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.

```
concatenateLists(x, y):
    if x is empty:
        return y
    if yis empty:
        return x

    current = x.head
    while current.next is not null:
        current = current.next

    current.next = y.head

    x' = x
```

return x'

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 class LinkedList {
    private Node head;

    public Node findSecondToLastNode() {
        if (head == null || head.next == null) {
            return null;
        }

        Node current = head;
        Node previous = null;

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

        return pre;
    }


    private class Node {
        private int data;
        private Node next;

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

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

```
public class SingularlyLinkedList {
    private Node head;

    public int size() {
        int counter = 0;
        Node current = head;

        while (current != null) {
            counter++;
            current = current.next;
        }

        return counter;
    }


    private class Node {
        private int data;
        private Node next;

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

}
```

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

```
public class SinglyLinkedList {
    private Node head;


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

        Node pre= null;
        Node current = head;

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

        current.next = head;
        head = current;
        pre.next = null;
    }

    private class Node {
        private int data;
        private Node next;

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

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 class SinglyLinkedList {
    private Node head;

    public void concatenate(SinglyLinkedList otherList) {
        if (head == null) {
            head = otherList.head;
        } else {
            Node cur = head;
            while (cur.next != null) {
                cur = cur.next;
            }
            cur.next = otherList.head;
        }
    }

    private class Node {
        private int data;
        private Node next;

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

}
```

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

```
public class SinglyLinkedList {
    private Node head;


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

        Node pre = null;
        Node cur = head;

        while (cur != null) {
            Node next = cur.next;
            cur.next = pre;
            pre = cur;
            cur = next;
        }

        head = pre;
    }

    private class Node {
        private int data;
        private Node next;

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

}
```