Topics

* Implement Node Class
* Generics
* Implement SinglyLinkedList Class
* Implement Basic Methods of SinglyLinkedList
* isEmpty()
* size()
* first()
* last()
* addFirst()
* addLast()
* removeFirst()

Homework

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

public boolean equals(Object obj) {

if (this == obj) return true;

if (obj == null || getClass() != obj.getClass()) return false

SinglyLinkedList<?> other = (SinglyLinkedList<?>) obj;

if (this.size() != other.size()) return false;

Node<?> currentA = this.head;

Node<?> currentB = other.head;

while (currentA != null) {

if (!currentA.data.equals(currentB.data)) return false;

currentA = currentA.next;

currentB = currentB.next;

}

return true;

}

-----------------------------------------------------------------------------------------------------------------

* 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<T> findSecondToLast() {

if (head == null || head.next == null) return null;

Node<T> current = head;

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

current = current.next;

}

return current;

}

-----------------------------------------------------------------------------------------------------------

* 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<T> current = head;

while (current != null) {

count++;

current = current.next; }

return count;

}

-----------------------------------------------------------------------------------------------------------------

* 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<T> oldHead = head;

head = head.next;

oldHead.next = null;

Node<T> tail = head;

while (tail.next != null) {

tail = tail.next;

}

tail.next = oldHead;

}

----------------------------------------------------------------------------------------------------------------

* 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<T> list) {

if (head == null) {

head = list.head;

return;

}

Node<T> current = head;

while (current.next != null) {

current = current.next;

}

current.next = list.head;

}

---------------------------------------------------------------------------------------------------------------------------

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

public void reverse() {

Node<T> prev = null;

Node<T> current = head;

while (current != null) {

Node<T> next = current.next;

current.next = prev;

prev = current;

current = next;

}

head = prev;

}