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.
- ...
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.

Data Representation

Data representation. Use one \(N\)-by-\(N\) 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.

Random percolation. Given an \(N\)-by-\(N\) 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 \(N\)-by-\(N\) 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.
Standard Array IO Library (Program 2.2.2)

public class StdArrayIO {
...
   // read M-by-N boolean matrix from standard input
   public static boolean[][] readBoolean2D() {
      int M = StdIn.readInt();
      int N = StdIn.readInt();
      boolean[][] a = new boolean[M][N];
      for (int i = 0; i < M; i++)
         for (int j = 0; j < N; j++)
            if (StdIn.readInt() != 0) a[i][j] = true;
      return a;
   }

   // print boolean matrix to standard output
   public static void print(boolean[][] a) {
      for (int i = 0; i < a.length; i++)
         for (int j = 0; j < a[i].length; j++)
            if (a[i][j]) StdOut.print("1 ");
            else StdOut.print("0 ");
      StdOut.println();
   }
}

Scaffolding

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

public class Percolation {
   // return boolean matrix representing full sites
   public static boolean[][] flow(boolean[][] open) {
      int N = open.length;
      boolean[][] full = flow(open);
      for (int j = 0; j < N; j++)
         if (full[N-1][j]) return true;
      return false;
   }

   // does the system percolate?
   public static boolean percolates(boolean[][] open) {
      boolean[][] full = flow(open);
      for (int j = 0; j < N; j++)
         if (full[0][j]) return true;
      return false;
   }

   public static void main(String[] args) {
      boolean[][] open = StdArrayIO.readBoolean2D();
      StdArrayIO.print(flow(open));
      StdOut.println(percolates(open));
   }
}

system percolates if any full site in bottom row

Vertical Percolation

Next step. Start by solving an easier version of the problem.

Vertical percolation. Is there a path of open sites from the top to the bottom that goes straight down?
Vertical Percolation

Q. How to determine if site \((i, j)\) is full?
A. It’s full if \((i, j)\) is open and \((i-1, j)\) is full.

Algorithm. Scan rows from top to bottom.

```java
public static boolean[][] flow(boolean[][] open) {
    int N = open.length;
    boolean[][] full = new boolean[N][N];
    full[0][0] = open[0][0];

    for (int i = 1; i < N; i++)
        for (int j = 0; j < N; j++)
            full[i][j] = open[i][j] && full[i-1][j];

    return full;
}
```

Vertical Percolation: Testing

Testing. Use standard input and output to test small inputs.

```shell
% more testT.txt
5
0 1 1 0 1
0 0 1 1 1
1 1 0 1 1
1 0 0 0 1
0 1 1 1 1
% java VerticalPercolation < testT.txt
true
% more testF.txt
5
1 0 1 0 0
1 0 1 1 1
1 1 0 1 1
1 0 0 0 1
0 0 0 1 1
% java VerticalPercolation < testF.txt
false
```

Vertical Percolation: Testing

Testing. Add helper methods to generate random inputs and visualize using standard draw.

```java
public class Percolation {
    ...
    // plot matrix to standard drawing
    public static void show(boolean[][] a, boolean foreground)
    }
```
Data Visualization

Visualization. Use standard drawing to visualize larger inputs.

```
public class Visualize {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double p = Double.parseDouble(args[1]);
        boolean[][] open = Percolation.random(N, p);
        boolean[][] full = Percolation.flow(open);
        StdDraw.setPenColor(StdDraw.BLACK);
        Percolation.show(open, false);
        StdDraw.setPenColor(StdDraw.CYAN);
        Percolation.show(full, true);
    }
}
```

Vertical Percolation: Probability Estimate

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

```
public class Estimate {
    public static double eval(int N, double p, int T) {
        int cnt = 0;
        for (int t = 0; t < T; t++) {
            boolean[][] open = Percolation.random(N, p);
            if (VerticalPercolation.percolates(open)) cnt++;
        }
        return (double) cnt / M;
    }
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double p = Double.parseDouble(args[1]);
        int T = Integer.parseInt(args[2]);
        StdOut.println(eval(N, p, T));
    }
}
```

General Percolation

Running time. Proportional to $T N^2$.

Memory consumption. Proportional to $N^2$.
Percolation. Given an \( N \times N \) 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.

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

| \% java Estimate 20 .5 100000 | 0.050953 |
| \% java Estimate 20 .6 100000 | 0.568869 |
| \% java Estimate 20 .7 100000 | 0.980804 |
| \% java Estimate 40 .6 100000 | 0.595995 |

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

**Plot results.** Plot the probability that an \( N \times N \) 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 \( p \)?

![too few points](image1)
![too many points](image2)
![judicious choice of points](image3)

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) {
            StdDraw.line(x0, y0, x1, y1);
            return;
        }
        curve(N, x0, y0, xm, fxm);
        StdDraw.filledCircle(xm, fxm, 0.005);
        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);
    }
}
```

**Plot results.** Plot the probability that an \( N \times N \) system percolates as a function of the site vacancy probability \( p \).

**Phase transition.** If \( p < 0.593 \), system almost never percolates; if \( p > 0.593 \), system almost always percolates.
Lessons

Expect bugs. Run code on small test cases.

Keep modules small. Enables testing and debugging.

Incremental development. Run and debug each module as you write it.

Solve an easier problem. Provides a first step.

Consider a recursive solution. An indispensable tool.

Build reusable libraries. StdArrayIO, StdRandom, StdIn, StdDraw, …