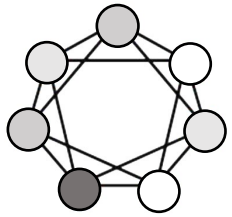
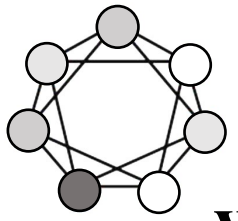


Tutorial 4
Linked List

Yaomin Wang

2025/2/25



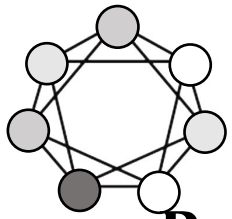
Linked Lists

What is a Linked list?

- A linked list is a linear collection of data elements whose order is not given by their physical placement in memory. Instead, each element points to the next.

Advantages:

- Do not use contiguous memory to complete dynamic operations.
- Insert and delete operations for easy insertion and removal internally
- Push and pop can be performed at both ends

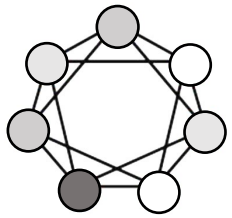


Linked Lists – Basic Operations

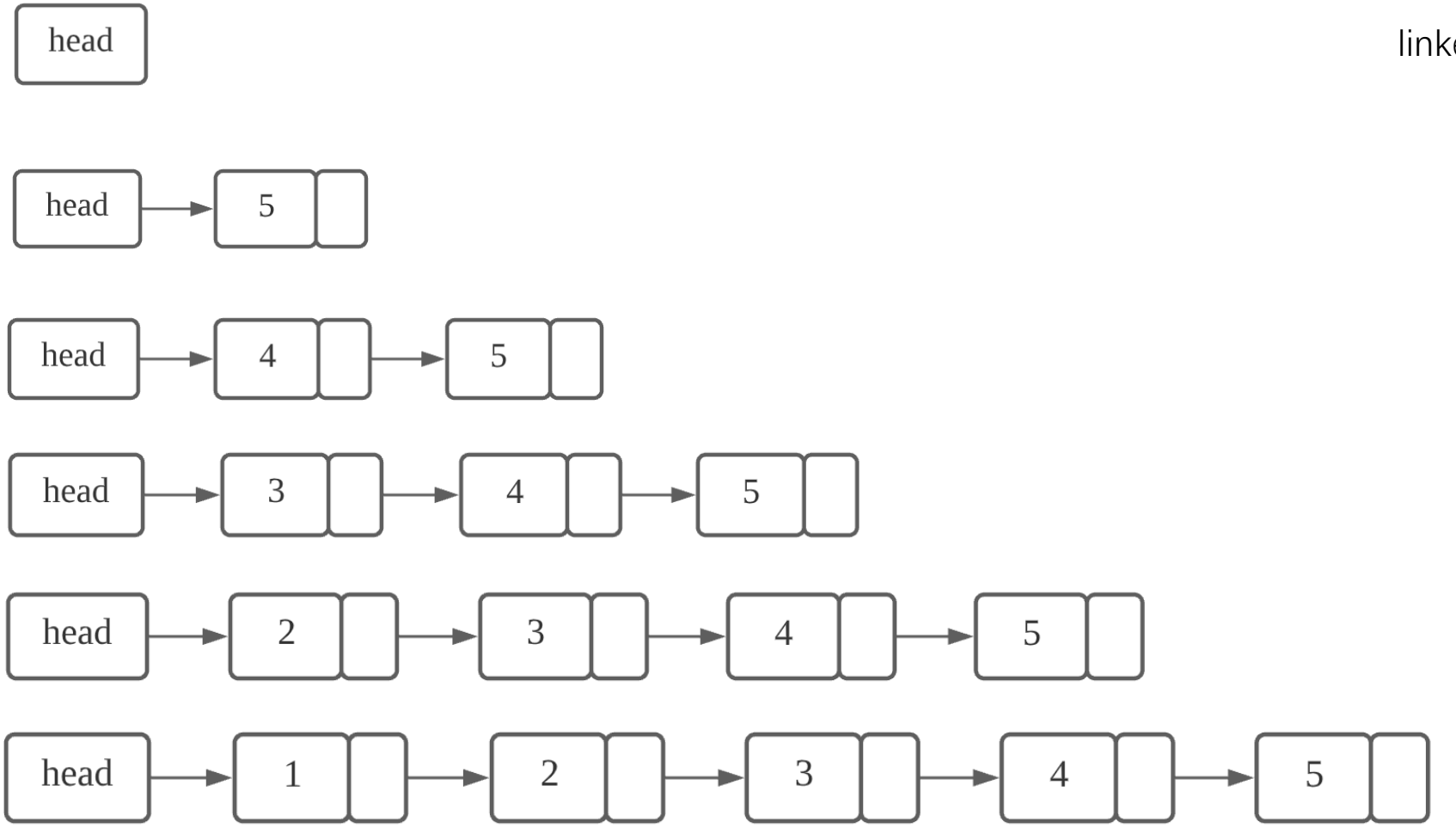
Basic operations:

- insertFirst()
- insertLast()
- deleteFirst()
- deleteLast()
- iterate()
- …

```
public void insertFirst(int data) {
    Node newNode = new Node(data);
    newNode.next = head;
    head = newNode;
}

public void iterate() {
    Node current = head;
    while (current != null) {
        // do with current ...
        current = current.next;
    }
}
```



A Toy Example




```
linkedList myLinkedList = new linkedList();
```

```
myLinkedList.insertFirst(5);
```

```
myLinkedList.insertFirst(4);
```

```
myLinkedList.insertFirst(3);
```

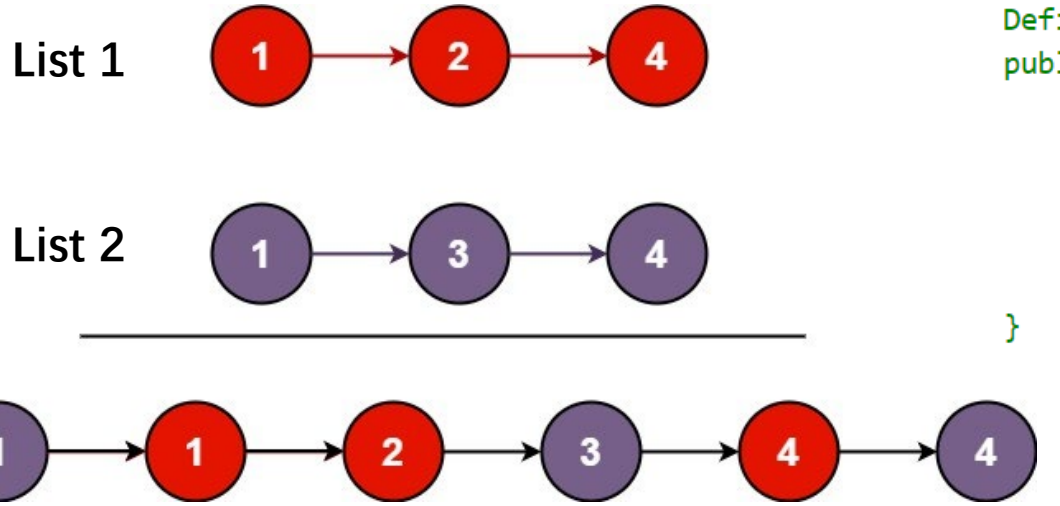
```
myLinkedList.insertFirst(2);
```

```
myLinkedList.insertFirst(1);
```

Linked Lists – Exercise 1

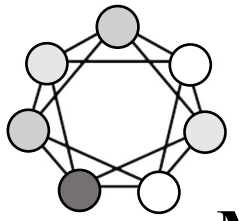
Merge Two Sorted Lists

- Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.




```
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; }
}
```

Chinese version: https://leetcode.cn/problems/merge-two-sorted-lists/
English version: https://leetcode.com/problems/merge-two-sorted-lists/



Linked Lists – Exercise 1

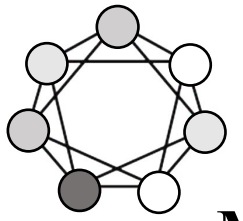
Merge Two Sorted Lists -- Example

```
Input: l1 = [1,2,4], l2 = [1,3,4]
Output: [1,1,2,3,4,4]

Input: l1 = [], l2 = []
Output: []

Input: l1 = [], l2 = [0]
Output: [0]
```

```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 mergeTwoLists(ListNode l1, ListNode l2) {

    }
}
```

Constraints:

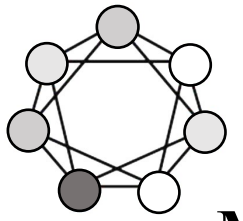
- The number of nodes in both lists is in the range `[0, 50]`.
- `-100 <= Node.val <= 100`
- Both `l1` and `l2` are sorted in **non-decreasing** order.



Linked Lists – Exercise 1

Merge Two Sorted Lists – Two Methods

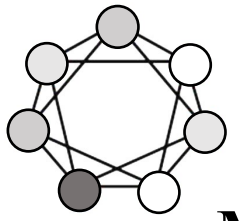
- **Iteration**: Compare the first node of these two linked lists. If the value is smaller, move to the second node and repeat comparing. Use this node with smaller value to construct a new list.
- **Recursion**: Compare the first node of these two linked list. After comparison, choose the node with smaller value to form the output list and link it to the new output of the function.



Linked Lists – Exercise 1

Merge Two Sorted Lists – Iteration Solution

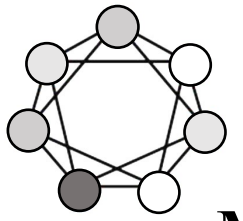
```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    ListNode head = new ListNode(0);
    ListNode current = head;
    while(list1 != null && list2 != null){
        if(list1.val < list2.val){
            current.next = list1;
            list1 = list1.next;
        }else{
            current.next = list2;
            list2 = list2.next;
        }
        current = current.next;
    }
    if(list1 == null){
        current.next = list2;
    }else{
        current.next = list1;
    }
    return head.next;
}
```



Linked Lists – Exercise 1

Merge Two Sorted Lists – Recursion Solution

```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if (list1 == null) {
        return list2;
    } else if (list2 == null) {
        return list1;
    } else if (list1.val < list2.val) {
        list1.next = mergeTwoLists(list1.next, list2);
        return list1;
    } else {
        list2.next = mergeTwoLists(list1, list2.next);
        return list2;
    }
}
```

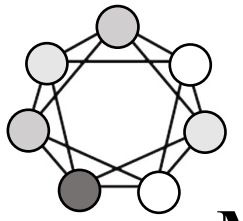


Linked Lists – Exercise 1

Merge Two Sorted Lists – Recursion Solution

```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if (list1 == null) {
        return list2;
    } else if (list2 == null) {
        return list1;
    } else if (list1.val < list2.val) {
        list1.next = mergeTwoLists(list1.next, list2);
        return list1;
    } else {
        list2.next = mergeTwoLists(list1, list2.next);
        return list2;
    }
}
```

Q: What is the time complexity of this code ?



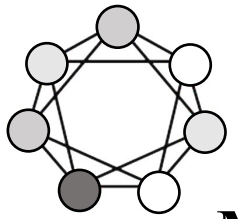
Linked Lists – Exercise 1

Merge Two Sorted Lists – Recursion Solution

```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if (list1 == null) {
        return list2;
    } else if (list2 == null) {
        return list1;
    } else if (list1.val < list2.val) {
        list1.next = mergeTwoLists(list1.next, list2);
        return list1;
    } else {
        list2.next = mergeTwoLists(list1, list2.next);
        return list2;
    }
}
```

Q: What is the time complexity of this code ?

A: $O(m+n)$, where m and n are the lengths of linked lists



Linked Lists – Exercise 1

Merge Two Sorted Lists – Recursion Solution

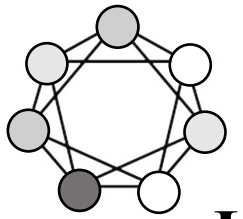
```
public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
    if (list1 == null) {
        return list2;
    } else if (list2 == null) {
        return list1;
    } else if (list1.val < list2.val) {
        list1.next = mergeTwoLists(list1.next, list2);
        return list1;
    } else {
        list2.next = mergeTwoLists(list1, list2.next);
        return list2;
    }
}
```

Q: What is the time complexity of this code ?

A: $O(m+n)$, where m and n are the lengths of linked lists

Why?

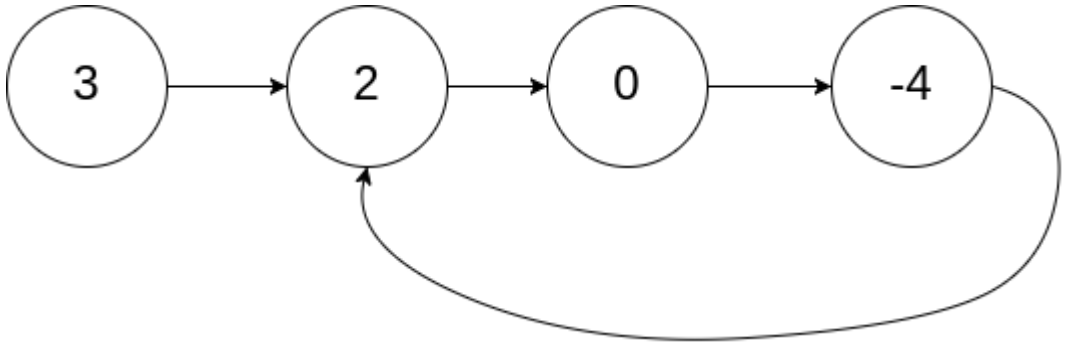
$MergeTwoLists()$ will only recursively compute each ListNode at most once!



Linked Lists – Exercise 2

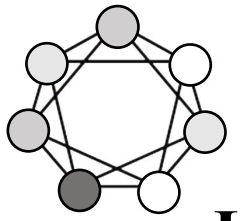
Linked list cycle

- Given a linked list(head the head of a linked list), determine if the linked list has a cycle in it.




```
Definition for singly-linked list.
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        val = x;
        next = null;
    }
}
```

Chinese version: https://leetcode.cn/problems/linked-list-cycle/
English version: https://leetcode.com/problems/linked-list-cycle/



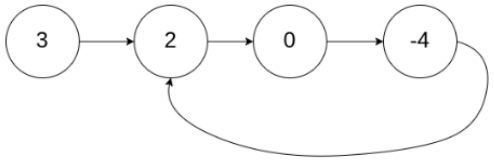
Linked Lists – Exercise 2

Linked list cycle -- Example

- Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

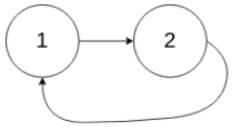
  Note that pos is not passed as a parameter.

Example 1:




```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list,
where the tail connects to the 1st node (0-indexed).
```

Example 2:



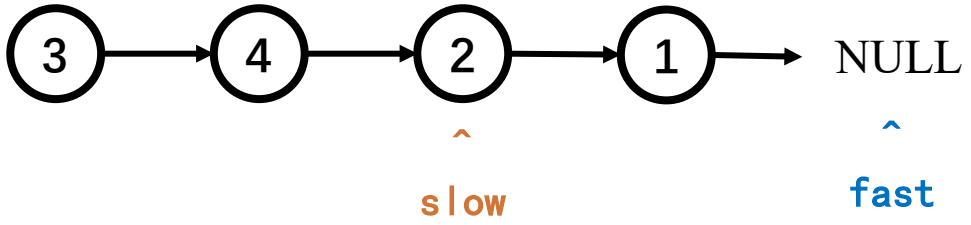

```
Input: head = [1,2], pos = 0
Output: true
Explanation: There is a cycle in the linked list,
where the tail connects to the 0th node.
```

Example 3:

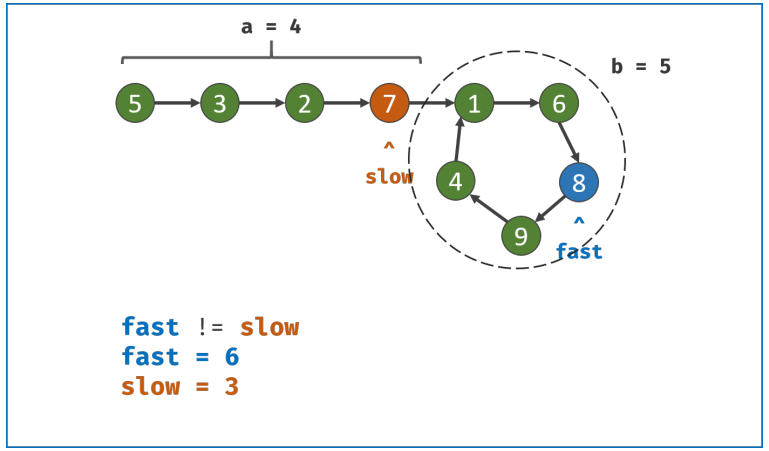

```
Input: head = [1], pos = -1
Output: false
Explanation: There is no cycle in the linked list.
```

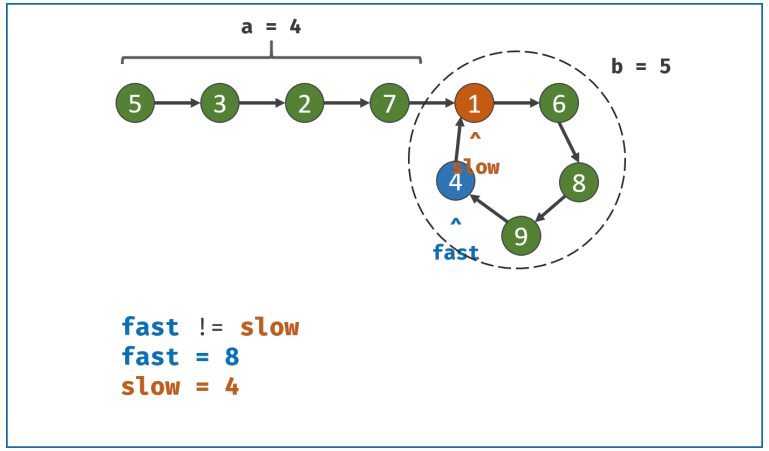
Linked Lists – Exercise 2

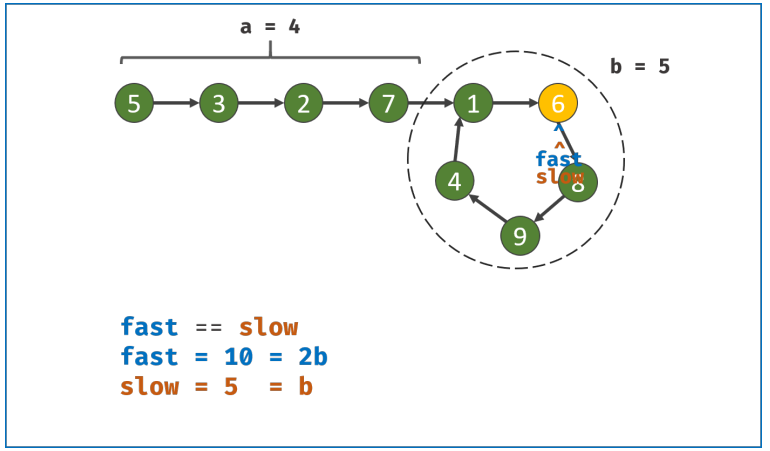
Fast & Slow pointers

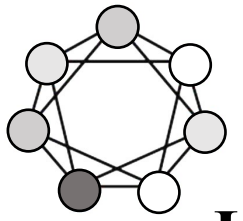


- Fast pointer go 2 steps every time;
- Slow pointer go 1 step every time.
- If there is no cycle: Fast point reaches the end (NULL)
- If there is a cycle: they will **meet eventually**!









Linked Lists – Exercise 2

Linked list cycle Solution – C++ & Java

```cpp
bool hasCycle(ListNode *head) {
    ListNode *fast = head;
    ListNode *slow = head;
    while(fast != NULL && slow != NULL){
        fast = fast->next;
        if(fast != NULL){
            fast = fast->next;
        }
        if(fast == slow)
            return true;
        slow = slow->next;
    }
    return false;
}
```

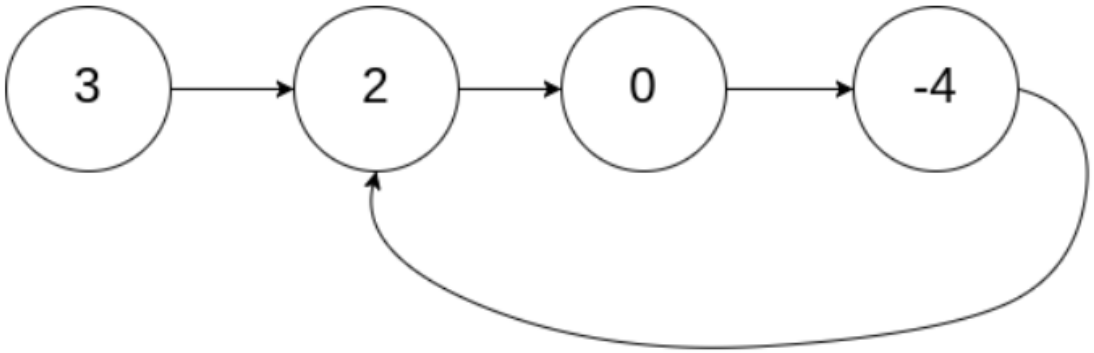
```java
public boolean hasCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;
    while(fast != null && slow != null){
        fast = fast.next;
        if(fast != null){
            fast = fast.next;
        }
        if(fast == slow)
            return true;
        slow = slow.next;
    }
    return false;
}
```

Linked Lists – Exercise 3

Linked list cycle – Find the node where the cycle begin

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

Example 1:



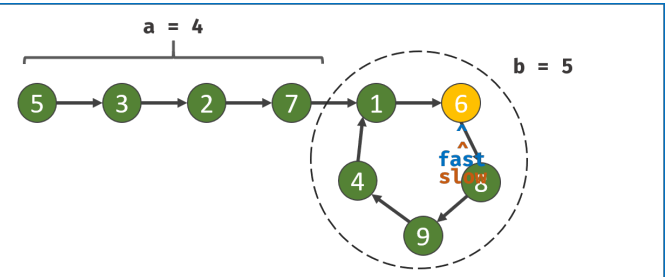

```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list,
where tail connects to the second node.
```

Chinese version: https://leetcode.cn/problems/linked-list-cycle-ii/
English version: https://leetcode.com/problems/linked-list-cycle-ii/

Linked Lists – Exercise 3

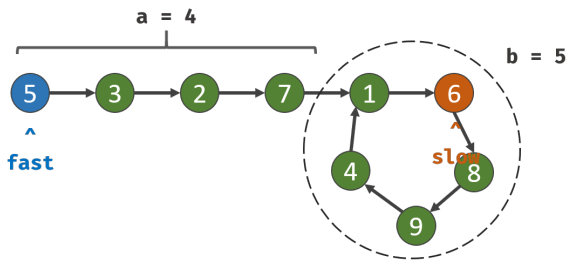
Fast & Slow pointers – analysis

- Let's use s(slow) and f(fast) to represent the number of nodes passed by the pointers **when they meet**
- $f = 2s;\ f = s + nb$; (fast passed n more circles than slow)
- Then we have $f = 2nb, s = nb$. We just need to make the slow pointer **take another step a** to reach the node where the cycle begins. In this way, $a + nb$ exactly indicates starting from scratch and walking n circles.
- How to take step a exactly? – Use another pointer from scratch.

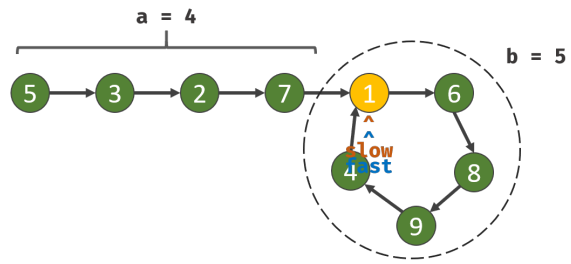



```
fast == slow
fast = 10 = 2b
slow = 5  = b
```

First Meet

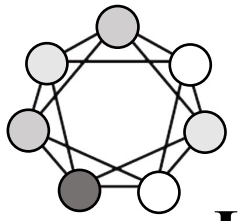



```
fast != slow
fast = 0
slow = 5  = b
```




```
fast != slow
fast = 4 = a
slow = 9 = a + b
```

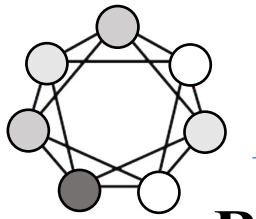
Second Meet



Linked Lists – Exercise 3

Linked list cycle – Find the node where the cycle begin Solution

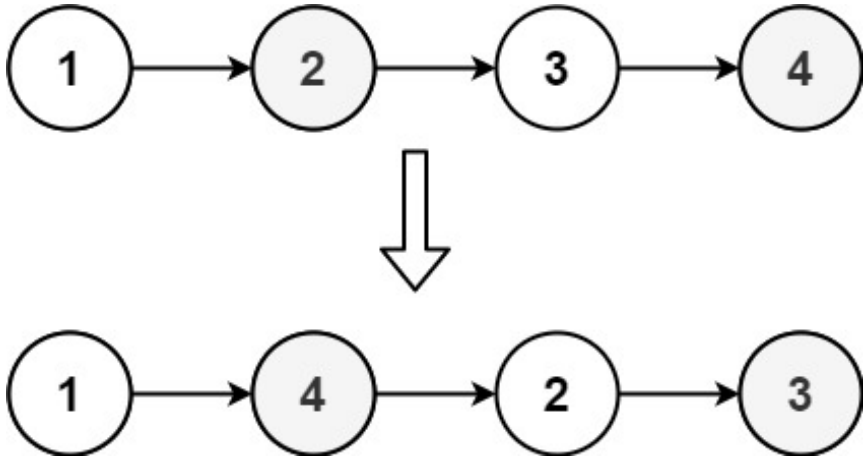
```java
public class Solution {
    public ListNode detectCycle(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;
        while(true){
            if(fast == null || fast.next == null)
                return null;
            fast = fast.next.next;
            slow = slow.next;
            if(fast == slow)
                break;
        }
        fast = head;
        while(fast != slow){
            fast = fast.next;
            slow = slow.next;
        }
        return slow;
    }
```



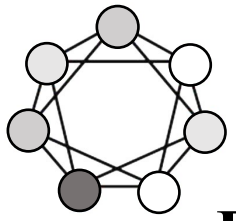
Linked Lists – Exercise 4

Reorder List – Node Exchange Operation

- You are given the head of a singly linked-list. The list can be represented as:
- L0 → L1 → … → Ln - 1 → Ln
- Reorder the list to be on the following form:
- L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …



Chinese version: https://leetcode.cn/problems/reorder-list/
English version: https://leetcode.com/problems/reorder-list/



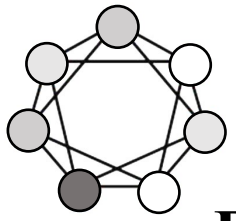
Linked Lists – Exercise 4

Reorder List – Node Exchange Operation

- **Find the middle node**: fast & slow points
- **Reverse the second half**: store the previous node
- **Merge two lists**: similar to exercise 1

Reverse
v

- L0 → L1 → … → **Ln/2** → … → *Ln - 1* → *Ln*
- L0 → Ln → L1 → Ln - 1 → L2 → Ln - 2 → …



Linked Lists – Exercise 4

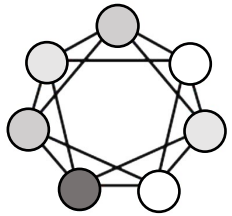
Reorder List Solution – Java version

- We use mid to represent the slow pointer

```java
class Solution {
    public void reorderList(ListNode head) {
        if(head == null)
            return;
        ListNode mid=head;
        ListNode fast=head;
        while(fast.next != null && fast.next.next != null){
            mid = mid.next;
            fast = fast.next.next;
        }
        ListNode l1 = head;
        ListNode l2 = mid.next;
        mid.next = null;
        l2 = reverseList(l2);
        mergeList(l1, l2);
    }
```

```java
public ListNode reverseList(ListNode head) {
    ListNode prev = null;
    ListNode curr = head;
    while (curr != null) {
        ListNode nextTemp = curr.next;
        curr.next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}
```

```java
public void mergeList(ListNode l1, ListNode l2) {
    ListNode l1_tmp;
    ListNode l2_tmp;
    while (l1 != null && l2 != null) {
        l1_tmp = l1.next;
        l2_tmp = l2.next;
        l1.next = l2;
        l1 = l1_tmp;
        l2.next = l1;
        l2 = l2_tmp;
    }
}
```



Thank You!