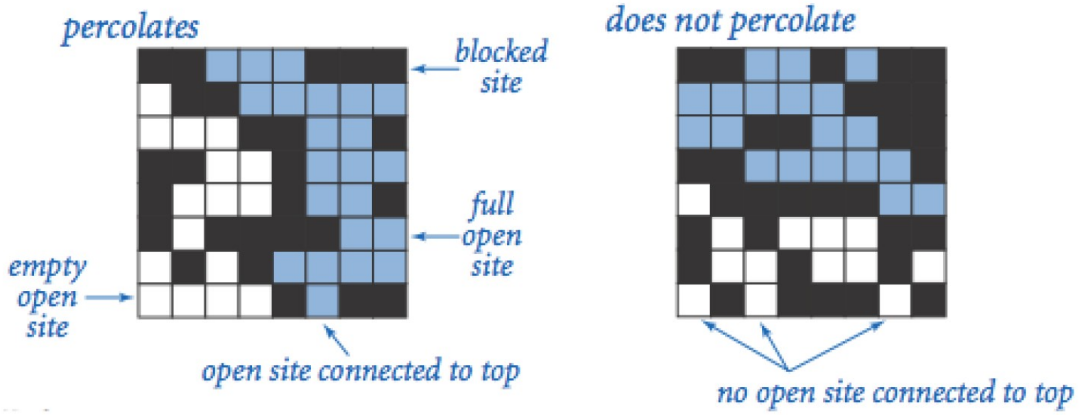
渗透问题（Percolation）

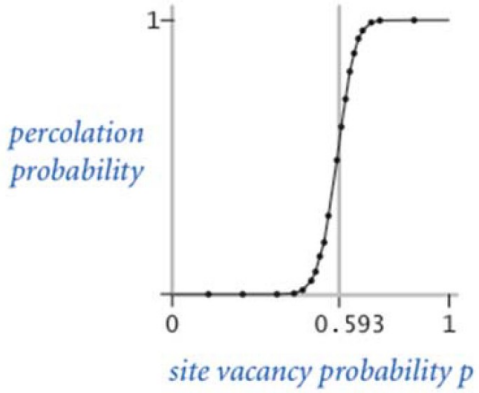
1 问题描述

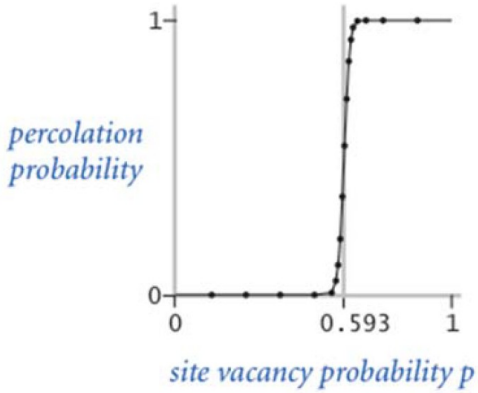
使用合并-查找（union-find）数据结构，编写程序通过蒙特卡罗模拟（Monte Carlo simulation）来估计渗透阈值。

模型： 我们使用 N×N 网格点来模型化一个渗透系统。每个格点或是 open 格点或是 blocked 格点。一个 full site 是一个 open 格点，它可以通过一系列的邻近（左、右、上、下）open 格点连通到顶行的一个 open 格点。如果在底行中存在一个 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。）



科学问题： 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 与渗滤概率。





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。你的任务是编写一个计算机程序来估计 p*。

2 实验内容

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 0，构造函数应该抛出 IllegalArgumentException 例外。构造函数应该与 N2 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 测量阈值的灵敏性。

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

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:

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

通过创建数据类型 PercolationStats 来执行一系列实验操作，包含一下 API。

```
public class PercolationStats {
    public PercolationStats(int N, int T)
    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 异常。

3 解决思路

初始思路是用一个复杂度 N*N 的两层 for 循环，检查每一个第一行节点和每一个第 N 行的节点是否相连，但是如果 N 非常大，该思路的效率就很低，于是思考，在第一行之上加一个虚拟节点 p，和第一行的每个 open 节点联通，在第 N 行之下加一个虚拟节点 q，和第 N 行的每个 open 节点联通，这样只用判断 p 和 q 是否连通即可。

4 程序实现

4.1 Percolation

```
public class Percolation {
    private boolean[][] grid;   // 网格
    private WeightedQuickUnionUF uf;
    private int siteSize; //规模

    public Percolation(int N) {
        grid = new boolean[N + 1][N];
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                grid[i][j] = false;
        uf = new WeightedQuickUnionUF(N * N + 2);
        siteSize = N;
        // 虚拟节点的实现
        for (int j = 1; j <= N; j++) {
            uf.union(getPos(N + 1, 1), getPos(1, j));
            uf.union(getPos(N + 1, 2), getPos(N, j));
        }
    }
    // 打开节点
    public void open(int i, int j) {
        if (grid[i - 1][j - 1]) return;
        grid[i - 1][j - 1] = true;
        final int[] dx = {-1, 1, 0, 0};
        final int[] dy = {0, 0, -1, 1};
        for (int k = 0; k < 4; k++) {
            int x = i + dx[k], y = j + dy[k];
            if (x < 1 || x > siteSize || y < 1 || y > siteSize) continue;
            // 若相邻节点也打开了则连接它们
            if (isOpen(x, y)) {
                uf.union(getPos(i, j), getPos(x, y));
            }
        }
    }

    // 二维转一维
    public int getPos(int i, int j) {
        return (i - 1) * (siteSize) + j - 1;
    }
```

```
    public boolean isOpen(int i, int j) {
        return grid[i - 1][j - 1];
    }

    public boolean isFull(int i, int j) {
        if (isOpen(i, j)) {
            int p = getPos(i, j);
            for (int k = 0; k < siteSize; k++)
                if (uf.connected(p, k))
                    return true;
        }
        return false;
    }
    // 判断是否渗漏
    public boolean percolate() {
        return uf.connected(getPos(siteSize + 1, 1), getPos(siteSize + 1, 2));
    }
}
```

4.2 PercolationStats

```
public class PercolationStats {
    private int expCnt;
    private Percolation pcl;
    private double[] fracs;

    public PercolationStats(int N, int T) {
        // 参数判断
        if (N < 0 || T < 0) throw new IllegalArgumentException("IllegalArgument");
        expCnt = T;
        fracs = new double[T];
        double time = 0.0;
        for (int k = 0; k < T; k++) {
            pcl = new Percolation(N);
            int openedSites = 0;
            // 记录运行时间
            Stopwatch stopwatch = new Stopwatch();
            while (!pcl.percolate()) {
                int i = StdRandom.uniform(1, N + 1);
                int j = StdRandom.uniform(1, N + 1);
```

```
                if (!pcl.isOpen(i, j)) {
                    pcl.open(i, j);
                    openedSites++;
                }
            }
            time += stopwatch.elapsedTime();
            var frac = (double) openedSites / (N * N);
            fracs[k] = frac;
        }
        System.out.printf("Average run time = %.4f\n" ,time / T * 1000.0);
    }
    // 获取均值
    public double mean() {
        return StdStats.mean(fracs);
    }
    // 获取标准差
    public double stddev() {
        return StdStats.stddev(fracs);
    }
    // 获取置信区间
    public double confidenceLo() {
        return mean() - 1.96 * stddev() / Math.sqrt(expCnt);
    }

    public double confidenceHi() {
        return mean() + 1.96 * stddev() / Math.sqrt(expCnt);
    }

    public static void main(String[] args) {
        int T = StdIn.readInt(), N = StdIn.readInt();
        PercolationStats ps = new PercolationStats(N, T);
        var confidence = ps.confidenceLo() + " " + ps.confidenceHi();
        StdOut.println("mean                    = " + ps.mean());
        StdOut.println("dev                     = " + ps.stddev());
        StdOut.println("95% confidence interval = " + confidence);
    }
}
```

4.3 WeightedQuickUnionUF

```
public class WeightedQuickUnionUF {
```

```
    private int[] id;
    private int[] sz;
    private int count;

    public WeightedQuickUnionUF(int N) {
        count = N;
        id = new int[N];
        sz = new int[N];
        for (int i = 0; i < N; i++) {
            id[i] = i;
        }
        for (int i = 0; i < N; i++)
            sz[i] = 1;
    }

    public int find(int p) {
        while (p != id[p]) {
            id[p] = id[id[p]]; // path compression by halving
            p = id[p];
        }
        return p;
    }

    public boolean connected(int p, int q) {
        return find(p) == find(q);
    }

    public void union(int p, int q) {
        int i = find(p);
        int j = find(q);
        if (i == j) return;
        if (sz[i] < sz[j]) {
            id[i] = j;
            sz[j] += sz[i];
        } else {
            id[j] = i;
            sz[i] += sz[j];
        }
        count--;
    }
}
```

4.4 除此之外还实现了 QuickFindUF、QuickUnionUF。

5 实验结果

5.1 示例输出

```
10000 100
Average run time = 0.4361
mean                    = 0.5927524099999986
dev                     = 0.015960819307516697
95% confidence interval = 0.5924395779415712 0.5930652420584259

Process finished with exit code 0
```

5.2 不同的合并-查找算法时间比较

	算法	运行时间 (ms)
N = 1000, T = 10	QuickFind	10.9670
	QuickUnion	3.1250
	WeightedQuickUnion	0.4670

6 实验心得

通过本次实验我认识到 Union-Find 可以被应用在很多的方面，并且认识到对于规模较大的数据，算法的改进优化在实际中的表现是非常有效的。