Data Structure Lab4 : Singly Linked List 2022-2023

Topics

1. Implement Node Class
2. Generics
3. Implement SinglyLinkedList Class
4. Implement Basic Methods of SinglyLinkedList
    - isEmpty()
    - size()
    - first()
    - last()
    - addFirst()
    - addLast()
    - removeFirst()

Homework

1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

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

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class SinglyLinkedList {
    Node head;
    public boolean equals(SinglyLinkedList other) {
        Node current1 = this.head;
        Node current2 = other.head;

        while (current1 != null && current2 != null) {
            if (current1.data != current2.data) {
                return false;
            }
            current1 = current1.next;
            current2 = current2.next;
        }

        return current1 == null && current2 == null;
    }
}
```

2. Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.

```
public Node findSecondToLast() {
    if (head == null || head.next == null) {
        return null;  // القائمة تحتوي على أقل من عقدتين
    }

    Node current = head;
    while (current.next.next != null) {  // توقف عندما تكون العقدة التالية هي الأخيرة
        current = current.next;
    }
    return current;  // هذه هي العقدة قبل الأخيرة
}
```

3. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.

```
public int size() {
    int count = 0;
    Node current = head;
    while (current != null) {
        count++;
        current = current.next;
}
    return count;
}
```

4. Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node.

```
public void rotate() {
    if (head == null || head.next == null) {
        return;
    }

    Node secondLast = head;
    while (secondLast.next.next != null) {
        secondLast = secondLast.next;
    }

    Node last = secondLast.next;
    secondLast.next = null;
    last.next = head;
    head = last;
}
```

5. Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.

```
public void concatenate(SinglyLinkedList other) {
    if (head == null) {
        head = other.head;
        return;
    }

    Node current = head;
    while (current.next != null) {
        current = current.next;
    }

    current.next = other.head;
}
```

6. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

```
public void reverse() {
    Node previous = null;
    Node current = head;

    while (current != null) {
        Node nextNode = current.next;
        current.next = previous;
        previous = current;
        current = nextNode;
    }

    head = previous;
}
```