实验一 渗透问题（Percolation）

一、实验目的

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）来估计渗透阈值的值。

二、内容描述

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渗水地形（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）？科学家们已经定义了一个称为渗透（percolation）的抽象过程来模拟这种情况。

如图 1.2.1 所示，我们使用 N×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。（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 格点对应于空格，水可能流过，从而渗透系统使水充满 open 格点，自顶向下流动。

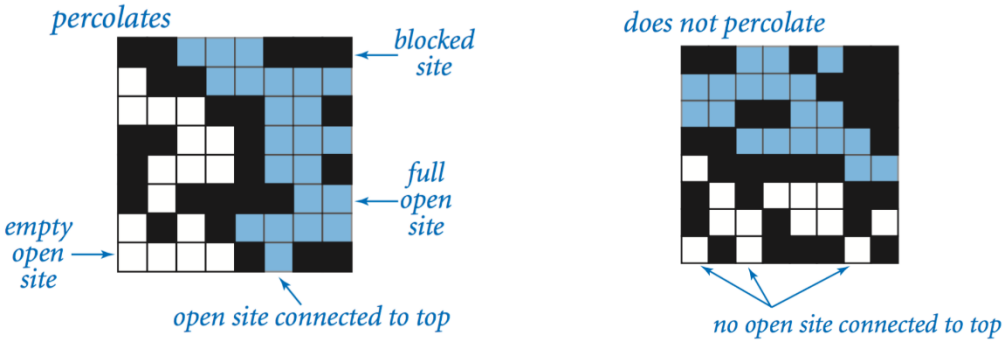

图 1.2.1

在上述模型中，我们将格点以空置概率 p 独立地设置为 open 格点（因此以概率 1-p 被设置为 blocked 格点）。当 p = 0 时，系统不会渗出；当 p=1 时，系统渗透。图 1.2.2 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 与渗滤概率。

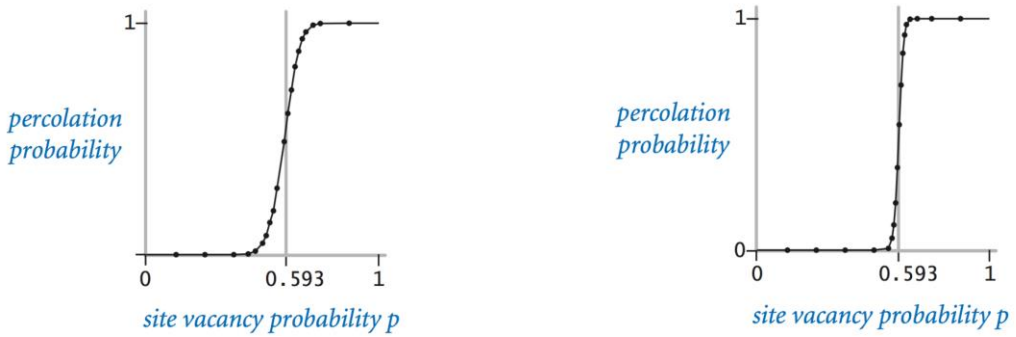

图 1.2.2

当 N 足够大时，存在阈值 p*，使得当 p <p*，随机 N×N 网格几乎不会

渗透，并且当 p> p*时，随机 N×N 网格几乎总是渗透。尚未得出用于确定渗滤阈值 p*的数学解。本次实验要求编写一个计算机程序来估计 p*。

三、模拟方法

本次实验，我们使用蒙特卡洛模拟来估计渗透阈值。蒙特卡洛模拟的步骤如下：

- 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；

我们利用 open 格点的比例来估计系统渗透时的渗透阈值。例如，如果在 20×20 的网格中，如果我们测得当第 204 个格点被 open 时系统渗透，那么对渗滤阈值的估计是 204/400 = 0.51。

通过重复该计算实验 T 次并对结果求平均值，可以获得了更准确的渗滤阈值估计。

同时，我们利用公式

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

以及公式

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

可以求得渗滤阈值提供 95%置信区间。

四、具体实现

1. Percolation 系统模型化

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

```
1.  public class Percolation {
2.      // 建立一个 N×N 的网格，初始化为 block 状态
3.      public Percolation(int n)
4.      // open 函数将网格从 block 变为 open 状态
5.      public void open(int row, int col)
6.      // 判断网格 (row, col) 是否为 open 状态
7.      public boolean isOpen(int row, int col)
8.      //  判断网格 (row, col) 是否为 full 状态
9.      public boolean isFull(int row, int col)
10.     // 返回 open sites 数量
11.     public int numberOfOpenSites()
12.     // 判断系统是否渗透
13.     public boolean percolates()
14.     // 主函数
15.     public static void main(String[] args)
16. }
```

其中，若网格(row, col)与顶部连通，则称该网格处于 full 状态。

a) Percolation 函数

```
1.  public Percolation(int n) {
2.          if (n <= 0) {
3.              throw new IllegalArgumentException("n 必须大于 0! ");
4.          }
5.          this.n = n;
6.          virtualtop = 0;//顶部虚节点存在数组头部
7.          virtualbtm = n * n + 1;//底部虚节点存在数组尾部
8.          virtualQU = new WeightedQuickUnionUF(n * n + 2);
9.          realQU = new WeightedQuickUnionUF(n * n + 1);
10.         isOpen = new boolean[n * n + 2];
11.         isOpen[virtualtop] = true;
12.         isOpen[virtualbtm] = true;//2 个虚节点均置为 open
13. }
```

在 Percolation 函数中通过传入参数 n 建立一个 n×n 的数组，分别建立大小为 n×n＋2 和 n×n＋1 的 QuickUnion 对象。其中建立大小为 n×n＋1 的 QuickUnion 对象是为了解决回溯（backwash）问题，这会在后面探讨。

b) Open()函数

```
1.  public void open(int row, int col) {
2.       int curIndex = linearIndex(row, col);
3.       isOpen[curIndex] = true;
4.       openCount++;
5.
6.       if (row == 1) {//第 1 行
7.           virtualQU.union(curIndex, virtualtop);
8.           realQU.union(curIndex, virtualtop);
9.       }
10.      if (row == n) {//最后一行
11.          virtualQU.union(curIndex, virtualbtm);
12.      }
13.      neighborConnect(row, col, row - 1, col);  // 上
14.      neighborConnect(row, col, row + 1, col);  // 下
15.      neighborConnect(row, col, row, col - 1);  // 左
16.      neighborConnect(row, col, row, col + 1);  // 右
17.  }
```

c) isopen()函数

```
1.    public boolean isOpen(int row, int col) {
2.         return isOpen[linearIndex(row, col)];
3.     }
```

d) isfull()函数

```
1.  public boolean isFull(int row, int col) {
2.          return realQU.connected(virtualtop,linearIndex(row, col)
3.         //必须用 realQU 判断而不能用 virtualQU，否则出现 backwash 现象
4.  }
```

这里用了大小为 n×n＋1 的 QuickUnion 对象来判断 isfull，是因为如果将虚拟底部节点（virtualbottom）算进来，那么实际未和顶部连通的方块可以通过虚拟底部节点（virtualbottom）与顶部连通。

e) Numberofopensites()函数

```
1.  public int numberOfOpenSites() {
2.          return openCount;
3.  }
```

f) Percolates()函数

```
1.  public boolean percolates() {
2.          return virtualQU.connected(virtualtop, virtualbtm);
3.          //判断渗透用 virtualQU 判断
4.  }
```

2. PercolationStats 测试模型

我们创建数据类型 PercolationStats 来执行一系列计算实验，包含以下 API ：

```
1.  public class PercolationStats {
2.      // perform independent trials on an n-by-n grid
3.      public PercolationStats(int n, int trials)
4.      // sample mean of percolation threshold
5.      public double mean()
6.      // sample standard deviation of percolation threshold
7.      public double stddev()
8.      // low endpoint of 95% confidence interval
9.      public double confidenceLo()
10.     // high endpoint of 95% confidence interval
11.     public double confidenceHi()
12.    // test client (see below)
13.    public static void main(String[] args)
14. }
```

此外，还包括一个 main( )方法，它取两个命令行参数 N 和 T，在 N×N 网格上进行 T 次独立的计算实验（上面讨论），并打印出均值、标准差和 95%渗透阈值的置信区间。

a) PercolationStats()函数

```
1.  public PercolationStats(int n, int trials) {
2.          if (n <= 0) {
3.              throw new IllegalArgumentException("必须大于 0");
4.          }
5.          if (trials <= 0) {
6.              throw new IllegalArgumentException("必须大于 0");
7.          }
8.          fractions = new double[trials];
9.          for (int i = 0; i < trials; i++) {
10.         //进行 trials 次随机实验
11.             Percolation percolation = new Percolation(n);
12.             int openedSites = 0;
13.             while (!percolation.percolates()) {
14.             //每次开 1 个方块直到 percolates 为止
15.                 int row = StdRandom.uniform(n) + 1;
16.                 int col = StdRandom.uniform(n) + 1; //随机 open
17.                 if (!percolation.isOpen(row, col)) {
18.                     percolation.open(row, col);
19.                     openedSites++;//每开 1 个格子 opensites 加 1
20.                 }
21.             }
22.             fractions[i] = openedSites * 1.0 / (n * n);
23.            //恰好渗透时 open 的格子数比例
24.         }
25. }
```

b) Mean()函数

```
1.  public double mean() {
2.          return StdStats.mean(fractions);
3.  }
```

c) Stddev()函数

```
1.  public double stddev() {
2.         return StdStats.stddev(fractions);
3.  }
```

d) confidenceLo()函数

```
1.  public double confidenceLo() {
2.          return mean() - CONFIDENCE_95 * stddev()
3.  }
```

e) confidenceHi()函数

```
1.  public double confidenceHi() {
2.         return mean() + CONFIDENCE_95 * stddev()
3.  }
```

五、源代码

Percolation.java：

```
1.  import edu.princeton.cs.algs4.StdOut;
2.  import edu.princeton.cs.algs4.WeightedQuickUnionUF;
3.
4.  public class Percolation {
5.
6.      private final WeightedQuickUnionUF realQU;//不包含 virtual bottom
7.      private final WeightedQuickUnionUF virtualQU;//包含 2 个虚节点
8.      private final boolean[] isOpen;//方块的状态参数
9.      private final int virtualtop;//顶部虚节点
10.     private final int virtualbtm;//底部虚节点
11.     private final int n;
12.     private int openCount;
13.     public Percolation(int n) {
14.         if (n <= 0) {
15.             throw new IllegalArgumentException("n 必须大于 0! ");
16.         }
17.         this.n = n;
18.         virtualtop = 0;//顶部虚节点存在数组头部
19.         virtualbtm = n * n + 1;//底部虚节点存在数组尾部
20.         virtualQU = new WeightedQuickUnionUF(n * n + 2);
21.         realQU = new WeightedQuickUnionUF(n * n + 1);
22.         isOpen = new boolean[n * n + 2];
23.         isOpen[virtualtop] = true;
24.         isOpen[virtualbtm] = true;//2 个虚节点均置为 open
25.     }
26.
27.     //将二维下标转化为一维下标
28.     private int linearIndex(int row, int col) {
29.         if (row < 1 || row > n) {
30.             throw new IndexOutOfBoundsException("行越界! ");
31.         }
32.         if (col < 1 || col > n) {
33.             throw new IndexOutOfBoundsException("列越界! ");
```

```
        }
        return (row - 1) * n + col;
    }

    public void open(int row, int col) {
        int curIndex = linearIndex(row, col);
        isOpen[curIndex] = true;
        openCount++;

        if (row == 1) {//第 1 行
            virtualQU.union(curIndex, virtualtop);
            realQU.union(curIndex, virtualtop);
        }
        if (row == n) {//最后一行
            virtualQU.union(curIndex, virtualbtm);
        }
        neighborConnect(row, col, row - 1, col);  // 上
        neighborConnect(row, col, row + 1, col);  // 下
        neighborConnect(row, col, row, col - 1);  // 左
        neighborConnect(row, col, row, col + 1);  // 右
    }

    //将 a 与 open 的邻居相连接
    private void neighborConnect(int rowA, int colA, int rowB, int colB) {
        if (0 < rowB && rowB <= n && 0 < colB && colB <= n
                && isOpen(rowB, colB)) {
            virtualQU.union(linearIndex(rowA, colA), linearIndex(rowB, colB));
            realQU.union(linearIndex(rowA, colA), linearIndex(rowB, colB));
        }
    }
    //计数
    public int numberOfOpenSites() {
        return openCount;
    }


    public boolean isOpen(int row, int col) {
        return isOpen[linearIndex(row, col)];
    }

    public boolean isFull(int row, int col) {
        return realQU.connected(virtualtop, linearIndex(row, col));
//必须用 realQU 判断而不能用 virtualQU，否则出现 backwash 现象
    }

    public boolean percolates() {
        return virtualQU.connected(virtualtop, virtualbtm);//判断渗透用 virtualQU
    判断
    }

    public static void main(String[] args) {
        StdOut.println("请运行 PercolationStats 程序！ ");
    }
}

```

PercolationStats.java：

```
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdRandom;
import edu.princeton.cs.algs4.StdStats;


public class PercolationStats {

    private final double[] fractions;
    private final double CONFIDENCE_95 = 1.96;
```

```
    public PercolationStats(int n, int trials) {
        if (n <= 0) {
            throw new IllegalArgumentException("n 必须大于 0! ");
        }
        if (trials <= 0) {
            throw new IllegalArgumentException("trials 必须大于 0! ");
        }
        fractions = new double[trials];
        for (int i = 0; i < trials; i++) {//进行 trials 次随机实验
            Percolation percolation = new Percolation(n);
            int openedSites = 0;
            while (!percolation.percolates()) {//每次开 1 个方块直到 percolates
                int row = StdRandom.uniform(n) + 1;
                int col = StdRandom.uniform(n) + 1; //随机 open 方块
                if (!percolation.isOpen(row, col)) {
                    percolation.open(row, col);
                    openedSites++;//每开 1 个格子 opensites 加 1
                }
            }
            fractions[i] = openedSites * 1.0 / (n * n);//恰好渗透 open 的格子比例
        }
    }

    //*p 均值
    public double mean() {
        return StdStats.mean(fractions);
    }

    //标准偏差
    public double stddev() {
        return StdStats.stddev(fractions);
    }

    //置信区间
    public double confidenceLo() {
        return mean() - CONFIDENCE_95 * stddev() / Math.sqrt(fractions.length);
    }


    public double confidenceHi() {
        return mean() + CONFIDENCE_95 * stddev() / Math.sqrt(fractions.length);
    }


    public static void main(String[] args) {

        int n = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        PercolationStats stats = new PercolationStats(n, trials);

        StdOut.println("mean                    = " + stats.mean());
        StdOut.println("stddev                  = " + stats.stddev());
        StdOut.println("95% confidence interval = "
                               + stats.confidenceLo() + ", "
                               + stats.confidenceHi());
    }
}
```

六、实验结果

输入：200 100

```
mean                    = 0.5947457499999999
stddev                  = 0.010452860185039303
95% confidence interval = 0.5926969894037322, 0.5967945105962675

Process finished with exit code 0
```

输入：200 100

```
mean                    = 0.59324125
stddev                  = 0.010217876105560578
95% confidence interval = 0.5912385462833102, 0.5952439537166899

Process finished with exit code 0
```

输入：2 10000

```
mean                    = 0.668525
stddev                  = 0.11718339289231955
95% confidence interval = 0.6662282054993106, 0.6708217945006895

Process finished with exit code 0
```

输入：2 100000

```
mean                    = 0.6664675
stddev                  = 0.11792194664108388
95% confidence interval = 0.6657366122024869, 0.6671983877975131

Process finished with exit code 0
```

七、实验总结

本次实验主要考察了 quickfind()、quickunion()方法在连通问题上的使用。实验过程中把握住以下两点即可较为容易的完成实验。

模型-算法抽象。在实验中将是否渗透问题转化为连通问题。巧妙地借助两个虚拟结点便将系统渗透等价为了两结点连通。

阈值估计模型。每个网格开放的概率 p 是抽象的，不好在实验中控制。因此，利用蒙特卡洛模型将阈值 p*转化为恰好渗透时 open 的格子数与总格子数之比，这样便可形象地估计出阈值 p*。

掌握了以上两点，问题便迎刃而解了，剩余的便是利用算法课程所学习的 quick union 算法编写程序了。