CIT 590 Assignment 10: Percolation

Fall 2016, David Matuszek

Purposes of this assignment

- To give you more practice with Java
- To get you to use some two-dimensional arrays
- To demonstrate the idea of simulation

General idea of the assignment

When it rains, water soaks into the ground. When sand grains are packed tightly, water doesn't soak as far down. Your task is to write a two-dimensional model of this situation.

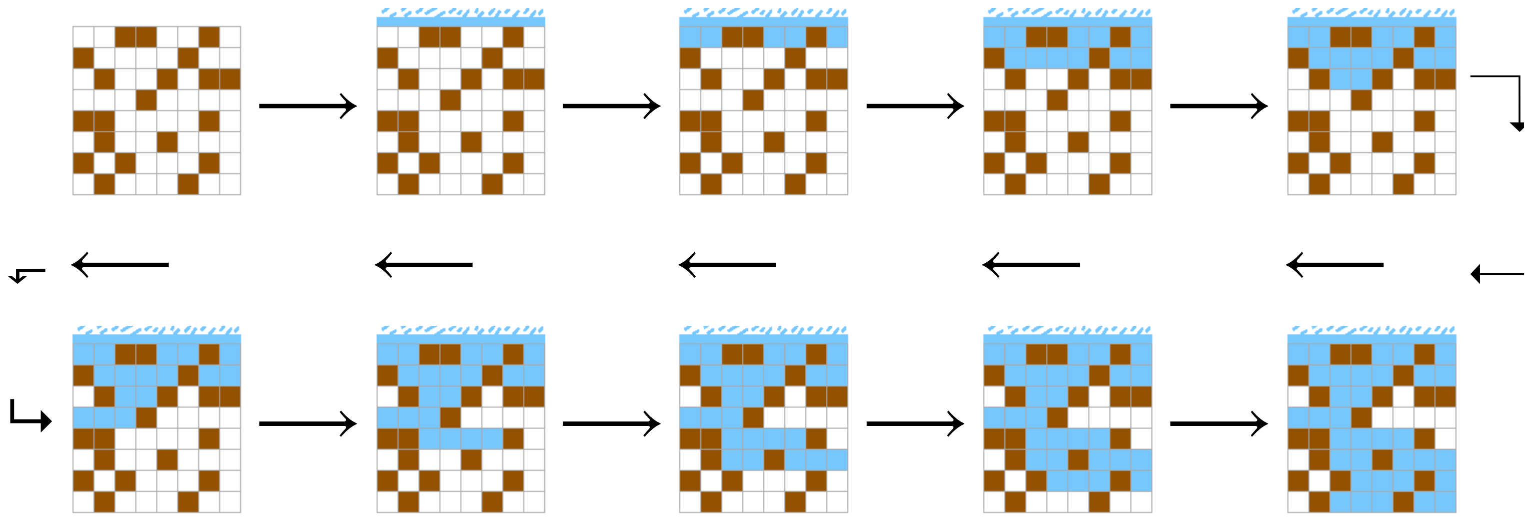
Details

Construct an NxN grid, that is, a square array. Populate the grid randomly with "sand" grains, such that each location has a sand grain with probability `p`, or is empty with probability `(1-p)`. You should import `java.util.Random` to help do this.

"Water" arrives at the top of this array, so that every empty location in the first row becomes filled with water.

Water will only flow down or horizontally, never up. For each succeeding row, if an empty space has water directly above it, or horizontally adjacent to it, it also fills with water. At each move, water can flow any distance horizontally.

If the ground is packed tightly with sand, water will not seep very far down into it. If the ground is very loosely packed, water will almost certainly seep all the way down. For an NxN container, there is some packing probability `p` such that water has exactly a 50% probability of seeping all the way to the bottom row. Your job is to find that probability.



Write at least the following functions:

`int[][] ground(int n, double p)`
: `ground(n, p)` returns an array of `n` arrays of integer, where each array is of length `n`, and each integer has probability `p` of being a sand grain, `(1-p)` of being empty (and dry). Use the encoding `0` = empty space, `1` = sand grain, `2` = water.

`void seep(int[][] ground, int row)`
: `seep(array, row)` causes water to flow from `row` into `row+1`, modifying the array. In other words, this function performs one step of the simulation.

`boolean percolate(int[][] ground)`
: Returns `true` if, after water has "seeped" as far as it can, water has reached the bottom row, and `false` otherwise. For the example above, the result would be `true`.

`double findProbability(int n)`
: For an `n` by `n` array, determines the packing probability `p` that causes the array to have a 50% probability of water seeping all the way to the bottom.

You might think that the probability of water percolating all the way to the bottom would decrease as `p` (probability of sand grains) increases. And it does, but in an interesting and extremely nonlinear fashion.

Specific values to use

Another interesting question is whether the probability depends on N, the array size. You don't have to answer this question, but provide data for it by determining the probability for arrays of size 50x50, 100x100, and 200x200.

How to go about it

One call of `percolate` won't tell you very much--it will just return `true` or `false`. If you run it multiple times for a given `p` (and a different `ground` each time!), that will provide better evidence as to whether `p` is too low or too high.

Here's what I would do. I would start with some reasonable value of `p`, possibly 0.5, and some amount to change `p` by (call it `delta`), possibly 0.05. Then, at each iteration, I would see if I needed to add `delta`, or subtract `delta`. Then every so often, I would make `delta` smaller (but not necessarily every time). This leaves open the question of (1) when to make delta smaller, (2) how much to make delta smaller, and (3) how to decide to end the iteration.

Here are two things to watch out for: (1) By sheer bad luck, `p` may become less than zero or greater than one, so guard against this case; and (2) if you overshoot the best value for `p`, `delta` should not be shrinking so rapidly that you can never get back to the best value.

For each of the three array sizes, I would like to see at least two digits of accuracy for `p`. Your program should be able to do this in well under a minute.

Testing

Write unit tests for `ground`, `seep`, and `percolate`.

Here is the syntax for declaring a literal array: `int[][] myArray = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};` You can use this to construct arrays for which you know whether the water will reach the bottom.

The `ground` method returns a random result, so you cannot expect a particular result. You can, however, determine whether the proportion of sand grains is reasonably close to `p`.

Due date

Zip your project and turn it in to Canvas before **11:59pm Tuesday, November 15**.