Queue applications

- Simulation, discrete-event simulation
  - How many toll-booths do we need? How many express lanes or self-checkout at grocery store? Runway access at airport?
  - Queues facilitate simulation with mathematical distributions governing events, e.g., Poisson distribution for arrival times

- Shortest path, e.g., in flood-fill to find path to some neighbor or in word-ladder
  - How do we get from "white" to "house" one-letter at a time?
    - white, while, whale, shale, shake, …?

Blob Finding with Queues

- An Alternative to Recursion
  - myGrid[row][col] = fillWith; // mark pixel
  - size++ // count pixel
  - myQueue.add(myPairGrid[row][col]);
  - while (myQueue.size() != 0){
    - Pair p = myQueue.remove();
    - for(int k=0; k < rowDelta.length; k++) {
      - row = p.row + rowDelta[k];
      - col = p.col + colDelta[k];
      - if (inRange(row, col) && myGrid[row][col] == lookFor)
        - myQueue.add(myPairGrid[row][col]);
        - myGrid[row][col] = fillWith;
        - size++;
    }
  }

Queue for shortest path (WordLadder APT)

```java
public boolean ladderExists(String[] words, String from, String to){
  Queue<String> q = new LinkedList<String>();
  Set<String> used = new TreeSet<String>();
  for(String s : words){
    if (oneAway(from, s)){
      q.add(s);
      used.add(s);
    }
  }
  while (q.size() != 0){
    String current = q.remove();
    if (oneAway(current, to)) return true;
    // add code here, what?
  }
  return false;
}
```

Shortest Path reprised

- How does use of Queue ensure we find shortest path?
  - Where are words one away from start?
  - Where are words two away from start?

- What would a recursive approach have given us?
  - What data structure does recursion seem to replace?

- Why do we need to avoid revisiting a word, when?
  - Why do we use a set for this? Why a TreeSet?
  - Alternatives?

- What if we want the ladder, not just whether it exists
  - What’s path from white to house? We know there is one.
  - Ideas? Options?