

Percolation

Team members: Zarina Zhakupova, Amir Azhibayev, Zhansaya Sabirova



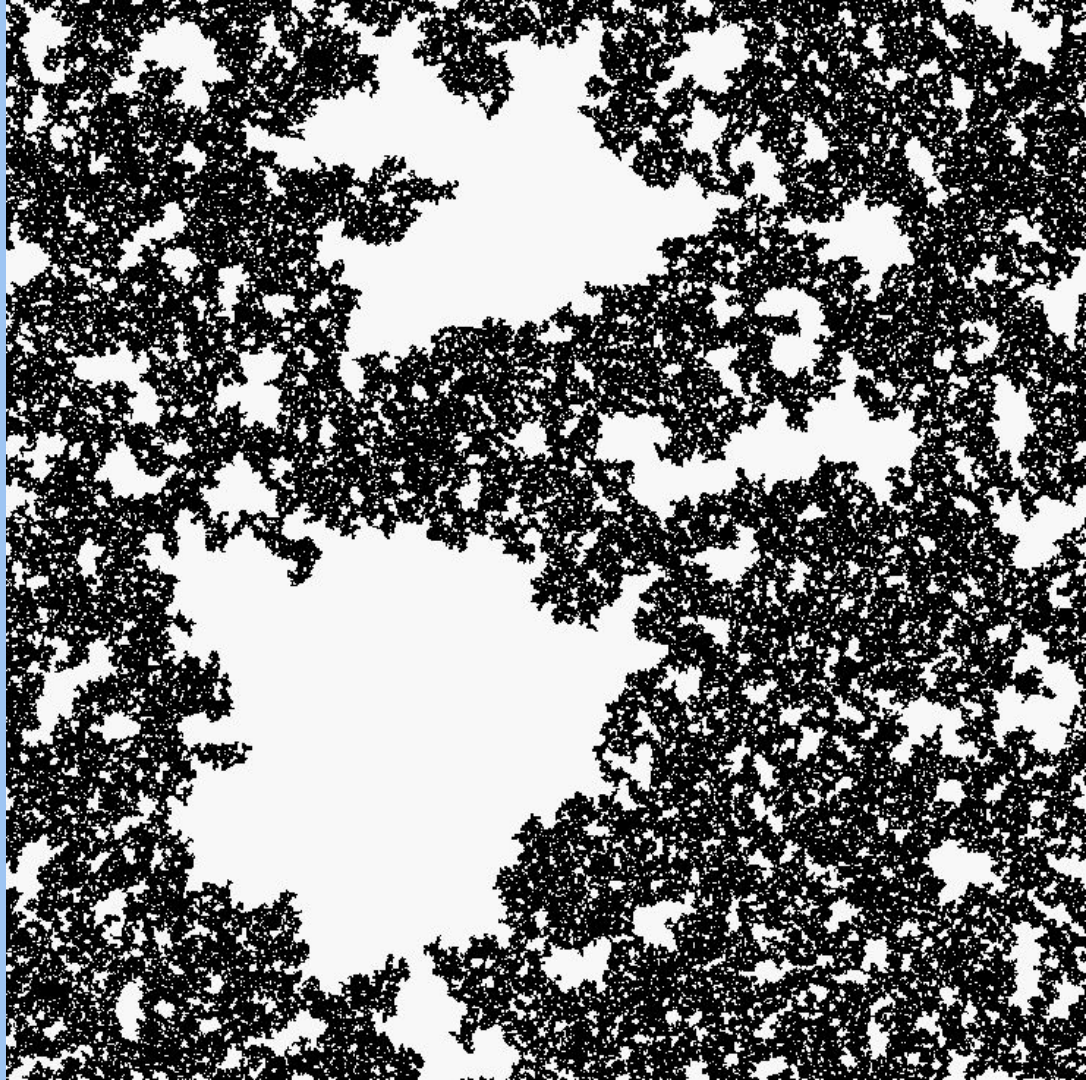
● Outline:

What is Percolation?;
Model;
Historical development;
Code and its explanation;
Conclusion.



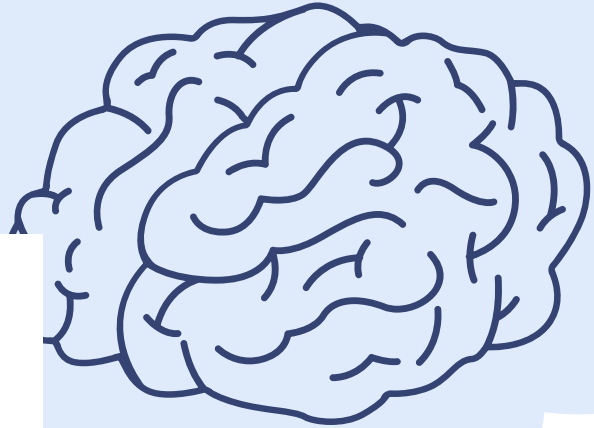
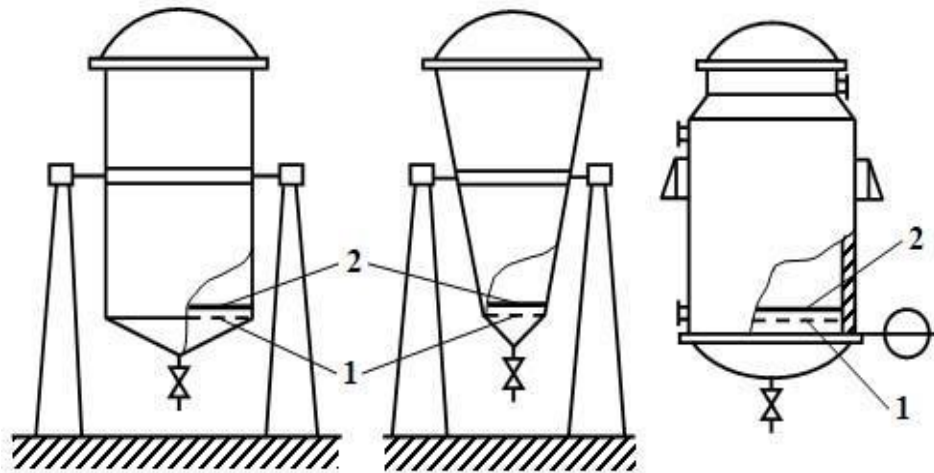


What is Percolation?



Model and FIRST MENTION

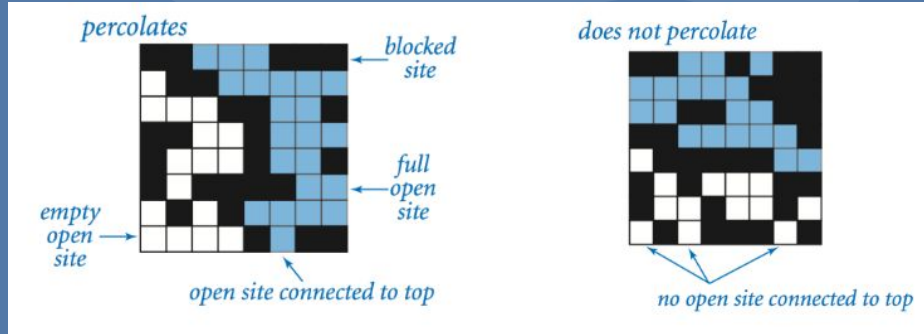
Percolator:



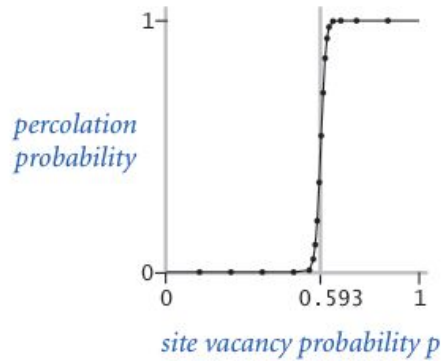
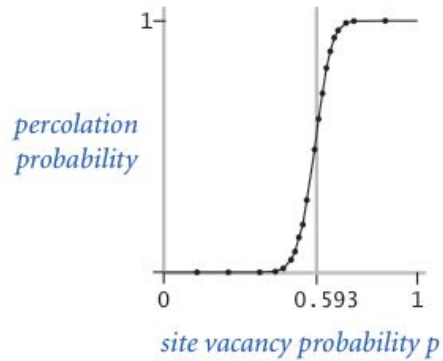


Task:

Given a composite systems comprised of randomly distributed insulating and metallic materials: what fraction of the materials need to be metallic so that the composite system is an electrical conductor?



The plots below show the site vacancy probability p versus the percolation probability for 20-by-20 random grid (left) and 100-by-100 random grid (right)



To model a percolation system, create a data type Percolation with the following API:

```
public class Percolation {  
    public Percolation(int N)  
    public void open(int i, int j)  
    public boolean isOpen(int i, int j)  
    public boolean isFull(int i, int j)  
    public boolean percolates()  
}
```





Code

```
public class Percolation {
    private static final int TOP = 0;
    private final boolean[][] opened;
    private final int size;
    private final int bottom;
    private int openSites;
    private final WeightedQuickUnionUF qf;
```

```
public Percolation(int n) {
    if (n <= 0) {
        throw new IllegalArgumentException();
    }
    size = n;
    bottom = size * size + 1;
    qf = new WeightedQuickUnionUF(size * size + 2);
    opened = new boolean[size][size];
    openSites = 0;
}
```



```
public void open(int row, int col) {  
    checkException(row, col);  
    opened[row - 1][col - 1] = true; // Open the box  
    ++openSites;|
```

```
if (row == 1) {  
    qf.union(getQuickFindIndex(row, col), TOP);  
if (row == size) {  
    qf.union(getQuickFindIndex(row, col), bottom);|
```

```
f (row > 1 && isOpen(row - 1, col)) {  
    qf.union(getQuickFindIndex(row, col), getQuickFindIndex(row - 1, col));  
}  
  
if (row < size && isOpen(row + 1, col)) {  
    qf.union(getQuickFindIndex(row, col), getQuickFindIndex(row + 1, col));  
}  
  
if (col > 1 && isOpen(row, col - 1)) {  
    qf.union(getQuickFindIndex(row, col), getQuickFindIndex(row, col - 1));  
}  
  
if (col < size && isOpen(row, col + 1)) {  
    qf.union(getQuickFindIndex(row, col), getQuickFindIndex(row, col + 1));  
}  
}|
```



```
private void checkException(int row, int col) {
    if (row <= 0 || row > size || col <= 0 || col > size) {
        throw new IllegalArgumentException();
    }
}
private void checkException(int row, int col) {
    if (row <= 0 || row > size || col <= 0 || col > size) {
        throw new IllegalArgumentException();
    }
}
```

```
public boolean isFull(int row, int col) {
    if ((row > 0 && row <= size) && (col > 0 && col <= size)) {
        return qf.find(TOP) == qf.find(getQuickFindIndex(row, col));
    } else throw new IllegalArgumentException();
}
```



```
1 import edu.princeton.cs.algs4.StdOut;
2 import edu.princeton.cs.algs4.StdRandom;
3 import edu.princeton.cs.algs4.StdStats;
4
5 public class PercolationStats {
6
7     private static final double CONFIDENCE_95 = 1.96;
8     private final int experimentsCount;
9     private final double[] fractions;
```

```
11     public PercolationStats(int n, int t) {
12         if (n <= 0 || t <= 0) {
13             throw new IllegalArgumentException("Given N <= 0 || T <= 0");
14         }
15         experimentsCount = t;
16         fractions = new double[experimentsCount];
17         for (int expNum = 0; expNum < experimentsCount; expNum++) {
18             Percolation pr = new Percolation(n);
19             int openedSites = 0;
20             while (!pr.percolates()) {
21                 int i = StdRandom.uniform(1, n + 1);
22                 int j = StdRandom.uniform(1, n + 1);
23                 if (!pr.isOpen(i, j)) {
24                     pr.open(i, j);
25                     openedSites++;
26                 }
27             }
28             double fraction = (double) openedSites / (n * n);
29             fractions[expNum] = fraction;
30         }
31     }
32 }
```



```
33
34     public double mean() {
35         |     return StdStats.mean(fractions);
36     }
37
38     public double stddev() {
39         |     return StdStats.stddev(fractions);
40     }
41
42
43     public double confidenceLo() {
44         |     return mean() - ((CONFIDENCE_95 * stddev()) / Math.sqrt(experimentsCount));
45     }
46
47
48     public double confidenceHi() {
49         |     return mean() + ((CONFIDENCE_95 * stddev()) / Math.sqrt(experimentsCount));
50     }
51
```



```
51
52     public static void main(String[] args) {
53         int n = Integer.parseInt(args[0]);
54         int t = Integer.parseInt(args[1]);
55         PercolationStats ps = new PercolationStats(n, t);
56
57         String confidence = ps.confidenceLo() + ", " + ps.confidenceHi();
58         StdOut.println("mean          = " + ps.mean());
59         StdOut.println("stddev       = " + ps.stddev());
60         StdOut.println("95% confidence interval = " + confidence);
61     }
62 }
63
```

OUTPUT:

```
mean          = 0.5924317499999998
stddev       = 0.010149171394127389
95% confidence interval = 0.5904425124867508, 0.5944289875932488
|
Process finished with exit code 0
```

Conclusion



The methods and tools that we used in the process of implementing the problem allow us to solve various other issues. In our example, a widely used computing technique known as Monte Carlo simulation to study a natural model known as percolation.