Programming Assignment 1 Checklist: Percolation

Frequently Asked Questions (General)

What's a checklist? The assignment provides the programming assignment specification; the checklist provides clarifications, test data, and hints that might be helpful in completing the assignment.

Where can I find the course policies regarding submission, lateness, grading, and collaboration? Please read the Assignments Page.

I haven't programmed in Java in a while. What material do I need to remember? For a review of our Java programming model (including our input and output libraries), read Sections 1.1 and 1.2 of *Algorithms, 4th Edition*.

Which Java programming environment should I use? We recommend the lightweight IDE DrJava along with the command line. If you use our Mac OS X or Windows installer, then everything should be configured and ready to go. If you prefer use another IDE (such as Eclipse), that's perfectly fine—be sure that you can use command-line arguments, read from standard input, redirect standard input and output, and add `stdlib.jar` and `algs4.jar` to your Java classpath.

What are the input and output libraries? We have designed a set of easy-to-use Java libraries in `stdlib.jar` for input and output that you are required to use in this course. Here are the APIs.

Where can I find the Java code for the algorithms and data structures from lecture and the textbook? They are in `algs4.jar`. Here are the APIs.

Can I use various Java libraries in this assignment, such as `java.util.LinkedList`, `java.util.ArrayList`, `java.util.TreeMap`, and `java.util.HashMap`? No. You should not use any Java libaries until we have implemented equivalent versions in lecture. Once we have introduced them in lecture, you are free to use either the Java library version or our equivalent. You are welcome to use common classes in the Java language such as `Math.sqrt()` and `Integer.parseInt()`.

How do I throw a java.lang.IndexOutOfBoundsException? Use a `throw` statement like the following:

```
if (i < 0 || i >= N) throw new IndexOutOfBoundsException("row index " + i + " must be between 0 and " + (N-1));
```

While you are required to *throw* exceptions using `throw` statements as specified, you should not *catch* exceptions with `try-catch` statements—this will interfere with our grading scripts.

How should I format and comment my code? Here are some recommended style guidelines. Below are some that are particularly important (and for which you will lose points if you ignore).

- Include a bold (or Javadoc) comment at the beginning of each Java file with your name, login, precept, date, the purpose of the program, and how to compile and execute it.
- Include a bold (or Javadoc) comment describing every method.
- Include a comment describing every instance variable.
- Indent consistently, using 3 or 4 spaces for each indentation level. Do not use hard tabs.
- Do not exceed 80 characters per line. This rule also applies to the `readme.txt` file.
- Avoid unexplained magic numbers, especially ones that are used more than once.

What's the easiest way to copy a subdirectory from the COS 226 ftp site to my computer?

- *Safari*. Click the ftp link above to mount the ftp site, then drag-and-drop the `percolation` folder from the ftp mount to the desired location. If prompted for a name and password, choose to Connect as a *Guest* instead of a *Register user*.
- *Firefox*. Install the DownThemAll! plugin to support downloading all of the links contained in a webpage and use it.
- *Internet Explorer*. Click the ftp link above, then click *Page* and select *Open FTP Site in Windows Explorer* from the dropdown menu. Drag the desired folder to your desktop.
- *Linux or OS X command line*. From the Linux or OS X command line, type `ftp ftp.cs.princeton.edu`; enter `anonymous` for your username and your email address for your password; type `cd pub/cs226` to get to the root of the cs226 ftp site.
- *Ftp client*. Using your favorite ftp client, go to ftp.cs.princeton.edu/pub/cs226, enter `anonymous` for your username and your email address for your password, and navigate to the `pub/cs226` subdirectory.

Frequently Asked Questions (Percolation)

What are the goals of this assignment?

- Set up a Java programming environment.
- Use our input and output libraries.

- Learn about a scientific application of the union–find data structure.
- Measure the running time of a program and use the doubling hypothesis to make predictions.
- Measure the amount of memory used by a data structure.

Can I add (or remove) methods to (or from) Percolation? No. You must implement the `Percolation` API exactly as specified, with the identical set of public methods and signatures or you will receive a substantial deduction. However, you are encouraged to add private methods that enhance the readability, maintainability, and modularity of your program. The one exception is `main()` — you are always permitted to add this method to test your code, but we will not call it unless we specify it in our API.

Why is it so important to implement the prescribed API? Writing to an API is an important skill to master because it is an essential component of modular programming, whether you are developing software by yourself or as part of a group. When you develop a module that properly implements an API, anyone using that module (including yourself, perhaps at some later time) does not need to revisit the details of the code for that module when using it. This approach greatly simplifies writing large programs, developing software as part of a group, or developing software for use by others.

Most important, when you properly implement an API, others can write software to use your module or to test it. We do this regularly when grading your programs. For example, your `PercolationStats` client should work with our `Percolation` data type and vice versa. If you add an extra public method to `Percolation` and call them from `PercolationStats`, then your client won't work with our `Percolation` data type. Conversely, our `PercolationStats` client may not work with your `Percolation` data type if you remove a public method.

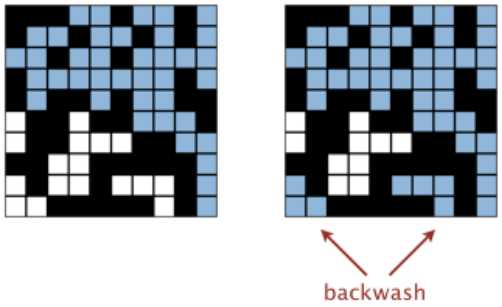
How many lines of code should my program be? You should strive for clarity and efficiency. Our reference solution for `Percolation.java` is about 80 lines, plus a test client. Our `PercolationStats.java` client is about 60 lines. If you are re-implementing the union-find data structure (instead of reusing the implementations provided), you are on the wrong track.

What assumptions can I make about the input to main() in PercolationStats? It can be any valid input: an integer $N \geq 1$ and an integer $T \geq 1$. In general, in this course you can assume that the input is of the specified format. But you do need to deal with pathological cases such as $N = 1$.

What should `stddev()` return if T equals 1? The sample standard deviation is undefined. We recommend returning `Double.NaN`.

After the system has percolated, PercolationVisualizer colors in light blue all sites connected to open sites on the bottom (in addition to those connected to open sites on the top). Is this "backwash" acceptable? No, this is likely a bug in your `Percolation`. It is only a minor deduction, so don't go crazy trying to get this detail.

```
% java PercolationVisualizer input10.txt
```



How do I generate a site uniformly at random among all blocked sites for use in PercolationStats? Pick a site at random (by using `StdRandom` to generate two integers between 0 (inclusive) and N (exclusive) and use this site if it is blocked; if not, repeat.

I don't get reliable timing information in PercolationStats when $N = 200$. What should I do? Increase the size of N (say to 400, 800, and 1600), until the mean running time exceeds its standard deviation.

Style and Bug Checkers

Style checker. We recommend using Checkstyle 5.5 (and the configuration file checkstyle.xml) to check the style of your Java programs. Here is a list of available Checkstyle checks.

Bug checker. We recommend using FindBugs 2.0.1 (and the configuration file findbugs.xml) to identify common bug patterns in your code. Here is a summary of FindBugs Bug descriptions.

Mac OS X and Windows installer. If you used our Mac OS X or Windows installer, these programs are already installed as command-line utilities. You can check a single file or multiple files via the commands:

```
% checkstyle HelloWorld.java
% checkstyle *.java
% findbugs HelloWorld.class
% findbugs *.class
```

Note that Checkstyle inspects the source code; Findbugs inspects the compiled code.

Eclipse. For Eclipse users, there is a Checkstyle plugin for Eclipse and a Findbugs plugin for Eclipse.

Caveat. The appearance of a warning message does not necessarily lead to a deduction (and, in some cases, it does not even indicate an error).

Testing

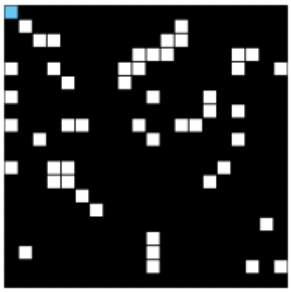
Testing. We provide two clients that serve as large-scale visual traces. We highly recommend using them for testing and debugging your `Percolation` implementation.

Visualization client. PercolationVisualizer.java animates the results of opening sites in a percolation system specified by a file by performing the following steps:

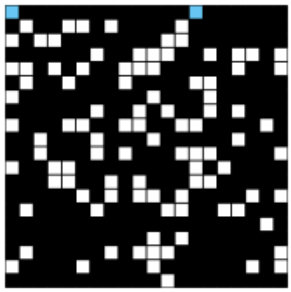
- Read the grid size *N* from the file.
- Create an *N*-by-*N* grid of sites (initially all blocked).
- Read in a sequence of sites (row *i*, column *j*) to open from the file. After each site is opened, draw full sites in light blue, open sites (that aren't full) in white, and blocked sites in black using *standard draw*, with site (0, 0) in the upper left-hand corner.

The program should behave as in this movie and the following snapshots when used with input20.txt.

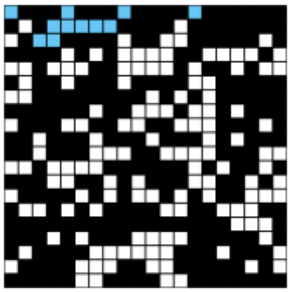
```
% java PercolationVisualizer input20.txt
```



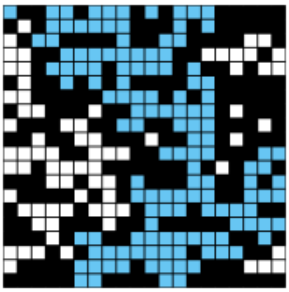
50 open sites



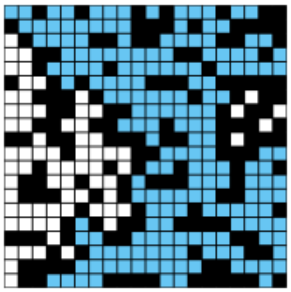
100 open sites



150 open sites



204 open sites



250 open sites

Sample data files. The directory percolation contains some sample files for use with the visualization client. Assoicated with each input `.txt` file is an output `.png` file that contains the desired graphical output at the end of the animation.

InteractiveVisualization client. InteractivePercolationVisualizer.java is similar to the first test client except that the input comes from a mouse (instead of from a file). It takes a command-line integer *N* that specifies the lattice size. As a bonus, it writes to standard output the sequence of sites opened in the same format used by `PercolationVisualizer`, so you can use it to prepare interesting files for testing. If you design an interesting data file, feel free to share it with us and your classmates by posting it in the discussion forums.

Possible Progress Steps

These are purely suggestions for how you might make progress. You do not have to follow these steps.