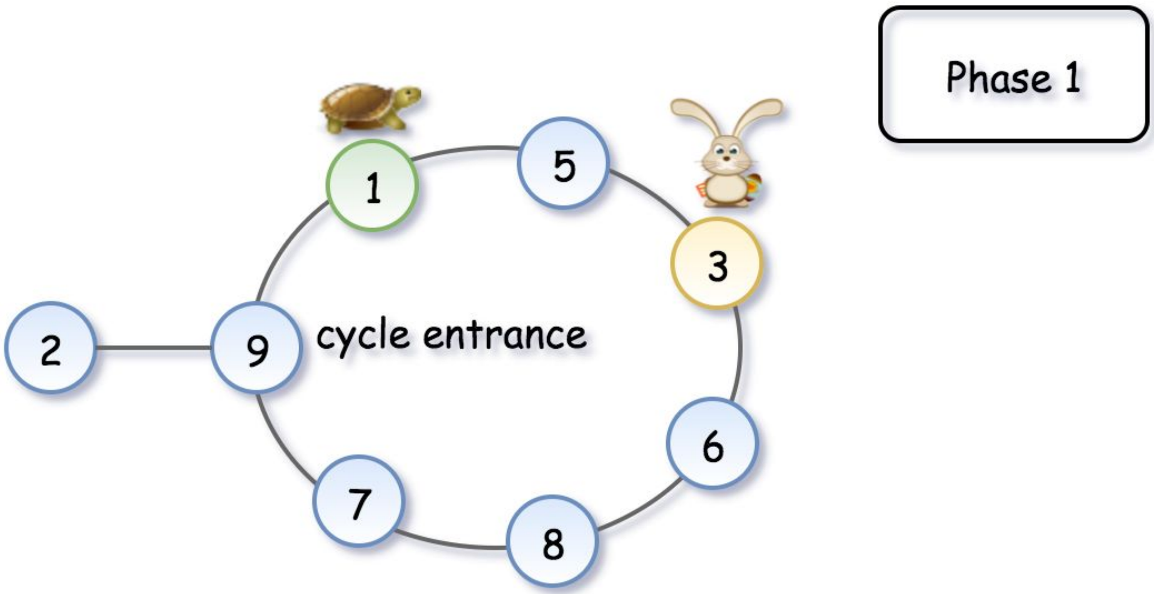
Phase 1



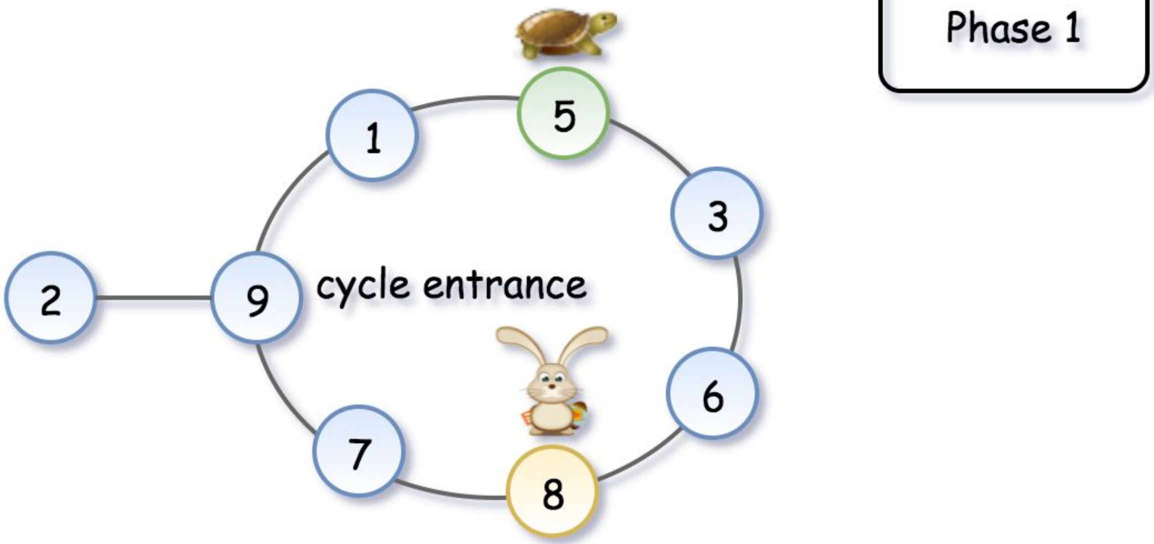
Phase 1



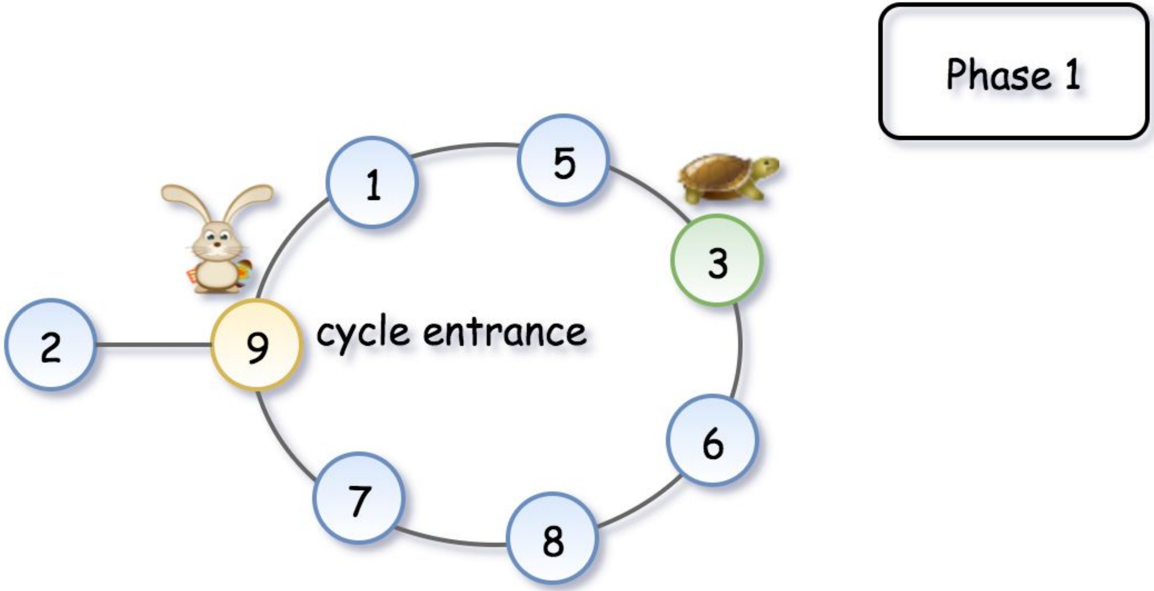
Phase 1



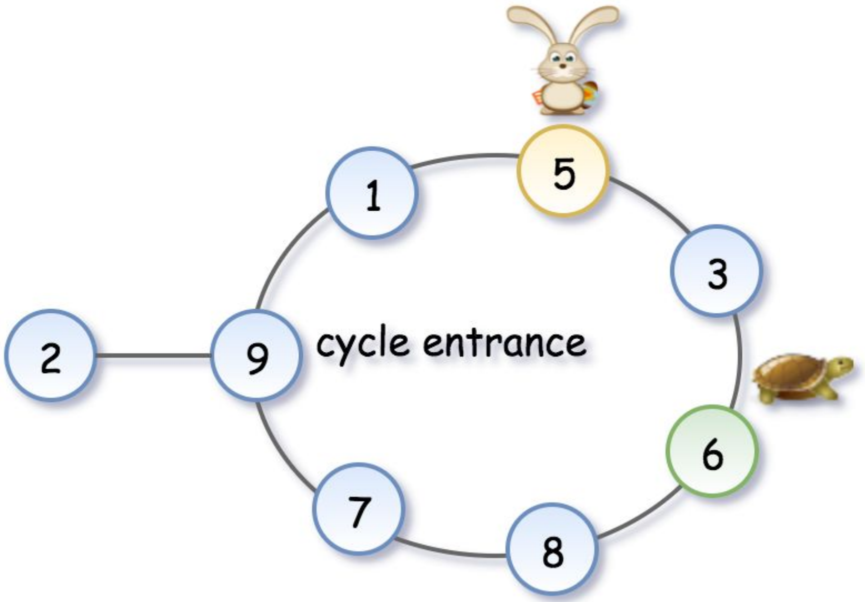
Phase 1



Phase 1

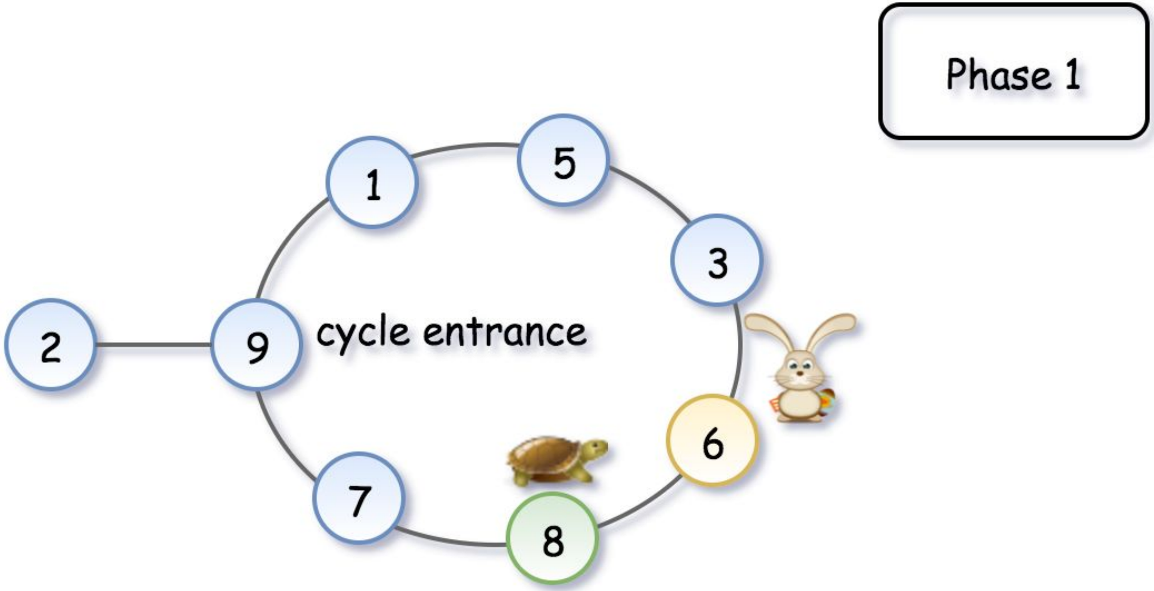


Phase 1

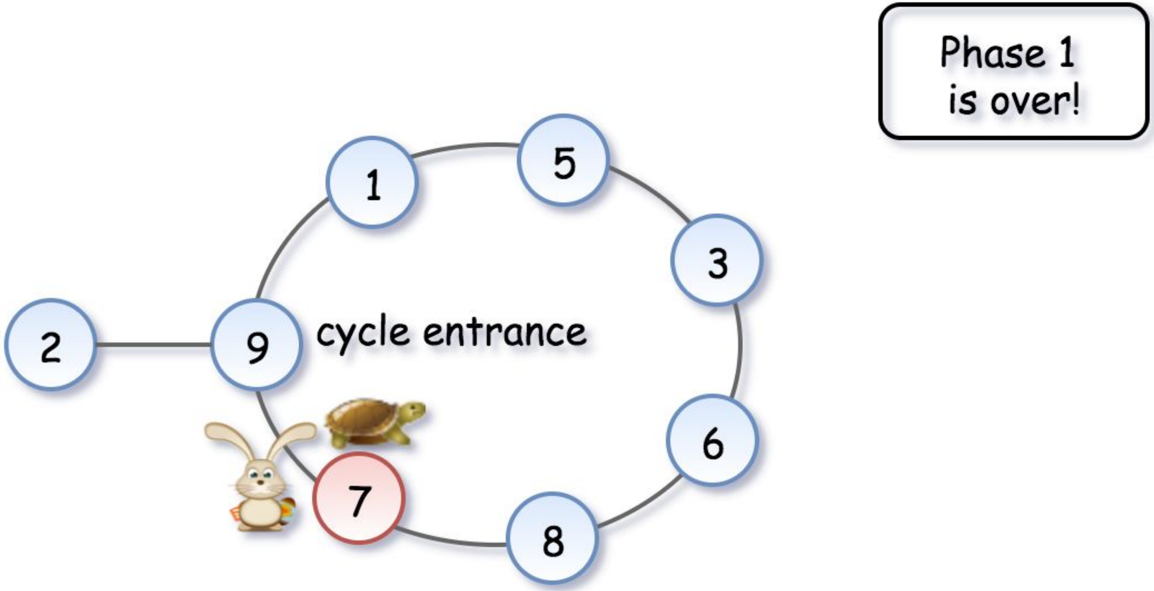


Phase 1

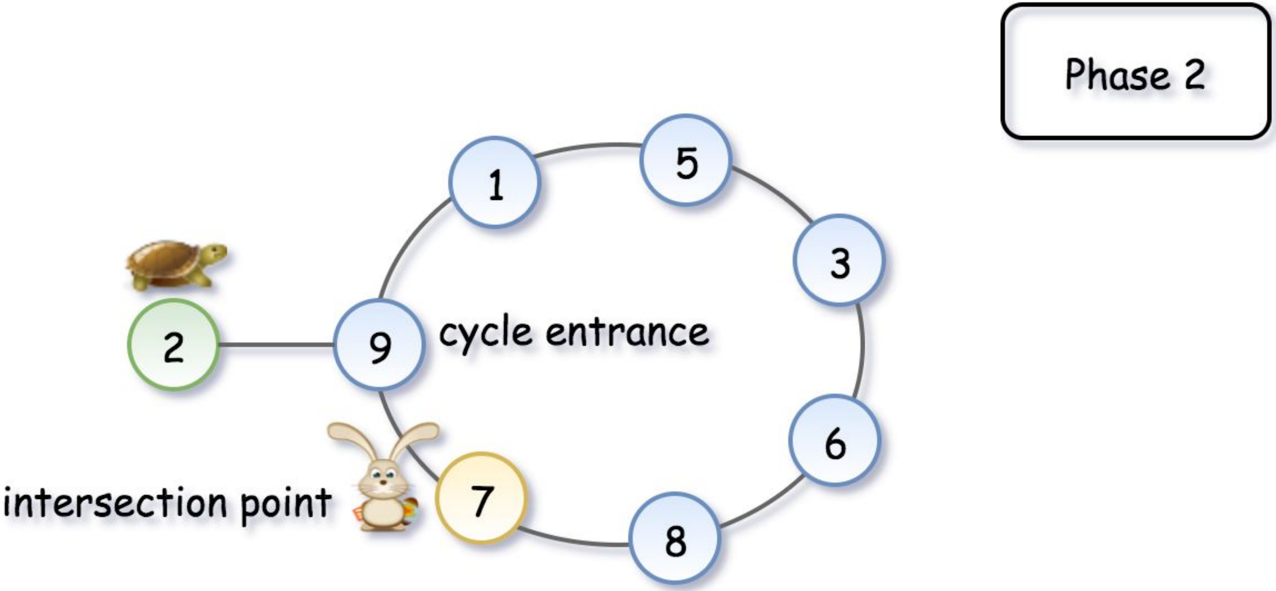
Phase 1



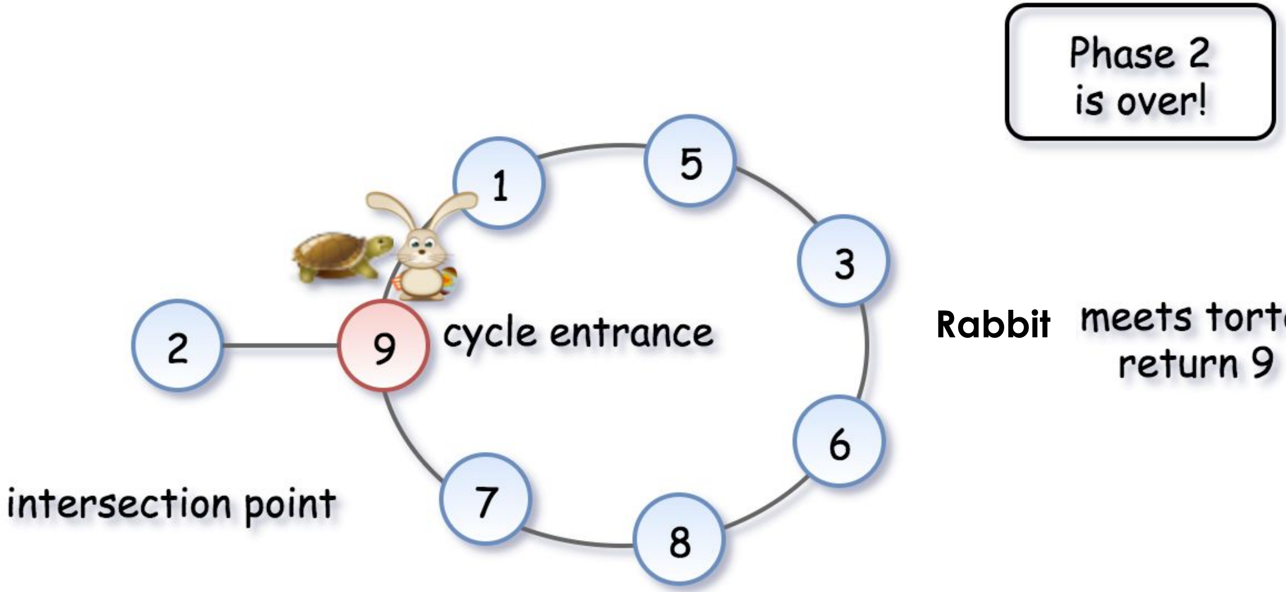
Phase 1 is over!



Phase 2



Phase 2 is over!



Rabbit meets tortoise -->
return 9

Implementation

```python
def detectCycle(self, head):
    # phase I
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            break

    # phase II
    while head != slow:
        slow = slow.next
        head = head.next

    return head
```

Pair Programming

[Linked List Cycle]


7:00

Things to pay attention

Pitfalls

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

Null Pointer Example

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

    return head.val == target
```

What if head is null?

A2SV
Africa To Silicon Valley

Null Pointer Example

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

    return head.val == target
```

Null Pointer Example

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

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

What if fast.next is null?

A2SV
Africa To Silicon Valley

Null Pointer Example

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

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

A2SV
Africa To Silicon Valley

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
```

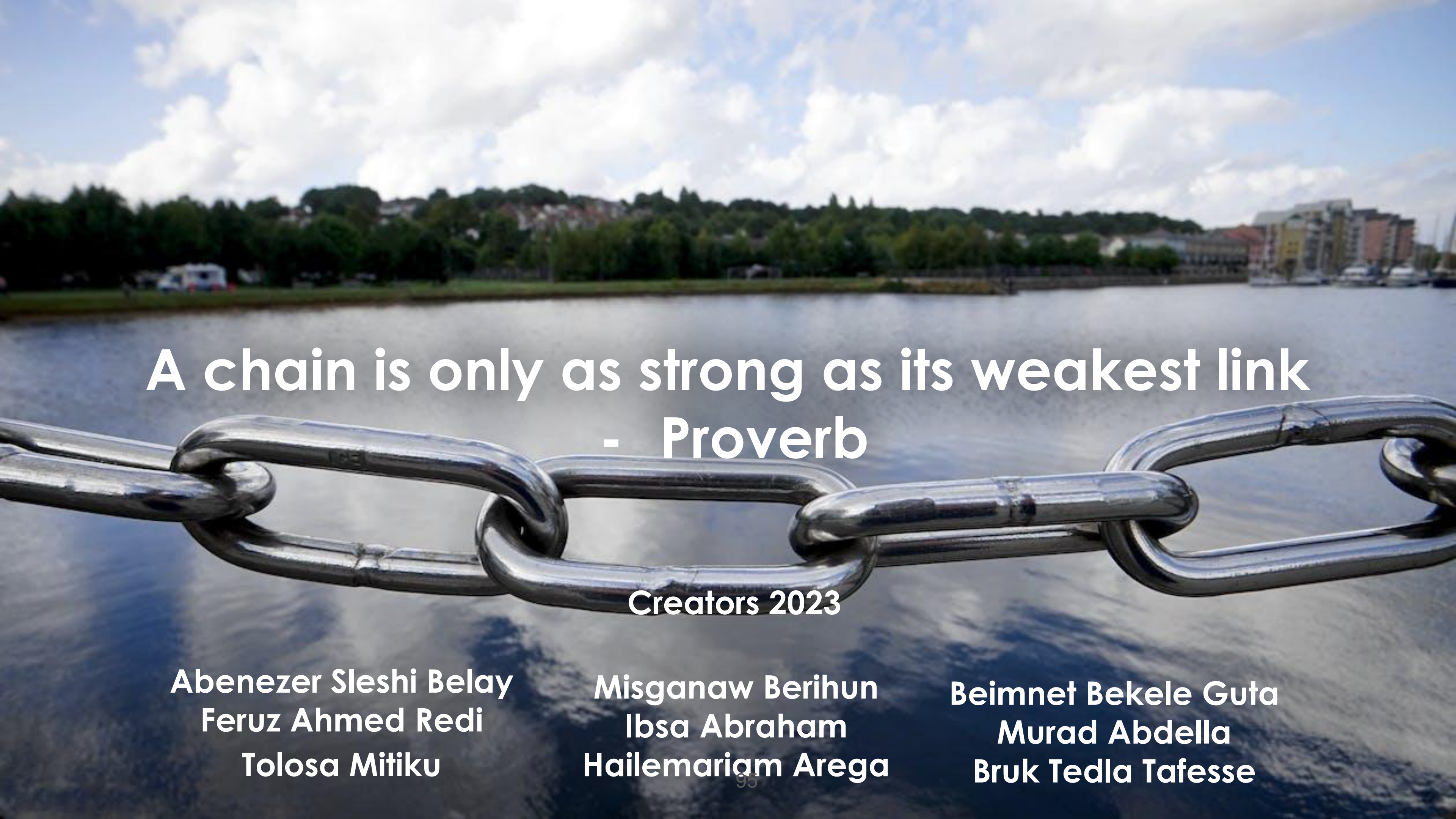
Is head the head of the modified linked list?

Practice Questions

- Design
  -
- Reverse Linked List
  - Reverse Linked Lists
  - Reverse Linked List II
  - Palindrome Linked List
- Reordering the nodes
  - Partition List
  - Rotate List
- Delete Nodes
  - Delete Node in a Linked List
  - Remove Nth Node from end of list
  - Remove duplicates from sorted list
- Sort linked lists
  a. Insertion Sort
- Two Pointers
  a. Parallel Pointers
    i. Add Two Numbers
    ii. Merge Two Sorted Lists
  b. Fast and Slow Pointers
    i. Middle of linked list
    ii. Linked List Cycle
    iii. Linked List Cycle II
- Miscellaneous
  a. Next Greater Node in a linked list
  b. Odd Even 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


A chain is only as strong as its weakest link
- Proverb
Creators 2023
Abenezer Sleshi Belay
Feruz Ahmed Redi
Tolosa Mitiku
Misganaw Berihun
Ibsa Abraham
Hailemariam Arega
Beimnet Bekele Guta
Murad Abdella
Bruk Tedla Tafesse
95