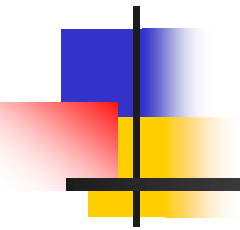


COS 126 – Atomic Theory of Matter

Goal of the Assignment

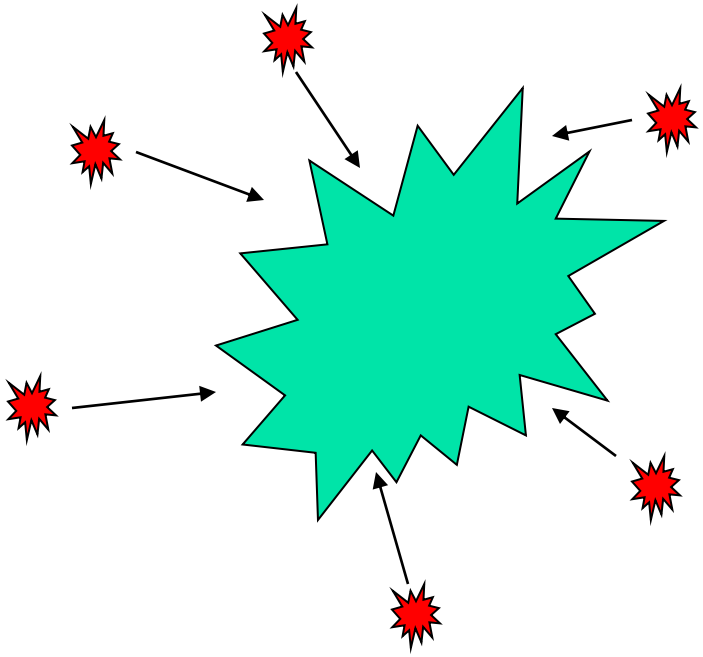
- Calculate Avogadro's number
  - Using Einstein's equations
  - Using fluorescent imaging
- Input data
  - Sequence of images
  - Each image is a rectangle of pixels
  - Each pixel is either light or dark
- Output
  - Estimate of Avogadro's number

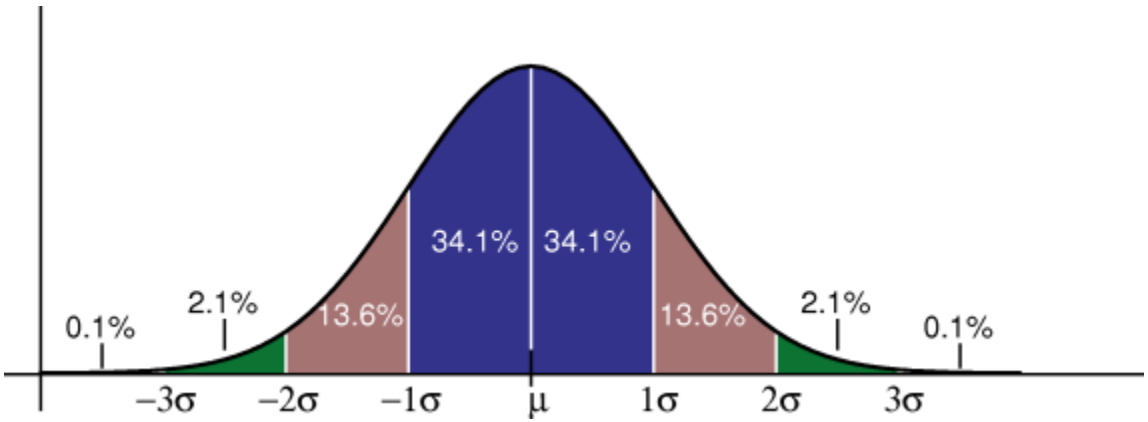
Assignment: Four Programs

- Blob data type
  - Maximal set of connected light pixels
- BlobFinder
  - Find *all* blobs in a JPEG image
  - List all the *big* blobs (aka beads)
- BeadTracker
  - Track beads from one image to the next
- Avogadro
  - Data analysis to estimate Avogadro's number from the motion of beads

Atomic Theory Overview

- Brownian Motion
  - Random collision of molecules
  - Displacement over time fits a Gaussian distribution



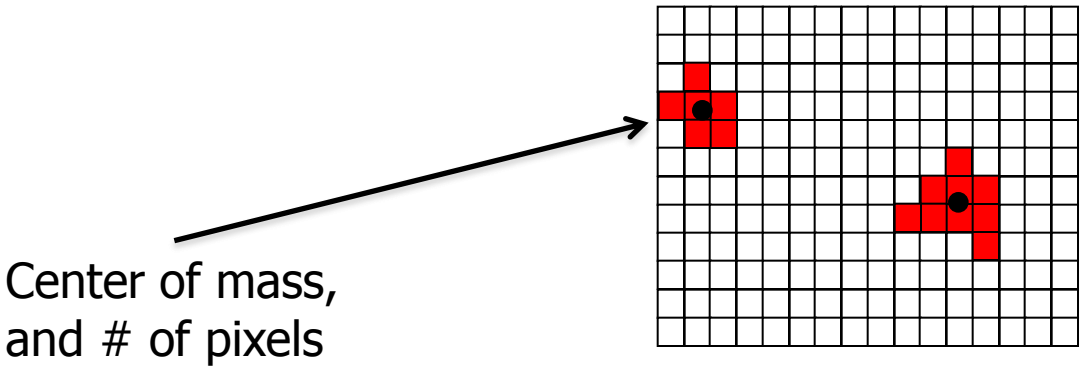


Atomic Theory Overview

- Avogadro's Number
  - Number of atoms needed to equal substance's atomic mass in grams
    - N_A atoms of Carbon-12 = 12 grams
  - Can calculate from Brownian Motion
    - Variance of Gaussian distribution is a function of resistance in water, number of molecules

Blob.java

- API for representing particles (blobs) in water
  - public Blob()
  - public void add(int i, int j)
  - public int mass() // number of pixels
  - public double distanceTo(Blob b) // from center (average)
  - public String toString()
- Only need *three* values to efficiently store
  - Do *not* store the positions of every pixel in the blob



Blob Challenges

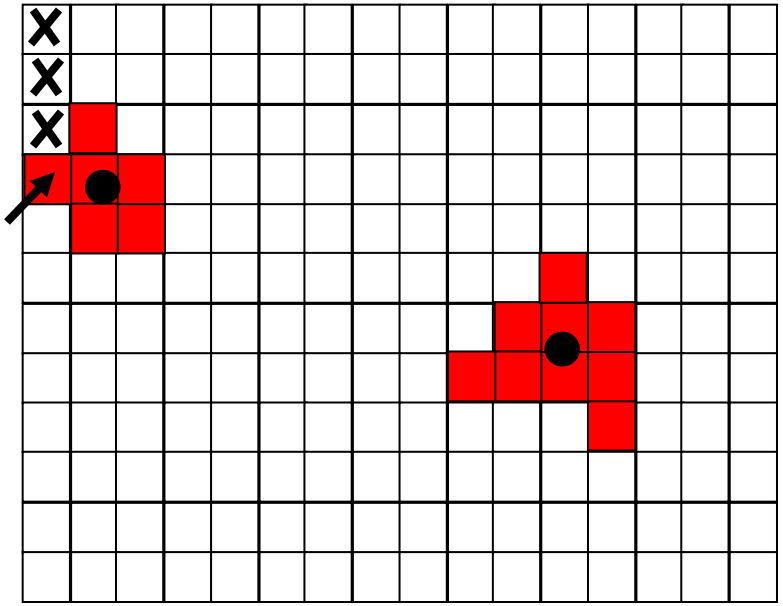
- Format numbers in a nice way
  - String.format("%2d (%8.4f, %8.4f)", mass, cx, cy);
  - (Use same format in System.out.printf())
  - E.g., "%6.3f" -> _2.354
  - E.g., "%10.4e" -> 1.2535e-23
- Thoroughly test
  - Create a simple main()

BlobFinder.java

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

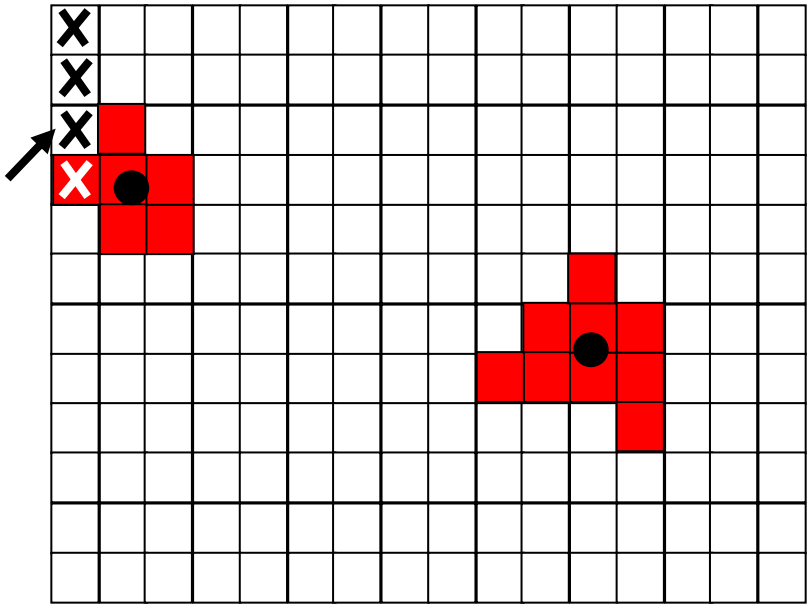
BlobFinder - Depth First Search

- 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



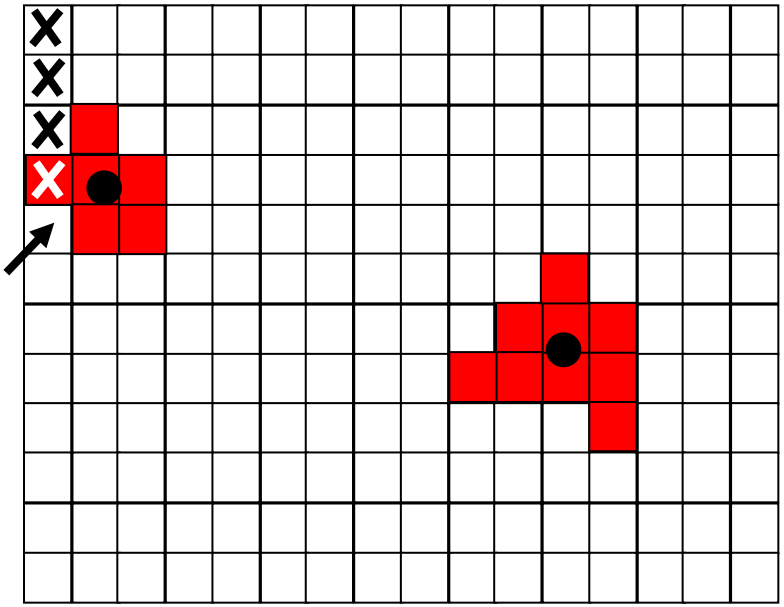
BlobFinder - Depth First Search

- 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



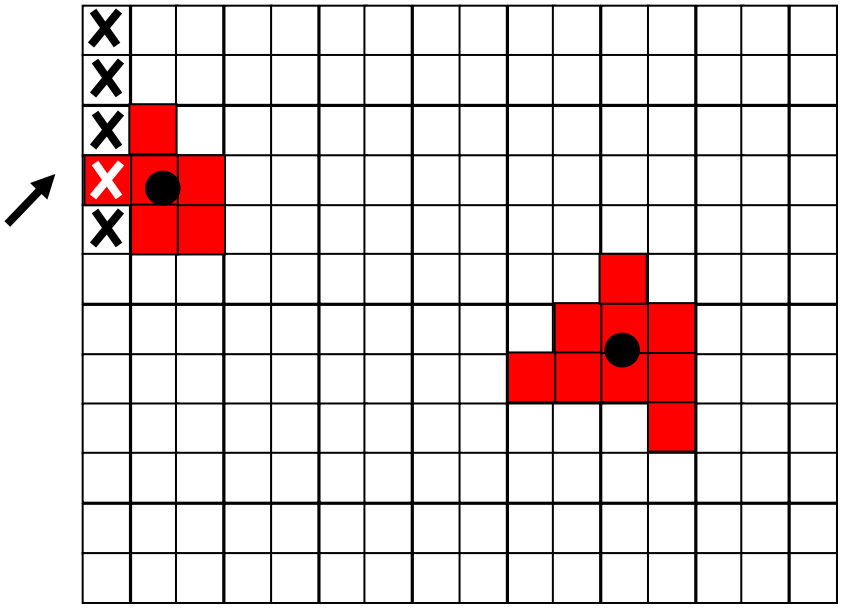
BlobFinder - Depth First Search

- 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



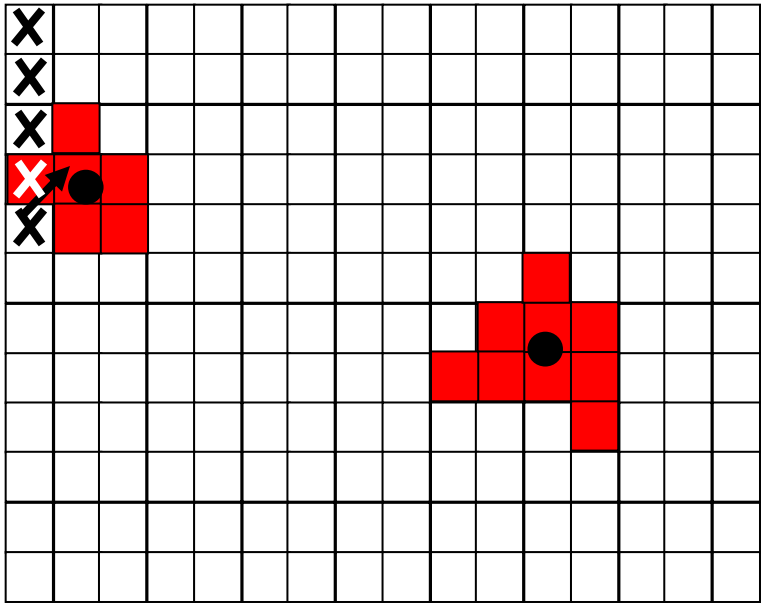
BlobFinder - Depth First Search

- 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



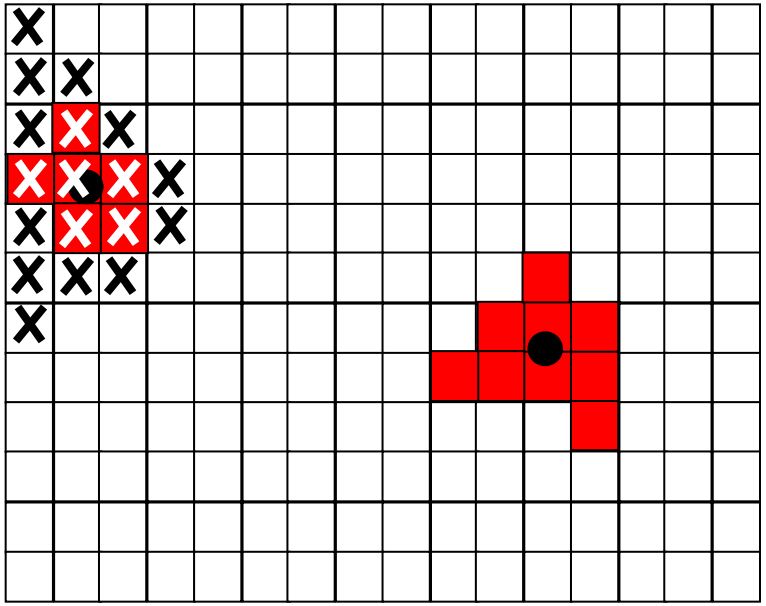
BlobFinder - Depth First Search

- 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



BlobFinder - 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



BlobFinder Challenges

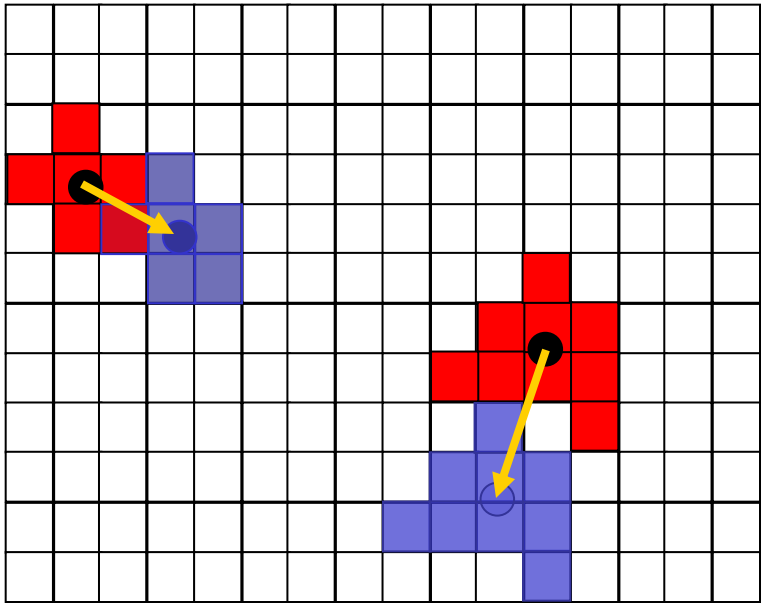
- Data structure for the collection of blobs
  - Store them any way you like
  - But be aware of memory use and timing

BlobFinder 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 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 BlobFinder change

BeadTracker Challenges

- Reading multiple input files
  - java 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
  - Only two need to be open at a time
- Recompiling
  - Recompile if Blob or BlobFinder change

Avogadro.java

- Analyze Brownian motion of all calculated displacements
  - Lots of crazy formulas, all given, pretty straightforward
  - 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 BlobFinder
  - Double check what happens at corner cases (e.g. at boundary pixels, or when luminance == tau, or mass == cutoff)
- Common errors in BlobFinder
  - 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()
  - BlobFinder, 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
  - BeadTracker is time sink, so analyze that
- How can you run 100 frames?