****

**College of Engineering, Software Engineering Department**

**SE 2231/36 Algorithms**

**Laboratory 1**

**Percolation**

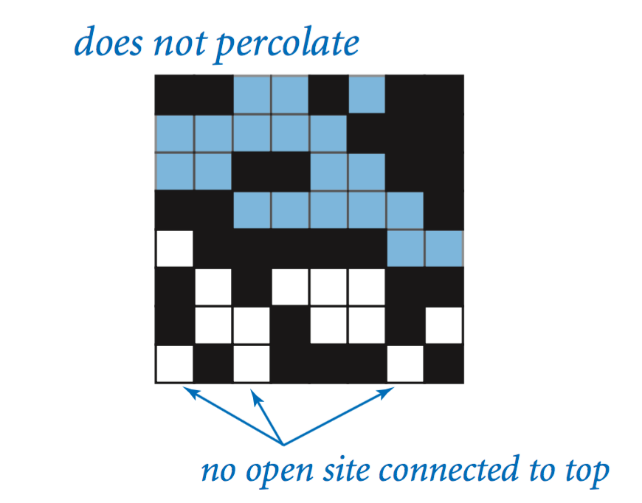
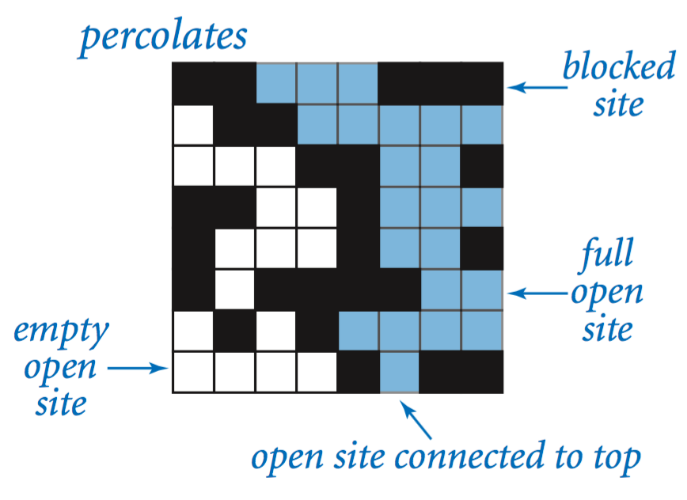
**Objectives:**

* Implement the weighted quick-union with path compression algorithm.
* Apply the algorithm on a real life problem.

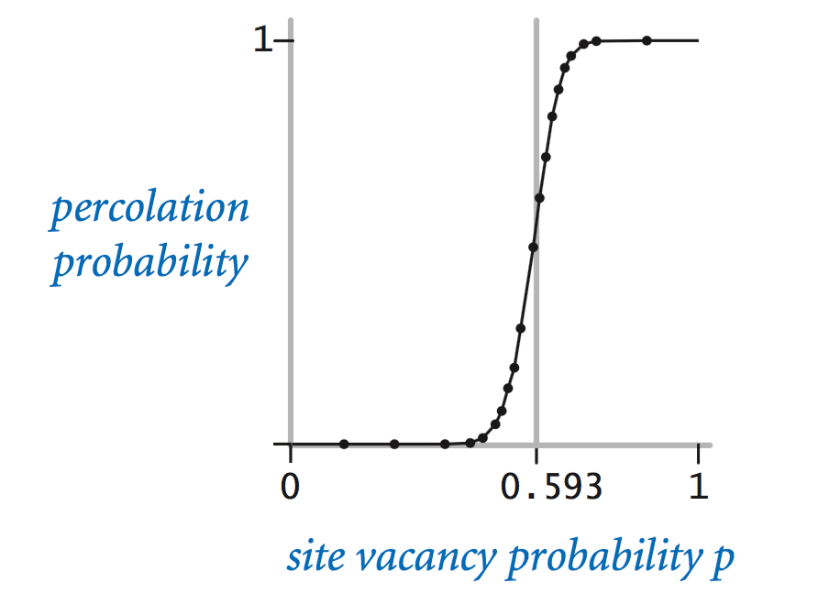
Write a program to estimate the value of the percolation threshold via Monte Carlo simulation. You

**Percolation.** Given a composite systems comprised of randomly distributed insulating and metallic materials: what fraction of the materials need to be metallic so that the composite system is an electrical conductor? Given a porous landscape with water on the surface (or oil below), under what conditions will the water be able to drain through to the bottom (or the oil to gush through to the surface)? Scientists have defined an abstract process known as *percolation* to model such situations.

**The model.** We model a percolation system using an *n*-by-*n* grid of *sites*. Each site is either *open* or *blocked*. A *full* site is an open site that can be connected to an open site in the top row via a chain of neighboring (left, right, up, down) open sites. We say the system *percolates* if there is a full site in the bottom row. In other words, a system percolates if we fill all open sites connected to the top row and that process fills some open site on the bottom row. (For the insulating/metallic materials example, the open sites correspond to metallic materials, so that a system that percolates has a metallic path from top to bottom, with full sites conducting. For the porous substance example, the open sites correspond to empty space through which water might flow, so that a system that percolates lets water fill open sites, flowing from top to bottom.)



**The problem.** In a famous scientific problem, researchers are interested in the following question: if sites are independently set to be open with probability *p* (and therefore blocked with probability 1 − *p*), what is the probability that the system percolates? When *p* equals 0, the system does not percolate; when *p* equals 1, the system percolates. The plots below show the site vacancy probability *p* versus the percolation probability for 20-by-20 random grid (left) and 100-by-100 random grid (right).



When *n* is sufficiently large, there is a *threshold* value *p*\* such that when *p* < *p*\* a random *n*-by-*n* grid almost never percolates, and when *p* > *p*\*, a random *n*-by-*n* grid almost always percolates. No mathematical solution for determining the percolation threshold *p*\* has yet been derived. Your task is to write a computer program to estimate *p*\*.

**Percolation data type.** To model a percolation system, create a class Percolation with the following API:

***class*** Percolation:

    # creates n-by-n grid, with all sites initially blocked

***def*** **\_\_init\_\_**(**self**, **n**: *int*):

**pass**

    # opens the site (row, col) if it is not open already

***def*** **open**(**self**, **row**: *int*, **col**: *int*) -> None:

**pass**

    # is the site (row, col) open?

***def*** **is\_open**(**self**, **row**: *int*, **col**: *int*) -> *bool*:

**pass**

    # is the site (row, col) full?

***def*** **is\_full**(**self**, **row**: *int*, **col**: *int*) -> *bool*:

**pass**

    # returns the number of open sites

***def*** **number\_of\_open\_sites**(**self**) -> *int*:

**pass**

    # does the system percolate?

***def*** **percolates**(**self**) -> *bool*:

**pass**

    # test client (optional)

**@***staticmethod*

***def*** **main**():

**pass**

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    Percolation.main()

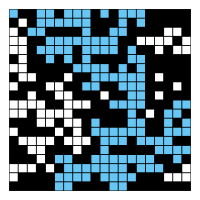
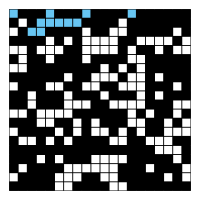
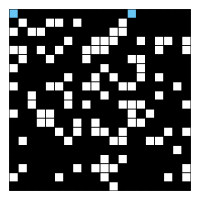
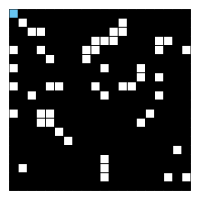
*Corner cases.* By convention, the row and column indices are integers between 1 and *n*, where (1, 1) is the upper-left site: Throw an IndexError if any argument to open(), is\_open(), or is\_full() is outside its prescribed range. Throw an Error in the constructor if *n* ≤ 0.

*Performance requirements.* The constructor must take time proportional to *n*2; all instance methods must take constant time plus a constant number of calls to union() and find().

**Monte Carlo simulation.** To estimate the percolation threshold, consider the following computational experiment:

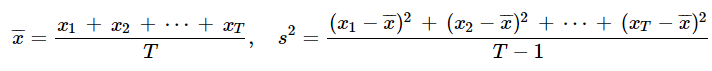
* Initialize all sites to be blocked.
* Repeat the following until the system percolates:
  + Choose a site uniformly at random among all blocked sites.
  + Open the site.
* The fraction of sites that are opened when the system percolates provides an estimate of the percolation threshold.

For example, if sites are opened in a 20-by-20 lattice according to the snapshots below, then our estimate of the percolation threshold is 204/400 = 0.51 because the system percolates when the 204th site is opened.

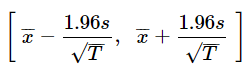


*50 open sites 100 open sites 150 open sites 204 open sites*

By repeating this computation experiment *T* times and averaging the results, we obtain a more accurate estimate of the percolation threshold. Let *xt* be the fraction of open sites in computational experiment *t*. The sample mean  provides an estimate of the percolation threshold; the sample standard deviation *s*; measures the sharpness of the threshold.



Assuming *T* is sufficiently large (say, at least 30), the following provides a 95% confidence interval for the percolation threshold:



To perform a series of computational experiments, create a class PercolationStats with the following API.

***class*** PercolationStats:

    # perform independent trials on an n-by-n grid

***def*** **\_\_init\_\_**(**self**, **n**: *int*, **trials**: *int*):

**pass**

    # sample mean of percolation threshold

***def*** **mean**(**self**) -> *float*:

**pass**

    # sample standard deviation of percolation threshold

***def*** **stddev**(**self**) -> *float*:

**pass**

    # low endpoint of 95% confidence interval

***def*** **confidence\_lo**(**self**) -> *float*:

**pass**

    # high endpoint of 95% confidence interval

***def*** **confidence\_hi**(**self**) -> *float*:

**pass**

    # test client (see below)

**@***staticmethod*

***def*** **main**():

**pass**

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    PercolationStats.main()

Throw an Error in the constructor if either *n* ≤ 0 or *trials* ≤ 0.

Also, include a main() method that takes two *command-line* *arguments n and T*, performs *T* independent computational experiments (discussed above) on an *n*-by-*n* grid, and prints the sample mean, sample standard deviation, and the *95% confidence interval* for the percolation threshold. You can use random for generating random numbers and std\_stats.py calculating mean, standard deviation, and confidence interval.

Here is what it would look like in the terminal:

~/Desktop/percolation> python percolation\_stats.py 200 100

mean = 0.5929934999999997

stddev = 0.00876990421552567

95% confidence interval = [0.5912745987737567, 0.5947124012262428]

~/Desktop/percolation> python percolation\_stats.py 200 100

mean = 0.592877

stddev = 0.009990523717073799

95% confidence interval = [0.5909188573514536, 0.5948351426485464]

~/Desktop/percolation> python percolation\_stats.py 2 10000

mean = 0.666925

stddev = 0.11776536521033558

95% confidence interval = [0.6646167988418774, 0.6692332011581226]

~/Desktop/percolation> python percolation\_stats.py 2 100000

mean = 0.6669475

stddev = 0.11775205263262094

95% confidence interval = [0.666217665216461, 0.6676773347835391]

**Analysis of running time and memory (optional and not graded).** Implement the Percolation data type using the *quick find* algorithm in QuickFindUF.

* Use Stopwatch to measure the total running time of PercolationStats for various values of *n* and *T*. How does doubling *n* affect the total running time? How does doubling *T* affect the total running time?

Now, implement the Percolation data type using the *weighted quick union* algorithm in WeightedQuickUnionUF. Answer the questions in the previous paragraph.

**Web submission.** Submit a .zip file containing only percolation.py, percolation\_stats.py and the analysis of running time and memory (if you did it). Your submissions may not call library functions except those in std\_in.py, std\_out.py, random, std\_stats.py, weighted\_quick\_union\_uf.py. If you really need a certain library please let me know first. Deadline: February 4, 2025 11:59pm

Submit to <https://www.dropbox.com/request/j96Ql6S5kK92g1cSZBZI> .

**For fun.** Create your own percolation input file and share it over on our discord server. For some inspiration, do an image search for “nonogram puzzles solved”.