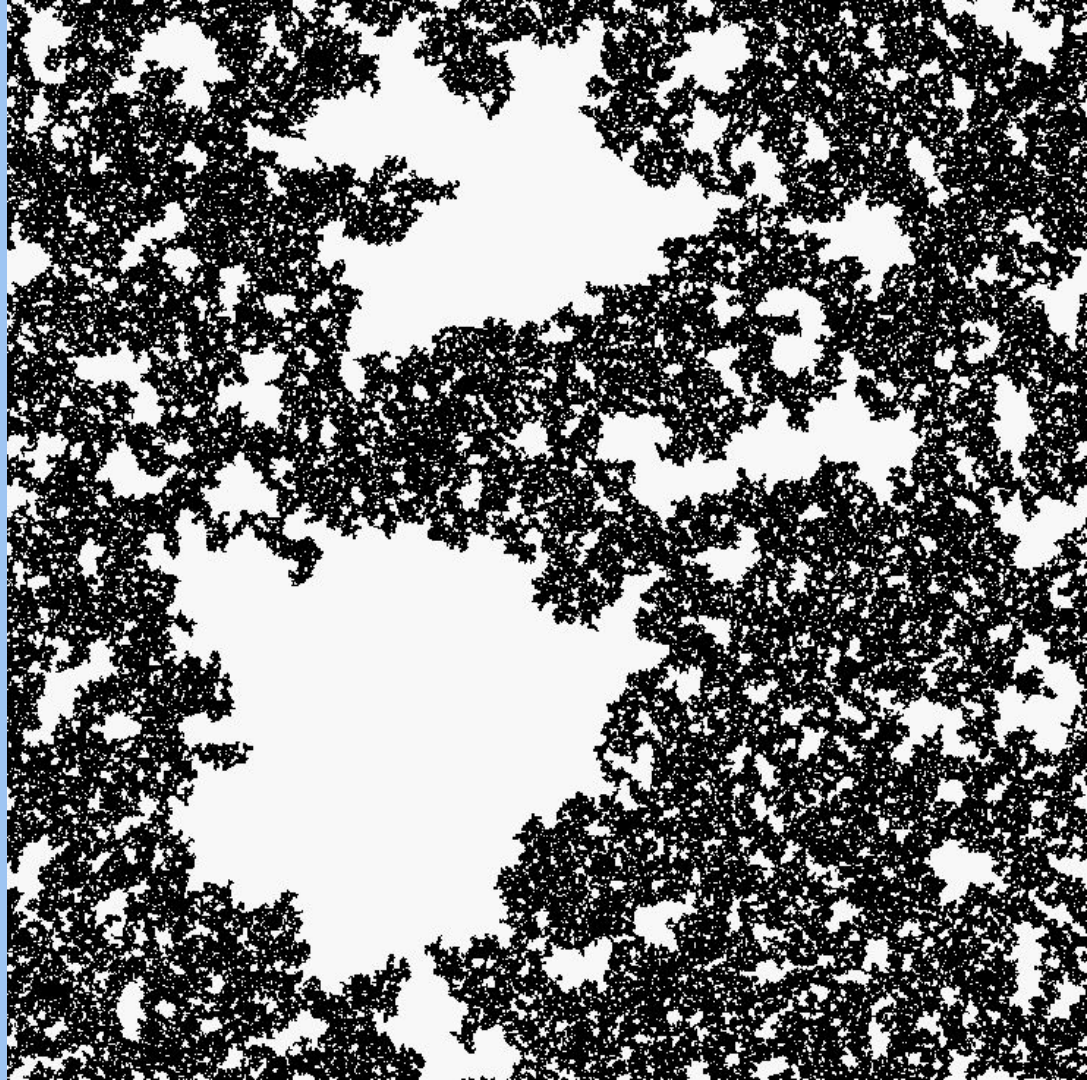
Percolation

Team members: Zarina Zhakupova, Amir Azhibayev, Zhansaya Sabirova

Outline:

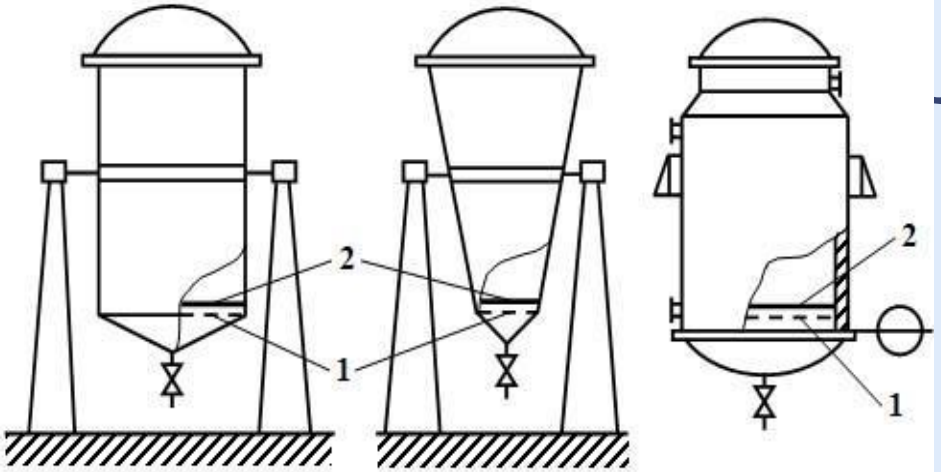
What is Percolation?;
Model;
Historical development;
Code and its explanation;
Conclusion.

What is Percolation?



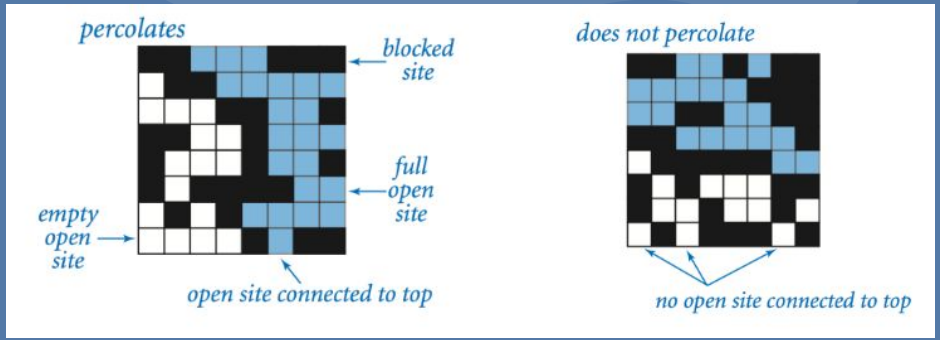
Model and FIRST MENTION

Percolator:

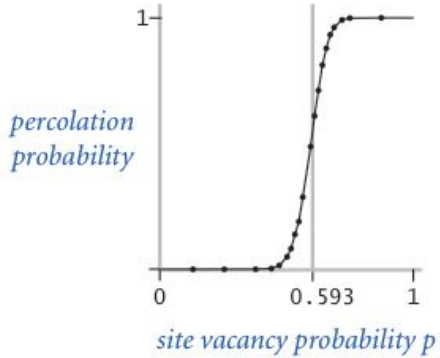


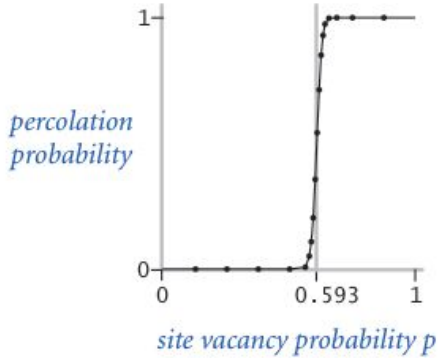
Task:

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

```java
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;
```

```java
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;
    }
```

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

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

```java
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));
        }
    }
```

```java
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();
        }
    }
```

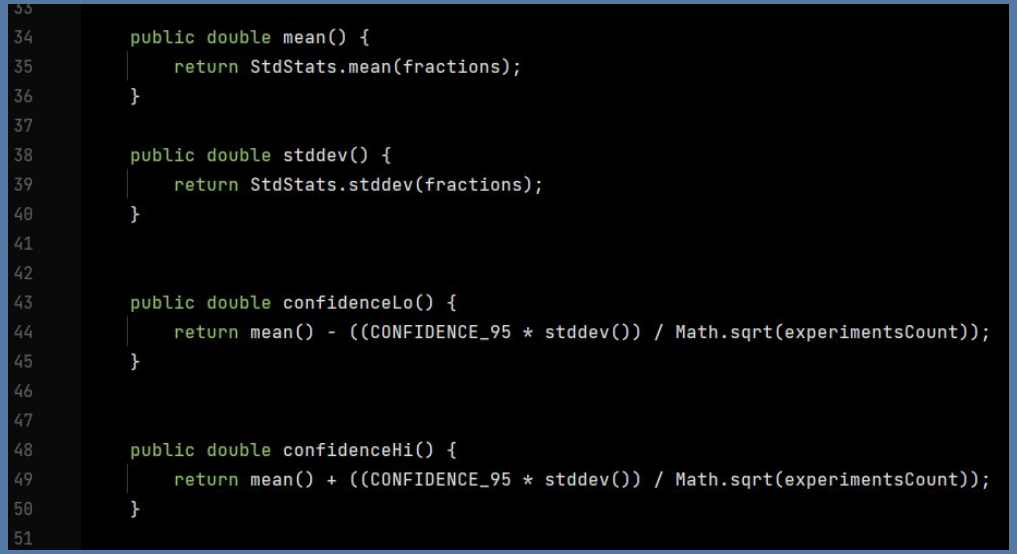
```java
public boolean isFull(int row, int col) {
        if ((row > 0 && row <= size) && (col > 0 && col <= size)) {
            return qf.find(TOP) == qf.find(getQuickFindIndex(row, col));
        } else throw new IllegalArgumentException();
    }
```

```java
import edu.princeton.cs.algs4.StdOut;
import edu.princeton.cs.algs4.StdRandom;
import edu.princeton.cs.algs4.StdStats;

public class PercolationStats {

    private static final double CONFIDENCE_95 = 1.96;
    private final int experimentsCount;
    private final double[] fractions;
```

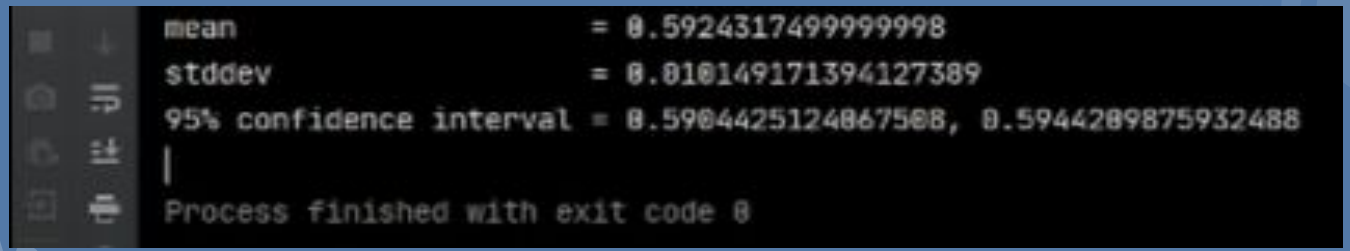
```java
    public PercolationStats(int n, int t) {
        if (n <= 0 || t <= 0) {
            throw new IllegalArgumentException("Given N <= 0 || T <= 0");
        }
        experimentsCount = t;
        fractions = new double[experimentsCount];
        for (int expNum = 0; expNum < experimentsCount; expNum++) {
            Percolation pr = new Percolation(n);
            int openedSites = 0;
            while (!pr.percolates()) {
                int i = StdRandom.uniform(1, n + 1);
                int j = StdRandom.uniform(1, n + 1);
                if (!pr.isOpen(i, j)) {
                    pr.open(i, j);
                    openedSites++;
                }
            }
            double fraction = (double) openedSites / (n * n);
            fractions[expNum] = fraction;
        }
    }
```



```
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:



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.