Case Study: Percolation Problem
Outline

1 Percolation

2 Vertical Percolation

3 General Percolation
If we pour liquid on top of some porous material, will the liquid reach the bottom?

Percolation refers to the abstract process that models such situations

Some Applications
- Spread of forest fires
- Flow of electricity through a network of resistors
- Permeation of gas in a coal mine through a gas mask filter
- Studying Fermi’s paradox, the apparent contradiction between the high estimates of the probability of the existence of extraterrestrial civilizations and the lack of evidence for them
Percolation

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

If sites are independently set to be open with vacancy probability $p$, what is the probability that the system percolates?

There is no known mathematical solution, so we take a computational (Monte Carlo simulation) approach

We use one $n$-by-$n$ boolean matrix to store which sites are open, and another to compute which sites are full
import stdarray
import stddraw
import stdrandom
import sys

def random(n, p):
a = stdarray.create2D(n, n, False)
    for i in range(n):
        for j in range(n):
            a[i][j] = stdrandom.bernoulli(p)
    return a

def draw(a, which):
n = len(a)
stddraw.setXscale(-.5, n)
stddraw.setYscale(-.5, n)
    for i in range(n):
        for j in range(n):
            if a[i][j] == which:
                stddraw.filledSquare(j, n - i - 1, .5)

def main():
n = int(sys.argv[1])
p = float(sys.argv[2])
test = random(n, p)
draw(test, False)
stddraw.show()

if __name__ == '__main__':
    main()
Percolation

$ python3 percolationio.py 10 .8

$ python3 percolationio.py 10 .2

$ python3 percolationio.py 100 .6
**Vertical Percolation**

We start by solving an easier version of the problem, namely vertical percolation: Is there a path of open sites from the top to the bottom that goes straight down?

A site \((i, j)\) is full if it is open and site \((i - 1, j)\) is full.

To determine if a system vertically percolates, scan rows from top to bottom.
import stdarray
import stdio

def flow(isOpen):
    n = len(isOpen)
    isFull = stdarray.create2D(n, n, False)
    for j in range(n):
        isFull[0][j] = isOpen[0][j]
    for i in range(1, n):
        for j in range(n):
            if isOpen[i][j] and isFull[i - 1][j]:
                isFull[i][j] = True
    return isFull

def percolates(isFull):
    n = len(isFull)
    for j in range(n):
        if isFull[n - 1][j]:
            return True
    return False

def main():
    isOpen = stdarray.readBool2D()
    isFull = flow(isOpen)
    stdarray.write2D(isFull)
    stdio.writeln(percolates(isFull))

if __name__ == '__main__':
    main()
Vertical Percolation

```
$ more test8.txt
8 8
0 0 1 1 1 0 0 0
1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 0
0 0 1 1 0 1 1 1
0 1 1 1 0 1 1 0
0 1 0 0 0 0 1 1
1 0 1 0 1 1 1 1
1 1 1 1 0 1 0 0

$ python3 percolationv.py < test8.txt
8 8
0 0 1 1 1 0 0 0
0 0 0 1 1 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
False
```
Vertical Percolation

visualizev.py: Accept integer \( n \), float \( p \), and integer \( \text{trials} \) as command-line arguments. Generate an \( n \times n \) random system with site vacancy probability \( p \). Compute the directed percolation flow, and draw result to standard draw. Repeat \( \text{trials} \) times.

```python
import percolationio
import percolationv
import stddraw
import sys

def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        stddraw.clear()
        stddraw.setPenColor(stddraw.BLACK)
        percolationio.draw(isOpen, False)
        stddraw.setPenColor(stddraw.BLUE)
        isFull = percolationv.flow(isOpen)
        percolationio.draw(isFull, True)
        stddraw.show(1000.0)
    stddraw.show()

if __name__ == '__main__':
    main()
```

10 / 21
Vertical Percolation

$ python3 visualizev.py 20 .65 1

$ python3 visualizev.py 20 .60 1

$ python3 visualizev.py 20 .55 1
Vertical Percolation

estimatev.py: Accept integer $n$, float $p$, and integer $trials$ as command-line arguments. Create $trials$ random $n$-by-$n$ systems with site vacancy probability $p$. Determine the fraction of them that percolate, and write that fraction to standard output.

```python
import percolationio
import percolationv
import stdio
import sys

def evaluate(n, p, trials):
    count = 0
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        isFull = percolationv.flow(isOpen)
        if (percolationv.percolates(isFull)):
            count += 1
    return 1.0 * count / trials

def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    q = evaluate(n, p, trials)
    stdio.writeln(q)

if __name__ == '__main__':
    main()
```

$ python3 estimatev.py 20 .65 1000
0.004
$ python3 estimatev.py 20 .60 1000
0.001
$ python3 estimatev.py 20 .55 1000
0.0
General Percolation

Given an $n$-by-$n$ system, is there any path of open sites from the top to the bottom?

To visit all sites reachable from site $(i, j)$, do depth first search (DFS)
- If $(i, j)$ already marked as reachable, return
- If $(i, j)$ not open, return
- Mark $(i, j)$ as reachable
- Visit the four neighbors of $(i, j)$ recursively

Percolation solution
- Run DFS from each site of top row
- Check if any site in bottom row is marked as reachable
import stdarray
import stdio

def _flow(isOpen, isFull, i, j):
    n = len(isFull)
    if (i < 0) or (i >= n):
        return
    if (j < 0) or (j >= n):
        return
    if not isOpen[i][j]:
        return
    if isFull[i][j]:
        return
    isFull[i][j] = True
    _flow(isOpen, isFull, i + 1, j)
    _flow(isOpen, isFull, i, j + 1)
    _flow(isOpen, isFull, i, j - 1)
    _flow(isOpen, isFull, i - 1, j)

def flow(isOpen):
    n = len(isOpen)
    isFull = stdarray.create2D(n, n, False)
    for j in range(n):
        _flow(isOpen, isFull, 0, j)
    return isFull
```python
def percolates(isFull):
    n = len(isFull)
    for j in range(n):
        if isFull[n - 1][j]:
            return True
    return False

def main():
    isOpen = stdarray.readBool2D()
    isFull = flow(isOpen)
    stdarray.write2D(isFull)
    stdio.writeln(percolates(isFull))

if __name__ == '__main__':
    main()
```
General Percolation

```bash
$ more test8.txt
8 8
0 0 1 1 1 0 0 0
1 0 0 1 1 1 1 1
1 1 1 0 0 1 1 0
0 0 1 1 0 1 1 1
0 1 1 1 0 1 1 0
0 1 0 0 0 0 1 1
1 0 1 0 1 1 1 1
1 1 1 1 0 1 0 0

$ python3 percolation.py < test8.txt
8 8
0 0 1 1 1 0 0 0
0 0 0 1 1 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 1 1 1
0 0 0 0 0 1 1 0
0 0 0 0 0 0 1 1
0 0 0 0 1 1 1 1
0 0 0 0 0 1 0 0
True
```
import percolation
import percolationio
import stddraw
import sys

def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        stddraw.clear()
        stddraw.setPenColor(stddraw.BLACK)
        percolationio.draw(isOpen, False)
        stddraw.setPenColor(stddraw.BLUE)
        isFull = percolation.flow(isOpen)
        percolationio.draw(isFull, True)
        stddraw.show(1000.0)
    stddraw.show()

if __name__ == '__main__':
    main()
General Percolation

$ python3 visualize.py 20 .65 1

$ python3 visualize.py 20 .60 1

$ python3 visualize.py 20 .55 1
General Percolation

estimate.py: Accept integer \( n \), float \( p \), and integer \( trials \) as command-line arguments. Create \( trials \) random \( n \)-by-\( n \) systems with site vacancy probability \( p \). Determine the fraction of them that percolate, and write that fraction to standard output.

```python
import percolation
import percolationio
import stdio
import sys

def evaluate(n, p, trials):
    count = 0
    for i in range(trials):
        isOpen = percolationio.random(n, p)
        isFull = percolation.flow(isOpen)
        if (percolation.percolates(isFull)):
            count += 1
    return 1.0 * count / trials

def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    q = evaluate(n, p, trials)
    stdio.writeln(q)

if __name__ == '__main__':
    main()
```

$ python3 estimate.py 20 .65 1000
0.868
$ python3 estimate.py 20 .60 1000
0.557
$ python3 estimate.py 20 .55 1000
0.222
percplot.py: Accept integer $n$ as a command-line argument. Plot to standard draw a graph that relates site vacancy probability (control variable) to percolation probability (experimental observations) for a $n$-by-$n$ system.

```python
import sys
import stddraw
import estimate

def curve(n, x0, y0, x1, y1, trials = 10000, gap = .01, err = .0025):
    xm = (x0 + x1) / 2.0
    ym = (y0 + y1) / 2.0
    fxm = estimate.evaluate(n, xm, trials)
    if (x1 - x0 < gap) or (abs(ym - fxm) < err):
        stddraw.line(x0, y0, x1, y1)
        stddraw.show(0.0)
        return
    curve(n, x0, y0, xm, fxm)
stddraw.filledCircle(xm, fxm, .005)
stddraw.show(0.0)
curve(n, xm, fxm, x1, y1)

def main():
    n = int(sys.argv[1])
    stddraw.setPenRadius(0.0)
    curve(n, 0.0, 0.0, 1.0, 1.0)
stddraw.show()

if __name__ == '__main__':
    main()
```
General Percolation

$ python3 percplot.py 10

$ python3 percplot.py 20

$ python3 percplot.py 30