COS 126 –
Atomic Theory of Matter
Goal of the Assignment

- Calculate Avogadro’s number
  - Using Einstein’s equations
  - Using fluorescent imaging

- Input data

- Output
  - Estimate of Avogadro’s number

<table>
<thead>
<tr>
<th>Video</th>
<th>Frames</th>
<th>Blobs/Beads</th>
<th>Displacements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>7.1833</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4.7932</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2.1693</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.5287</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.4292</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Overview – Four Classes

**Blob**
- Maximal set of connected light pixels

**BeadFinder**
- Find all blobs in a JPEG image
- List all the big blobs (aka beads)

**Avogadro**
- Data analysis to estimate Avogadro’s number from the motion of beads

**BeadTracker**
- Track beads from one image to the next

**Boltzmann Avogadro**

**Command line args**

**Pictures (frames)**

**Video data**
Atomic Theory Overview

- **Brownian Motion**
  - Random collision of molecules
  - Displacement over time fits a Gaussian distribution
Atomic Theory Overview

- Avogadro’s Number
  - Number of atoms needed to equal substance’s atomic mass in grams
    - $N_A$ atoms of Carbon-12 = 12 grams
    - $N_A = 6.0221367 \times 10^{23}
  - Can calculate from Brownian Motion
    - Variance of Gaussian distribution is a function of resistance in water, number of molecules
API for representing particles (blobs) in water

- public Blob()
  - constructor
- public void add(int i, int j)
  - add pixel at i,j to Blob
- public int mass()
  - number of pixels in Blob
- public double distanceTo(Blob b)
  - Euclidean distance between the center of masses between Blobs
- public String toString()
  - a string representation of this Blob
- public static void main(String[] args)
  - unit tests all methods in the Blob data type
Blob.java

- Center of mass
- Only need three *instance variables*
  - Do *not* store the positions of every pixel in the blob
- Two alternatives:
  - number of points, x-coordinate center of mass, and y-coordinate center of mass) or
  - number of points, sum of x-coordinates, and sum of y-coordinates) needed to compute the center-of-mass

Center is not integer values!
Blob Challenges

- Format numbers in a nice way
  - `String.format("%2d (%8.4f, %8.4f)", mass, cx, cy);`
  - (Use same format in `System.out.printf()`)
  - E.g., "%6.3f" -> _2.354
  - E.g., "%10.4e" -> 1.2535e-23

- Thoroughly test
  - Create a simple main()
  - Test ALL methods
BeadFinder.java

- Locate all blobs in a given image
  - And identify large blobs (called beads)

- API
  - public BeadFinder(Picture picture, double threshold)
    - Calculate luminance (see Luminance.java, 3.1)
      - Include pixels with a luminance >= threshold
    - Find blobs with DFS (see Percolation.java, 2.4)
      - The hard part, next slide...

- public Blob[] getBeads(int minSize)
  - Returns all beads with at least minSize pixels
  - Array must be of size equal to number of beads
BeadFinder - Depth First Search

- Use boolean[][] array to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BeadFinder - Depth First Search

- Use boolean[][] array to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BeadFinder - Depth First Search

- Use boolean[][][] array to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
Use boolean[][][] array to mark visited

Traverse image pixel by pixel
- Dark pixel
  - Mark as visited, continue
- Light pixel
  - Create new blob, call DFS

DFS algorithm
- Base case: simply return if
  - Pixel out-of-bounds
  - Pixel has been visited
  - Pixel is dark (and mark as visited)
- Add pixel to current blob, mark as visited
- Recursively visit up, down, left, and right neighbors
BeadFinder - Depth First Search

- Use boolean[][] array to mark visited
- Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS
- DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
Use boolean[][] array to mark visited

Traverse image pixel by pixel
  - Dark pixel
    - Mark as visited, continue
  - Light pixel
    - Create new blob, call DFS

DFS algorithm
  - Base case: simply return if
    - Pixel out-of-bounds
    - Pixel has been visited
    - Pixel is dark (and mark as visited)
  - Add pixel to current blob, mark as visited
  - Recursively visit up, down, left, and right neighbors
BeadFinder Challenges

- Data structure for the collection of blobs
  - Store them any way you like
  - But be aware of memory use and timing
BeadFinder Challenges

- Data structure for the collection of blobs
  - Store them any way you like
  - But be aware of memory use and timing
- Array of blobs?
  - But how big should the array be?
- Linked list of blobs?
  - Memory efficient, but harder to implement
  - Avoid traversing whole list to add a blob!
- Anything else?
  - Submit your (extra) object classes if not in 4.3
BeadTracker.java

- Track beads between successive images
- Single main function
  - Take in a series of images
  - Output distance traversed by all beads for each time-step
    - For each bead found at time t+1, find closest bead at time t and calculate distance
      - Not the other way around!
      - Don’t include if distance > 25 pixels (new bead)
BeadTracker Challenges

- Reading multiple input files
  - java-introcs BeadTracker run_1/*.jpg
  - Expands files in alphabetical order
  - End up as args[0], args[1], ...

- Avoiding running out of memory
  - How?

- Recompiling
  - Recompile if Blob or BeadFinder change
BeadTracker Challenges

- Reading multiple input files
  - `java-introcs BeadTracker run_1/*.jpg`
  - Expands files in alphabetical order
  - End up as `args[0], args[1], ...`

- Avoiding running out of memory
  - Do *not* open all picture files at same time
  - Various ways to do this

- Recompiling
  - Recompile if Blob or BeadFinder change
BeadTracker Challenges

- Avoid running out of memory (1)

File containing frame $t_0$
- beads $t_0$
- $radial\ displacement_0$

File containing frame $t_1$
- beads $t_1$

File containing frame $t_2$

... ...

File containing frame $t_n$

Only need two open at a time.
BeadTracker Challenges

- Avoid running out of memory (1)

File containing frame $t_0$

File containing frame $t_1$

File containing frame $t_2$

... File containing frame $t_n$

beads $t_1$

beads $t_2$

radial displacement $t_1$

Only need two open at a time.
BeadTracker Challenges

- Avoid running out of memory (2)

File containing frame $t_0$

File containing frame $t_1$

File containing frame $t_2$

File containing frame $t_n$

beads $t_0$

beads $t_1$

beads $t_2$

radial displacement$_0$, radial displacement$_1$

No need to re-find beads – use the beads $t_1$ found in radial displacement$_0$ computation in the computation of radial displacement$_1$.
Analyze Brownian motion of all calculated displacements
  - Intricate formulas, all given
  - Be careful about units in the math, convert pixels to meters, etc.

Can test without the other parts working
  - We provide sample input files
  - Can work on it while waiting for help
Conclusion: Final Tips

- Avoiding subtle bugs in BeadFinder
  - Double check what happens at corner cases (e.g., at boundary pixels, or when luminance == tau, or mass == cutoff)

- Common errors in BeadFinder
  - NullPointerException
  - StackOverflowError (e.g., if no base case)
  - No output (need to add prints)

- Look at checklist Q&A
Conclusion: Final Tips

- Testing with a main()
  - Blob
    - Test all methods
  - BeadFinder, BeadTracker, and Avogadro
    - Must have a main() that can handle I/O described in Testing section of checklist

- Timing analysis
  - Look at feedback from earlier assignments
  - \( N = \) number of pixels (not frames)
  - How can you run 100 frames?