CS 310: Stacks/Queues by Arrays/Links

Chris Kauffman

Week 3-2
Logistics

At Home
- Weiss Ch 15 on ArrayLists
- Weiss Ch 16 Stacks/Queues
- Weiss Ch 17 Linked Lists (next time)
- HW 1 Milestone: Due Tomorrow by 11:59pm
- HW 1 Final: Due 8 days
- Questions on HW 1?

Goals Today
- Finish up ArrayList
- Implementation of Stacks and Queues

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>
Simple structure, supports few operations:

- `T s.getTop()`: return whatever is on top
- `s.push(T x)`: put `x` on top
- `void s.pop()`: remove whatever is on top
- `boolean s.isEmpty()`: true when nothing is in it, false o/w

Questions

- Examples of stacks?
- How would you implement a stack using arrays?
Array Based Implementation

Just use ArrayList to make an AStack

class AStack<T>{
    private ArrayList<T> stuff;
    public AStack(); // Constructor
    public void push(T x); // Like add(x)
    public void pop(); // Like remove(size()-1)
    public T getTop(); // Like get(size()-1)
    public boolean isEmpty(); // Like size()==0
}

See: weiss/nonstandard/ArrayStack.java

Work It

- Stacks: more or less functionality than ArrayList?
- Worst and Amortized Complexity of stack operations?
- Can we do better?
Nodes

To get worst-case $O(1)$ push, need to change the underlying representation of the stack implementation.

- Simplest unit to support linked data structure
- ListNode in text
- `Cons box` in Lisp
- Tracks a piece of data and the next node in a sequence
- String them together by setting next

Node Class

```java
class Node<T>{
    public T data;
    public Node<T> next;
    public Node(T d, Node<T> n){
        this.data = d;
        this.next = n;
    }
}
```
Linked Nodes

Can string Nodes together by manipulating the next field

class Node<T>{
    public T data;
    public Node<T> next;

    public Node(T d, Node<T> n ){
        this.data = d;
        this.next = n;
    }
}

Node<Integer> n3 =
    new Node<T>(10,null);
Node<Integer> n2 =
    new Node<T>(22, n3);
Node<Integer> n1 =
    new Node<T>( 5, n2);
Node<Integer> head = n1;
Implement a Stack with linked Nodes

class LinkedStack<T> {
    // Fields, probably involving nodes
    public LinkedStack(); // Constructor
    public void push(T x); // Push an element
    public void pop(); // Pop an element
    public T getTop(); // Return top element
    public boolean isEmpty(); // True only when empty
}

Consider

- Which end of the stack needs to be tracked
- Is a size required
Implementations of Stacks

Weiss Textbook Source

- package weiss.nonstandard.*
  - Stack interface, ArrayStack and ListStack implementations
  - ListNode and LinkedList classes
- package weiss.util.*
  - Reimplements java.util.* collections
  - Stack.java is based on arrays
- Included in today’s code pack

Java

- Deque interface - slight generalization of stack/queue
- ArrayDeque implements with arrays
- LinkedList implements with linked nodes
Stack Implementation with Nodes

**Generic**

Stacks can hold any type of thing

```java
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

LinkedStack

```java
public class LinkedStack<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;
        }
    }
}
```
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 ) = new Stack();</td>
<td>( s ) = new Stack();</td>
</tr>
<tr>
<td>( s ).push(4);</td>
<td>( s ).push(4);</td>
</tr>
<tr>
<td>( s ).push(10);</td>
<td>( s ).push(10);</td>
</tr>
<tr>
<td>( s ).push(5);</td>
<td>( s ).push(5);</td>
</tr>
<tr>
<td>( s ).pop();</td>
<td>( s ).pop();</td>
</tr>
<tr>
<td>( s ).push(11);</td>
<td>( s ).push(11);</td>
</tr>
</tbody>
</table>

### Array-based Stack

- \( s \) is an array at 2048.
- \( size \) is 3.
- \( data \) is a doubly-linked list:
  - \([0]\) 4
  - \([1]\) 10
  - \([2]\) 11
  - \([3]\) 0
  - \([4]\) 0
- \( length \) is 5.

### Node-based Stack

- \( topNode \) is at 4048.
- \( data \) is a singly-linked list:
  - \( 11 \) at 4048
  - \( 4 \) at 4600
  - \( 10 \) at 5050
- \( next \) is:
  - \( null \) at 4604
  - \( 4600 \) at 5054
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

LinkedQueue: 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
LinkedQueue<String> bsg = new LinkedQueue<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

```java
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