YaQ, YaQ, haha! (Yet Another Queue)

- What is the dequeue policy for a Queue?
  - Why do we implement Queue with LinkedList
    - Interface and class in java. util
  - Can we remove an element other than first?

- How did queue help in word-ladder/shortest path?
  - First item enqueued/added is the one we want
  - What if different element is “best”?

- PriorityQueue is like a queue, but different dequeue policy
  - Best item is dequeued, queue manages itself to ensure operations are efficient

Digression: word ladders

- How many ladders from cart to dire as shown?
  - Enqueue dare more than once?
  - Downside?

- We want to know number of ladders that end at W:
  - What do we know initially?
  - When we put something on the queue, what do we know?
  - How do we keep track?

- Similar to tree-counting?

Digression: Bogglescore APT

- How many ways to find Aha?!
  - Each one scores points
  - Related to genomics problem

- Number of ways that AHA ends at (2,1)?
  - What do we know initially?
  - When we extend search what do we know?
  - How do we keep track?

- Similar to tree-counting?

Why use PriorityQueue?

- Implementation of several algorithms facilitated by using pq, efficient implementation helps ensure algorithm efficiency
  - Mapquest, Googlemap, shortest path
    - How is this like word-ladder? How different?
  - Connecting all outlets in a house with minimal wiring

- Data compression facilitated by using priority queue
  - Alltime best assignment in a CompSci 100 course?
    - Subject to debate, of course
  - From A-Z, soup-to-nuts, bits to abstractions
## Data Compression

- Compression is a high-profile application
  - .zip, .mp3, .jpg, .gif, .gz, ...
  - What property of MP3 was a significant factor in what made Napster work (why did Napster ultimately fail?)

- Why do we care?
  - Secondary storage capacity doubles every year
  - Disk space fills up quickly on every computer system
  - More data to compress than ever before

## More on Compression

- What’s the difference between compression techniques?
  - .mp3 files and .zip files?
  - .gif and .jpg?
  - Lossless and lossy

- Is it possible to compress (lossless) every file? Why?

- Lossy methods
  - Good for pictures, video, and audio (JPEG, MPEG, etc.)

- Lossless methods
  - Run-length encoding, Huffman, LZW, ...

## Priority Queue

- Compression motivates the study of the ADT priority queue
  - Supports two basic operations
    - add/insert -- an element into the priority queue
    - remove/delete -- the minimal element from the priority queue
  - Implementations may allow getmin/peek separate from delete
    - Analogous to top/pop, peek/dequeue in stacks, queues

- See PQDemo.java,
  - code below sorts, complexity?

```java
Scanner s = ... // initialize;
PriorityQueue<String> pq = new PriorityQueue<String>();
while (s.hasNext()) pq.add(s.next());
while (pq.size() > 0) {
    System.out.println(pq.remove());
}
```

## Priority Queue implementations

<table>
<thead>
<tr>
<th></th>
<th>Insert average</th>
<th>Getmin (delete)</th>
<th>Insert worst</th>
<th>Getmin (delete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsorted vector</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Sorted vector</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Search tree</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>Balanced tree</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
<tr>
<td>Heap</td>
<td>0(1)</td>
<td>log n</td>
<td>log n</td>
<td>log n</td>
</tr>
</tbody>
</table>

- Heap has O(1) find-min (no delete) and O(n) build heap
Priority Queue implementation

- PriorityQueue uses heaps, fast and reasonably simple
  - Why not use inheritance hierarchy as was used with Map?
  - Trade-offs when using HashMap and TreeMap:
    - Time, space
    - Ordering properties, e.g., what does TreeMap support?
  - Changing method of comparison when calculating priority?
    - Create object to replace, or in lieu of compareTo
      - Comparable interface compares this to passed object
      - Comparator interface compares two passed objects
    - Both comparison methods: compareTo() and compare()
      - Compare two objects (parameters or self and parameter)
      - Returns -1, 0, +1 depending on <, ==, >

Creating Heaps

- Heap is an array-based implementation of a binary tree used for implementing priority queues, supports:
  - add/insert, peek/getmin, remove/deleteMin, O(???)
  - Using array minimizes storage (no explicit pointers), faster too — children are located by index/position in array
- Heap is a binary tree with shape property, heap/value property
  - shape: tree filled at all levels (except perhaps last) and filled left-to-right (complete binary tree)
  - each node has value smaller than both children

Sorting w/o Collections.sort(...) public static void sort(ArrayList<String> a) {
    PriorityQueue pq = new PriorityQueue<String>();
    pq.addAll(a);
    for(int k=0; k < a.size(); k++) a.set(k, pq.remove());
}

- How does this work, regardless of pq implementation?
- What is the complexity of this method?
  - add O(1), remove O(log n)? If add O(log n)?
  - heapsort uses array as the priority queue rather than separate pq object.
  - From a big-Oh perspective no difference: O(n log n)

PriorityQueue.java (Java 5)

- What about objects inserted into pq?
  - For "min-heap", what properties must inserted objects have, e.g., insert non-comparable?
  - Change what minimal means?
  - Implementation uses heap
- If we use a Comparator for comparing entries we can make a min-heap act like a max-heap, see PQDemo
  - Where is class Comparator declaration? How used?
  - What if we didn't know about Collections.reverseOrder?
    - How do we make this ourselves?
Array-based heap
- Store "node values" in array beginning at index 1
- For node with index k
  - Left child: index 2*k
  - Right child: index 2*k+1
- Why is this conducive for maintaining heap shape?
- What about heap property?
- Is the heap a search tree?
- Where is minimal node?
- Where are nodes added? Deleted?

Thinking about heaps
- Where is minimal element?
  - Root, why?
- Where is maximal element?
  - Leaves, why?
- How many leaves are there in an N-node heap (big-Oh)?
  - O(n), but exact?
- What is complexity of find max in a min heap? Why?
  - O(n), but \( \frac{1}{2} N \)?
- Where is second smallest element? Why?
  - Near root?

Adding values to heap
- To maintain heap shape, must add new value in left-to-right order of last level
  - Could violate heap property
  - Move value "up" if too small
- Change places with parent if heap property violated
  - Stop when parent is smaller
  - Stop when root is reached
- Pull parent down, swapping isn't necessary (optimization)

Adding values, details (pseudocode)
```java
void add(Object elt)
{
    // Add elt to heap in myList
    myList.add(elt);
    int loc = myList.size() - 1;
    while (1 < loc &&
          elt < myList.get(loc/2))
    {
        myList.set(loc, myList.get(loc/2));
        loc = loc/2; // go to parent
    }
    // what's true here?
    myList.set(loc, elt);
}
```
Removing minimal element

- Where is minimal element?
  - If we remove it, what changes, shape/property?
- How can we maintain shape?
  - “last” element moves to root
  - What property is violated?
- After moving last element, subtrees of root are heaps, why?
  - Move root down (pull child up) does it matter where?
- When can we stop “re-heaping”?
  - Less than both children
  - Reach a leaf

Anita Borg 1949-2003

- “Dr. Anita Borg tenaciously envisioned and set about to change the world for women and for technology. ... she fought tirelessly for the development technology with positive social and human impact.”
- “Anita Borg sought to revolutionize the world and the way we think about technology and its impact on our lives.”