CS062
DATA STRUCTURES AND ADVANCED PROGRAMMING

11: Stacks and Queues

Alexandra Papoutsaki
she/her/hers
Lecture 11: Stacks and Queues

- Stacks
- Queues
- Applications
- Java Collections

Some slides adopted from Algorithms 4th Edition and Oracle tutorials
Stacks

- Dynamic linear data structures.
- Elements are inserted and removed following the LIFO paradigm.
- **LIFO**: Last In, First Out.
  - Remove the most recent element.
- Similar to lists, there is a sequential nature to the data.

- Metaphor of cafeteria plate dispenser.
  - Want a plate? **Pop** the top plate.
  - Add a plate? **Push** it to make it the new top.
  - Want to see the top plate? **Peek**.
  - We want to make push and pop as time efficient as possible.
**Example of stack operations**

<table>
<thead>
<tr>
<th>push</th>
<th>To</th>
<th>be</th>
<th>or</th>
<th>not</th>
<th>to</th>
<th>-</th>
<th>be</th>
<th>-</th>
<th>-</th>
<th>that</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>to</td>
<td>be</td>
<td>not</td>
<td>that</td>
<td>or</td>
<td>be</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Push to top**
- **Pop from top**

**Stacks**

- Last In First Out (LIFO)
- First In First Out (FIFO)
Implementing stacks with ArrayLists

- Where should the top go to make push and pop as efficient as possible?
- The end/rear represents the top of the stack.
- To push an element `add(E element)`.
  - Adds at the end. Amortized $O^+(1)$.
- To pop an element `remove()`.
  - Removes and returns the element from the end. Amortized $O^+(1)$.
- To peek `get(size()-1)`.
  - Retrieves the last element. $O(1)$.
- If the front/beginning were to represent the top of the stack, then:
  - Push, pop would be $O(n)$ and peek $O(1)$. 
Implementing stacks with singly linked lists

- Where should the top go to make push and pop as efficient as possible?
- The head represents the top of the stack.
- To push an element `add(E element)`.
  - Adds at the head. $O(1)$.
- To pop an element `remove()`.
  - Removes and retrieves from the head. $O(1)$.
- To peek `get(0)`.
  - Retrieves the head. $O(1)$.
- If the last node were to represent the top of the stack, then:
  - Push, pop, peek would all be $O(n)$. 
Implementing stacks with doubly linked lists

- Where should the top go to make push and pop as efficient as possible?
- The head represents the top of the stack.
- To push an element `addFirst(E element)`.
  - Adds at the head. $O(1)$.
- To pop an element `removeFirst()`.
  - Removes and retrieves from the head. $O(1)$.
- To peek `get(0)`.
  - Retrieves the head’s element. $O(1)$.
- If the `tail` were to represent the top of the stack, we’d need to use `addLast(E element), removeLast(), and get(size()-1)` to have $O(1)$ complexity.
- Guaranteed constant performance but memory overhead with pointers.
Implementation of stacks

- Stack.java: simple interface with push, pop, peek, isEmpty, and size methods.
- ArrayListStack.java: for implementation of stacks with ArrayLists. Must implement methods of Stack interface.
- LinkedStack.java: for implementation of stacks with singly linked lists. Must implement methods of Stack interface.
Lecture 11: Stacks and Queues

- Stacks
- Queues
- Applications
- Java Collections
Queues

- Dynamic linear data structures.
- Elements are inserted and removed following the FIFO paradigm.
- **FIFO**: First In, First Out.
  - Remove the *least* recent element.
- Similar to lists, there is a sequential nature to the data.

- Metaphor of a line of people waiting to buy tickets.
- Just arrived? **Enqueue** person to the end of line.
- First to arrive? **Dequeue** person at the top of line.
- We want to make enqueue and dequeue as time efficient as possible.
### Example of queue operations

<table>
<thead>
<tr>
<th>enqueue</th>
<th>To</th>
<th>be</th>
<th>or</th>
<th>not</th>
<th>to</th>
<th>be</th>
<th>-</th>
<th>-</th>
<th>that</th>
<th>-</th>
<th>-</th>
<th>-</th>
<th>is</th>
</tr>
</thead>
<tbody>
<tr>
<td>dequeue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To</td>
<td>be</td>
<td>or</td>
<td>not</td>
<td>to</td>
</tr>
</tbody>
</table>

**enqueue at end**

**dequeue from beginning**

**First In, First Out**
Implementing queue with ArrayLists

- Where should we enqueue and dequeue elements?
- To enqueue an element `add()` at the end of `ArrayList`. Amortized $O^+(1)$.
- To dequeue an element `remove(0)`. $O(n)$.
- What if we add at the beginning and remove from end?
  - Now dequeue is cheap ($O^+(1)$) but enqueue becomes expensive ($O(n)$).
Implementing queue with singly linked list

- Where should we enqueue and dequeue elements?
  - To enqueue an element `add()` at the head of SLL ($O(1)$).
  - To dequeue an element `remove(size() - 1)` ($O(n)$).
- What if we add at the end and remove from beginning?
  - Now dequeue is cheap ($O(1)$) but enqueue becomes expensive ($O(n)$).
- $O(1)$ for both if we have a tail pointer.
  - enqueue at the tail, dequeue from the head.
  - Simple modification in code, big gains!
  - Version that recommended textbook follows.
Implementing queue with doubly linked list

• Where should we enqueue and dequeue elements?
  • To enqueue an element \texttt{addLast()} at the tail of DLL \(O(1)\).
  • To dequeue an element \texttt{removeFirst()} \(O(1)\).
• What if we add at the head and remove from tail?
  • Both are \(O(1)\)!
• A lot of extra pointers! Also, in practice, "jumping" around the memory can increase significantly the running time.
Implementation of queues

- **Queue.java**: simple interface with `enqueue`, `dequeue`, `peek`, `isEmpty`, and `size` methods.
- **ArrayListQueue.java**: for implementation of queues with ArrayLists. Must implement methods of `Queue` interface.
- **LinkedQueue.java**: for implementation of queues with doubly linked lists. Must implement methods of `Queue` interface.
Lecture 11: Stacks and Queues

- Stacks
- Queues
- Applications
- Java Collections
Stack applications

- Java Virtual Machine.
- Basic mechanisms in compilers, interpreters (see CS101).
- Back button in browser.
- Undo in word processor.
- Postfix expression evaluation.
Postfix expression evaluation example

Example: \((52 - ((5 + 7) \times 4))\Rightarrow 52\ 5\ 7\ +\ 4\ \times\ -\)

1. \(52\) pushed
2. \(5\) pushed
3. \(7\) pushed
4. \(12\) calculated: \(v1=pop()=7\)
5. \(4\) pushed
6. \(v2=pop()=5\)
7. \(12\) calculated: \(v1=pop()=5\)
8. \(52\) pushed
9. \(v2=pop()=12\)
10. \(48\) calculated: \(v2=pop()=12\)
11. \(v1=pop()=48\)
12. \(4\) pushed
13. \(v2=pop()=52\)
14. \(4\) calculated: \(v2=pop()=52\)
15. \(peek()=4\)
Queue applications

- Spotify playlist.
- Data buffers (netflix, Hulu, etc.).
- Asynchronous data transfer (file I/O, sockets).
- Requests in shared resources (printers).
- Traffic analysis.
- Waiting times at calling center.
Lecture 11: Stacks and Queues

- Stacks
- Queues
- Applications
- Java Collections
The Java Collections Framework
Deque in Java Collections

- Do not use Stack. Deprecated class.

- Queue is an interface...

- It’s recommended to use the Deque interface instead.
  - Double-ended queue (can add and remove from either end).

```java
java.util.Deque;

public interface Deque<E> extends Queue<E>
```

- You can choose between LinkedList and ArrayDeque implementations.

  ```java
  Deque deque = new ArrayDeque(); //preferable
  ```

[https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html](https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html)
Lecture 11: Stacks and Queues

- Stacks
- Queues
- Applications
- Java Collections
ASSIGNED READINGS AND PRACTICE PROBLEMS

Readings:

- Oracle’s guides:
  - Collections: https://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - Deque: https://docs.oracle.com/javase/8/docs/api/java/util/Deque.html
  - ArrayList: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

- Recommended Textbook:
  - Chapter 1.3 (Page 126-157)

- Recommended Textbook Website:
  - Stacks and Queues: https://algs4.cs.princeton.edu/13stacks/

Code

- Lecture 11 code

Practice Problems:

- 1.3.2-1.3.8, 1.3.32-1.3.33