Algorithms Homework Assignment #2

"Percolation"

This homework is adopted from the Princeton University "Algorithms, Part I" by Kevin Wayne, Robert Sedgewick.

Due date: Wednesday, April 2nd 2018, 9AM
- Submit softcopy on the server.

In this homework assignment, you'll write a program to estimate the value of the *percolation threshold* via Monte Carlo simulation. To do this, you need to first implement a data structure that can represent the percolation state using the Union-Find algorithm discussed during the lecture.

Template code for the homework assignment, as well as some testing code, will be provided to you;
- "Percolation.java" and "PercolationStats.java" are what you need to implement.
- "WeightedQuickUnionUF.java", "StdDraw.java", and "PercolationVisulizer.java" are provided for you. ➔ (This is already done for you. Nothing to do in these files.)
- You can download these files from e-Class website.

Your task is to correctly implement "Percolation.java" and "PercolationStats.java" so that you can verify percolation visually, as well as getting a good estimate of the value of the percolation threshold.

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.)
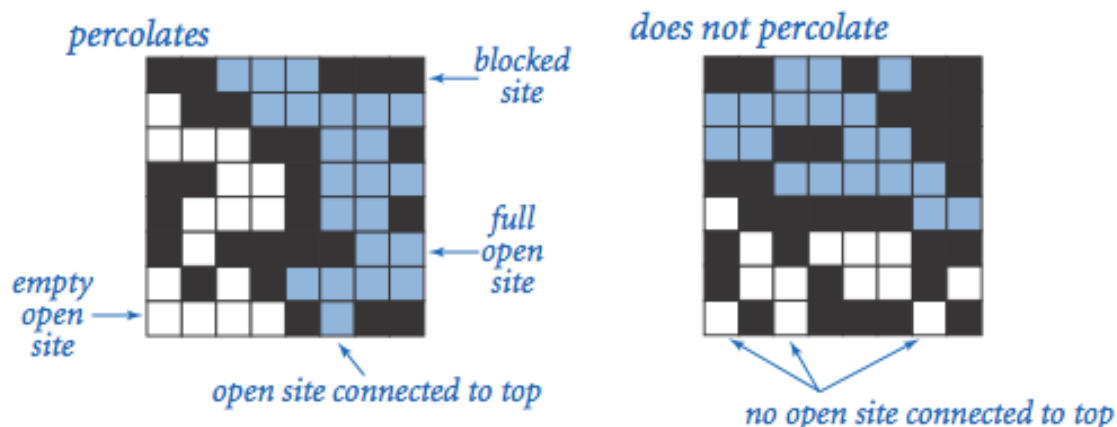
The problem. In a famous scientific problem, researchers are interested in the following question: if sites are independently set to be open with probability p (and therefore blocked with probability $1 - p$), what is the probability that the system percolates? When p equals 0, the system does not percolate; 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).
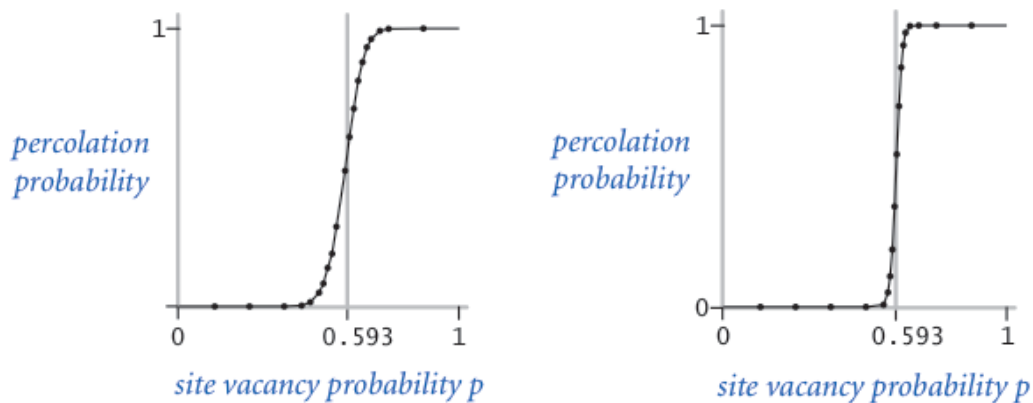
When N is sufficiently large, there is a *threshold* value p^* such that when $p < p^*$ a random N-by-N grid almost never percolates, and when $p > p^*$, a random N-by-N grid almost always percolates. No mathematical solution for determining the percolation threshold p^* has yet been derived. Your task is to write a computer program to estimate p^*.

Percolation data type. To model a percolation system, create a data type Percolation in "Percolation.java" 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 (already done for you)
}
```

You should implement the Percolation data type using the weighted quick-union algorithm as implemented in the WeightedQuickUnionUF.java provided to you.

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.IndexOutOfBoundsException if any argument to open(), isOpen(), or isFull() is outside its prescribed range. The constructor should throw a java.lang.IllegalArgumentException if $N \leq 0$. ➔ This is already done for you!

Performance requirements. The constructor should take time proportional to N^2; all methods should take constant time plus a constant number of calls to the union-find methods union(), find(), connected(), and count().

Test program. The "Percolation.java" has a main() function that can test your implementation of the Percolation data type. (➔ This is already done for you!). It reads an input file which has the N value in the first line, and an (i, j) pairs in each line for the list of open sites. It will then print out whether the system percolates or not. You can use this to check whether your implementation is correct or not.

A sample input file look like below;

```
$ cat test/input4.txt
4                                        // N
 4 1                                     // ex> (i,j) = (4,1) is open site
 3 1
 2 1
 1 1
 1 4
 2 4
 4 4
 3 4
```

Example outputs;

```
$ java Percolation testinput/input6.txt
N = 6
This system percolates
running time = 0.019 sec

$ java Percolation testinput/input8-no.txt
N = 8
This system does NOT percolate
running time = 0.019 sec

$ java Percolation testinput/input8.txt
N = 8
This system percolates
running time = 0.02 sec

$ java Percolation testinput/input50.txt
N = 50
This system percolates
running time = 0.036 sec
```

Test program2. The "PercolationVisualizer.java" is a program that can visually show the progress and result of running the Percolation test program. (➔ This is already done for you!). It works in the same way as the test program in "Percolation.java", except that it has a GUI.

Example output;

```
$ java PercolationVisualizer test/input8.txt

$ java PercolationVisualizer test/input8-no.txt
```
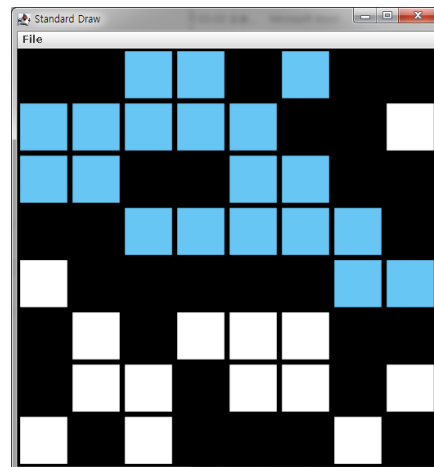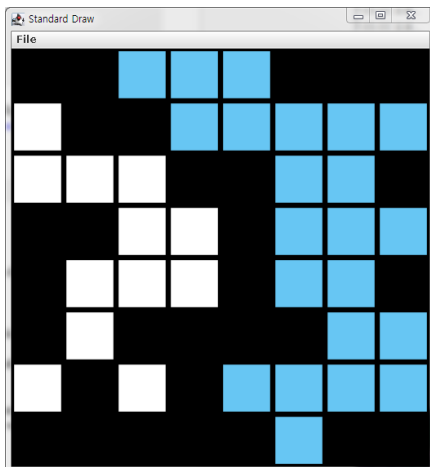

To see this output, you might need to setup X forwarding on your computer (e.g. Xming)

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).
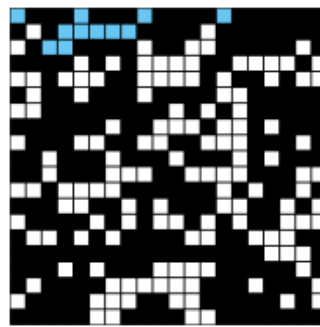- The fraction of sites that are opened when the system percolates provides an estimate of the percolation threshold.

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.
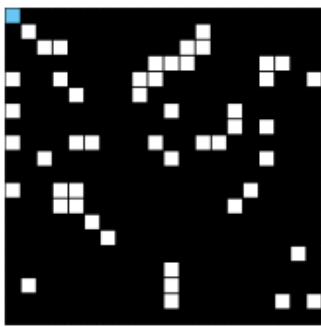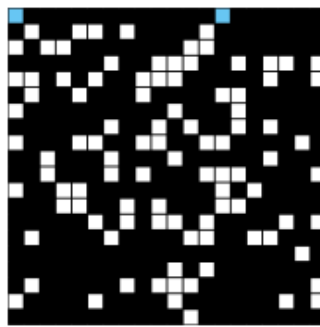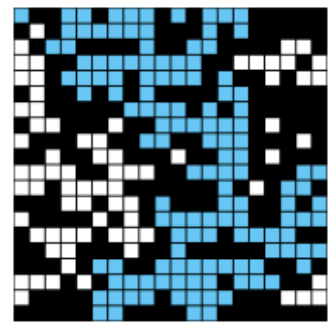

50 open sites   *100 open sites*   *150 open sites*   *204 open sites*

By repeating this computation experiment T times and averaging the results, we obtain a more accurate estimate of the percolation threshold. Let x_t 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.

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

Assuming T is sufficiently large (say, at least 30), the following provides a 95% confidence interval for the percolation threshold: ➔ This calculation is already done for you!

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

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 $N \le 0$ or $T \le 0$.
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 *Random* class to generate random numbers; use *standard statistics* to compute the sample mean and standard deviation. . ➔ Some of these are already done for you!

Example output:

```
% 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
```


What and how to submit
- You must submit softcopy of "Percolation.java" and "PercolationStats.java" file, along with all other required files for your program to run.
- Here is the instruction on how to submit the softcopy files :
  ① Login to your server account at nsl2.cau.ac.kr.
  ② In your home directory, create a directory "submit_alg/submit_<student ID>_hw2" (ex> "/student/20149999/submit_alg/submit_20149999_hw2")
  ③ Put your "Percolation.java" and "PercolationStats.java" file in that directory together with all the files that you downloaded from e-Class (we need those files to compile your program).

Other requirements:
- Your program must run on the class Linux server at nsl2.cau.ac.kr.
- Your code must include your name student ID at the beginning of the code as a comment.
- Your code should be easily readable and include sufficient comments for easy understanding.
- Your code should not include any Korean characters.

Grading criteria:
- You get 3 points
  - if all of your programs work correctly, AND
  - if you meet all of above requirements, AND
  - if your code handles all the exceptional cases that might occur.
- Otherwise, partial deduction may apply.
- No delayed submissions are accepted.
- Copying other student's work will result in negative points.
- Code that does not compile or code that does not run will result in negative points