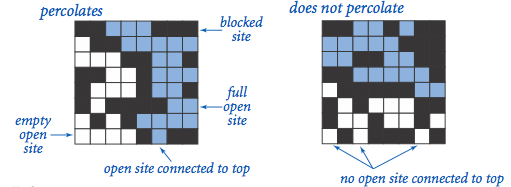
## Programming Assignment 1: Percolation

Write a program to estimate the value of the percolation threshold via Monte Carlo simulation.

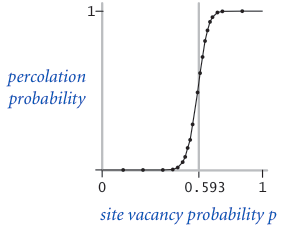
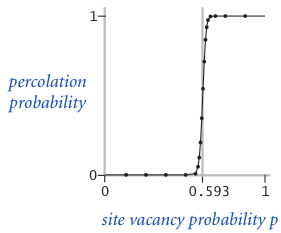
**Install a Java programming environment.** Install a Java programming environment on your computer by following these step-by-step instructions for your operating system [ [Mac OS X](http://algs4.cs.princeton.edu/mac) · [Windows](http://algs4.cs.princeton.edu/windows) · [Linux](http://algs4.cs.princeton.edu/linux) ]. After following these instructions, the commands javac-algs4 and java-algs4 will classpath in both [stdlib.jar](http://algs4.cs.princeton.edu/code/stdlib.jar) and [algs4.jar](http://algs4.cs.princeton.edu/code/algs4.jar): the former contains libraries for reading data from standard input, writing data to standard output, drawing results to standard draw, generating random numbers, computing statistics, and timing programs; the latter contains all of the algorithms in the textbook.

**Percolation.** Given a composite systems 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.** We model a percolation system using an N-by-N grid of sites. Each site is either open or blocked. A full site is an open site that can be connected to an open site in the top row via a chain of neighboring (left, right, up, down) open sites. We say the system 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 site on the bottom row. (For the insulating/metallic materials example, the open sites correspond to metallic materials, so that a 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 that a system that percolates 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 20-by-20 random grid (left) and 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 almost never percolates, and when p > p\*, a random N-by-N grid almost always percolates. No mathematical solution for determining the percolation threshold p\* has yet been derived. Your task is to write a computer program to estimate p\*.

**Percolation data type.** To model a percolation system, create a data type Percolation with the following API:

**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? **public static void main(String[] args** // test client (optional) **}**

Corner cases.  By convention, the row and column indices i and j are integers between 1 and N, where (1, 1) is the upper-left site: Throw a java.lang.IndexOutOfBoundsExceptionif any argument to open(), isOpen(), or isFull() is outside its prescribed range. The constructor should throw a java.lang.IllegalArgumentException if N ≤ 0.

Performance requirements.  The constructor should take time proportional to N2; all methods should take constant time plus a constant number of calls to the union-find methodsunion(), find(), connected(), and count().

**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 provides an estimate of the percolation threshold.

For example, if sites are opened in a 20-by-20 lattice 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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ercolation 50 sites  50 open sites | ercolation 100 sites  100 open sites | ercolation 150 sites  150 open sites | ercolation 204 sites  204 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 provides an estimate of the percolation threshold; the sample standard deviation σ measures the sharpness of the threshold.

stimating the sample mean and variance

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

5% confidence interval for percolation threshold

To perform a series of computational experiments, create a data type PercolationStats with the following API.

**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 confidenceLo()**  // low endpoint of 95% confidence interval  **public double confidenceHi()**  // high endpoint of 95% confidence interval  **public static void main(String[] args)**  // test client (described below) **}**

The constructor should throw a java.lang.IllegalArgumentException if either N ≤ 0 or T ≤ 0.

Also, include a main() method that takes two command-line arguments N and T, performs T independent computational experiments (discussed above) on an N-by-N grid, and prints out the mean, standard deviation, and the 95% confidence interval for the percolation threshold. Use standard random from our standard libraries to generate random numbers; usestandard statistics to compute the sample mean and standard deviation.

% **java PercolationStats 200 100** mean = 0.5929934999999997 stddev = 0.00876990421552567 95% confidence interval = 0.5912745987737567, 0.5947124012262428 % **java PercolationStats 200 100** mean = 0.592877 stddev = 0.009990523717073799 95% confidence interval = 0.5909188573514536, 0.5948351426485464 % **java PercolationStats 2 10000** mean = 0.666925 stddev = 0.11776536521033558 95% confidence interval = 0.6646167988418774, 0.6692332011581226 % **java PercolationStats 2 100000** mean = 0.6669475 stddev = 0.11775205263262094 95% confidence interval = 0.666217665216461, 0.6676773347835391

**Analysis of running time and memory usage (optional and not graded).** Implement the Percolation data type using the quick-find algorithm [QuickFindUF.java](http://algs4.cs.princeton.edu/15uf/QuickFindUF.java.html) from algs4.jar.

* Use the stopwatch data type from our standard library to measure the total running time of PercolationStats. 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, 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 algorithm [WeightedQuickUnionUF.java](http://algs4.cs.princeton.edu/15uf/WeightedQuickUnionUF.java.html) from algs4.jar. Answer the questions in the previous paragraph.

**Deliverables.** Submit only Percolation.java (using the weighted quick-union algorithm as implemented in the WeightedQuickUnionUF class) and PercolationStats.java. We will supply stdlib.jar and WeightedQuickUnionUF. Your submission may not call any library functions other than those in java.lang, stdlib.jar, and WeightedQuickUnionUF.

**For fun.** Create your own percolation input file and share it in the discussion forums. For some inspiration, see these [nonogram puzzles](http://photo.qip.ru/users/syusyu/2208847/?sort=visitors&page=1).

This assignment was developed by Bob Sedgewick and Kevin Wayne.   
Copyright © 2008.