Special Linked Structures

CS 112 - Recitation 4

Ashwin Haridas
ah1058@rutgers.edu

Let’s review: CLL and DLL

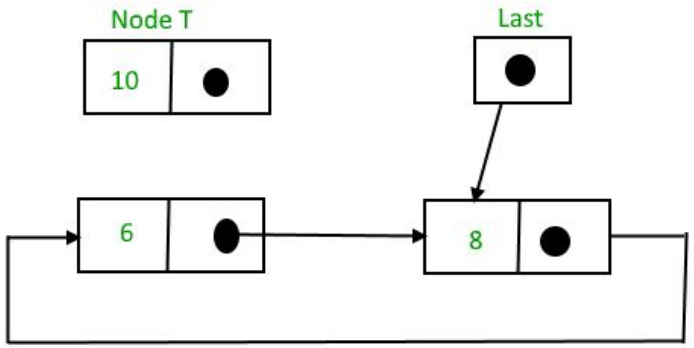
Warm-Up 1 Circular Linked List - Insert to the front

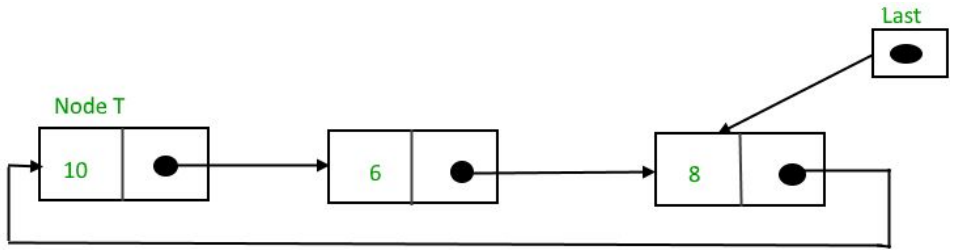
Given the head of a circular linked list, insert a node given data to the front (after last node). Use the following method signature:

```
public static Node addBegin(Node last, int data)
```

Think: What are the cases that need to be covered?

.





Warm-Up 1 Solution

```
public static Node addBegin(Node last, int data)
{
    if (last == null)
        return addToEmpty(last, data);

    // Creating a node dynamically
    Node temp = new Node();

    // Assigning the data
    temp.data = data;

    // Adjusting the links
    temp.next = last.next;
    last.next = temp;

    return last;
}
```

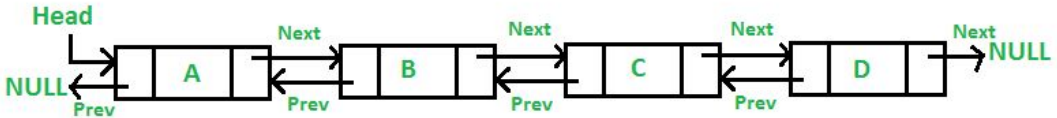
Warm-Up 2 Doubly Linked List - Insert After

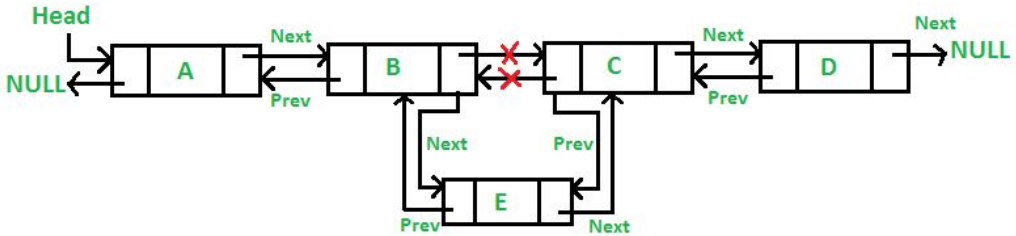
Assuming following DLL structure:

```
public class DLL{

  Node head;

  int data;

  Node prev;

  Node next;

 public Node (int d){

  this.data = d;

}
```

Write a method to insert a node after a target node in DLL using following method signature:

```
public void InsertAfter(Node prev_Node, int
```





Warm-Up 2 Solution

```
/* Given a node as prev_node, insert a new node after the given node */
public void InsertAfter(Node prev_Node, int new_data)
{

    /*1. check if the given prev_node is NULL */
    if (prev_Node == null) {
        System.out.println("The given previous node cannot be NULL ");
        return;
    }

    /* 2. allocate node
    * 3. put in the data */
    Node new_node = new Node(new_data);

    /* 4. Make next of new node as next of prev_node */
    new_node.next = prev_Node.next;

    /* 5. Make the next of prev_node as new_node */
    prev_Node.next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node.prev = prev_Node;

    /* 7. Change previous of new_node's next node */
    if (new_node.next != null)
        new_node.next.prev = new_node;
}
```

Q1 Implement Queue with CLL

Write a Queue implementation that uses a circular linked list, which is the same as a linked list except that no links are null and the value of last.next is first whenever the list is not empty.

Keep only one Node instance variable (last).

Implement Queue with CLL: Solution

```
public void enqueue(Item item) {
    if (isEmpty()) {
        last = new Node();
        last.item = item;
        last.next = last;
    } else {
        Node node = new Node();
        node.item = item;

        if (size == 1) {
            last.next = node;
            node.next = last;
        } else {
            node.next = last.next;
            last.next = node;
        }
        last = node;
    }
    size++;
}
```

```
public Item dequeue() {
    if (isEmpty()) {
        return null;
    }

    Item item;

    if (size == 1) {
        item = last.item;
        last = null;

    } else {
        item = last.next.item;
        last.next = last.next.next;
    }
    size--;

    return item;
}
```

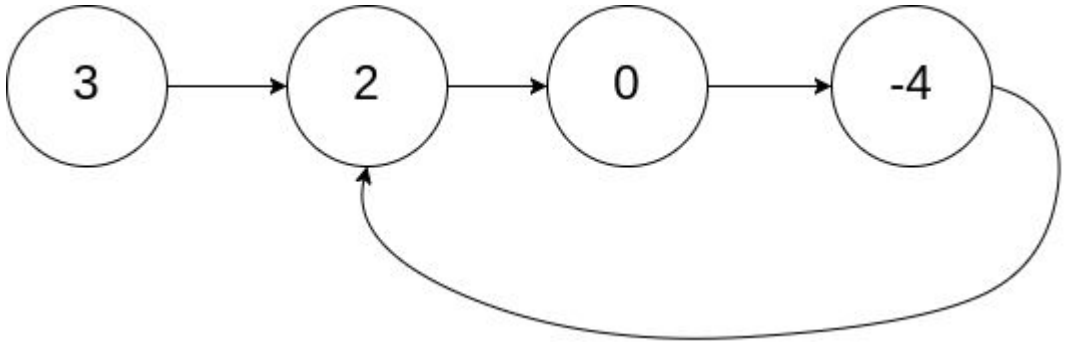
Q2 Linked List Cycle I

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

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. 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**.

Return true *if there is a cycle in the linked list*. Otherwise, return false.

Hint: Floyd's Algorithm!



```
class ListNode {
        int val;
        ListNode next;
}


public boolean hasCycle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head;

        while (_____________________________________________________) {
                fast = _____________________________
                slow = _____________________________
                if (_____________________________________) {
                        return ____________________
                }
        }

        return false;
}
```

Q2 Solution

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

    }

}
```

Youtube References: https://www.youtube.com/watch?v=agkyC-rbgKM

Good Work!

Go to https://dynrec.cs.rutgers.edu/live/

Enter the Quiz Code: