CSC3100 Data Structures
Tutorial 4

Lai Wei (USTF, SDS, 120090485)

laiwei1@link.cuhk.edu.cn

https://github.com/I-am-Future

https://i-am-future.github.io/

Contents

- Linked Lists (Prof. Fang's lec 7, Prof. Yu's lec 6)

- 2 x Coding Questions

- Q & A

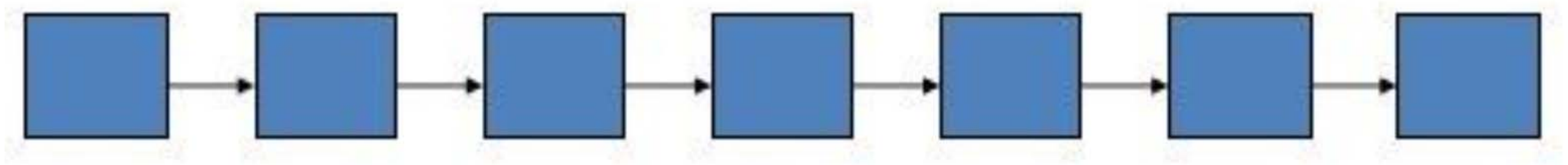
1. Linked Lists (1)

- A list represents a countable number of <u>ordered</u> <u>values</u>, where the same value may occur more than once.

- 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

1. Linked Lists (2)

- 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.



1. Linked Lists (3)

- It is a data structure consisting of a collection of nodes which together represent a sequence. In its most basic form, each node contains: data, and a reference (in other words, a link) to the next node in the sequence.

```
class Node {
public:
    int data;
    Node* next;
};
```

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

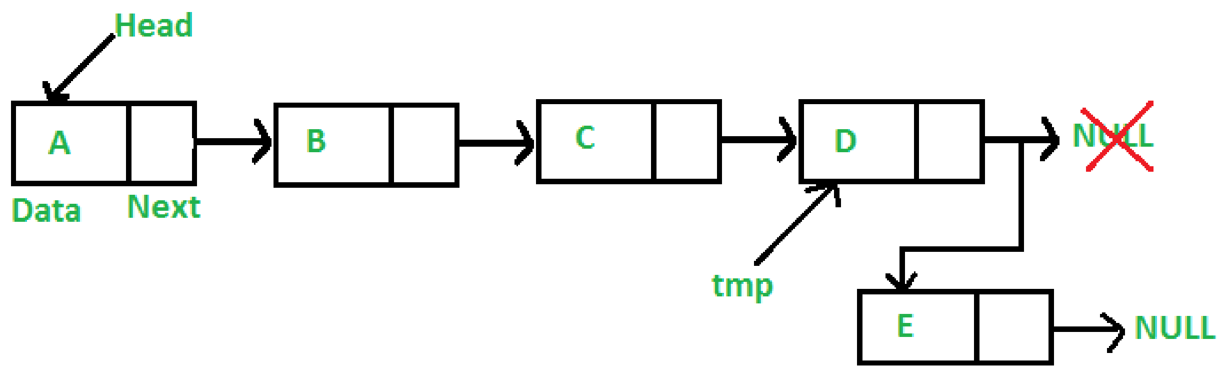
```
class Node {
    int data;
    Node next;
}
```

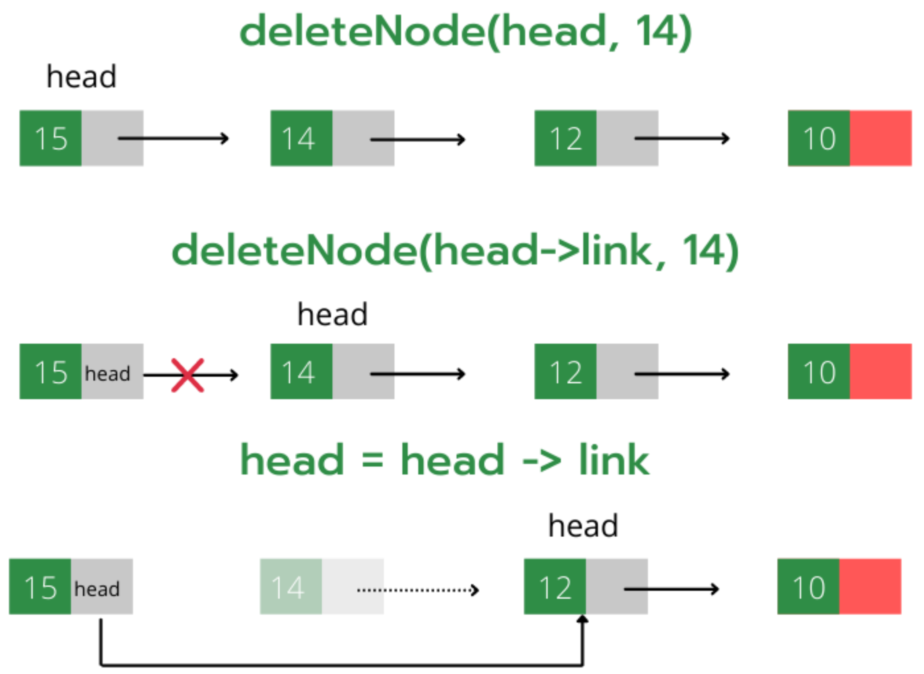
Example of a “Node” for practical use

```c
typedef struct PSNode{
    /* Process info field */
    char              name[64];        // process name
    pid_t             pid;             // this pid
    pid_t             ppid;            // parent pid
    ProcType          type;            // process type

    /* Data structure field (as a list or tree) */
    struct PSNode*    list_next;       // As linked list, next node pointer
    struct PSNode*    thischild_head;  // As tree, this children's list head
    int               thischild_size;  // As tree, len of this children's list
    struct PSNode*    parentchild_next;// As tree, parent children's list next

    /* Helper var field */
    int               compactNum;
    bool              compactSubtree;  // true if a subtree is compacted
} PSNode;
```

1. Linked Lists (3)

- This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration.





1. Linked Lists (4)

- Basic operations:
  - insertFirst()
  - insertLast()
  - deleteFirst()
  - deleteLast()
  - iterate()

Just think carefully with procedures!

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

1. Linked Lists (5)

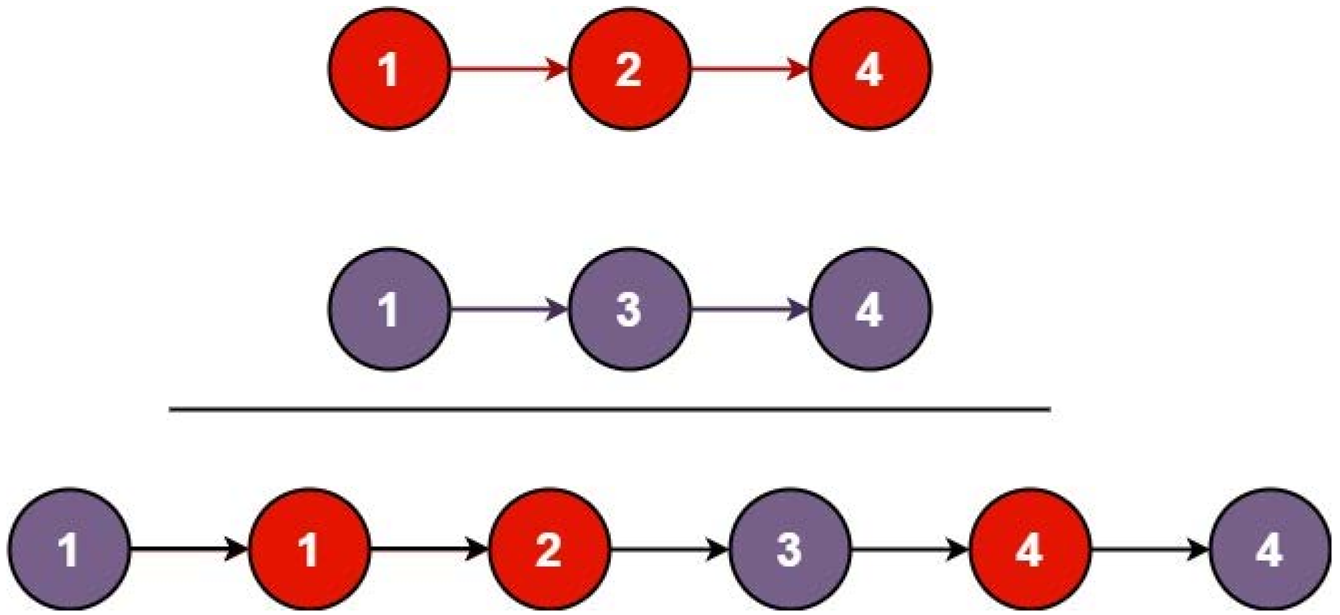
- Different implementations.
- Some Linked Lists only remember the “head”, while other implementation of Linked Lists remember both “head” and “tail”. Therefore, complexities are different n two cases.

	“head”	“head” and “tail”
• insertFirst()	O(1)	O(1)
• insertLast()	O(n)	O(1)
• deleteFirst()	O(1)	O(1)
• deleteLast()	O(n)	O(n)

2. Exercise 1

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

- You are given the heads of two sorted linked lists list1 and list2.
- Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.
- Return the head of the merged linked list.



2. Exercise 1

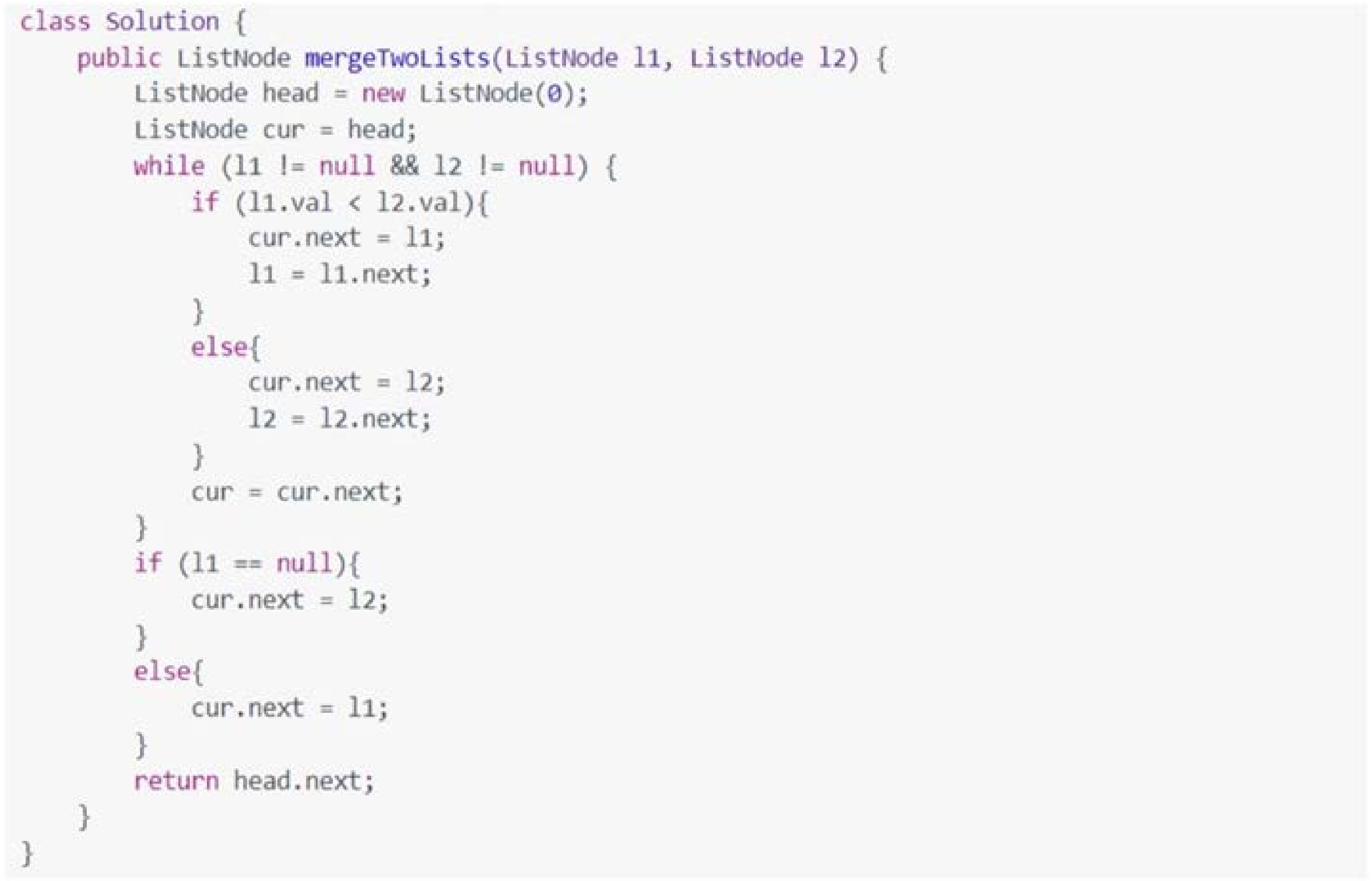
Chinese version:
https://leetcode.cn/problems/merge-two-sorted-lists/
English version:
https://leetcode.com/problems/merge-two-sorted-lists/

- This is a simple example for linked lists.
- **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.

2. Exercise 1

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

Ex. 1 Merge Two Sorted Lists - iteration

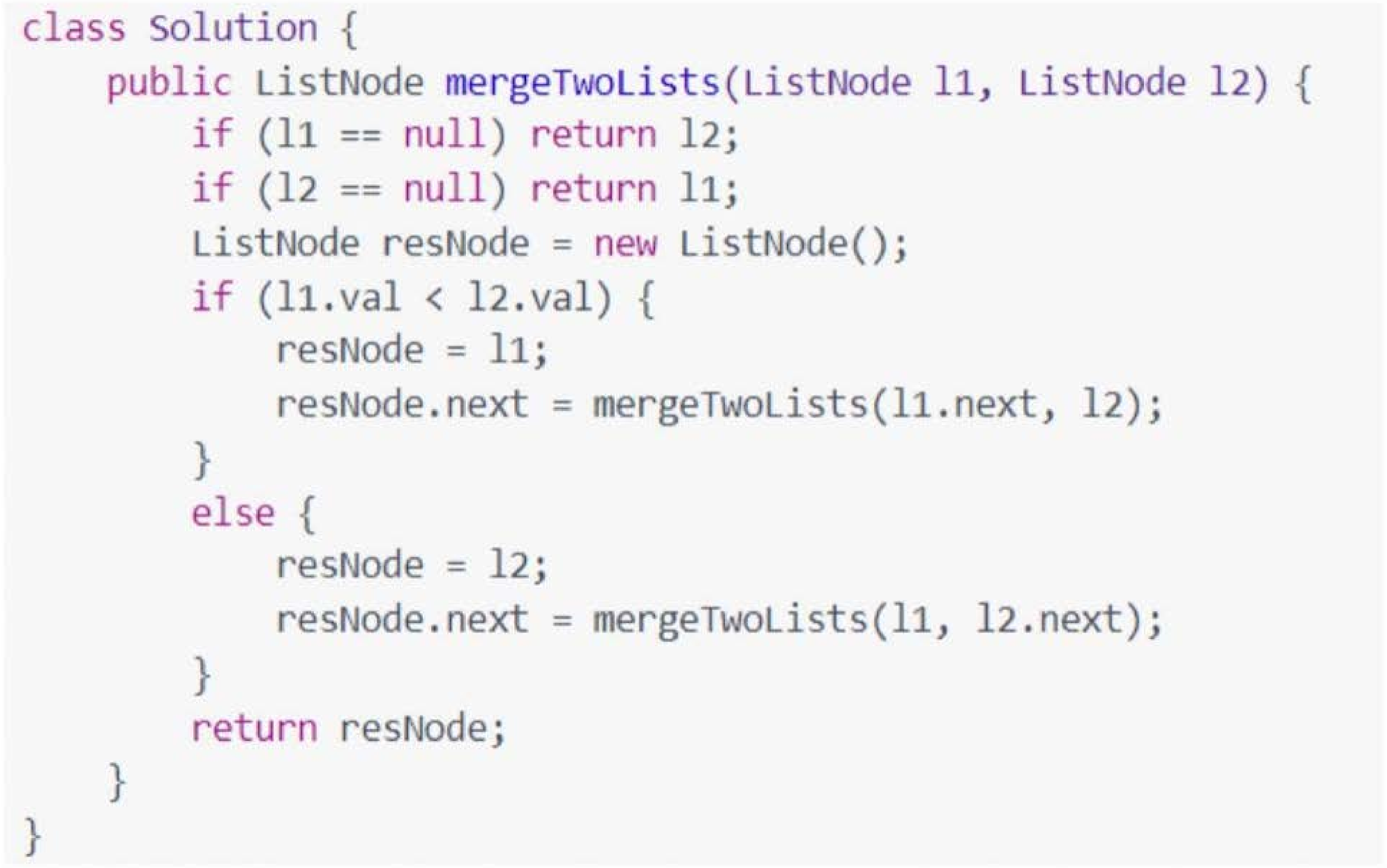


```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        ListNode head = new ListNode(0);
        ListNode cur = head;
        while (l1 != null && l2 != null) {
            if (l1.val < l2.val){
                cur.next = l1;
                l1 = l1.next;
            }
            else{
                cur.next = l2;
                l2 = l2.next;
            }
            cur = cur.next;
        }
        if (l1 == null){
            cur.next = l2;
        }
        else{
            cur.next = l1;
        }
        return head.next;
    }
}
```

2. Exercise 1

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

Ex. 1 Merge Two Sorted Lists - recursion

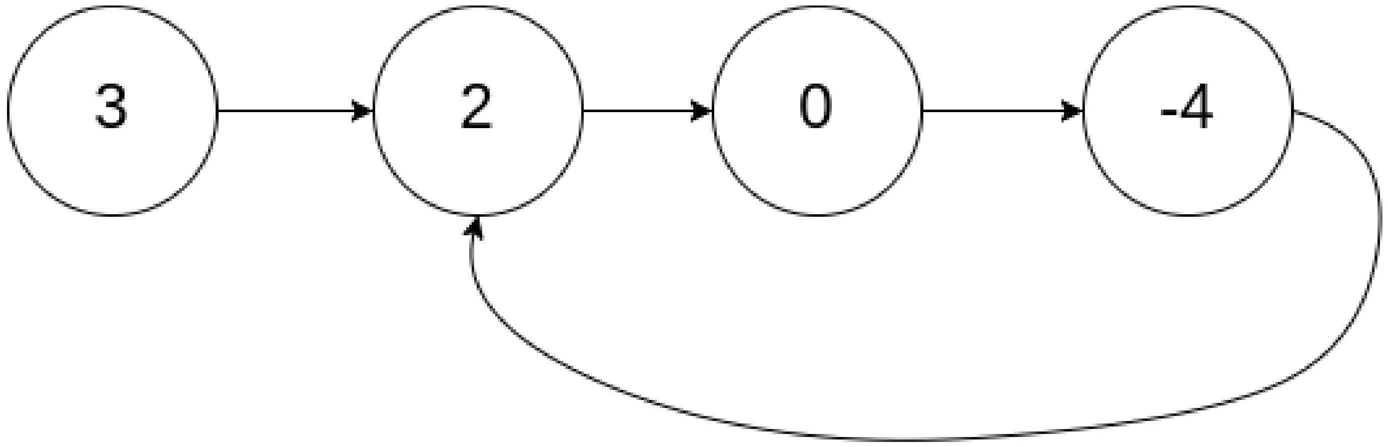


```java
class Solution {
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;
        ListNode resNode = new ListNode();
        if (l1.val < l2.val) {
            resNode = l1;
            resNode.next = mergeTwoLists(l1.next, l2);
        }
        else {
            resNode = l2;
            resNode.next = mergeTwoLists(l1, l2.next);
        }
        return resNode;
    }
}
```

2. Exercise 2

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

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



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

2. Exercise 2

- 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!

Visualization:
【算法动画图解：leetcode.141.环形链表（有环）】
https://www.bilibili.com/video/BV12M4y1d7QP/?share_source=copy_web&vd_source=0c067fd928325f3684e2a932b9539e44

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

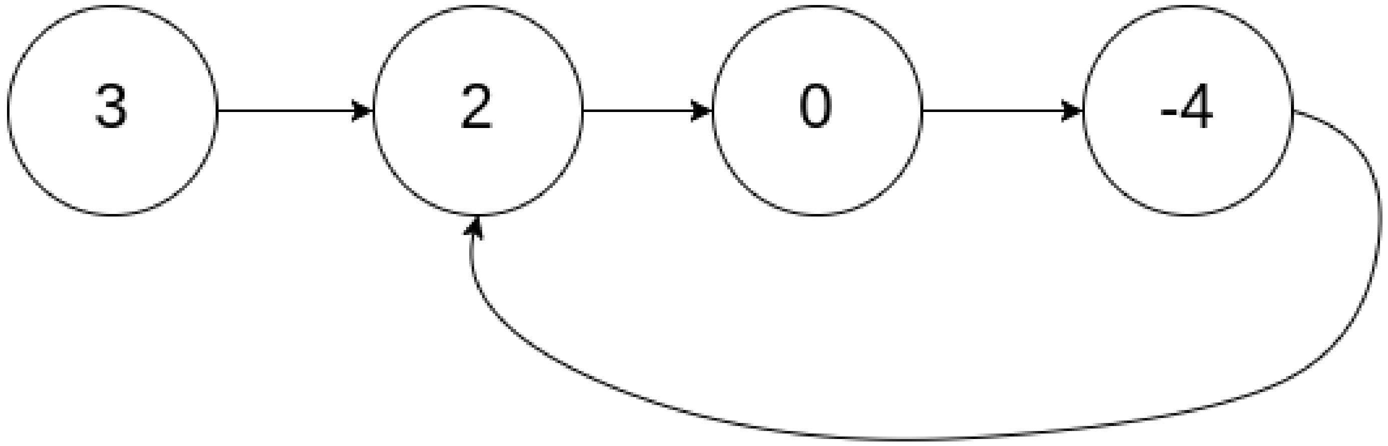
2. Exercise 2

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

2. Exercise - bonus

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

- Given head, the head of a linked list, determine if the linked list has a cycle in it.
- Given the head of a linked list, return the node where the cycle begins. If there is no cycle, return null.



3. Q & A