




Lecture 13: Doubly
Linked Lists and Queues

+ Today

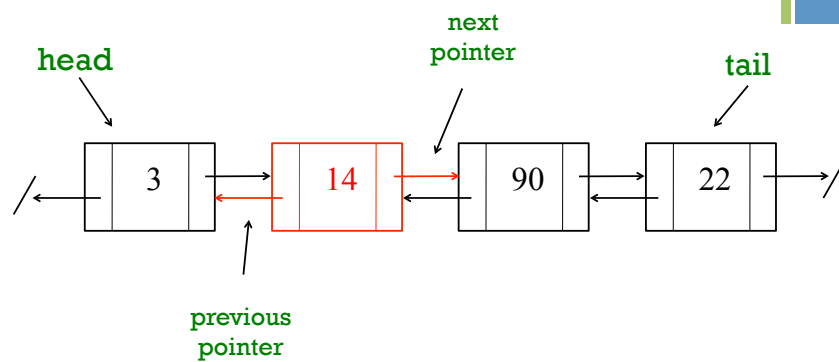


- Reading
 - JS Ch 9 (Linked Lists) and JS Ch 10.2 (Queues)
- Objectives
 - Review Linked Lists worksheet
 - Week 4 Assignment (Doubly linked lists)
 - Stacks and Queues

+ Announcements

- No quiz this Friday!
- Last faculty candidate this Friday
- Posted on Piazza
 - pdf with neater (more informative) JavaDoc comments for methods in CurDoublyLinkedList
 - Solutions to Linked List worksheet

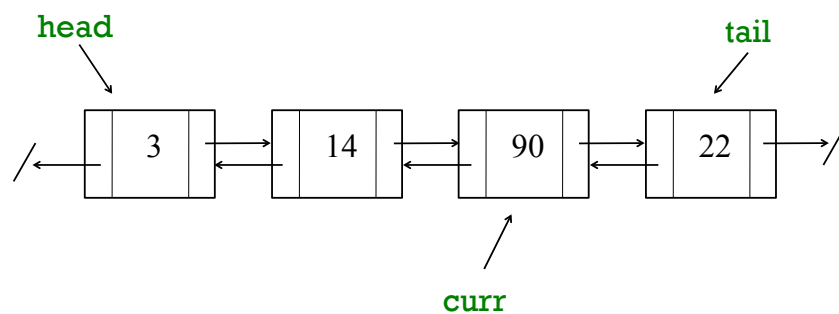
+ Doubly Linked List



+ Doubly Linked List

- Useful if you need to traverse in both directions
- Must change twice as many links when adding or removing!
- `DoublyLinkedListNode<E>` class now contains pointer to previous node
- `java.util.LinkedList` provides a doubly linked list
- We'll use `DoublyLinkedList` from Bailey

+ Doubly Linked List



How do we remove the node pointed to by curr?

+ Removing from Doubly Linked List

```
public E remove(E value) {
    DoublyLinkedNode<E> curr = head;
    while( curr != null && !curr.value().equals(value)) {
        curr = curr.next();
    }

    if(curr != null) {
        if(curr.previous() != null) {
            curr.previous().setNext(curr.next());
        }else{
            head = curr.next();
        }
    }

    if(curr.next() != null) {
        curr.next().setPrevious(curr.previous());
    }else {
        tail = curr.previous();
    }
    count--;
    return curr.value();
}
return null;
}
```

+ Review of Stacks

- Analogy: stack of plates
- Operations
 - LIFO: “last in, first out”
 - Push(), Pop()
- Limit access to data
 - Facilitates certain algorithms
 - Less easily corruptible w/o random access



+ Using a stack

```
Stack<String> s = new Stack<String>();

for (int i = 0; i < 4; ++i) {
    s.push("" + i);
}

while (s.size() > 0) {
    System.out.print(s.pop());
}
```

3210



+ Queue

- Analogy: line of people
- Operations
 - FIFO: "first in first out"
 - **enqueue()** at **end**
 - **dequeue()** from **beginning**
- Applications
 - Simulations
 - Event queue
 - Keeping track when searching



+ Using a queue

```
Queue<String> q = new Queue<String>();

for (int i = 0; i < 4; ++i) {
    q.enqueue("" + i);
}

while (q.size() > 0) {
    System.out.print(q.dequeue());
}
```

0 1 2 3



+ Queue Implementations

- Doubly Linked List
 - Head of list is front of queue
 - Tail of list is end of queue
 - How complex for enqueue, dequeue?
- ArrayList:
 - Which end should be front, rear?
 - How complex for enqueue, dequeue?
- Array
 - Which end should be front, rear?
 - How complex for enqueue, dequeue?