Linked List II

Doubly Linked Lists

Reminder

- Project 1
  - If not already picked up, do so after class.
  - Still have some Exam 1's left
- Project 2
  - Due this Sunday
    - Submit early!
    - Submit often!
  - Miss the minimum? Please see me after class.
  - Questions?

Exam 2

- Will return after doubly linked lists.

Announcement

- Final Exam
  - Wednesday, February 25, 2004
  - 8:00am – 10:00 am
  - 70-3435

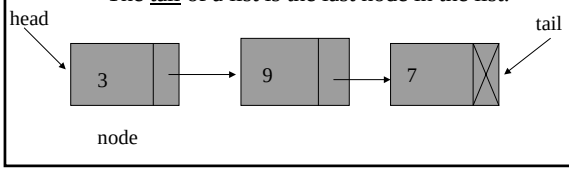
Any questions

Linked Lists

- Sequence of elements
  - arranged one after the other
  - Each element has
    - Some piece of data
    - a link to the next element in the sequence
    - The "next" link for the last element is null.
  - Basic linked list need not be sorted.

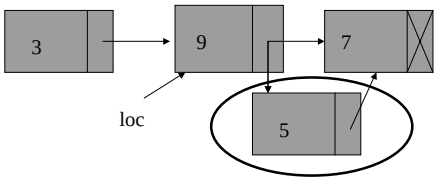
Linked Lists

- Implementation
  - Like with trees, a list can be seen as a collection of "nodes"
  - The head of a list is the first node in the list
  - The tail of a list is the last node in the list.



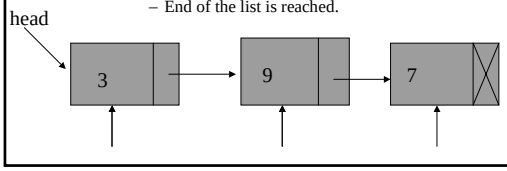
Linked List – Remove

- Removing an interior node
  - Need a reference to the node before the node to be deleted (loc)
  - Loc's next will point to whatever the deleted node's next was pointing to.



Linked List – Find

- Finding an item in a linked list
  - Basic idea
    - Start at head of the list
    - Follow the links until
      - Object searched for is found or
      - End of the list is reached.



Linked List – Remove

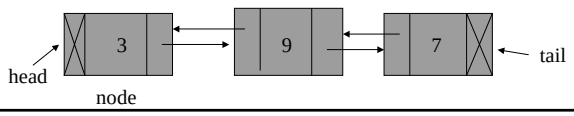
- Remove is a bit awkward
  - Suppose we want to remove the node returned by `find`.
    - We'll need to maintain a pointer to the node before this node.
    - Very cumbersome.
    - Possible solution
      - Have a node have a pointer to it's previous element in the list as well.
- Doubly-linked list

Doubly Linked Lists

- Sequence of elements
  - Each element has
    - Some piece of data
    - a link to the next element in the sequence
      - The "next" link for the last element is null.
    - a link to the previous element in the sequence
      - The "prev" link for the first element is null.

Doubly Linked Lists

- Implementation
  - Like with trees, a list can be seen as a collection of "nodes"
  - The head of a list is the first node in the list
  - The tail of a list is the last node in the list.
  - Allows for forward and backwards traversal



Doubly Linked Lists

- Implementation

```
public class DListNode {
      Object data;
      DListNode next;
      DListNode prev;
}
```

DListNodes

- Operations on DListNodes
  - Constructor
    - initialData – data to be placed in the node
    - initialNext – reference to next node
    - initialPrev – reference to previous node
  - Get Methods
    - Object getData()
    - DListNode getNext()
    - DListNode getPrev()
  - Set Methods
    - setData(Object O)
    - setNext (DListNode N)
    - setPrev (DListNode N)

DListNodes – constructor

```
public DListNode (Object initialData, DListNode
   initialNext, DListNode initialPrev)
{
    // set your data
    data = initialData;

    // set your next
    setNext (initialNext);

    // set prev
    setPrev(initialPrev);
}
```

DListNodes – setMethods

```
public void setNext (DListNode N)
{
   if (next != N) {
      next = N;
      if (N != null) N.setPrev (this);
   }
}

public void setPrev (DListNode N)
{
   if (prev != N) {
      prev = N;
      if (N != null) N.setNext (this);
   }
}
```

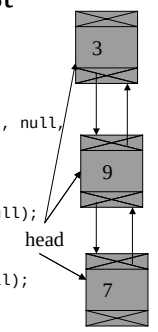
Doubly Linked List

- Let's create a simple list:

```
// Add the 7
DListNode head = new DListNode (new Integer(7), null,
                                null);

// Add the 9
head = new DListNode (new Integer(9), head, null);

// Add the 3
head = new DListNode (new Integer(3), head,null);
```



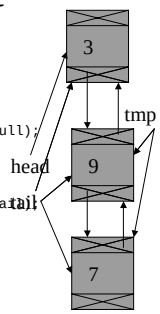
Doubly Linked List

- Let's create a simple list (another way)

```
// Add the 3
DListNode head = new DListNode (new Integer(3), null,null);
DListNode tail = head;

// Add the 9
DListNode tmp = new DListNode (new Integer(9), null, tail);
tail = tmp;

// Add the 7
tmp = new DListNode (new Integer(7), null, tail);
tail = tmp;
```



Doubly Linked List

- Operations on entire list
  - Find
  - Add
  - Remove

Doubly Linked List – Find

```
public DListNode find (DListNode head, Object
  target)
{
   DListNode found = null;
   DListNode cur = head;
   while ((found == null) && (cur != null)) {
      if (target.equals (cur.getData())
         found = cur;
      else
         cur = cur.getNext();
   }
   return found
}
```

Doubly Linked List – Find

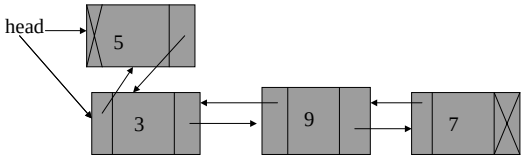
```
public DListNode findBackwards (ListNode tail,
  Object target)
{
   DListNode found = null;
   DListNode cur = tail;
   while ((found == null) && (cur != null)) {
      if (target.equals (cur.getData())
         found = cur;
      else
         cur = cur.getPrev();
   }
   return found
}
```

Doubly Linked List – Add

- Adding will depend on where you wish to add the new node
  - Add before the head of the list
  - Add to the interior of the list

Doubly Linked List – Add

- Adding before the head of the list
  - Make the new node the new head



Doubly Linked List – Add

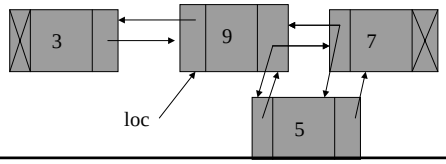
```
// Add the 7
DListNode head = new DListNode (new Integer(7), null,
  null);

// Add the 9
head = new DListNode (new Integer(9), head, null );

// Add the 3
head = new DListNode (new Integer(3), head, null);
```

Doubly Linked List – Add

- Adding to the interior of the list
  - Need a reference to the node before the location of the new node (loc)
  - New node's next will point to whatever loc's next is pointing to
  - loc.next will point to new node
  - prev pointers will get updated appropriately.



Doubly Linked List – Add

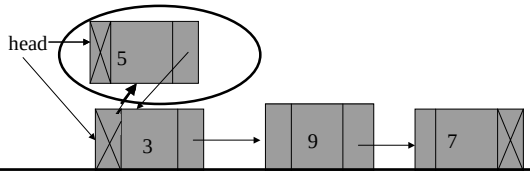
```
public void addAfter (DListNode loc,
  Object target)
{
  DListNode tmp = new DListNode (target,
                                 loc.getNext(),
  loc);
}
```

Doubly Linked List – Remove

- Remove will depend on where the node you wish to remove is
  - Removing the node at the head of the list
  - Removing an interior node.

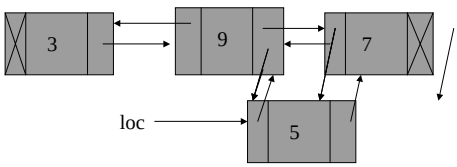
Doubly Linked List – Remove

- Removing the node at the head of the list
  - Have the head point to the node after the old head
    - `head = head.getNext();`
    - `head.setPrev (null);`



Doubly Linked List – Remove

- Removing an interior node
  - Need a reference to the node to be deleted (loc) (NOT the node before)
  - Set loc's next to be loc's prev's next



Doubly Linked List – Remove

```
public void remove (DListNode loc)
{
  DListNode nxt = loc.getNext();
  DListNode prv = loc.getPrev();
  if (prv != null)
     prv.setNext(nxt);
}
```

Lists vs. Arrays

- Arrays are better at random access
  - I.e give me the 4th element in the collection
- Linked lists are better for internal additions and deletions
  - I.e. delete the 5th element
- Doubly linked lists are better if you need to iterate backwards
- Lists perform dynamic sizing
  - It is expensive for arrays to resize themselves dynamically

Summary

- Doubly Linked Lists
- DListNode
- Operations
  - Add
  - Find
  - Remove
- Linked Lists vs. Arrays