Linked Lists

Read Chapter 4
Questions

• Given a singly linked list, devise a time- and space-efficient algorithm to find the \textit{mth-to-last element} of the list. Implement your algorithm, taking care to handle relevant error conditions. Define \textit{mth to last} such that \textit{m=0}, the last element of the list is returned.

• You are given a linked list that is either NULL-terminated (acyclic), or ends in a cycle (cyclic), write a function that takes a pointer to the head of a list and determines if the list is \textit{cyclic or acyclic}. Your function should return 0 if the list is acyclic and 1 if it is cyclic. You may not modify the list in any way.
Linked list

• Singly linked list
• Doubly linked list
  – With dummy nodes
• Circularly linked list
Linked List

- **Linked List**
  - A sequence of elements arranged one after another
  - Each element connected to the next by a “link”

- **Node**
  - Element
  - Link to the next element
  - Last node

12 → 28 → 0 → 34 end
Declarations for Linked Lists

- Each node in the linked list is a class, as shown here.

```java
public class IntNode
{
    private int data;
    private IntNode link;
    ...
}
```

12  →  28  →  0  →  34  end
List end

- **null reference** used as the final node of a linked list
How to access a linked list?
Declarations for Linked Lists

• A program can keep track of the front node by using an IntNode reference variable head.
  – Notice that head is not an IntNode -- it is a reference to an IntNode.
Declarations for Linked Lists

• A program can keep track of the front node by using an IntNode reference variable head.
  – Notice that head is not an IntNode -- it is a reference to an IntNode.
• We represent the empty list by storing null in the head reference.
Declarations for Linked Lists

- A program can keep track of
  - the front node by using an IntNode reference variable `head`.
  - the last node by using an IntNode reference variable `tail`.

![Diagram showing a linked list with nodes 12, 28, 0, 34, and null, with `head` pointing to 12 and `tail` pointing to 34.]
IntNode
Members

• 2 instance variables

• Constructor
  – public IntNode(int initialData, IntNode initialLink)
  – Demonstrate the execution of one constructor.

• Methods
  – public int getData()
  – public IntNode getLink()
  – public void setData(int newData)
  – public void setLink(IntNode newLink)
Members

• Methods
  – public void addNodeAfter(int e)
  – public void removeNodeAfter()
addNodeAfter

• Add a node with element 25 after the “sel” node
addNodeAfter

• link = new IntNode(25,link);

• What if the selected node is the tail of a list?
removeNodeAfter

• Remove the node after the node with value 28
removeNodeAfter

• Remove the node after the node with value 28

```java
sel = the node with value 28
sel.removeNodeAfter()

link = sel.getLink().getLink();
Or
link = sel.link.link;
```
removeNodeAfter

• Special case: remove the last node

link = node with value 34; link.link is null;
removeNodeAfter

- Special case: tail node activate this method?

Precondition: link!=null
LinkedList
public class LinkedList{
    private IntNode head=null;

    public void addFromFront(int e)
    public void removeHead();
}

LinkedList
Inserting an IntNode at the Front

We want to add a new entry, 13, to the front of the linked list shown here.
Inserting an IntNode at the Front

1. Create a new node...
Inserting an IntNode at the Front

1. Create a new node...
2. Place the data in the new node's data field.
Inserting an IntNode at the Front

1. Create a new node...
2. Place the data in the new node's data field....
3. Connect the new node to the front of the list.
Inserting an IntNode at the Front

1. Create a new node...
2. Place the data in the new node's data field....
3. Connect the new node to the front of the list.
4. Make the head refer to the new head of the linked list.
Inserting an IntNode at the Front

1. Create a new node...
2. Place the data in the new node's data field....
3. Connect the new node to the front of the list.
4. Make the head refer to the new head of the linked list.

head = new IntNode(13, head);
Inserting an IntNode at the Front

```java
public IntNode(int initialData, IntNode initialLink)
{
    data = initialEntry;
    link = initialLink;
}
```

Suppose head is null and we execute the assignment shown here:

```
head = new IntNode(13, head);
```

null

head
Inserting an IntNode at the Front

When the statement finishes, the linked list has one node, containing 13.

```
public IntNode(int initialData, IntNode initialLink)
{
    data = initialEntry;
    link = initialLink;
}
```

head = new IntNode(13, head);
Caution!

• Always make sure that your linked list methods work correctly with an empty list.
Pseudocode for Removing IntNodes

• A technique for removing a node from the front of a list,
• A technique for removing a node from elsewhere.
Removing the Head IntNode

head = head.link;

Draw the change that this statement will make to the linked list.
Removing the Head IntNode

head = head.link;

```
head
  13 10 15 null
head
```
Removing the Head IntNode

• Here’s what the linked list looks like after the removal finishes.
Special case

• What if a linked list contains only one node?
Use LinkedList
Method - listLength

//a method in IntNode
public static int listLength(IntNode head)
{
    int answer = 0
    for(IntNode cursor = head; cursor!=null; cursor=cursor.link)
    {
        answer++;
    }
    return answer;
}
//a method in LinkedList

public int listLength()
{
    return IntNode.listLength(head);
}

Methods - listSearch

//a method in IntNode
public static IntNode listSearch(IntNode head, int target) {
    IntNode answer = null;
    for(IntNode cursor=head; cursor!=null; cursor = cursor.link) {
        if(cursor.data==target) {
            answer = cursor;
            break;
        }
    }
    return answer;
}
Method-listPosition

//a method in IntNode
//pos starts from 1
public static IntNode listPosition(IntNode head, int pos)
{
    int index = 1;
    IntNode cursor=head;
    for(; (cursor!=null)&&(index<pos); index++){
        cursor = cursor.link;
    }
    return cursor;
}
findMiddle

- Special cases
- List has 0 elements? pos = 0
- List has 1 element? pos = 1
- List has 2 elements? pos = 2
- List has 3 elements? Pos = 2
- Any rule?
- How to declare it in IntNode and in LinkedList?
public static IntNode listCopy(IntNode source) {
    IntNode copyHead;
    IntNode copyTail;

    // Handle the special case of the empty list.
    if (source == null) return null;

    // Make the first node for the newly created list.
    copyHead = new IntNode(source.data, null);
    copyTail = copyHead;

    // Make the rest of the nodes for the newly created list.
    while (source.link != null) {
        source = source.link;
        copyTail.addNodeAfter(source.data);
        copyTail = copyTail.link;
    }

    // Return the head reference for the new list.
    return copyHead;
}
Summary of LinkedList

• IntNode
  – Instance variable
  – Constructor
  – Methods
    • Accessor/Modification methods
    • Static methods
      – addNodeAfter
      – removeNodeAfter
      – listLength, listSearch, listPosition
      – listCopy
Linked list

• Singly linked list
• **Doubly linked list**
  – With dummy nodes
• Circularly linked list
Doubly Linked List

- Doubly Linked List
- Node
  - Element
  - Link to the next/previous element
Doubly Linked List

public class DLinkedList {
    private DIntNode head;
    private DIntNode tail;
    private int manyItems;

    public DLinkedList() {
        head = tail = null;
        manyItems = 0;
    }
}

CS272, Huiping Cao
Doubly Linked List with Dummy Nodes

public class DLinkedList {
    private DIntNode head;
    private DIntNode tail;
    private int manyItems;

    public DoublyLinkedListDummy() {
        head = new DIntNode();
        tail = new DIntNode();
        head.setNext(tail);
        head.setPrev(tail);
        tail.setPrev(head);
        manyItems = 0;
    }
}

CS272, Huiping Cao
Outline

- DoublyLinkedList with dummy head and tail
public void addAfter(DIntNode v, int element) {
    DIntNode newNode = new DIntNode(element, null, null);
    // (1) Make newNode's prev link point to v
    newNode.setPrev(v);
    // (2) Make newNode's next link point to w
    DIntNode vNext = v.getNext();
    newNode.setNext(v.getNext());
    // (3) Make vNext's prev link point to newNode
    vNext.setPrev(newNode);
    // (4) Make v's next link point to newNode
    v.setNext(newNode);
    // Update many items
    manyItems++;
}

CS272, Huiping Cao
public boolean remove(DIntNode v) {
    //Pseudo-codes
    //Special consideration
    DIntNode vPrev = v.getPrev();
    DIntNode vNext = v.getNext();

    //Make vNext's prev link point to vPrev
    vNext.setPrev(vPrev);

    //Make vPrev's next link point to vNext
    vPrev.setNext(vNext);

    //Update many items
    manyItems--;
    return true;
}

Linked list

• Singly linked list
• Doubly linked list
  – With dummy nodes
• **Circularly linked list**
Circularly Linked List

- Circularly Linked List
- IntNode
  - Element
  - Link to the next element

```plaintext
12 → 28 → 12 → 34
```

`cursor`