CS 310: Linked Queues and Array Queues

Chris Kauffman

Week 4-1
Announcement: IBM’s Master the Main Frame Contest

Contest for students sponsored by IBM. More Info Here
No experience with mainframes is necessary. In fact, the contest is designed for students with little or no mainframe experience, increasing with difficulty as the contest progresses. Students just need to bring drive and competitive spirit and be ready to compete!

Frequently Asked Questions/Answers: When is the contest?
The contest runs from October 1, 2015 - December 31, 2015.

Why compete in a Mainframe Contest? Prizes!
- IBM Master the Mainframe T-shirts
- Prize Packs worth over $100 in IBM swag
- Tablet Computers

My email address for registration: kauffman@cs.gmu.edu
Logistics

HW
- HW 1 now late
- HW 2 up a few days

Reading
- Weiss Ch 16 Stacks/Queues
- Weiss Ch 17 Linked Lists

Goals Today
- Stack Review
- Queues two ways

HW 1 Worst to Best
- Noncompiling code
- Code that compiles
- Code that compiles and passes most/all tests
- All of the above PLUS is clean and understandable
- All of the above PLUS code clearly meets complexity bounds, perhaps justified in comments
Last Time: ArrayList complexities

- ArrayList of size $N$
- Space complexity: $O(N)$

<table>
<thead>
<tr>
<th>Operation</th>
<th>Method</th>
<th>Worst Time</th>
<th>Average Time</th>
<th>Worst Space</th>
<th>Average Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size()</td>
<td>al.size()</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Get(i)</td>
<td>al.get(i)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Set(i,x)</td>
<td>al.set(i,x)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Add(x)</td>
<td>al.add(x)</td>
<td>O(N)</td>
<td>O(1)</td>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Insert(i,x)</td>
<td>al.add(i,x)</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Remove(i)</td>
<td>al.remove(i)</td>
<td>O(N)</td>
<td>O(N)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
Previously: Stacks two ways

<table>
<thead>
<tr>
<th>Operation</th>
<th>Run Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>isEmpty()</td>
<td></td>
</tr>
<tr>
<td>push(x)</td>
<td></td>
</tr>
<tr>
<td>pop()</td>
<td></td>
</tr>
<tr>
<td>getTop()</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Array Stack</th>
<th>Array Stack</th>
<th>Linked Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td></td>
<td>Worst</td>
<td>Worst</td>
</tr>
</tbody>
</table>

- What do those operations do?
- What fields does the array-based stack have?
- What fields does the linke-based stack have?
Stack Implementation Notes

Generic

Stacks can hold any type of thing

class Stack<T>{...}
class UseStack{
    main(){
        Stack<String> strs =
            new Stack<String>();
        strs.push("Hi");
        String s = strs.top();
        strs.pop();

        Stack<Integer> ints =
            new Stack<Integer>();
        ints.push(1);
        int one = ints.top();
        ints.pop();
    }
}

Inside Classes

Node class could live inside

ListStack

public class ListStack<T>{
    Node top;
    public void push(T t){...}
    public void pop(){...}
    public T top(){...}

    static class Node<X>{
        X data; Node<X> next;
        public Node(X data, Node<X> next){
            this.data = data;
            this.next = next;
        }
    }
}

More on inside classes on
Wednesday
Note: Contiguous vs. Non-contiguous memory

<table>
<thead>
<tr>
<th>Array-based Stack</th>
<th>Node-based Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>1024</td>
<td>1024</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>size</td>
<td>top</td>
</tr>
<tr>
<td>3</td>
<td>4048</td>
</tr>
<tr>
<td>2048</td>
<td>2048</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>data</td>
<td>data</td>
</tr>
<tr>
<td>4048</td>
<td>11</td>
</tr>
<tr>
<td>2052</td>
<td>4048</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>length</td>
<td>next</td>
</tr>
<tr>
<td>5</td>
<td>5050</td>
</tr>
<tr>
<td>2044</td>
<td>4052</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>[0]</td>
<td>data</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>4048</td>
<td>4600</td>
</tr>
<tr>
<td>[1]</td>
<td>next</td>
</tr>
<tr>
<td>10</td>
<td>null</td>
</tr>
<tr>
<td>4052</td>
<td>4604</td>
</tr>
<tr>
<td>[2]</td>
<td>data</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>4056</td>
<td>5050</td>
</tr>
<tr>
<td>[3]</td>
<td>next</td>
</tr>
<tr>
<td>0</td>
<td>null</td>
</tr>
<tr>
<td>4060</td>
<td>5054</td>
</tr>
<tr>
<td>[4]</td>
<td>data</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>4064</td>
<td>5050</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

s = new Stack();
s.push(4);
s.push(10);
s.push(5);
s.pop();
s.push(11);
Get in Line

Queues are pervasive in computing and life

▷ Examples?
▷ Semantics?

Source: kittylittered
Queues

Support 4 operations

- enqueue(x): x enters at the back
- dequeue(): front leaves
- getFront(): return who’s in front
- isEmpty(): true when nothing is in it, false o/w

Goal:

- Worst case $O(1)$ for all ops
- $O(n)$ space

ListQueue: Ideas

- **Draw pictures** showing data changes for the following code
- **Draw the Nodes** and connections at each step
- **Decide what parts of the queue** need to be tracked with fields

```java
ListQueue<String> bsg = new ListQueue<String>();
bsg.enqueue("Adama"); // Add to back
bsg.enqueue("Tye");
bsg.enqueue("Starbuck");
bsg.dequeue(); // Remove from front
String col = bsg.getFront(); // Who’s in front
bsg.dequeue();
bsg.enqueue("Apollo");
bsg.enqueue("Baltar");
bsg.dequeue();
bsg.enqueue("Number 6");
bsg.dequeue();
```
Queue Picture Demo

- In weiss/nonstandard/ListQueue.java
- Also in code pack from last week
- Uses ListNode.java, more verbose Node
- JGrasp can draw these reasonably well
Create a LinkedQueue with Nodes

class LinkedQueue<T>{
    Node<T> front, back;
    public LinkedQueue();
    public void enqueue(T x);  // x enters a back
    public void dequeue();      // front leaves
    public T getFront();        // return who’s in front
    public boolean isEmpty();   // true when empty
}

Consider

- Worst case $O(1)$ for all ops
- How to remove the front?
- How to add to the back?
ArrayQueue: Ideas

Use an array / ArrayList to implement a queue.

- Easy… or is it? Beware of memory use: $O(N)$
- Draw pictures, figure out fields

```java
ArrayQueue<String> bsg = new ArrayQueue<String>();
bsg.enqueue("Adama"); // Add to back
bsg.enqueue("Tye");
bsg.enqueue("Starbuck");
bsg.dequeue(); // Remove from front
String col = bsg.getFront(); // Who’s in front
bsg.dequeue();
bsg.enqueue("Apollo");
bsg.enqueue("Baltar");
bsg.dequeue();
bsg.enqueue("Number 6");
bsg.dequeue();
String doc = bsg.getFront(); // Who’s in front
bsg.enqueue("Boomer");
bsg.enqueue("Helo");
```
ArrayQueue: Code

Use an array / ArrayList to implement a queue.

- Easy... or is it?

Define

- ArrayQueue data members
- enqueue(x): x enters at the back
- dequeue(): front leaves
- getFront(): return who’s in front
- isEmpty(): true when nothing is in it, false o/w

Goals

- Amortized $O(1)$ for all ops
- $O(N)$ space
- Worst-case $O(N)$ enqueue is fine
- Most enqueue() ops should be $O(1)$
- Control memory use
- Hint: track index of front and rear, wrap arrays around
ArrayQueue Demo

- In weiss/nonstandard/ArrayQueue.java
- Slightly modified demo version in today's code pack
- Uses plain java arrays, not ArrayList
- Array doubling in size done manually

Note interesting functions in Weiss’s version

```java
private int increment(int x){...
private void doubleQueue() { ... }
```
jGrasp Drawing

Drawing isn’t uber smart

- May have to manually turn on display of some fields
  - back in ListQueue
  - Wrench Button -> Fields Display
- Doesn’t get nested arrays or ArrayLists (row/col for Board)
Next Time

Lists Proper: singly and doubly linked lists