Project 1 (Percolation) Checklist

Prologue

Project goal: write a program to estimate the percolation threshold of a system

Relevant files:

- ⇝ `project1.pdf` (project writeup)
- ⇝ `project1_checklist.pdf` (checklist)
- ⇝ `project1.zip` (starter files, test data, and `report.txt`)

Exercises

Exercise 1. (*Great Circle Distance*) Write a program `GreatCircle.java` that accepts x_1 (double), y_1 (double), x_2 (double), and y_2 (double) as command-line arguments representing the latitude and longitude (in degrees) of two points on earth, and writes to standard output the great-circle distance (in km) between the two points, given by the formula

$$d = 111 \arccos(\sin(x_1)\sin(x_2) + \cos(x_1)\cos(x_2)\cos(y_1 - y_2)).$$

```
>_ ~/workspace/project1

$ java GreatCircle 48.87 -2.33 37.8 -122.4
8701.389543238289
```

Exercises

GreatCircle.java

```java
import stdlib.StdOut;

public class GreatCircle {
    // Entry point.
    public static void main(String[] args) {
        // Accept x1 (double), y1 (double), x2 (double), and y2 (double) as command-line arguments.
        ...

        // Convert the angles to radians.
        ...

        // Calculate great-circle distance d.
        ...

        // Write d to standard output.
        ...
    }
}
```

Exercise 2. (*Counting Primes*) Implement the static method `isPrime()` in `PrimeCounter.java` that accepts an integer x and returns `true` if x is prime and `false` otherwise. Also implement the static method `primes()` that accepts an integer n and returns the number of primes less than or equal to n — a number x is prime if it is not divisible by any number $i \in [2, \sqrt{x}]$.

```
>_ ~/workspace/project1

$ java PrimeCounter 1000
168
```

Exercises

PrimeCounter.java

```java
import stdlib.StdOut;

public class PrimeCounter {
    // Entry point. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        StdOut.println(primes(n));
    }

    // Returns true if x is prime; and false otherwise.
    private static boolean isPrime(int x) {
        // For each 2 <= i <= x / i, if x is divisible by i, then x is not a prime. If no such i
        // exists, then x is a prime.
        ...
    }

    // Returns the number of primes <= n.
    private static int primes(int n) {
        // For each 2 <= i <= n, use isPrime() to test if i is prime, and if so increment a count.
        // At the end return the count.
        ...
    }
}
```

Exercise 3. (*Euclidean Distance*) Implement the static method `distance()` in `Distance.java` that accepts position vectors x and y — each represented as a 1D array of doubles — and returns the Euclidean distance between the two vectors, calculated as the square root of the sums of the squares of the differences between the corresponding entries.

```
>_ ~/workspace/project1

$ java Distance
5
-9 1 10 -1 1
5
-5 9 6 7 4
13.0
```

Exercises

Distance.java

```java
import stdlib.StdArrayIO;
import stdlib.StdOut;

public class Distance {
    // Entry point. [DO NOT EDIT]
    public static void main(String[] args) {
        double[] x = StdArrayIO.readDouble1D();
        double[] y = StdArrayIO.readDouble1D();
        StdOut.println(distance(x, y));
    }

    // Returns the Euclidean distance between the position vectors x and y.
    private static double distance(double[] x, double[] y) {
        // Sum up the squares of (x[i] - y[i]), where 0 <= i < x.length, and return the square
        // root of the sum.
        ...
    }
}
```

Exercises

Exercise 4. (*Matrix Transpose*) Implement the static method `tranpose()` in `Transpose.java` that accepts a matrix x — represented as a 2D array of doubles — and returns a new matrix that is the transpose of x.

```
>_ ~/workspace/project1

$ Transpose
3 3
1 2 3
4 5 6
7 8 9
3 3
  1.00000   4.00000   7.00000
  2.00000   5.00000   8.00000
  3.00000   6.00000   9.00000
```

Exercises

Transpose.java

```
import stdlib.StdArrayIO;

public class Transpose {
    // Entry point. [DO NOT EDIT]
    public static void main(String[] args) {
        double[][] x = StdArrayIO.readDouble2D();
        StdArrayIO.print(transpose(x));
    }

    // Returns a new matrix that is the transpose of x.
    private static double[][] transpose(double[][] x) {
        // Create a new 2D matrix t (for transpose) with dimensions n x m, where m x n are the
        // dimensions of x.
        ...

        // For each 0 <= i < m and 0 <= j < n, set t[j][i] to x[i][j].
        ...

        // Return t.
        ...
    }
}
```

Exercises

Exercise 5. (*Rational Number*) Implement an immutable data type `Rational` that represents a rational number, ie, a number of the form a/b where a and $b \neq 0$ are integers. The data type must support the following API:

Rational	
`Rational(long x)`	constructs a rational number whose numerator is `x` and denominator is 1
`Rational(long x, long y)`	constructs a rational number given its numerator `x` and denominator `y` (†)
`Rational add(Rational other)`	returns the sum of this rational number and `other`
`Rational multiply(Rational other)`	returns the product of this rational number and `other`
`boolean equals(Object other)`	returns `true` if this rational number is equal to `other`, and false otherwise
`String toString()`	returns a string representation of this rational number

† Use the private method `gcd()` to ensure that the numerator and denominator never have any common factors. For example, the rational number 2/4 must be represented as 1/2.

```
>_ ~/workspace/project1

$ java Rational 10
a            = 1 + 1/2 + 1/4 + ... + 1/2^10 = 1023/512
b            = (2^10 - 1) / 2^(10 - 1) = 1023/512
a.equals(b) = true
```

Exercises

Rational.java

```java
import stdlib.StdOut;

public class Rational {
    private long x; // numerator
    private long y; // denominator

    // Constructs a rational number whose numerator is x and denominator is 1.
    public Rational(long x) {
        // Set this.x to x and this.y to 1.
        ...
    }

    // Constructs a rational number given its numerator x and denominator y.
    public Rational(long x, long y) {
        // Set this.x to x / gcd(x, y) and this.y to y / gcd(x, y).
        ...
    }

    // Returns the sum of this rational number and other.
    public Rational add(Rational other) {
        // Sum of rationals a/b and c/d is the rational (ad + bc) / bd.
        ...
    }

    // Returns the product of this rational number and other.
    public Rational multiply(Rational other) {
        // Product of rationals a/b and c/d is the rational ac / bd.
        ...
    }

    // Returns true if this rational number is equal to other, and false otherwise.
    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
```

Exercises

Rational.java

```java
        if (other == this) {
            return true;
        }
        if (other.getClass() != this.getClass()) {
            return false;
        }

        // Rationals a/b and c/d are equal iff a == c and b == d.
        ...
    }

    // Returns a string representation of this rational number.
    public String toString() {
        long a = x, b = y;
        if (a == 0 || b == 1) {
            return a + "";
        }
        if (b < 0) {
            a *= -1;
            b *= -1;
        }
        return a + "/" + b;
    }

    // Returns gcd(p, q), computed using Euclid's algorithm.
    private static long gcd(long p, long q) {
        return q == 0 ? p : gcd(q, p % q);
    }

    // Unit tests the data type. [DO NOT EDIT]
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Rational total = new Rational(0);
        Rational term = new Rational(1);
        for (int i = 1; i <= n; i++) {
```

Exercises

Rational.java

```
            total = total.add(term);
            term = term.multiply(new Rational(1, 2));
        }
        Rational expected = new Rational((long) Math.pow(2, n) - 1, (long) Math.pow(2, n - 1));
        StdOut.printf("a           = 1 + 1/2 + 1/4 + ... + 1/2^%d = %s\n", n, total);
        StdOut.printf("b           = (2^%d - 1) / 2^(%d - 1) = %s\n", n, n, expected);
        StdOut.printf("a.equals(b) = %b\n", total.equals(expected));
    }
}
```

Exercise 6. (*Harmonic Number*) Write a program `Harmonic.java` that accepts n (int) as command-line argument, computes the nth harmonic number H_n as a rational number, and writes the value to standard output.

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}.$$

```
>_ ~/workspace/project1

$ java Harmonic 5
137/60
```

Exercises

Harmonic.java

```java
import stdlib.StdOut;

public class Harmonic {
    // Entry point.
    public static void main(String[] args) {
        // Accept n (int) as command-line argument.
        ...

        // Set total to the rational number 0.
        ...

        // For each 1 <= i <= n, add the rational term 1 / i to total.
        ...

        // Write total to standard output.
        ...
    }
}
```

Problems

The guidelines for the project problems that follow will be of help only if you have read the description of the project and have a general understanding of the problems involved. It is assumed that you have done the reading.

Problems

Problem 1. (*Model a Percolation System*)

Hints:

- Model percolation system as an $n \times n$ array of booleans (`true` $\Longrightarrow$ open site and `false` $\Longrightarrow$ blocked site)
- Create an UF object with $n^2 + 2$ sites and use the private `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
- 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

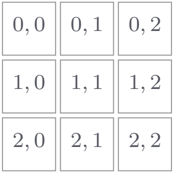
	0	
1	2	3
4	5	6
7	8	9
	10	

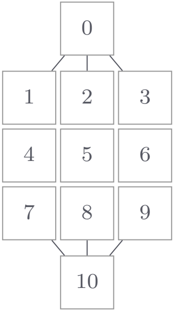
Problems

- 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 $(1 \dots n^2)$ corresponding to the percolation system site (i, j)

Problems

- `public Percolation(int n)`
  - Initialize instance variables
  - Connect the `uf` sites corresponding to first and last rows of the percolation system with the source and sink sites respectively
  - The 3×3 system with its top and bottom rows connected to the source and sink



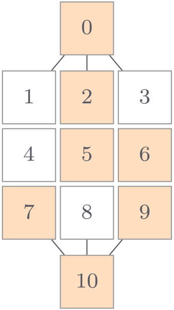


Problems

- `void open(int i, int j)`
  - Open the site `(i, j)` if it is not already open
  - Increment `openSites` by one
  - Check if any of the neighbors to the north, east, west, and south of `(i, j)` is open, and if so, connect the `uf` site corresponding to `(i, j)` with the `uf` site corresponding to that neighbor
- `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

Problems

- 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



- The site $(2, 0)$ is technically not full since it is not connected to an open site in the top row via a path of neighboring (north, east, west, and south) open sites, 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 receive full credit, you must resolve the back wash problem

Problem 2. (*Estimate Percolation Threshold*)

Hints:

- Instance variables
  - Number of independent experiments, `int m`
  - Percolation thresholds for the `m` experiments, `double[] x`
- `PercolationStats(int n, int m)`
  - Initialize instance variables
  - Perform 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[]`

Problems

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

Problems

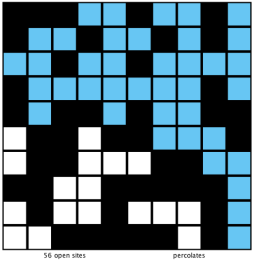
The `data` directory contains some input `.txt` files for the percolation visualization programs, and associated with each file is an output `.png` file that shows the desired output; for example

```
>_ ~/workspace/project1

$ cat data/input10.txt
10
9 1
1 9
...
7 9
```

```
>_ ~/workspace/project1

$ display data/input10.png
```

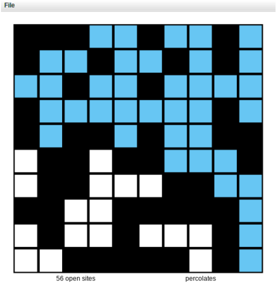


Problems

The visualization program `PercolationVisualizer` accepts as command-line argument the name of an input file, and visually reports if the system percolates or not

```
>_ ~/workspace/project1

$ java PercolationVisualizer data/input10.txt
```

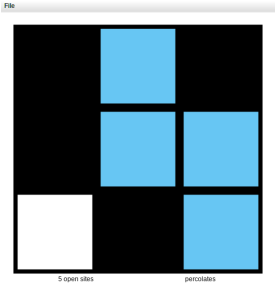


Problems

The visualization program `InteractivePercolationVisualizer` constructs an $n \times n$ percolation system, where n is specified as command-line argument, and allows you to interactively open sites in the system by clicking on them and visually inspect if the system percolates or not

```
>_ ~/workspace/project1

$ java InteractivePercolationVisualizer 3
3
0 1
1 2
1 1
2 0
2 2
```



Epilogue

Use the template file `report.txt` to write your report for the project

Your report must include:

- Time (in hours) spent on the project
- Difficulty level (1: very easy; 5: very difficult) of the project
- A short description of how you approached each problem, issues you encountered, and how you resolved those issues
- Acknowledgement of any help you received
- Other comments (what you learned from the project, whether or not you enjoyed working on it, etc.)

Epilogue

Before you submit your files:

- Make sure your programs meet the style requirements by running the following command on the terminal

```
>_ ~/workspace/project1

$ check_style src/*.java
```

- Make sure your code is adequately commented, is not sloppy, and meets any project-specific requirements, such as corner cases and time complexities
- Make sure your report uses the given template, isn't too verbose, doesn't contain lines that exceed 80 characters, and doesn't contain spelling mistakes

Epilogue

Files to submit:

1. `GreatCircle.java`
2. `PrimeCounter.java`
3. `Distance.java`
4. `Transpose.java`
5. `Rational.java`
6. `Harmonic.java`
7. `Percolation.java`
8. `PercolationStats.java`
9. `report.txt`