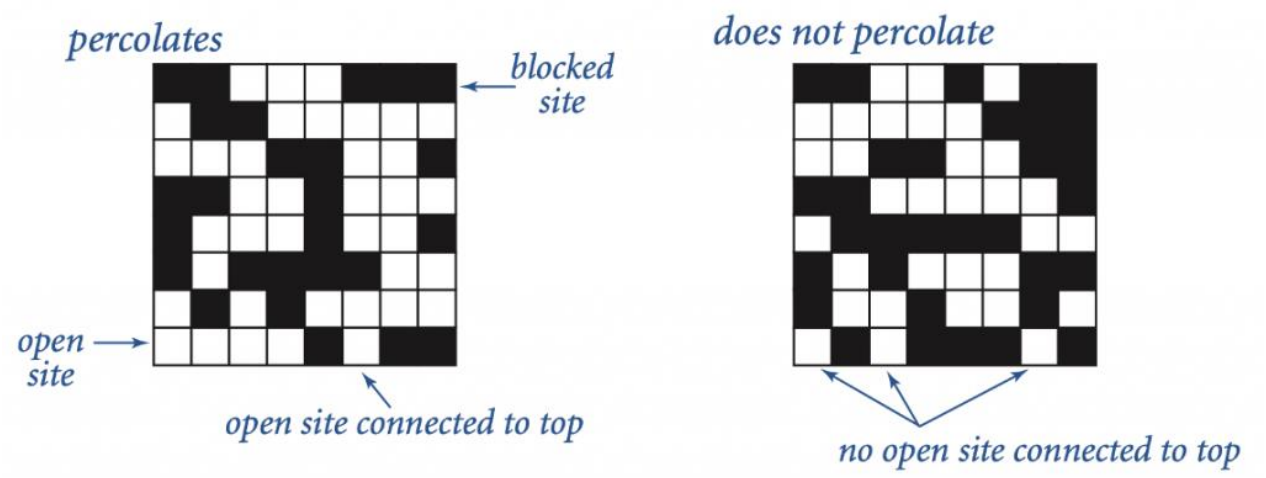
Background

Percolation serves as an abstract model applicable to numerous physical systems. In this context, we consider a grid comprising n-by-n sites, where each site is open with a given probability, denoted as p. The crux of percolation lies in the system's ability to percolate, which occurs if and only if there exists a connection between an open site in the top row of the grid and an open site in the bottom row, facilitated by a series of interconnected open sites. This concept provides a valuable framework for understanding and modeling various real-world phenomena and systems.



As a dynamic-connectivity problem, we model percolation and apply union-find to decide whether the system percolates.

How to model Percolation as a dynamic-connectivity problem?

1. Initialize a *n*-by-*n* grid of sites, where all sites are closed/blocked.
2. Initialize a weighted quick union-find object containing all the sites in the grid plus two additional sites: a virtual top and a virtual bottom.
3. Open all sites.
4. Check if the system percolates.

Opening all sites

- Starting at the first site (0,0), one row at a time, open each site with probability *p*.

Open site

1. If site is closed, open. Otherwise, do nothing.
2. If opened site is in the first row then connect to virtual top.

3. If opened site is in the last row then connect to virtual bottom.
4. Connect opened site to any adjacent site that is open. An adjacent site is a site to the left, right, top, or bottom of the site. (Not diagonals).

Does the system percolate?

The system percolates if virtual top site is connected to virtual bottom site.

Overview of files

You are tasked with completing 3 methods under the Percolation class: openSite, openAllSites, and percolationCheck.

Files provided

- StdDraw.java
  - Used by Percolation.java to draw the grid.
- StdRandom.java
  - Used by Percolation.java to generate random numbers.
- WeightedQuickUnionFind.java
  - Used by Percolation.java to store information about which open sites are connected or not.
- Percolation.java
  - This is the file you will update and submit. It contains the information for the grid.

Percolation.java

Instance variables

- **boolean[][] grid**: boolean 2D array representing the grid. Each (row, col) is a site. (row, col) is true if the site is open, false if the site is closed.
- **int gridSize**: the size of the grid.
- **int gridSquared**: the number of sites in a grid.
- **WeightedQuickUnionFind wquFind**: Weighted quick union-find object used to keep track of all connected/opened sites.
- **int virtualTop**: index of a virtual top in the size and parent arrays in *WeightedQuickUnionFind*. Connect the virtual top to every open site in the first row of the grid.
- **int virtualBottom**: index of a virtual bottom in the size and parent arrays in WeightedQuickUnionFind.

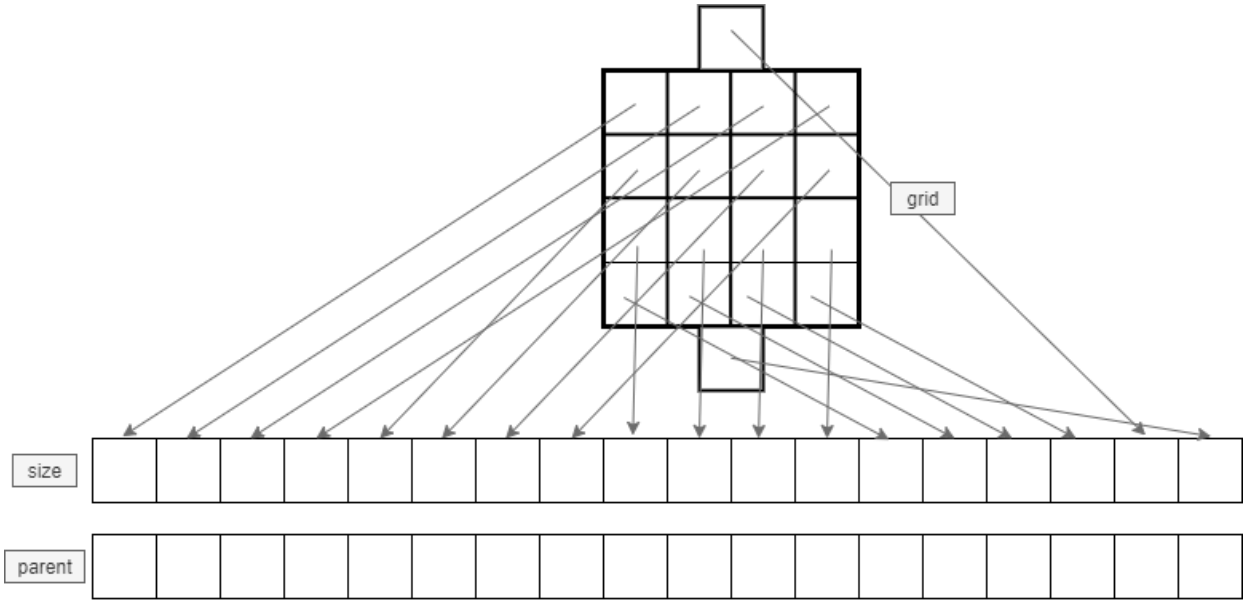
Methods

- **constructor**: initializes the object's instance variables. Do not update.

- **openSite()**: opens a site at (row, col). If the site is already open, do nothing. You complete this method.
- **openAllSites()**: opens all sites in the grid. Starting at the first site at index (0,0) and moving row wise through all the sites, each site is opened with probability *p*. You complete this method. Use **StdRandom.uniform()** to generate a random number.
- **percolationCheck()**: returns true if the system percolates. You complete this method.
- **displayGrid()**: displays the grid. An open site is colored blue, a closed site is colored black.
- **main()**: for testing only, update freely.

The following picture depicts the relationship of the grid sites to the size and parent arrays in the WeightedQuickUnionFind object.

- The 2D array grid is flattened into a 1D array.
- The 1D array has an additional two sites: the virtual top and bottom sites.
- The virtual top corresponds to the next to last index in size/parent array, the virtual bottom corresponds to the last index in the size/parent array.



Implementation

- You are to complete the methods openSite(), openAllSites(), and percolationCheck() in the Percolation.java file.
- YOU MAY only update the methods openSite(), openAllSites(), and percolationCheck().
- YOU MAY call displayGrid() from inside the main method only.
- DO NOT add any public methods to the percolation class.
- YOU MAY add private methods to the percolation class.
- Provide 10 percolation problems for one run