Data Structures and Algorithms in Java

Assignment 3 (Percolation) Discussion

Introduction

The percolation threshold of a system is a measure of how porous the system needs to be so that it percolates

Goal: write programs to estimate the percolation threshold of a system

We model percolation system as an $n \times n$ array of booleans (`true` $\implies$ open site and `false` $\implies$ blocked site)

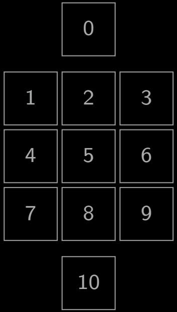
We use an `UF` object with $n^2 + 2$ sites and the `encode()` method to translate sites $(0, 0), (0, 1), \ldots, (n - 1, n - 1)$ of the array to sites $1, 2, \ldots, n^2$ of the `UF` object

Sites 0 (source) and $n^2 + 1$ (sink) are virtual, ie, not part of the percolation system

Introduction

A 3×3 percolation system and its `UF` representation

$0, 0$	$0, 1$	$0, 2$
$1, 0$	$1, 1$	$1, 2$
$2, 0$	$2, 1$	$2, 2$



Problem 1 (Percolation Data Type)

Create a data type called `Percolation` that supports the following API

`Percolation(int n)`	constructs an `n x n` percolation system, with all sites blocked
`void open(int i, int j)`	opens site `(i, j)` if it is not already open
`boolean isOpen(int i, int j)`	returns `true` if site `(i, j)` is open, and `false` otherwise
`boolean isFull(int i, int j)`	returns `true` if site `(i, j)` is full, and `false` otherwise
`int numberOfOpenSites()`	returns the number of open sites
`boolean percolates()`	returns `true` if this system percolates, and `false` otherwise

```
× ~/workspace/percolation
```

```
$ javac -d out src/Percolation.java
$ java Percolation data/input10.txt
10 x 10 system:
  Open sites = 56
  Percolates = true
$ java Percolation data/input10-no.txt
10 x 10 system:
  Open sites = 55
  Percolates = false
```

Problem 1 (Percolation Data Type)

Instance variables

- Percolation system size, `int n`
- Percolation system, `boolean[][] open`
- Number of open sites, `int openSites`
- Union-find representation of the percolation system, `WeightedQuickUnionUF uf`

`private int encode(int i, int j)`

- Return the `uf` site $(1, 2, \ldots, n^2)$ corresponding to the percolation system site `(i, j)`

`public Percolation(int n)`

- Initialize instance variables

`void open(int i, int j)`

- If site `(i, j)` is not open
    - Open the site
    - Increment `openSites` by one
    - If the site is in the first (or last) row, connect the corresponding `uf` site with the source (or sink)
    - If any of the neighbors to the north, east, west, and south of site `(i, j)` is open, connect the `uf` site corresponding to site `(i, j)` with the `uf` site corresponding to that neighbor

Problem 1 (Percolation Data Type)

`boolean isOpen(int i, int j)`

- Return whether site `(i, j)` is open or not

`boolean isFull(int i, int j)`

- Return whether site `(i, j)` is full or not — a site is full if it is open and its corresponding `uf` site is connected to the source

`int numberOfOpenSites()`

- Return the number of open sites

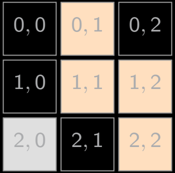
`boolean percolates()`

- Return whether the system percolates or not — a system percolates if the sink is connected to the source

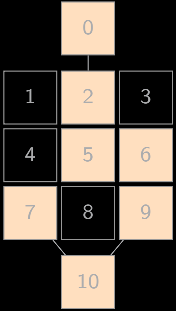
Back Wash

Using virtual source and sink sites introduces what is called the *back wash* problem

In the 3×3 system, consider opening the sites $(0, 1)$, $(1, 2)$,$(1, 1)$, $(2, 0)$, and $(2, 2)$, and in that order; the system percolates once $(2, 2)$ is opened



0, 0	0, 1	0, 2
1, 0	1, 1	1, 2
2, 0	2, 1	2, 2



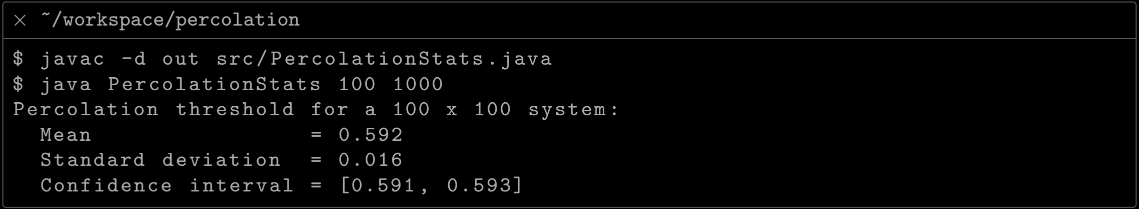
The site $(2, 0)$ is not full, but the corresponding `uf` site 7 is connected to the source, so is incorrectly reported as being full — this is the back wash problem

To solve the back wash problem, create another `WeightedQuickUnionUF` object

Problem 2 (Estimation of Percolation Threshold)

Create an immutable data type called `PercolationStats` that supports the following API

`PercolationStats(int n, int m)`	performs `m` independent experiments on an `n x n` percolation system
`double mean()`	returns sample mean of percolation threshold
`double stddev()`	returns sample standard deviation of percolation threshold
`double confidenceLow()`	returns low endpoint of 95% confidence interval
`double confidenceHigh()`	returns high endpoint of 95% confidence interval



```
$ javac -d out src/PercolationStats.java
$ java PercolationStats 100 1000
Percolation threshold for a 100 x 100 system:
  Mean                = 0.592
  Standard deviation  = 0.016
  Confidence interval = [0.591, 0.593]
```

Problem 2 (Estimation of Percolation Threshold)

Instance variables

- Number of independent experiments, `int m`
- Percolation thresholds for the `m` experiments, `double[] x`

`PercolationStats(int n, int m)`

- Initialize instance variables
- Repeat the following experiment `m` times
    - Create an $n \times n$ percolation system
    - Until the system percolates, choose a site (i, j) at random and open it if it is not already open
    - Calculate percolation threshold as the fraction of sites opened, and store the value in `x[]`

`double mean()`

- Return the mean μ of the values in `x[]`

`double stddev()`

- Return the standard deviation σ of the values in `x[]`

`double confidenceLow()`

- Return $\mu - \frac{1.96\sigma}{\sqrt{m}}$

`double confidenceHigh()`

- Return $\mu + \frac{1.96\sigma}{\sqrt{m}}$