

三、实验内容及基本要求

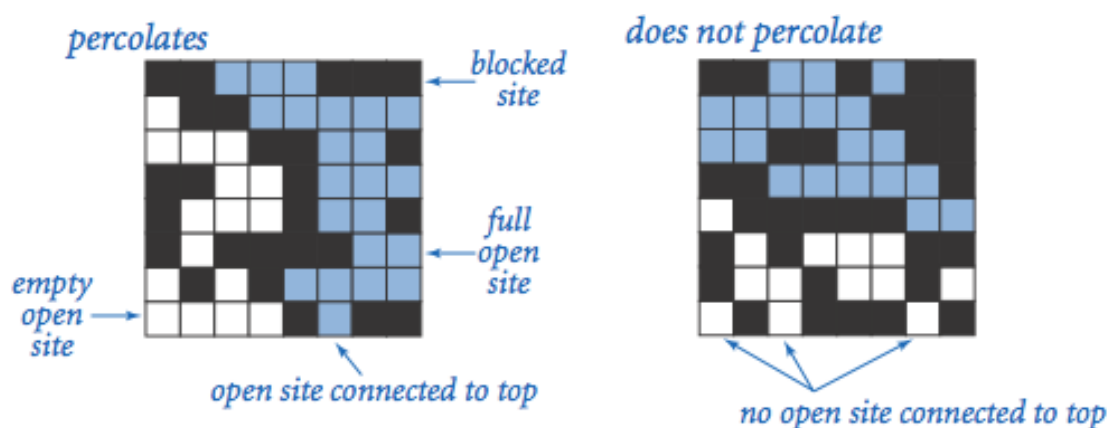
(一) 渗透问题 (Percolation)

使用合并-查找 (union-find) 数据结构, 编写程序通过蒙特卡罗模拟 (Monte Carlo simulation) 来估计渗透阈值。

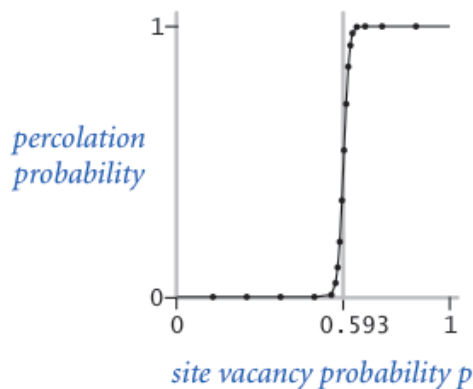
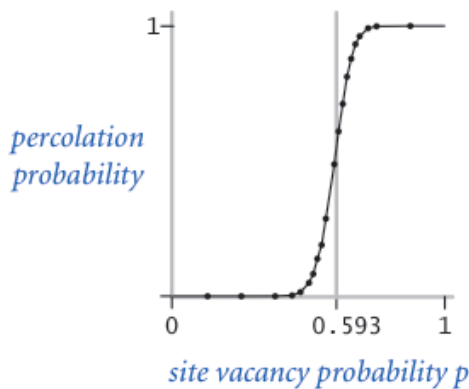
安装 Java 编程环境。 按照以下各步指令, 在计算机上 (操作系统 [Mac OS X](http://algs4.cs.princeton.edu/mac) (<http://algs4.cs.princeton.edu/mac>) · [Windows](http://algs4.cs.princeton.edu/windows) (<http://algs4.cs.princeton.edu/windows>) · [Linux](http://algs4.cs.princeton.edu/linux) (<http://algs4.cs.princeton.edu/linux>) 安装 Java 编程环境。执行这些指令后, 在你 Java classpath 下会有 [stdlib.jar](#) and [algs4.jar](#)。前者包含库: 从标准输入读数据、向标准输出写数据以及向标准绘制绘出结果, 产生随机数、计算统计量以及计时程序; 后者包含了教科书中的所有算法。

给定由随机分布的绝缘材料和金属材料构成的组合系统: 金属材料占多大比例才能使组合系统成为电导体? 给定一个表面有水的多孔景观 (或下面有油), 水将在什么条件下能够通过底部排出 (或油渗透到表面)? 科学家已经定义了一个称为渗透 (*percolation*) 的抽象过程来模拟这种现象。

模型。 使用 $N \times N$ 网格点来建模一个渗透系统。每个格点或是 *open* 格点或是 *blocked* 格点。定义 *full site* 格点为 *open* 格点。如果在网格最底行中存在一个 *full site* 格点, 它可以通过一系列近邻 (左、右、上、下) *open* 格点连通到顶行的一个 *open* 格点, 则称该系统是渗透的。(对于绝缘/金属材料的例子, *open* 格点对应于金属材料, 渗透系统有一条从顶行到底行的金属路径, 且 *full sites* 格点导电。对于多孔物质示例, *open* 格点对应于空, 水可能流过, 从而渗透系统使水充满 *open* 格点, 自顶向下流动。)



科学问题: 如果将格点以概率 p 独立地设置为 *open* 格点 (因此格点被设置为 *blocked* 格点的概率为 $1 - p$), 那么系统渗透的概率是多少? 当 $p = 0$ 时, 系统不会渗出; 当 $p = 1$ 时, 系统必定渗透。下图显示了 20×20 随机网格 (左) 和 100×100 随机网格 (右) 的格点空 (对应 *open* 格点) 置概率 p 与渗透概率。



当 N 足够大时, 存在阈值 p^* , 使得当 $p < p^*$, 随机 $N \times N$ 网格几乎不会渗透, 并且当 $p > p^*$ 时, 随机 $N \times N$ 网格几乎总是渗透。尚未得出用于确定渗透阈值 p^* 的数学解。你的任务是编写一个计算机程序来估计 p^* 的值。

Percolation 数据类型。 模型化一个 Percolation 系统, 创建含有以下 API 的数据类型 Percolation。

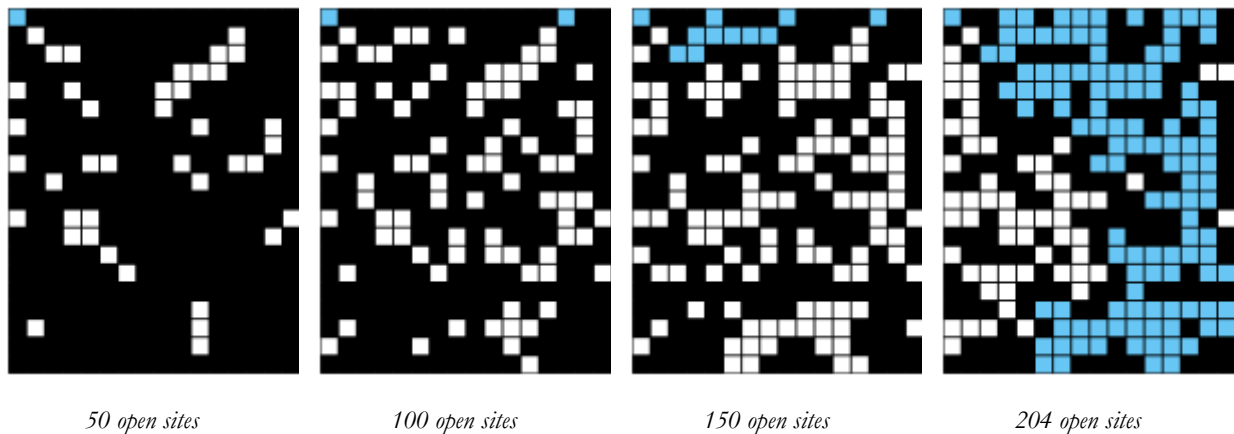
```
public class Percolation {
    public Percolation(int N)           // create N-by-N grid, with all sites blocked
    public void open(int i, int j)       // open site (row i, column j) if it is not already
    public boolean isOpen(int i, int j)  // is site (row i, column j) open?
    public boolean isFull(int i, int j)  // is site (row i, column j) full?
    public boolean percolates()          // does the system percolate?
    public static void main(String[] args) // test client, optional
}
```

约定行 i 列 j 下标在 1 和 N 之间, 其中 (1, 1) 为左上格点位置: 如果 `open()`, `isOpen()`, or `isFull()` 不在这个规定的范围, 则抛出 `IndexOutOfBoundsException` 例外。如果 $N \leq 0$, 构造函数应该抛出 `IllegalArgumentException` 例外。构造函数应该与 N^2 成正比。所有方法应该为常量时间加上常量次调用合并-查找方法 `union()`, `find()`, `connected()`, and `count()`。

蒙特卡洛模拟 (Monte Carlo simulation)。 要估计渗透阈值, 考虑以下计算实验:

- 初始化所有格点状态为 *blocked*。
- 重复以下操作直到系统渗出:
 - 在所有 *blocked* 的格点之间随机均匀选择一个格点 (row i , column j)。
 - 设置这个格点 (row i , column j) 为 *open* 格点。
- *open* 格点的比例提供了系统渗透时渗透阈值的一个估计。

例如, 如果在 20×20 的网格中, 根据以下快照的 *open* 格点数, 那么对渗透阈值的估计是 $204/400 = 0.51$, 因为当第 204 个格点被 *open* 时系统渗透。



通过重复该计算实验 T 次并对结果求平均值，我们获得了更准确的渗滤阈值估计。令 x_t 是第 t 次计算实验中 *open* 格点所占比例。样本均值 μ 提供渗滤阈值的一个估计值；样本标准差 σ 测量阈值的灵敏性。

$$\mu = \frac{x_1 + x_2 + \dots + x_T}{T}, \quad \sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \dots + (x_T - \mu)^2}{T-1}$$

假设 T 足够大（例如至少 30），以下为渗滤阈值提供 95% 置信区间：

$$\left[\mu - \frac{1.96\sigma}{\sqrt{T}}, \mu + \frac{1.96\sigma}{\sqrt{T}} \right]$$

通过创建数据类型 `PercolationStats` 来执行一系列计算实验，包含以下 API。

```
public class PercolationStats {
    public PercolationStats(int N, int T) //perform T independent computational experiments on an NxN grid
    public double mean()                  // sample mean of percolation threshold
    public double stddev()                // sample standard deviation of percolation threshold
    public double confidenceLo()          // returns lower bound of the 95% confidence interval
    public double confidenceHi()          // returns upper bound of the 95% confidence interval
    public static void main(String[] args) // test client, described below
}
```

在 $N \leq 0$ 或 $T \leq 0$ 时，构造函数应该抛出 `java.lang.IllegalArgumentException` 例外。

此外，还包括一个 `main()` 方法，在命令行输入两个参数 N 和 T ，在 $N \times N$ 网格上进行 T 次独立计算实验（上面讨论），并打印出均值 μ 、标准差 σ 和 95% 渗透阈值的置信区间。使用标准库中的标准随机数生成随机数；使用标准统计库来计算样本均值和标准差。

```
Example values after creating PercolationStats(200, 100)
mean()                = 0.5929934999999997
stddev()              = 0.00876990421552567
confidenceLow()       = 0.5912745987737567
confidenceHigh()      = 0.5947124012262428
```

```
Example values after creating PercolationStats(200, 100)
mean()                = 0.592877
stddev()              = 0.009990523717073799
```

```
confidenceLow()          = 0.5909188573514536
confidenceHigh()         = 0.5948351426485464
```

Example values after creating PercolationStats(2, 100000)

```
mean()                   = 0.6669475
stddev()                 = 0.11775205263262094
confidenceLow()          = 0.666217665216461
confidenceHigh()         = 0.6676773347835391
```

运行时间和内存占用分析。

使用 *quick-find* 算法（[QuickFindUF.java](#) from algs4.jar）实现 Percolation 数据类型。进行实验表明当 N 加倍时对运行时间的影响；使用近似表示法，给出在计算机上的总时间，它是输入 N 和 T 的函数表达式。

使用 *weighted quick-union* 算法（[WeightedQuickUnionUF.java](#) from algs4.jar）实现 Percolation 数据类型。进行实验表明当 N 加倍时对运行时间的影响；使用近似表示法，给出在计算机上的总时间，它是输入 N 和 T 的函数表达式。

注：这个问题的实验由 Bob Sedgewick 和 Kevin Wayne 设计开发（Copyright © 2008）。更多信息可参考 <http://algs4.cs.princeton.edu/home/>。