Outline

- Stacks
- Queues
  - Priority Queues
    - Implementation
- Conclusion
Priority Queues

- Regular Queues implement FIFO order
  - Sometimes, the arrival order shouldn't determine queue order

- A *priority queue* reorganizes contents based on some "natural ordering" (wink wink) of its contents
  - It has the same API as regular Queues
Applications of Priority Queues

- Organization: What should I do next?
  - Take selfies (important)
  - Do homework (somewhat important)
  - Shower (least important)

- In Operating Systems
  - Many programs in contention to be run on 1 CPU.
    - FIFO is fair, but it isn't great in real life. (Why?)
    - Which program to run next?

- Sorting
- Dijkstra's Shortest Path Algorithm
- Data compression (Hwk 7)
Giving a Natural Ordering to Objects

- Seems like a familiar concept...
  - Hey we've had to do this before to sort objects (Shapes)
  - Before, we ordered Shapes by their areas

- Recall the Comparable<T> interface?

```java
public interface Comparable<T> {

    /**
     * Compares this object with the specified object for order.
     * @return a negative integer, zero, or a positive integer as this object
     * is less than, equal to, or greater than the specified object.
     */
    int compareTo(T o);
}
```
A College Class

- Want: colleges with more students to be prioritized in a queue
  - Write a `compareTo()` in such a way that a college is ordered before another college if it has more students

```java
public class College implements Comparable<College> {
    protected List<Student> students;

    // other methods omitted

    /**
     * @return a negative integer, zero, or a positive integer as this object
     * is "less than," "equal to," or "greater than" the specified object.
     */
    @Override
    public int compareTo(College other) {
        if (this.students.size() == other.students.size()) {
            return 0;
        } else if (this.students.size() < other.students.size()) {
            return 1;
        } else {
            return -1;
        }
    }
}
```
College c1 = new College(2900); // filled with 2900 students
College c2 = new College(500); // filled with 500 students
College c3 = new College(60000); // filled with 60000 students

PriorityQueue<College> queue = new PriorityQueue<>();
queue.offer(c1);
queue.offer(c2);
queue.offer(c3);
queue.poll(); // c3 dequeued
queue.poll(); // c1 dequeued
queue.poll(); // c2 dequeued
Thoughts on PriorityQueue Implementation?

- Let's assume that a LinkedList<E> is used to store the queue
  - No changes to `poll(..)`, `peek(..)`, etc.

- One change: When `offer(E item)` is called:
  - May need to walk entire list to find place to insert item: $O(n)$
    - *Wait, couldn't we speed this up using binary search?*
  - Then link up the new item: $O(1)$
  - Total: $O(n)$
Analysis: PriorityQueue

- Well, a LinkedList worked well for Queues, so...
  - What if we used a LinkedList to implement a PriorityQueue

<table>
<thead>
<tr>
<th>Signature</th>
<th>Time Complexity</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean offer(E item)</td>
<td>O(n)</td>
<td>Need to traverse the LinkedList to find where item should be placed</td>
</tr>
<tr>
<td>public E remove()</td>
<td>O(1)</td>
<td>Remove from head of LinkedList</td>
</tr>
<tr>
<td>public E poll()</td>
<td>O(1)</td>
<td>Remove from head of LinkedList</td>
</tr>
<tr>
<td>public E peek()</td>
<td>O(1)</td>
<td>Look in head of LinkedList</td>
</tr>
<tr>
<td>public E element()</td>
<td>O(1)</td>
<td>Look in head of LinkedList</td>
</tr>
</tbody>
</table>
Eventually, we'll learn how to use a "heap" to implement PriorityQueue

- Heap-based PriorityQueue\(<E>\) implementation will give us:

<table>
<thead>
<tr>
<th>Signature</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean offer(E item)</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>public E remove()</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>public E poll()</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>public E peek()</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>public E element()</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

**Tradeoff:**
Dequeuing gets slower for faster enqueuing

This is the standard PriorityQueue implementation
Takeaway: Java's PriorityQueue\<E\\>  

- Used when FIFO is the wrong policy for determining order in queue  
  - Very common, it turns out

- Know this:  
  - Java's PriorityQueue\<E\\> is heap-based, not list-based  
  - Objects the PriorityQueue stores must implement Comparable\<T\\>
Dijkstra's Single-Source Shortest Paths (SSSP) Algorithm

- Used in just about every GPS navigational system

```java
/** @pre Nodes are ordered by distance */
public void SSSP(List<Node> nodes, Node source) {
    PriorityQueue<Node> que = new PriorityQueue<>();
    Node[] pred = new Node[nodes.size()];

    // initialize: set distances and build priority queue
    for (int i = 0; i < nodes.size(); i++) {
        Node n = nodes.get(i);
        double dist = (n.equals(source)) ? 0 : INFINITY;
        n.setDistFromSrc(dist);
        que.offer(n);
    }

    while (!que.isEmpty()) {
        Node curr = que.poll(); // grab the closest node
        for each (neighbor of curr) {
            double dist = curr.getDistFromSrc() + distance(curr, neighbor);
            if (dist < curr.getDistFromSrc()) {
                neighbor.setDistFromSrc(dist);
                que.offer(que.remove(neighbor)); // re-insert neighbor to update priority in que
                pred[neighbor] = curr; // save predecessor Node
            }
        }
    }
}
```
Outline

- Stacks
- Queues
- Priority Queues
- Conclusion
Conclusion

- Stacks and queues are two of the most commonly used data structures
  - Lots of problems can be simplified with their usage
  - Their usage will occur over and over again in this class and beyond

- Deep down, both are just Lists (FIFO vs. LIFO)
  - In fact less general
  - You can only operate on one end (Stack) or both ends (Queue)
  - Priority queues offer another way ordering items
What You Need to Know

- Their interfaces and implementation choices
  - How different choices of underlying List affects runtime of various operations

- How to get a PriorityQueue to automatically organize your objects
  - Comparable<T> interface

- Applications of stack and queues
  - Know when to use them when given a problem to solve
  - Look over the airport simulation code!
Administrivia 3/6

- Reminder about Friday Workday
- Hwk 3 solution posted
- Hwk 4 due 3/26
- Lab 7 postmortem (due tonight)
  - Learn about the structure of a time-based simulator
  - Strongly encouraged to implement the simulator supporting multiple runways
    - One giant queue for N runways?
    - N queues for N runways?
    - Would the choice make a difference? (Yes, but the reason might surprise you...)
Administrivia 10/25

- Hwk 3
  - Regrade due in a week!
- Hwk 4 due tonight!
- Hwk 5 (Schelling' Model) posted!
  - Due Friday 11/8; Bonus points available (+5 on midterm 2)
- Participate in CS student focus groups
  - Only 9 have signed up. Need more participation!
  - If interested, please email alchambers@pugetsound.edu
    - By 10/30!!!!
Hwk 5 (Schelling' Model)
   • Due Friday 11/8; Bonus points available (+5 on midterm 2)

Lab tomorrow (Melody)
   • Stack and Queue programming
   • Bring your API handouts

Last time...
   • Stack usage, palindrome checker
   • Today: Dijkstra's algorithm, stack implementation
Melody Lab Post-mortem

- Tricky size() issue (and how I got around it)
- Repetition in play() can be challenging

Last time...

- Dijkstra's expression solver, stack implementation

Today

- Queue implementation
- PriorityQueue
- Dijkstra's single-source shortest paths algorithm
Last time...

- Queue implementation
- PriorityQueue
- Dijkstra's shortest paths

Today

- PriorityQueue usage and implementation (naive)
- Start recursion