Spring 2024

Course Title: Sequential and parallel Algorithm
Course Number & Section: COMP 275-01
Group Programming Project Assignment #1: Percolation

Objective

This assignment entails writing a program to estimate the value of the *percolation threshold* by leveraging the Monte Carlo simulation concept.

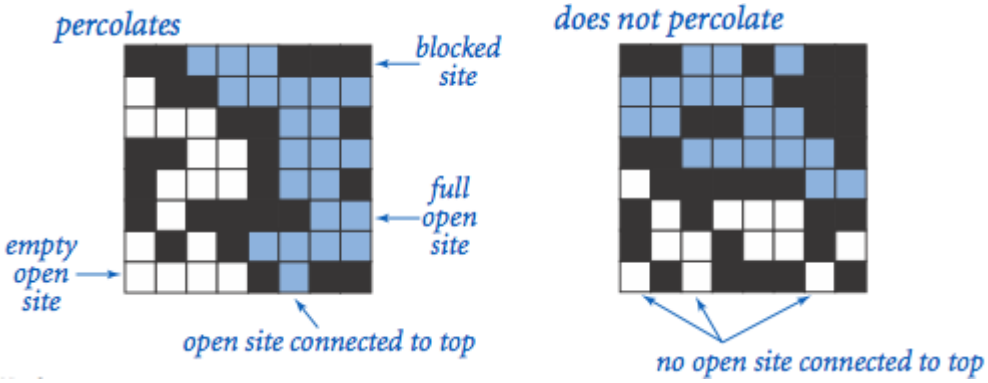
Definition and Brief Intuitive Inquiry on Percolation.

Definition: In physics, chemistry, and materials science, percolation refers to the movement and filtering of fluids through porous materials.

Intuitive Inquiries: Given a composite system comprised of randomly distributed insulating and metallic materials, what fraction of the materials need to be metallic so that the composite system is an electrical conductor? Given a porous landscape with water on the surface (or oil below), under what conditions will the water be able to drain through to the bottom (or the oil to gush through to the surface)? Scientists have defined an abstract process known as *percolation* to model such situations.

The model Description.

In this assignment you are to model a percolation system using an *N*-by-*N* grid of *sites(arrays)*. Each site is either *open* or *blocked*. A *full*-site is an open-site that connect to an open-site in the top row via a chain of neighboring (left, right, up, down) open sites. A system is said to *percolates* if there is a full site in the bottom row. In other words, a system percolates if we fill all open sites connected to the top row, and that process fills some open sites on the bottom row. for the insulating/metallic materials example, the open sites correspond to metallic materials. So, a metallic system that percolates has a metallic path from top to bottom, with full sites conducting. For the porous substance example, the open sites correspond to "empty" space through which water might flow. So, such a system percolates when it lets water fill open sites, flowing from top to bottom.

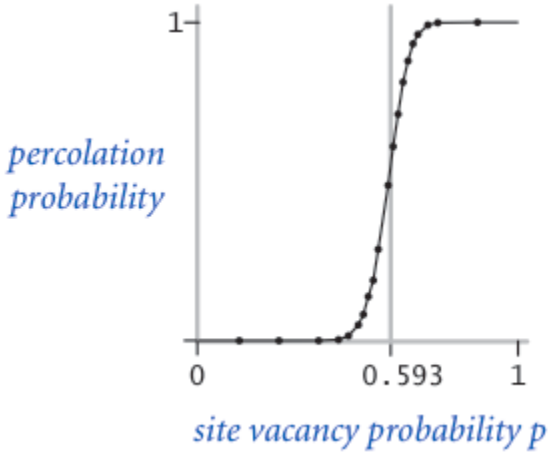


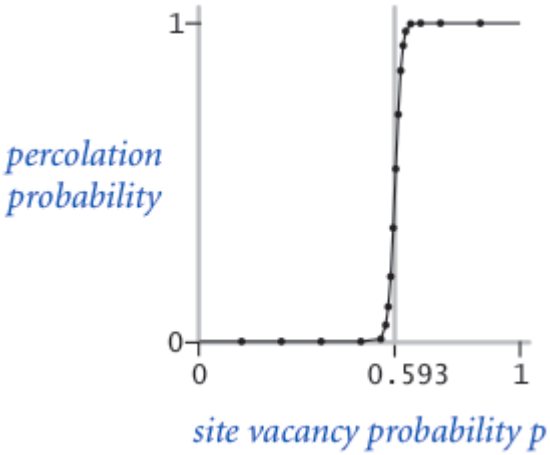
The problem.

In a famous scientific problem, researchers are interested in the following question:

- If sites are "independently" set to be open with probability p (and therefore blocked with probability $1 - p$), what is the probability that the system percolates? When p equals 0, the system does not percolate; when p equals 1, the system percolates.

The plots below show the site vacancy probability p versus the percolation probability for a 20-by-20 random grid (left) and a 100-by-100 random grid (right).





When N is sufficiently large, there is a *threshold* value p^* such that when $p < p^*$, a random N-by-N grid rarely percolates, and when $p > p^*$, a random N-by-N grid almost always percolates. ***No mathematical solution for determining the percolation threshold p* has been derived yet. Your task is to write a computer program to estimate p*.***

Resources Provides

The resources (code & test input data) that will aid you to accomplish the objectives of this assignment are uploaded to Moodle.

Among the code resources, you are to complete the following APIs:

1. **Percolation Data Type(Used for sites simulation):**

```
public class Percolation {
   public Percolation(int N)        // create N-by-N grid, with all sites blocked
   public void open(int i, int j)   // open site (row i, column j) if it is not open already
   public boolean isOpen(int i, int j)     // is site (row i, column j) open?
   public boolean isFull(int i, int j)     // is site (row i, column j) full?
   public boolean percolates()             // does the system percolate?
}
```

Hint on corner cases for percolation API. By convention, the row and column indices *i* and *j* are integers between 0 and $N - 1$, where (0, 0) is the upper-left site: Throw a java.lang.IndexOutOfBoundsException if any argument to open(), isOpen(), or isFull() is outside its prescribed range. The constructor should throw a java.lang.IllegalArgumentException if $N \leq 0$.

2. Percolation Statistics API (used for simulating p*→ Monte Carlo Simulation):

```
public class PercolationStats {
  public PercolationStats(int N, int T)   // perform T independent experiments on an N-by-N grid
  public double mean()                    // sample mean of percolation threshold
  public double stddev()                  // sample standard deviation of percolation threshold
  public double confidenceLow()           // low endpoint of 95% confidence interval
  public double confidenceHigh()          // high endpoint of 95% confidence interval
}
```

Hints: The constructor should throw a java.lang.IllegalArgumentException if either $N \leq 0$ or $T \leq 0$.
The constructor should take two arguments *N* and *T*, and perform *T* independent computational experiments (discussed above) on an *N*-by-*N* grid. Using this experimental data, it should calculate the mean, standard deviation, and the *95% confidence interval* for the percolation threshold. Use *standard random* from stdlib.jar to generate random numbers; use *standard statistics* from stdlib.jar to compute the sample mean and standard deviation.

Example values after creating **PercolationStats(200, 100)**
mean() = 0.5929934999999997
stddev() = 0.00876990421552567
confidenceLow() = 0.5912745987737567
confidenceHigh() = 0.5947124012262428

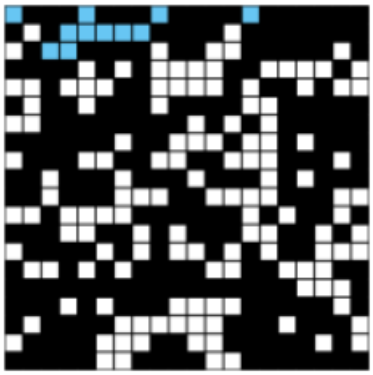
Example values after creating **PercolationStats(200, 100)**
mean() = 0.592877
stddev() = 0.009990523717073799
confidenceLow() = 0.5909188573514536
confidenceHigh() = 0.5948351426485464

Example values after creating **PercolationStats(2, 100000)**
mean() = 0.6669475
stddev() = 0.11775205263262094
confidenceLow() = 0.666217665216461
confidenceHigh() = 0.6676773347835391

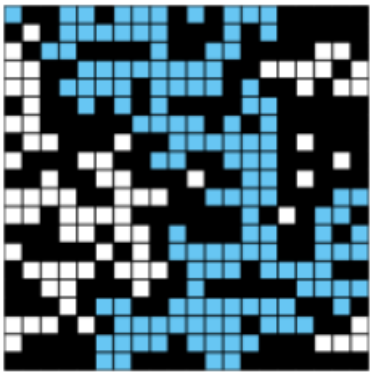
Monte Carlo simulation. To estimate the percolation threshold, consider the following computational experiment:

- Initialize all sites to be blocked.
- Repeat the following until the system percolates:
  - Choose a site (row *i*, column *j*) uniformly at random among all blocked sites.
  - Open the site (row *i*, column *j*).
- The fraction of sites that are opened when the system percolates provide an estimate of the percolation threshold.

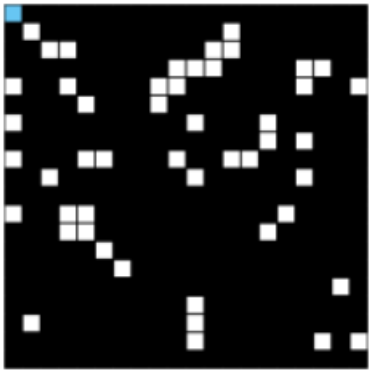
For example, if sites are opened in a 20-by-20 grid according to the snapshots below, then our estimate of the percolation threshold is 204/400 = 0.51 because the system percolates when the 204th site is opened.



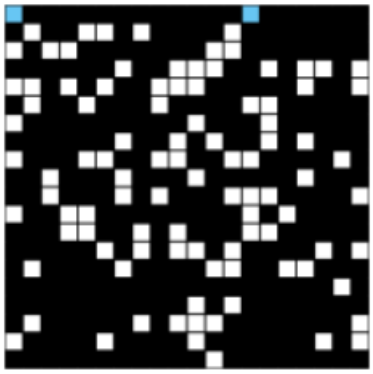
150 open sites



204 open sites



50 open sites



100 open sites

By repeating this computation experiment *T* times and averaging the results, we obtain a more accurate estimate of the percolation threshold. Let *xt* be the fraction of open sites in computational experiment *t*. The sample mean (μ) is used to estimate of the percolation threshold; the sample standard deviation σ measures the sharpness of the "threshold".

$$\mu = \frac{x_1 + x_2 + \cdots + x_T}{T}, \quad \sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_T - \mu)^2}{T - 1}$$

Assuming T is sufficiently large (say, at least 30), the following provides a 95% confidence interval for the percolation threshold:

$$\left[\mu - \frac{1.96\sigma}{\sqrt{T}}, \ \mu + \frac{1.96\sigma}{\sqrt{T}}\right]$$

Test Input Data.

- The data input for test you program is also available in the assignment folder on Moodle.

Analysis of running time and memory usage.

Implement the Percolation data type using the *quick-find* data type **QuickFindUF.java**.

- Use the *stopwatch data type* **Stopwatch.java** to measure the total running time of **PercolationStats.java**. How does doubling N affect the total running time? How does doubling T affect the total running time? Give a formula (using tilde notation) of the total running time on your computer (in seconds) as a single function of both N and T.
- Using the 64-bit memory-cost model from lecture and Section 1.4 of the recommended textbook for this class, give the total memory usage in bytes (using tilde notation) that a Percolation object uses to model an N-by-N percolation system. Count all memory that is used, including memory for the union-find data structure.

Now, implement the Percolation data type using the *weighted quick-union* data type **WeightedQuickUnionUF.java**. Answer the same questions in the previous two bullets.

Deliverables

1. Percolation.java → 40 pts
2. PercolationStats.java → 40 pts
3. Analysis report → 20 pts