2.4 A Case Study: Percolation

Percolation. Pour liquid on top of some porous material. Will liquid reach the bottom?

Applications. [chemistry, materials science, ...]
- Chromatography.
- Spread of forest fires.
- Natural gas through semi-porous rock.
- Flow of electricity through network of resistors.
- Permeation of gas in coal mine through a gas mask filter.
- ...

Abstract model.
- N-by-N grid of sites.
- Each site is either blocked or open.
A Case Study: Percolation

**Percolation.** Pour liquid on top of some porous material. Will liquid reach the bottom?

**Abstract model.**
- $N$-by-$N$ grid of sites.
- Each site is either blocked or open.
- An open site is full if it is connected to the top via open sites.

**Random percolation.** Given an $N$-$N$ system where each site is vacant with probability $p$, what is the probability that system percolates?

**Remark.** Famous open question in statistical physics.

**Recourse.** Take a computational approach: Monte Carlo simulation.

Data Representation

**Data representation.** Use one $N$-$N$ boolean matrix to store which sites are open; use another to compute which sites are full.

**Standard array I/O library.** Library to support reading and printing 1- and 2-dimensional arrays.

<table>
<thead>
<tr>
<th>0 0 1 1 1 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 0 0 1 1 1 0</td>
</tr>
<tr>
<td>0 0 1 1 0 1 1 1</td>
</tr>
<tr>
<td>0 1 1 0 1 1 0 1</td>
</tr>
<tr>
<td>0 1 0 0 0 1 1 1</td>
</tr>
<tr>
<td>1 0 1 0 1 1 1 1</td>
</tr>
<tr>
<td>1 1 1 0 1 0 0 1</td>
</tr>
</tbody>
</table>

**shorthand:** 0 for blocked, 1 for open

<table>
<thead>
<tr>
<th>0 0 0 1 1 1 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 1 1 1 0 1</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 1 1 1 0</td>
</tr>
<tr>
<td>0 0 0 0 0 1 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 1</td>
</tr>
</tbody>
</table>

**shorthand:** 0 for not full, 1 for full
Scaffolding

Approach. Write the easy code first. Fill in details later.

```java
public class Percolation {
    // return boolean matrix representing full sites
    public static boolean[][] flow(boolean[][] open) {
        // does the system percolate?
        public static boolean percolates(boolean[][] open) {
            int N = open.length;
            boolean[][] full = flow(open);
            for (int j = 0; j < N; j++)
                if (!full[N-1][j]) return false; // system percolates if any full site in bottom row
            return false;
        }
        // test client
        public static void main(String[] args) {
            boolean[][] open = StdArrayIO.readBoolean2D();
            StdArrayIO.print(flow(open));
            StdOut.println(percolates(open));
        }
    }
}
```

General Percolation: Recursive Solution

Percolation. Given an N-by-N system, is there any path of open sites from the top to the bottom.

Depth first search. To visit all sites reachable from i-j:
- If i-j already marked as reachable, return.
- If i-j not open, return.
- Mark i-j as reachable.
- Visit the 4 neighbors of i-j recursively.

Percolation solution.
- Run DFS from each site on top row.
- Check if any site in bottom row is marked as reachable.

```java
// return boolean matrix representing full sites
public static boolean[][] flow(boolean[][] open) {
    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
        if (open[0][j]) flow(open, full, 0, j);
    return full;
}
public static void flow(boolean[][] open, boolean[][] full, int i, int j) {
    int N = full.length;
    if (i < 0 || i >= N || j < 0 || j >= N) return;
    if (!open[i][j]) return;
    if (!full[i][j]) return;
    full[i][j] = true; // mark
    flow(open, full, i+1, j); // down
    flow(open, full, i, j+1); // right
    flow(open, full, i, j-1); // left
    flow(open, full, i-1, j); // up
}
```

General Percolation: Probability Estimate

Analysis. Given N and p, run simulation T times and report average.

Running time. Still proportional to \(T N^2\).
Memory consumption. Still proportional to \(N^2\).
In Silico Experiment

Plot results. Plot the probability that an $N$-by-$N$ system percolates as a function of the site vacancy probability $p$.

Design decisions.
- How many values of $p$?
- For which values of $p$?
- How many experiments for each value of $p$?

Adaptive Plot

Adaptive plot. To plot $f(x)$ in the interval $[x_0, x_1]$:
- Stop if interval is sufficiently small.
- Divide interval in half and compute $f(x_m)$.
- Stop if $f(x_m)$ is close to $\frac{1}{2} (f(x_0) + f(x_1))$.
- Recursively plot $f(x)$ in the interval $[x_0, x_m]$.
- Plot the point $(x_m, f(x_m))$.
- Recursively plot $f(x)$ in the interval $[x_m, x_1]$.

Net effect. Short program that judiciously chooses values of $p$ to produce a "good" looking curve without excessive computation.

Percolation Plot: Java Implementation

```java
public class PercolationPlot {
    public static void curve(int N, double x0, double y0, double x1, double y1) {
        double gap = 0.05;
        double error = 0.005;
        int T = 10000;
        double xm = (x0 + x1) / 2;
        double ym = (y0 + y1) / 2;
        double fxm = Estimate.eval(N, xm, T);
        if (x1 - x0 < gap && Math.abs(ym - fxm) < error) {
            return;
        }
        curve(N, x0, y0, xm, fxm);
        StdDraw.filledCircle(xm, fxm, .005);
        curve(N, xm, fxm, x1, y1);
    }

    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        curve(N, 0.0, 0.0, 1.0, 1.0);
    }
}
```

Phase transition. If $p < 0.593$, system almost never percolates; if $p > 0.593$, system almost always percolates.