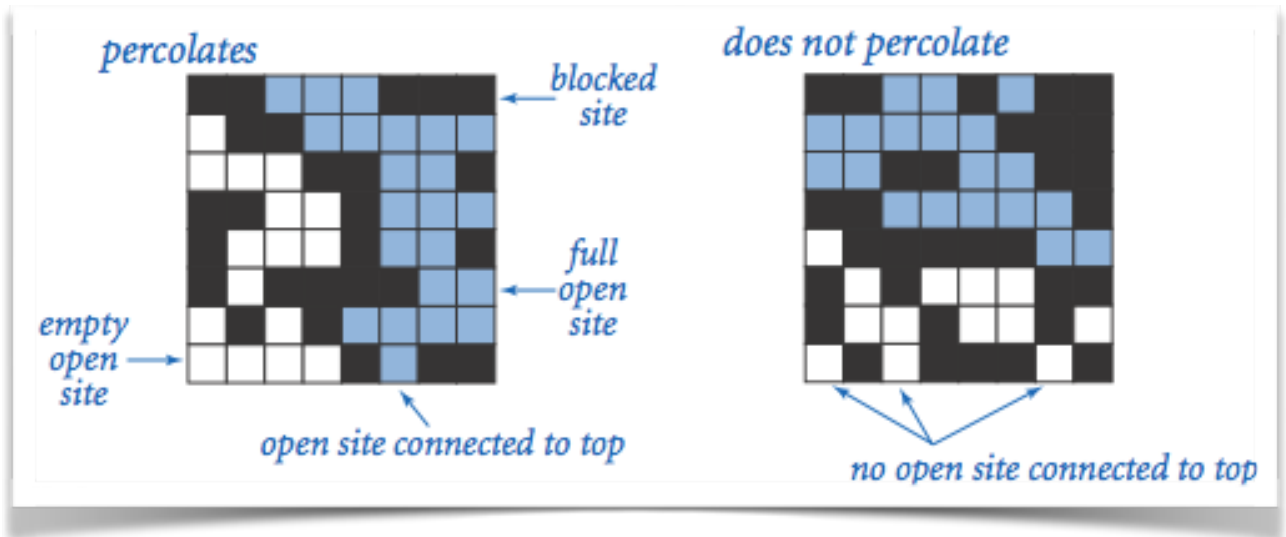


渗透问题(Percolation)

一、模型

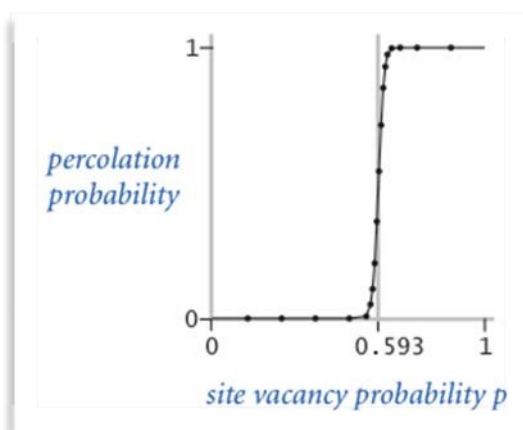
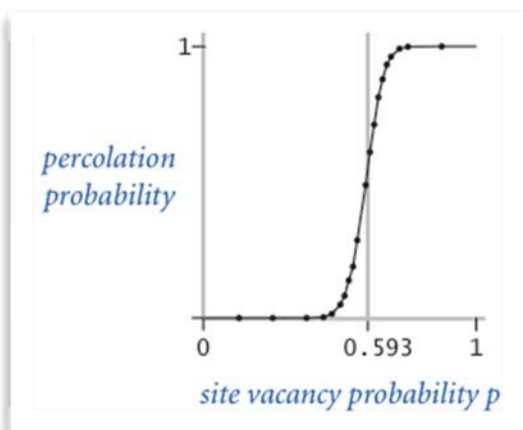
我们使用 $N \times N$ 网格点来模型一个渗透系统。每个格点或是 open 格点或是 blocked 格点。一个 full site 是一个 open 格点，它可以通过一连串的邻近（左，右，上，下）open 格点连通到顶行的一个 open 格点。如果在底行中有一个 full site 格点，则称系统是渗透的。



二、问题描述

如果将格点以概率 p 独立地设置为 open 格点（因此以概率 $1-p$ 被设置为 blocked 格点），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。



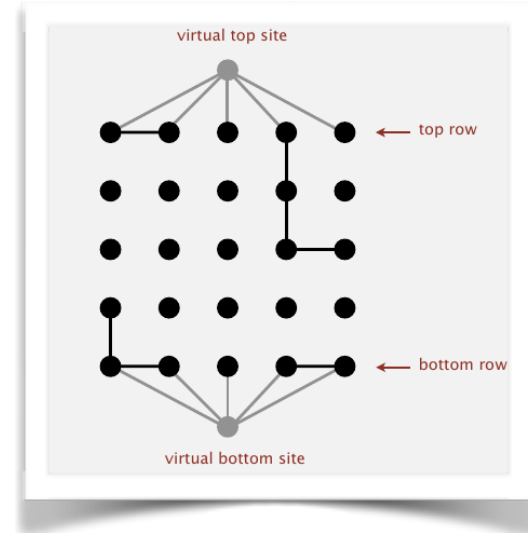
当 N 足够大时, 存在阈值 p^* , 使得当 $p < p^*$, 随机 $N \times N$ 网格几乎不会渗透, 并且当 $p > p^*$ 时, 随机 $N \times N$ 网格几乎总是渗透。尚未得出用于确定渗透阈值 p^* 的数学解。要求编写一个计算机程序来估计

p^* 。

三、解决思路

1. 渗透模型是连通性问题的一个应用。如果按照正常的思路, 要使得渗透系统渗透, 其必要条件是网格的顶部一行至少要有有一个open格点, 同时底部也至少要有有一个open格点。

其实更加巧妙的做法是: 多加两个虚拟点(virtual site), 如果这个模型是渗透的, 那么虚顶点和虚底点就是连通的。问题一下子就简单了。如右图所示。



2. 渗透模型渗透的概率是多少呢? 假设有一个20乘20的模型, 网格中白色(unblocked)的概率是 p , 则黑色(blocked)的概率是 $1-p$, 那么这个模型渗透的概率是多少呢? 看样子不太好用数学公式描述, 事实上它就是不能解的。解决方法是用大量模拟实验(simulation)来找到这个概率, 也就是蒙特卡罗采样了。当 p 比某个值大的时候, 模型几乎总是渗透, p 比某个值小的时候模型几乎不渗透。这么非线性看来确实不好用数学公式描述。

四、渗透模型所需的API及其实现

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。

```
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

2

IllegalArgumentException 例外。构造函数应该与 *N* 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()。

1. public Percolation(int N){}构造函数。如果 $N \leq 0$ ，构造函数抛出异常。

```
/**
 * 创建一个N*N的渗透系统，所有的格子都为blocked
 */
public Percolation(int N) {
    if (N <= 0) {
        throw new IllegalArgumentException(N + " is illegal!");
    }
    this.grid = new boolean[N][N];
    this.size = N;
    this.bottom = N * N + 1;
    uf = new WeightedQuickUnionUF(n: N * N + 2);
}
```

2. void open(int i, int j){}方法。

```

/**
 * 打开(i, j)这个格子。
 * 如果已经打开了, 就返回;
 * 如果是阻塞的, 就把这个格子打开
 */
public void open(int i, int j) {
    if (isOpen(i, j)) {
        return;
    }

    if (!isOpen(i, j)) {
        grid[i - 1][j - 1] = true;    //如果是阻塞的, 那就打开它
        count++;

        int site = gridSite(i, j);

        if (i == 1) {
            uf.union(p: 0, site);
        }

        if (i == site) {
            uf.union(bottom, site);
        }

        int rowOfSite, columnOfSite;
        for (int k = 0; k < 4; k++) {
            rowOfSite = i;
            columnOfSite = j;
            switch (k) {
                case 0:
                    columnOfSite--;    //site左边的格子
                    break;
                case 1:
                    rowOfSite--;    //site上边的格子
                    break;
                case 2:
                    columnOfSite++;    //site右边的格子
                    break;
            }
        }
    }
}

/**
 * 判断(i, j)这个格子是否是开着的
 */
public boolean isOpen(int i, int j) {    //is site (row i, column j) open?
    validate(i, j);
    return grid[i - 1][j - 1];
}
}
}
}

```

3. boolean isOpen(int i, int j) {} 方法。

{ } 方法。

4. public boolean isFull(int i, int j)

```

/**
 * 如果该格子和顶部的虚拟格子(用0表示, 即和第一排所有格子)相连通, 返回true
 */
public boolean isFull(int i, int j) {    //is site full?
    validate(i, j);
    int site = gridSite(i, j);
    return uf.connected(site, q: 0);
}

```

5. public boolean percolates() {} 方法

```
/**
 * 如果虚顶点和虚底点是连通的，那么渗透系统是渗透的
 */
public boolean percolates() { //does the system percolate?
    return uf.connected( p: 0, bottom);
}
```

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。

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透阈值估计。令 x_t 是第 *t* 次计算实验中 **open** 格点所占比例。样本均值 μ 提供渗透阈值的一个估计值；样本标准差 σ 测量阈值的灵敏性。

我们创建数据类型 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
    N-by-N 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。

1. public PercolationStats(int *N*, int *T*) 构造函数。

```

public class PercolationStats {
    private double totalOfThreshold ; // 阈值之和
    private int times;                // 实验次数
    private ArrayList<Double> threshold = new ArrayList<>();

    public PercolationStats(int N, int T) {
        this.times = T;

        if (N <= 0 || T <= 0) {
            throw new IllegalArgumentException("N or T is illegal!");
        }
        while (T > 0) {
            T--;
            Percolation percolation = new Percolation(N);
            totalOfThreshold += percolation.threshold(N, percolation);
            threshold.add(percolation.threshold(N, percolation));
        }
    }
}

```

2. public double mean(){} 方法。

```

// 求阈值的平均值
public double mean() {
    return totalOfThreshold / times;
}

```

3. public double stddev(){} 方法。

```

public double stddev() {
    double stddev = 0; // 方差
    for (int i = 0; i < times; i++) {
        stddev += (threshold.get(i) - mean()) * (threshold.get(i) - mean());
    }
    return Math.sqrt(stddev) / (times - 1);
}

```

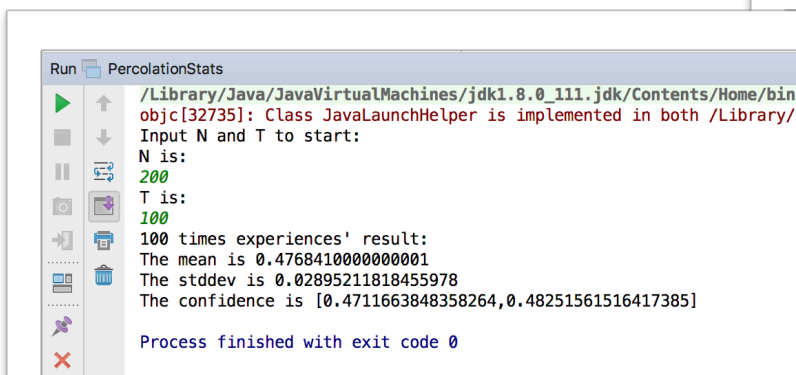
4. 置信区间。

```
public double confidenceLo() { return mean() - ((1.96 * stddev()) / Math.sqrt(times)); }  
public double confidenceHi() { return mean() + ((1.96 * stddev()) / Math.sqrt(times)); }
```

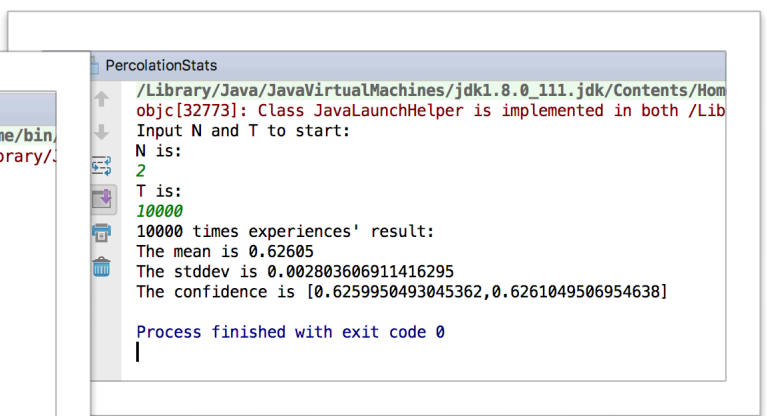
5. 渗透阈值的估计。

```
public double threshold(int n, Percolation test) { //渗透阈值的估计  
    while (!test.percolates()) {  
        int r = 1 + (int) (Math.random() * n);  
  
        int c = 1 + (int) (Math.random() * n);  
        if (test.isOpen(r, c)) {  
  
        } else {  
            test.open(r, c);  
        }  
    }  
    return (double) count / (n * n);  
}
```

五、运行结果



```
Run PercolationStats  
/Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Home/bin/  
objc[32735]: Class JavaLaunchHelper is implemented in both /Library/  
Input N and T to start:  
N is:  
200  
T is:  
100  
100 times experiences' result:  
The mean is 0.4768410000000001  
The stddev is 0.02895211818455978  
The confidence is [0.4711663848358264,0.48251561516417385]  
Process finished with exit code 0
```



```
PercolationStats  
/Library/Java/JavaVirtualMachines/jdk1.8.0_111.jdk/Contents/Hom  
objc[32773]: Class JavaLaunchHelper is implemented in both /Lib  
Input N and T to start:  
N is:  
2  
T is:  
10000  
10000 times experiences' result:  
The mean is 0.62605  
The stddev is 0.002803606911416295  
The confidence is [0.6259950493045362,0.6261049506954638]  
Process finished with exit code 0
```