## 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.

**1. Develop an implementation of the equals method in the context of the SinglyLinkedList class.**  
**Answer:**  
To implement the equals method:

1. Check if the sizes of both lists are equal. If not, return false.
2. Use two pointers to traverse both lists simultaneously.
3. Compare the values of the corresponding nodes in both lists.
4. If a mismatch is found, return false.
5. If all nodes match, return true.

Example Code:

java

نسخ الكود

public boolean equals(SinglyLinkedList<E> other) {

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

Node<E> currentA = this.head;

Node<E> currentB = other.head;

while (currentA != null) {

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

currentA = currentA.next;

currentB = currentB.next;

}

return true;

}

**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.**  
**Answer:**

1. If the list has fewer than two nodes, return null.
2. Use a pointer (current) starting at the head.
3. Traverse the list until current.next.next == null.
4. Return current as it is the second-to-last node.

**3. Give an implementation of the size() method for the SinglyLinkedList class, assuming that we did not maintain size as an instance variable.**  
**Answer:**  
To compute the size dynamically:

1. Initialize count = 0.
2. Use a pointer (current) starting at the head.
3. Traverse the list, incrementing count for every node encountered.
4. Return count after reaching the end of the list.

Example Code:

java

نسخ الكود

public int size() {

int count = 0;

Node<E> 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.**  
**Answer:**

1. Save the current head in a temporary variable.
2. Update the head to head.next.
3. Traverse to the end of the list using a pointer.
4. Set the last node’s next to the saved node.
5. Set the next of the saved node to null.

Example Code:

java

نسخ الكود

public void rotate() {

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

Node<E> oldHead = head;

head = head.next;

Node<E> current = head;

while (current.next != null) {

current = current.next;

}

current.next = oldHead;

oldHead.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.**  
**Answer:**

1. If L is empty, set its head to M.head and return.
2. Use a pointer to traverse to the end of L.
3. Set the next of the last node in L to M.head.
4. Clear M's head (optional, to avoid confusion).

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

1. Initialize three pointers:
   * prev = null (to hold the previous node).
   * current = head (to track the current node).
   * next (to temporarily store the next node).
2. Traverse the list:
   * Save current.next in next.
   * Set current.next to prev.
   * Move prev to current and current to next.
3. Once the traversal is complete, set head = prev (the new head).

4o

**أنت قلت:**

is there any question need to write the code if there is write the code

**قال ChatGPT:**

ChatGPT

Here are the questions that require code and their implementations:

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

This question needs code because the equals method is key to comparing two linked lists.

java

نسخ الكود

public boolean equals(SinglyLinkedList<E> other) {

// Check if the other list is null

if (other == null) return false;

// Start from the head nodes of both lists

Node<E> currentA = this.head;

Node<E> currentB = other.head;

// Traverse both lists simultaneously

while (currentA != null && currentB != null) {

// Compare the data of current nodes

if (!currentA.data.equals(currentB.data)) {

return false;

}

currentA = currentA.next;

currentB = currentB.next;

}

// Check if both lists are fully traversed (same size)

return currentA == null && currentB == null;

}

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

This requires code to compute the size dynamically.

java

نسخ الكود

public int size() {

int count = 0;

Node<E> current = head; // Start from the head

while (current != null) { // Traverse the list

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.**

This needs code as it modifies the structure of the list.

java

نسخ الكود

public void rotate() {

// Check if the list is empty or has only one node

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

Node<E> oldHead = head; // Save the current head

head = head.next; // Move the head to the second node

oldHead.next = null; // Disconnect the old head

Node<E> current = head; // Start from the new head

while (current.next != null) { // Traverse to the last node

current = current.next;

}

current.next = oldHead; // Attach the old head at the end

}

**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.**

This also needs code to merge two lists.

java

نسخ الكود

public void concatenate(SinglyLinkedList<E> other) {

if (this.head == null) { // If the first list is empty

this.head = other.head; // Set its head to the second list's head

return;

}

Node<E> current = this.head; // Start from the head of the first list

while (current.next != null) { // Traverse to the last node of the first list

current = current.next;

}

current.next = other.head; // Connect the last node to the head of the second list

}

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

This involves iterative reversal, so it requires code.

java

نسخ الكود

public void reverse() {

Node<E> prev = null; // Initialize the previous pointer

Node<E> current = head; // Start from the head of the list

Node<E> next = null; // Initialize the next pointer

while (current != null) { // Traverse the list

next = current.next; // Store the next node

current.next = prev; // Reverse the current node's pointer

prev = current; // Move the previous pointer forward

current = next; // Move the current pointer forward

}

head = prev; // Update the head to the last node

}