19CSE 212: Data structures and Algorithms

Lab Sheet 4

Circular Linked List and Doubly Linked List

Patel Rajkumar Pankajbhai

AM.EN.U4CSE20349

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

1. Implement the following in a circular singly linked list.
   1. Insert at head.
   2. Insert at last.
   3. Insert after a node.
   4. Delete a node with given data item.
   5. Delete a node at a given position.
   6. Searching an element.

Code :

package circular*L*inked*L*ist;

class Node{

*int* data;

Node next;

Node(*int* *d*){

data = *d*;

next = null;

}

}

class CircularLinkedList{

Node head;

*int* length = 0;

public *void* printCircularLinkedList() {

Node curr = head;

if(curr == null) {

System.out.print("The circular linked list is empty");

}

else {

System.out.print(curr.data+" ");

curr = curr.next;

while(curr != head) {

System.out.print(curr.data+" ");

curr = curr.next;

}

}

System.out.println();

}

public *void* insertAtHead(*int* *data*) {

Node newNode = **new** Node(*data*);

Node curr = head;

length++;

if(curr == null) {

head = newNode;

newNode.next = head;

}

else {

newNode.next = head;

head = newNode;

// now go to last element and resign it to the newly created node in both if and else case //

while(curr.next != head.next) {

curr = curr.next;

}

curr.next = head;

}

}

public *void* insertAtLast(*int* *data*) {

Node newNode = **new** Node(*data*);

Node curr = head;

length++;

if(curr == null) {

head = newNode;

newNode.next = head;

}

else {

while(curr.next != head) {

curr = curr.next;

}

curr.next = newNode;

newNode.next = head;

}

}

public *void* insertAfterNode(*int* *n*,*int* *data*) {

Node newNode = **new** Node(*data*);

Node curr = head;

if(*n* > length || *n* < 1) {

System.out.print("Please enter valid position to insert");

return;

}

else if(*n* == length) {

*this*.insertAtLast(*data*);

}

else if(*n* == 1) {

*this*.insertAtHead(*data*);

}

else {

*int* currPos = 1;

while(currPos < *n*) {

currPos++;

curr = curr.next;

}

newNode.next = curr.next;

curr.next = newNode;

length++;

}

}

public *void* deleteData(*int* *data*) {

Node curr = head;

Node prevHead = head;

if(curr == null) {

System.out.print("Circular Linked List is empty");

return;

}

else if(curr.data == *data*) {

head = curr.next;

// update last element link coz its circula

while(curr.next != prevHead) {

curr = curr.next;

}

curr.next = head;

}

else {

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

curr = curr.next;

}

curr.next = curr.next.next;

}

length--;

}

public *void* deleteAtPosition(*int* *pos*) {

Node curr = head;

*int* currPos = 1;

if(curr == null) {

System.out.print("The linked list is emtpy");

return;

}

else if(*pos* < 1 || *pos* > length) {

System.out.println("Please enter valid position");

return;

}

else if(*pos* == 1) {

while(curr.next != head) {

curr = curr.next;

}

curr.next = curr.next.next;

head = head.next;

}

else {

while(currPos < *pos*-1) {

currPos++;

curr = curr.next;

}

curr.next = curr.next.next;

}

length--;

}

public *boolean* findElement(*int* *data*) {

Node curr = head;

while(curr != null && curr.next != head) {

if(curr.data == *data*) {

return true;

}

else {

curr = curr.next;

}

}

return false;

}

}

public class Driver {

public static *void* main(String[] *args*) {

CircularLinkedList cl = **new** CircularLinkedList();

cl.insertAtHead(2);

cl.insertAtHead(1);

cl.printCircularLinkedList();

cl.insertAtLast(4);

cl.insertAtLast(6);

cl.printCircularLinkedList();

cl.insertAfterNode(2,3);

cl.insertAfterNode(4,5);

cl.insertAfterNode(6,8);

cl.insertAfterNode(6,7);

cl.printCircularLinkedList();

cl.deleteData(1);

cl.deleteData(8);

cl.deleteData(7);

cl.printCircularLinkedList();

cl.deleteAtPosition(0);

cl.printCircularLinkedList();

cl.deleteAtPosition(1);

cl.printCircularLinkedList();

cl.deleteAtPosition(3);

cl.printCircularLinkedList();

cl.deleteAtPosition(5);

cl.printCircularLinkedList();

}

}

Output :

Text

Description automatically generated

1. Implement the following in a doubly linked list.
   1. Insert at a position, after a given node.
   2. Delete a node with given data.
   3. Sort the list.
   4. Reverse first k elements.

Code :

package doubly*L*inked*L*ist;

import java.util.Scanner;

class Node {

*int* data;

Node next,prev;

Node(*int* *data*){

*this*.data = *data*;

next = null;

prev = null;

}

}

class DoublyLinkedList {

Node head;

*int* length = 0;

public *void* printDoublyLinkedList() {

Node curr = head;

if(curr == null) {

System.out.println("The linked list is empty");

}

else {

while(curr != null) {

System.out.print(curr.data+" ");

curr = curr.next;

}

}

System.out.println("...");

}

public *void* printDoublyLinkedListR() {

Node curr = head;

while(curr.next != null) curr = curr.next;

while(curr != null) {

System.out.print(curr.data+" ");

curr = curr.prev;

}

System.out.println("...");

}

public *void* insertAtPosition(*int* *pos*,*int* *data*) {

Node newNode = **new** Node(*data*);

Node curr = head;

if(head == null) {

head = newNode;

}

else if(*pos* == length+1) {

// System.out.println("at end");

while(curr.next != null) {

curr = curr.next;

}

curr.next = newNode;

newNode.prev = curr;

}

else if(*pos* == 1) {

// System.out.println("at start");

newNode.next = head;

head.prev = newNode;

head = newNode;

}

else {

// System.out.println("at pos "+data);

*int* currPos = 1;

while(currPos < *pos* - 1) {

curr = curr.next;

currPos++;

}

newNode.next = curr.next;

newNode.prev = curr;

curr.next = newNode;

newNode.next.prev = newNode;

}

length++;

}

public *void* deleteNode(*int* *data*) {

Node curr = head;

if(curr == null) {

System.out.print("The linkedlist is empty");

return;

}

else if(curr.data == *data*) {

head = curr.next;

curr = head;

curr.prev = null;

}

else {

while(curr != null) {

if(curr.data == *data*) {

//do alteration

curr.prev.next = curr.next;

if(curr.next != null) {

curr.next.prev = curr.prev;

}

break;

}

else {

curr = curr.next;

}

}

}

}

public *void* sortDoublyLinkedList() {

Node ptrF = head;

Node ptrB = head;

while(ptrF.next != null) {

ptrB = ptrF;

while(ptrB != null) {

if(ptrF.data > ptrB.data) {

*int* temp = ptrF.data;

ptrF.data = ptrB.data;

ptrB.data = temp;

}

ptrB = ptrB.next;

}

ptrF = ptrF.next;

}

}

public *void* reverseFirstKElements(*int* *k*) {

//breaking the LinkedList into two parts;

Node newHead;

Node curr = head;

if(*k* > length || *k* < 1) {

System.out.println("Please do enter a valid position");

return;

}

while(*k* > 1) {

*k*--;

curr = curr.next;

}

newHead = curr.next;

curr.next = null;

//now reversing the first LinkedList which is pointed by head pointer

curr = head;

Node next = head;

Node prev = null;

while(curr != null) {

next = curr.next;

curr.next = prev;

curr.prev = next;

prev = curr;

curr = next;

}

head = prev;

//now combining both the linked list;

curr = head;

while(curr.next != null) {

curr = curr.next;

}

curr.next = newHead;

if(newHead != null) {

newHead.prev = curr;

}

}

}

public class Driver {

public static *void* main(String[] *args*) {

DoublyLinkedList dl = **new** DoublyLinkedList();

dl.insertAtPosition(1, 1);

dl.insertAtPosition(2, 2);

dl.insertAtPosition(3, 4);

dl.insertAtPosition(3, 3);

dl.insertAtPosition(1, 0);

dl.printDoublyLinkedList();

dl.printDoublyLinkedListR();

dl.deleteNode(0);

dl.deleteNode(2);

dl.deleteNode(4);

dl.printDoublyLinkedList();

dl.printDoublyLinkedListR();

dl.insertAtPosition(1, 2);

dl.insertAtPosition(3, 5);

dl.insertAtPosition(1, 4);

dl.printDoublyLinkedList();

dl.sortDoublyLinkedList();

dl.printDoublyLinkedList();

Scanner sc = **new** Scanner(System.in);

System.out.print("Enter K to reverse K elements: ");

dl.reverseFirstKElements(sc.nextInt());

sc.close();

dl.printDoublyLinkedList();

}

}

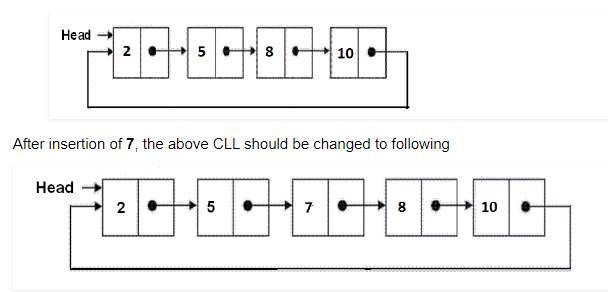
Output :

Text

Description automatically generated

1. Implement a procedure Sorted \_insert () for circular singly linked list.

The function Sorted \_insert() should insert a new value in a sorted Circular Linked List (CLL). For example, if the input CLL is following.



Code :

package sorted*C*ircular*L*inked*L*ist;

class Node{

*int* data;

Node next;

Node(*int* *data*){

*this*.data = *data*;

*this*.next = null;

}

}

class SortedCircularLinkedList{

Node head;

public *void* printSorted() {

Node curr = head;

if(curr == null) {

System.out.println("The list is empty!");

}

else {

System.out.print("Printing Sorted Circular Linked List: ");

System.out.print(curr.data+" ");

curr = curr.next;

while(curr != head) {

System.out.print(curr.data+" ");

curr = curr.next;

}

System.out.println();

}

}

public *void* insertAtFront(*int* *data*,Node *newNode*) {

*newNode*.next = head;

Node curr = head;

while(curr.next != head) {

curr = curr.next;

}

curr.next = *newNode*;

head = *newNode*;

}

public *void* sortedInsert(*int* *data*) {

Node newNode = **new** Node(*data*);

if(head == null) {

head = newNode;

newNode.next = head;

}

else {

Node curr = head;

if(curr.data > *data*) {

*this*.insertAtFront(*data*,newNode);

}

else {

while(curr.next.data < *data* && curr.next != head) {

curr = curr.next;

}

if(curr.next == head) {

curr.next = newNode;

newNode.next = head;

}

else {

newNode.next = curr.next;

curr.next = newNode;

}

}

}

}

}

public class Driver {

public static *void* main(String[] *args*) {

SortedCircularLinkedList scll = **new** SortedCircularLinkedList();

scll.sortedInsert(2);

scll.sortedInsert(8);

scll.sortedInsert(3);

scll.sortedInsert(9);

scll.sortedInsert(1);

scll.sortedInsert(4);

scll.printSorted();

}

}

Output :

Text

Description automatically generated

1. Implement the SortedMerge() function that takes two doubly-linked lists, each of which is sorted in increasing order, and merges the two together into one list which is in increasing order. SortedMerge() should return the new list. The new list should be made by splicing together with the nodes of the first two lists. For example, if the first list a is 5->10->15

and the other list b is 2->3->20, then SortedMerge() should return a pointer to the head node of the merged list 2->3->5->10->15->20.

Code :

package sorted*M*erge;

class Node {

*int* data;

Node prev;

Node next;

Node(*int* *data*){

*this*.data = *data*;

*this*.prev = null;

*this*.next = null;

}

}

class DoublyLinkedList {

Node head;

Node tail;

public *void* print() {

Node curr = head;

if(curr == null) {

System.out.println("The given Linked list is empty");

}

else {

while(curr != null) {

System.out.print(curr.data+" ");

curr = curr.next;

}

System.out.println();

}

}

public *void* insert(*int* *data*) {

Node newNode = **new** Node(*data*);

if(head == null) {

head = newNode;

tail = newNode;

}

else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

public DoublyLinkedList sortedMerge(DoublyLinkedList *List2*) {

DoublyLinkedList result = **new** DoublyLinkedList();

Node curr1 = *this*.head;

Node curr2 = *List2*.head;

while(curr1 != null && curr2 != null) {

if(curr1.data < curr2.data) {

result.insert(curr1.data);

curr1 = curr1.next;

}

else {

result.insert(curr2.data);

curr2 = curr2.next;

}

}

while(curr1 != null) {

result.insert(curr1.data);

curr1 = curr1.next;

}

while(curr2 != null) {

result.insert(curr2.data);

curr2 = curr2.next;

}

return result;

}

}

public class Driver {

public static *void* main(String[] *args*) {

DoublyLinkedList LinkedListOne = **new** DoublyLinkedList();

DoublyLinkedList LinkedListTwo = **new** DoublyLinkedList();

LinkedListOne.insert(1);

LinkedListOne.insert(4);

LinkedListOne.insert(9);

LinkedListOne.insert(15);

LinkedListOne.insert(19);

LinkedListOne.insert(20);

System.out.print("Sorted Doubly Linked List 1: ");

LinkedListOne.print();

LinkedListTwo.insert(0);

LinkedListTwo.insert(5);

LinkedListTwo.insert(6);

LinkedListTwo.insert(25);

System.out.print("Sorted Doubly Linked List 2: ");

LinkedListTwo.print();

DoublyLinkedList result = LinkedListOne.sortedMerge(LinkedListTwo);

System.out.print("The new sorted merged Doubly Linked List is: ");

result.print();

}

}

Output:

Text

Description automatically generated

1. Implement the sumof pair() function that takes a sorted doubly linked list of positive distinct elements and find pairs in a doubly linked list whose sum is equal to the given value x.

Example:

Input : : 1 <-> 3<->4 <-> 5 <-> 7 <-> 8 <-> 9

x = 8

Output: (1,7), (3,5)

Code :

package sum*O*f*P*airs;

import java.util.Scanner;

class Node {

*int* data;

Node prev;

Node next;

Node(*int* *data*){

*this*.data = *data*;

*this*.prev = null;

*this*.next = null;

}

}

class DoublyLinkedList {

Node head;

Node tail;

public *void* insert(*int* *data*) {

Node newNode = **new** Node(*data*);

if(head == null) {

head = newNode;

tail = newNode;

}

else {

newNode.prev = tail;

tail.next = newNode;

tail = newNode;

}

}

public *void* FindSumOfPairs(*int* *sum*) {

Node ptrS = head;

Node ptrF;

while(ptrS.next != null) {

ptrF = ptrS.next;

while(ptrF != null) {

if(ptrF.data + ptrS.data == *sum*) {

System.out.println(ptrF.data +" "+ ptrS.data);

}

ptrF = ptrF.next;

}

ptrS = ptrS.next;

}

}

}

public class Driver {

public static *void* main(String[] *args*) {

DoublyLinkedList dll = **new** DoublyLinkedList();

Scanner sc = **new** Scanner(System.in);

System.out.print("Enter the number of elements: ");

*int* size = sc.nextInt();

System.out.print("Enter "+size+" elements in ascending order: ");

while(size-- > 0) {

dll.insert(sc.nextInt());

}

System.out.print("Enter the SUM value: ");

*int* sum = sc.nextInt();

dll.FindSumOfPairs(sum);

sc.close();

}

}

Output :

Text

Description automatically generated