**实验一 渗透问题（Percolation）**

1. **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。

1. **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。

如图一些文字和图案

中度可信度描述已自动生成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。

文本

低可信度描述已自动生成

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\*。

1. **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

,

以及公式

可以求得渗滤阈值提供95％置信区间。

1. **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。

1. **Percolation函数**
2. public Percolation(int n) {
3. if (n <= 0) {
4. throw new IllegalArgumentException("n必须大于0！");
5. }
6. this.n = n;
7. virtualtop = 0;//顶部虚节点存在数组头部
8. virtualbtm = n \* n + 1;//底部虚节点存在数组尾部
9. virtualQU = new WeightedQuickUnionUF(n \* n + 2);
10. realQU = new WeightedQuickUnionUF(n \* n + 1);
11. isOpen = new boolean[n \* n + 2];
12. isOpen[virtualtop] = true;
13. isOpen[virtualbtm] = true;//2个虚节点均置为open
14. }

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。

1. **Open()函数**
2. public void open(int row, int col) {
3. int curIndex = linearIndex(row, col);
4. isOpen[curIndex] = true;
5. openCount++;
7. if (row == 1) {//第1行
8. virtualQU.union(curIndex, virtualtop);
9. realQU.union(curIndex, virtualtop);
10. }
11. if (row == n) {//最后一行
12. virtualQU.union(curIndex, virtualbtm);
13. }
14. neighborConnect(row, col, row - 1, col);  // 上
15. neighborConnect(row, col, row + 1, col);  // 下
16. neighborConnect(row, col, row, col - 1);  // 左
17. neighborConnect(row, col, row, col + 1);  // 右
18. }
19. **isopen()函数**
20. public boolean isOpen(int row, int col) {
21. return isOpen[linearIndex(row, col)];
22. }
23. **isfull()函数**
24. public boolean isFull(int row, int col) {
25. return realQU.connected(virtualtop,linearIndex(row, col)
26. //必须用realQU判断而不能用virtualQU，否则出现backwash现象
27. }

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。

1. **Numberofopensites()函数**
2. public int numberOfOpenSites() {
3. return openCount;
4. }
5. **Percolates()函数**
6. public boolean percolates() {
7. return virtualQU.connected(virtualtop, virtualbtm);
8. //判断渗透用virtualQU判断
9. }
   1. **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。

1. **PercolationStats()函数**
2. public PercolationStats(int n, int trials) {
3. if (n <= 0) {
4. throw new IllegalArgumentException("必须大于0");
5. }
6. if (trials <= 0) {
7. throw new IllegalArgumentException("必须大于0");
8. }
9. fractions = new double[trials];
10. for (int i = 0; i < trials; i++) {
11. //进行trials次随机实验
12. Percolation percolation = new Percolation(n);
13. int openedSites = 0;
14. while (!percolation.percolates()) {
15. //每次开1个方块直到percolates为止
16. int row = StdRandom.uniform(n) + 1;
17. int col = StdRandom.uniform(n) + 1; //随机open
18. if (!percolation.isOpen(row, col)) {
19. percolation.open(row, col);
20. openedSites++;//每开1个格子opensites加1
21. }
22. }
23. fractions[i] = openedSites \* 1.0 / (n \* n);
24. //恰好渗透时open的格子数比例
25. }
26. }
27. **Mean()函数**
28. public double mean() {
29. return StdStats.mean(fractions);
30. }
31. **Stddev()函数**
32. public double stddev() {
33. return StdStats.stddev(fractions);
34. }
35. **confidenceLo()函数**
36. public double confidenceLo() {
37. return mean() - CONFIDENCE\_95 \* stddev()
38. }
39. **confidenceHi()函数**
40. public double confidenceHi() {
41. return mean() + CONFIDENCE\_95 \* stddev()
42. }
43. **源代码**

**Percolation.java：**

1. import edu.princeton.cs.algs4.StdOut;
2. import edu.princeton.cs.algs4.WeightedQuickUnionUF;
4. public class Percolation {
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. }
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！");
34. }
35. return (row - 1) \* n + col;
36. }
38. public void open(int row, int col) {
39. int curIndex = linearIndex(row, col);
40. isOpen[curIndex] = true;
41. openCount++;
43. if (row == 1) {//第1行
44. virtualQU.union(curIndex, virtualtop);
45. realQU.union(curIndex, virtualtop);
46. }
47. if (row == n) {//最后一行
48. virtualQU.union(curIndex, virtualbtm);
49. }
50. neighborConnect(row, col, row - 1, col);  // 上
51. neighborConnect(row, col, row + 1, col);  // 下
52. neighborConnect(row, col, row, col - 1);  // 左
53. neighborConnect(row, col, row, col + 1);  // 右
54. }
56. //将a与open的邻居相连接
57. private void neighborConnect(int rowA, int colA, int rowB, int colB) {
58. if (0 < rowB && rowB <= n && 0 < colB && colB <= n
59. && isOpen(rowB, colB)) {
60. virtualQU.union(linearIndex(rowA, colA), linearIndex(rowB, colB));
61. realQU.union(linearIndex(rowA, colA), linearIndex(rowB, colB));
62. }
63. }
64. //计数
65. public int numberOfOpenSites() {
66. return openCount;
67. }

70. public boolean isOpen(int row, int col) {
71. return isOpen[linearIndex(row, col)];
72. }
74. public boolean isFull(int row, int col) {
75. return realQU.connected(virtualtop, linearIndex(row, col));
76. //必须用realQU判断而不能用virtualQU，否则出现backwash现象
77. }
79. public boolean percolates() {
80. return virtualQU.connected(virtualtop, virtualbtm);//判断渗透用virtualQU判断
81. }
83. public static void main(String[] args) {
84. StdOut.println("请运行PercolationStats程序！");
85. }
86. }

**PercolationStats.java：**

1. import edu.princeton.cs.algs4.StdOut;
2. import edu.princeton.cs.algs4.StdRandom;
3. import edu.princeton.cs.algs4.StdStats;

6. public class PercolationStats {
8. private final double[] fractions;
9. private final double CONFIDENCE\_95 = 1.96;
10. public PercolationStats(int n, int trials) {
11. if (n <= 0) {
12. throw new IllegalArgumentException("n必须大于0！");
13. }
14. if (trials <= 0) {
15. throw new IllegalArgumentException("trials必须大于0！");
16. }
17. fractions = new double[trials];
18. for (int i = 0; i < trials; i++) {//进行trials次随机实验
19. Percolation percolation = new Percolation(n);
20. int openedSites = 0;
21. while (!percolation.percolates()) {//每次开1个方块直到percolates
22. int row = StdRandom.uniform(n) + 1;
23. int col = StdRandom.uniform(n) + 1; //随机open方块
24. if (!percolation.isOpen(row, col)) {
25. percolation.open(row, col);
26. openedSites++;//每开1个格子opensites加1
27. }
28. }
29. fractions[i] = openedSites \* 1.0 / (n \* n);//恰好渗透open的格子比例
30. }
31. }
33. //\*p均值
34. public double mean() {
35. return StdStats.mean(fractions);
36. }
38. //标准偏差
39. public double stddev() {
40. return StdStats.stddev(fractions);
41. }
43. //置信区间
44. public double confidenceLo() {
45. return mean() - CONFIDENCE\_95 \* stddev() / Math.sqrt(fractions.length);
46. }

49. public double confidenceHi() {
50. return mean() + CONFIDENCE\_95 \* stddev() / Math.sqrt(fractions.length);
51. }

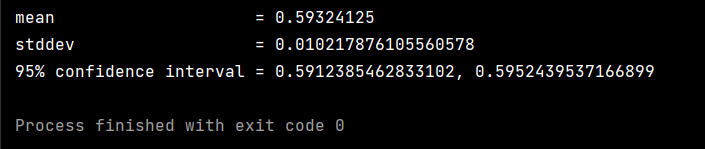
54. public static void main(String[] args) {
56. int n = Integer.parseInt(args[0]);
57. int trials = Integer.parseInt(args[1]);
58. PercolationStats stats = new PercolationStats(n, trials);
60. StdOut.println("mean                    = " + stats.mean());
61. StdOut.println("stddev                  = " + stats.stddev());
62. StdOut.println("95% confidence interval = "
63. + stats.confidenceLo() + ", "
64. + stats.confidenceHi());
65. }
66. }
67. **实验结果**

输入：200 100

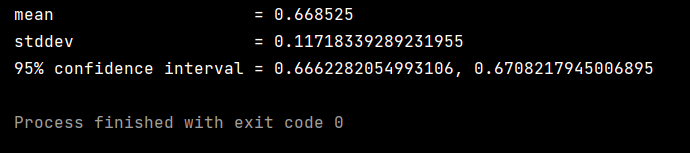
文本

描述已自动生成

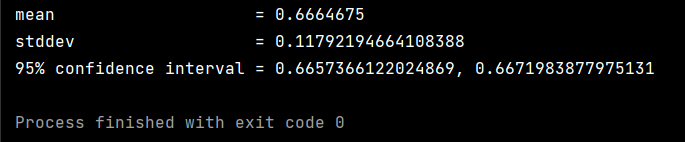
输入：200 100



输入：2 10000



输入：2 100000



1. **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。