Singly Linked List

Operations Without Tail Pointer Of Unsorted Singly

Linked List

AddAtFront(element)

- List having only head pointer.

- Make space for new element, say newNode.

- Stop. - Set newNode’s next to head.

- Set head to newNode

AddAtRear(element) - with no tail pointer.

- Make memory for new element, say newNode.

- Store element in newNode’s data.

- Set newNode’s next to empty.

- if list is empty then

- Set head to newNode.

- Stop. // List is not empty. Traverse the list to find the last node.

- Set current to head.

- while (current’s next is not empty)

- Set current to current’s next.

- Set current node’s next to newNode

================================================================

Operations With Tail Pointer

AddAtFront(element)

- Make space for new element, say newNode.

- Store element in newNode’s data.

- Set newNode’s next to empty.

- if list is empty then

- Set head and tail to newNode.

- Stop.

- Set newNode’s next to head.

- Set head to newNode.

AddAtRear(element) - with tail pointer.

- Make memory for new element, say newNode.

- Store element in newNode’s data.

- Set newNode’s next to empty.

- if list is empty then

- Set head and tail to newNode.

- Stop.

- Set tail node’s next to newNode.

- Set tail to newNode

================================================

Operations Without Tail Pointer Of Unsorted Singly

Linked List(Complete Example)(Unsorted)

==========================================

\*SinglyLinkedNode\*

**public** **class** SinglyListNode {

**public** **int** data;

**public** SinglyListNode next;

**public** SinglyListNode(**int** element) {

data = element;

next = **null**;

}

}

========================================

\*LinkedListIntf\*

**public** **interface** LinkedListIntf {

**public** **void** AddAtFront(**int** element);

**public** **void** AddAtRear(**int** element);

**public** **int**[] GetAllElements();

}

========================================

\*SinglyLinkedList\*

**public** **class** SinglyLinkedList **implements** LinkedListIntf {

**private** SinglyListNode head;

**public** SinglyLinkedList() {

head = **null**;

}

@Override

**public** **void** AddAtFront(**int** element) {

// Make space for element,say newNode

// Store element in newNode's data

SinglyListNode newNode = **new** SinglyListNode(element);

// Set newNode’s next to head.

newNode.next = head;

// Set head to newNode.

head = newNode;

}

@Override

**public** **void** AddAtRear(**int** element) {

// Make memory for new element, say newNode.

// Store element in newNode’s data.

SinglyListNode newNode = **new** SinglyListNode(element);

/\*- if list is empty then

- Set head to newNode.

- Stop \*/

**if** (head == **null**) {

head = newNode;

} **else**

/\*

\* List is not empty. -Traverse the list to find the last node. - Set current to

\* head

\*/

{

SinglyListNode curr = head;

/\*- while (current’s next is not empty)

- Set current to current’s next.

- Set current node’s next to newNode\*/

**while** (curr.next != **null**) {

curr = curr.next;

}

curr.next = newNode;

}

}

@Override

**public** **int**[] GetAllElements() {

**int**[] listElements;

/\*

\* if (head == null) { listElements = new int[0]; return listElements; }

\*/

listElements = **new** **int**[GetNodeCount()];

**int** i = 0;

SinglyListNode curr = head;

**while** (curr != **null**) {

listElements[i] = curr.data;

++i;

curr = curr.next;

}

**return** listElements;

}

**public** **int** GetNodeCount() {

**int** count = 0;

SinglyListNode curr = head;

// Algo: While current is not empty.

**while** (curr != **null**) {

++count;

curr = curr.next;

}

**return** count;

}

}

==============================================

\*SinglyLinkedListTest\*

**import** **static** org.junit.jupiter.api.Assertions.\*;

**import** org.junit.jupiter.api.Test;

**class** SinglyLinkedListTest {

@Test

**void** testEmptyList() {

SinglyLinkedList slist = **new** SinglyLinkedList();

**int**[] listElements = slist.GetAllElements();

*assertTrue*(listElements.length == 0);

}

@Test

**void** testAddAtFrontOneElement() {

SinglyLinkedList slist = **new** SinglyLinkedList();

**int** element = 5;

slist.AddAtFront(element);

**int**[] listElements = slist.GetAllElements();

*assertTrue*((listElements.length == 1) && (listElements[0] == element));

}

@Test

**void** testAddAtFrontTwoElement() {

SinglyLinkedList slist = **new** SinglyLinkedList();

**int** element1 = 5;

**int** element2 = 10;

slist.AddAtFront(element1); // head -> 5

slist.AddAtFront(element2); // head -> 10 -> 5

**int**[] listElements = slist.GetAllElements(); // 10, 5

*assertTrue*((listElements.length == 2) &&

(listElements[0] == element2) && (listElements[1] == element1));

}

@Test

**void** testAddAtRearElement() {

SinglyLinkedList slist = **new** SinglyLinkedList();

**int** element1 = 5;

**int** element2 = 10;

**int** element3 = 20;

slist.AddAtFront(element1);

slist.AddAtFront(element2);

slist.AddAtRear(element3);

**int**[] listElements = slist.GetAllElements();

*assertTrue*((listElements.length ==3) && (listElements[0]== element2) && (listElements[2]== element3));

}

}

==============================================================

Linked List(Complete Example)(Unsorted)

\*SinglyLinkedNode\*

**public** **class** SinglyListNode {

**public** **int** data;

**public** SinglyListNode next;

**public** SinglyListNode(**int** element) {

data = element;

next = **null**;

}

}

========================================

\*SortedLinkedListIntf\*

**public** **interface** SortedLinkedListIntf {

**public** **void** Insert(**int** element);

**public** **int**[] GetAllElements();

}

=======================================================

\*SortedLinkedListImpl\*

**public** **class** SortedLinkedListImpl **implements** SortedLinkedListIntf {

**private** SinglyListNode head;

@Override

**public** **void** Insert(**int** element) {

// - Make memory for new element, say newNode.

// - Store element in newNode’s data

SinglyListNode newNode = **new** SinglyListNode(element);

/\*- if list is empty then

- Set head to newNode.

- Stop \*/

**if** (head == **null**) {

head = newNode;

} **else**

/\*

\* List is not empty. Traverse the list to fin previous and current nodes,

\* because newNode will be inserted between previous and current nodes..

\* - Set current to head.

\* - Set previous to empty.

\*/

{

SinglyListNode curr = head;

SinglyListNode prev = **null**;

/\*- while ( (current is not empty) and (current node’s data < element) )

- Set previous to current.

- Set current to current’s next\*/

**while** ((curr != **null**) && (curr.data < element)) {

prev = curr;

curr = curr.next;

}

/\*

\* // Adding before first node? (Adding smallest element).

- if current is head node then

- Set newNode’s next to head.

- Set head to newNode.

- Stop.

- Set previous node’s next to newNode.

- Set newNode’s next to current\*/

**if** (curr == head) {

newNode.next = head;

head = newNode;

} **else** {

prev.next = newNode;

newNode.next = curr;

}

}

}

@Override

**public** **int**[] GetAllElements() {

**int**[] listElements;

listElements = **new** **int**[GetNodeCount()];

**int** i = 0;

SinglyListNode curr = head;

**while** (curr != **null**) {

listElements[i] = curr.data;

++i;

curr = curr.next;

}

**return** listElements;

}

**public** **int** GetNodeCount() {

**int** count = 0;

SinglyListNode curr = head;

// Algo: While current is not empty.

**while** (curr != **null**) {

++count;

curr = curr.next;

}

**return** count;

}

}

==============================================================

\*SortedLinkedListImplTest

import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.Test;

class SortedLinkedListImplTest {

@Test

void testForSortedList() {

SortedLinkedListImpl slist = new SortedLinkedListImpl();

int element1 = 5;

int element2 = 10;

int element3 = 7;

slist.Insert(element1); // 5

slist.Insert(element2); // 5 10

slist.Insert(element3); // 5 7 10

slist.Insert(15); // 5 7 10 15

slist.Insert(3); // 3 5 7 10 15

int[] listElements = slist.GetAllElements();

assertTrue((listElements.length == 5) &&

(listElements[2] == element3) && (listElements[0] == 3) && (listElements[1] == 5));

}

}

==================================================================