ALGONQUIN COLLEGE

CST8130 – Data Structures

Professor : Dr. Anu Thomas
Email: thomasa@algonquincollege.com
Office: T314

Singly Linked Lists

General hint when writing code related to Linked List (class LList and LLNode)

- Consider the effect on head data member (ie first element in list)
- Consider the effect in middle of list
- Consider the effect on last member of list
- Consider the effect if list is empty

Single Linked list - Traditional

Data members

- **LList** class has a head (reference to first **LLNode** in the list);
- **LLNode** contains reference to next node in list

Methods

- add at head method – O(1)
- search method – sequential through the list – O(n)
- sort method – doesn't really work – need addInOrder which would have O(n)
- delete from head method – O(1)
- delete method of particular node – O(n)

Single Linked List – options

- Data member – tail – reference to last node in the list
- Allows delete at tail – O(n)
- Allows add at tail – O(1)
- All other methods same

Single Linked List - options

- Data member - numItems in list

- Can provide useful information without having to traverse list to count

Doubly Linked Lists

Doubly Linked List

- LLNode would contain links to both previous and next nodes in the list
- LList still contains
  - head – reference to first node in the list,
  - usually also contains a tail – reference to the last node in the list

Node

```
public class DLLNode {
    private String data;
    private DLLNode next;
    private DLLNode prev;

public DLLNode (String newData) {
    this.data = newData;
    this.next = null;
    this.prev = null;
}
```

Node (contd.)

```
public void updateNodeNext (DLLNode nextOne) {
    this.next = nextOne;
}
public void updateNodePrev (DLLNode prevOne) {
    this.prev = prevOne;
}
public String toString () {      return  this.data;}
public DLLNode getNext() {   return this.next;}
public DLLNode getPrev() {   return this.prev;}
```

List

```
public class DLList {
        private DLLNode head;
        private DLLNode tail;

public DLList() {    head = null;  tail=null;}
public void addAtHead (String newData) {
        DLLNode newNode = new DLLNode (newData);
        if (head != null)
                head.updateNodePrev(newNode);
        newNode.updateNodeNext(head);
        head = newNode;
        if (tail == null)
                tail = newNode;
        }
```

List (contd.)

```
public void displayFromHead() {
        DLLNode temp = head;
        while (temp != null) {
                System.out.println (temp);
                temp = temp.getNext();
        }
}
public void displayFromTail() {
        DLLNode temp = tail;
        while (temp != null) {
                System.out.println (temp);
                temp = temp.getPrev();
        }
}
```

Method main

```
public static void main(String[] args) {
    DLList list = new DLList();

    list.addAtHead(“Anu");
    list.addAtHead(“Thomas");
    System.out.println("The list from head  is ");
    list.displayFromHead();

    System.out.println("The list from tail  is ");
    list.displayFromTail();
```

Delete a node

What would delete look like?

```
public boolean searchAndDelete (String oneToDelete) {
        if (head == null)     return false;
        DLLNode current = head;
        // walk through list, compare each node with search value
        while (current!= null &&  !(current.toString().equals(oneToDelete))) {
                        current = current.getNext();
         }
        if (current != null) { // must have found it
                if (head == current && tail == current){ // deleting only item in list
                        head = null;                tail = null;
                } else if (current == head)   {   // deleting first item
                        head = current.getNext();
                        current.getNext().updateNodePrev(null);
                } else if (current == tail) {  // deleting last item
                        tail  = current.getPrev();
                        current.getPrev().updateNodeNext(null);
                } else {      current.getPrev().updateNodeNext(current.getNext());
                        current.getNext().updateNodePrev(current.getPrev());
                }
                return true;
         }   else  return false;    // didn't find it }
}
```

Practice…..write these methods

public boolean addAtTail(String s)

public DLLNode deleteAtHead()

public DLLNode deleteAtTail()

public boolean addInOrder (String s)

Questions?