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.

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.

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

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

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.

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

**Solution**

Class Node<T> {

T data;

Node<T> next;

Node(T data) {

This.data = data;

This.next = null;

}

}

Class SinglyLinkedList<T> {

Private Node<T> head;

SinglyLinkedList() {

Head = null;

}

Boolean isEmpty() {

Return head == null;

}

Int size() {

Int count = 0;

Node<T> current = head;

While (current != null) {

Count++;

Current = current.next;

}

Return count;

}

T first() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Return head.data;

}

T last() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

Node<T> current = head;

While (current.next != null) {

Current = current.next;

}

Return current.data;

}

Void addFirst(T data) {

Node<T> newNode = new Node<>(data);

newNode.next = head;

head = newNode;

}

Void addLast(T data) {

Node<T> newNode = new Node<>(data);

If (isEmpty()) {

Head = newNode;

} else {

Node<T> current = head;

While (current.next != null) {

Current = current.next;

}

Current.next = newNode;

}

}

T removeFirst() {

If (isEmpty()) {

Throw new NoSuchElementException("List is empty");

}

T removedData = head.data;

Head = head.next;

Return removedData;

}

Boolean equals(SinglyLinkedList<T> otherList) {

If (size() != otherList.size()) {

Return false;

}

Node<T> currentSelf = head;

Node<T> currentOther = otherList.head;

While (currentSelf != null) {

If (!currentSelf.data.equals(currentOther.data)) {

Return false;

}

currentSelf = currentSelf.next;

currentOther = currentOther.next;

}

Return true;

}

Node<T> findSecondToLastNode() {

If (isEmpty() || head.next == null) {

Return null;

}

Node<T> current = head;

While (current.next.next != null) {

Current = current.next;

}

Return current;

}

Int calculateSize() {

Int count = 0;

Node<T> current = head;

While (current != null) {

Count++;

Current = current.next;

}

Return count;

}

Void rotate() {

If (!isEmpty() && head.next != null) {

Node<T> firstNode = head;

Head = head.next;

Node<T> current = head;

While (current.next != null) {

Current = current.next;

}

Current.next = firstNode;

firstNode.next = null;

}

}

Void concatenateLists(SinglyLinkedList<T> otherList) {

If (isEmpty()) {

Head = otherList.head;

} else {

Node<T> current = head;

While (current.next != null) {

Current = current.next;

}

Current.next = otherList.head;

}

}

Void reverse() {

Node<T> previous = null;

Node<T> current = head;

Node<T> next = null;

While (current != null) {

Next = current.next;

Current.next = previous;

Previous = current;

Current = next;

}

Head = previous;

}

}