Coursera公开课–Algorithm–Percolation

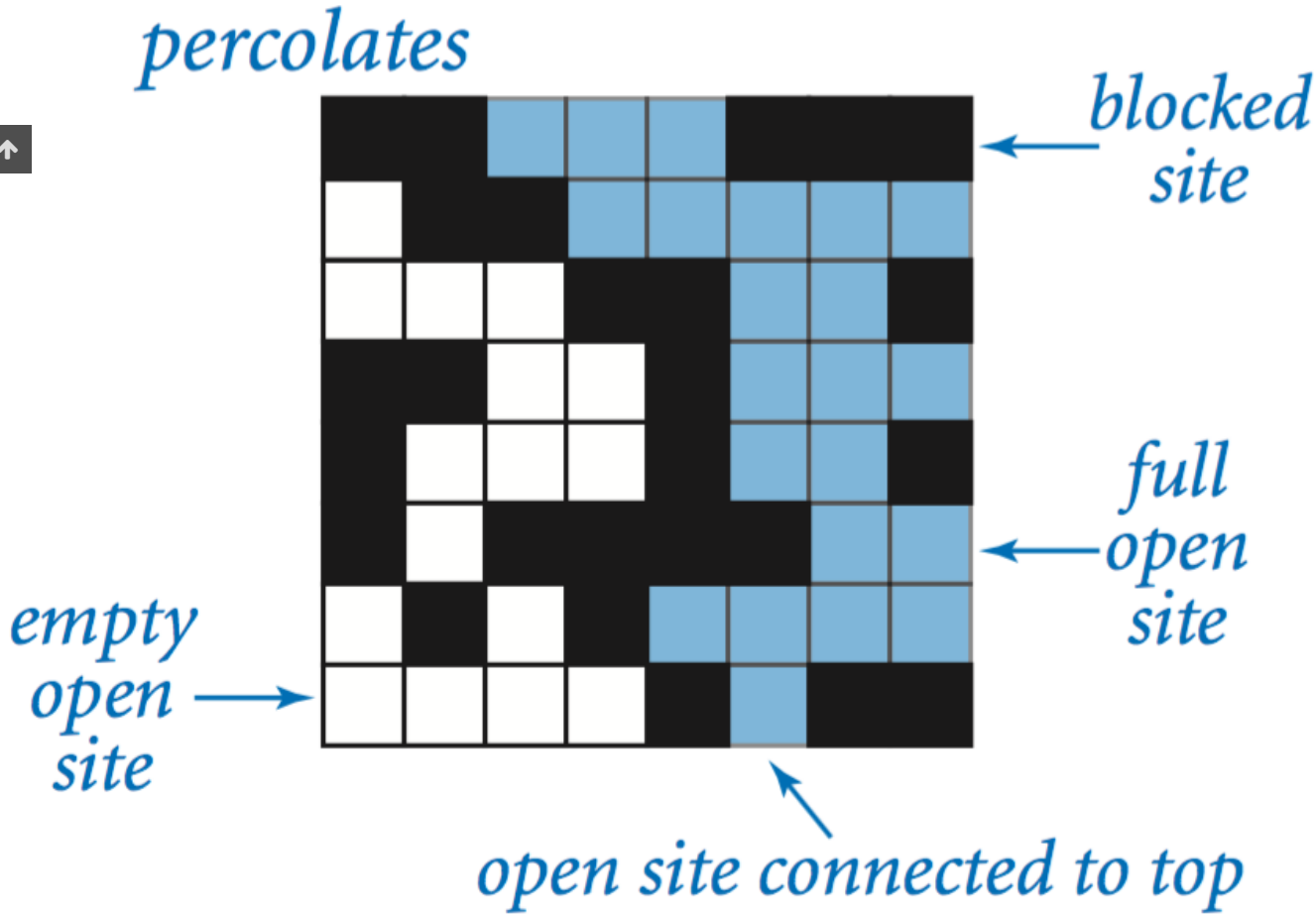
2017-12-24 | Algorithm | | 0

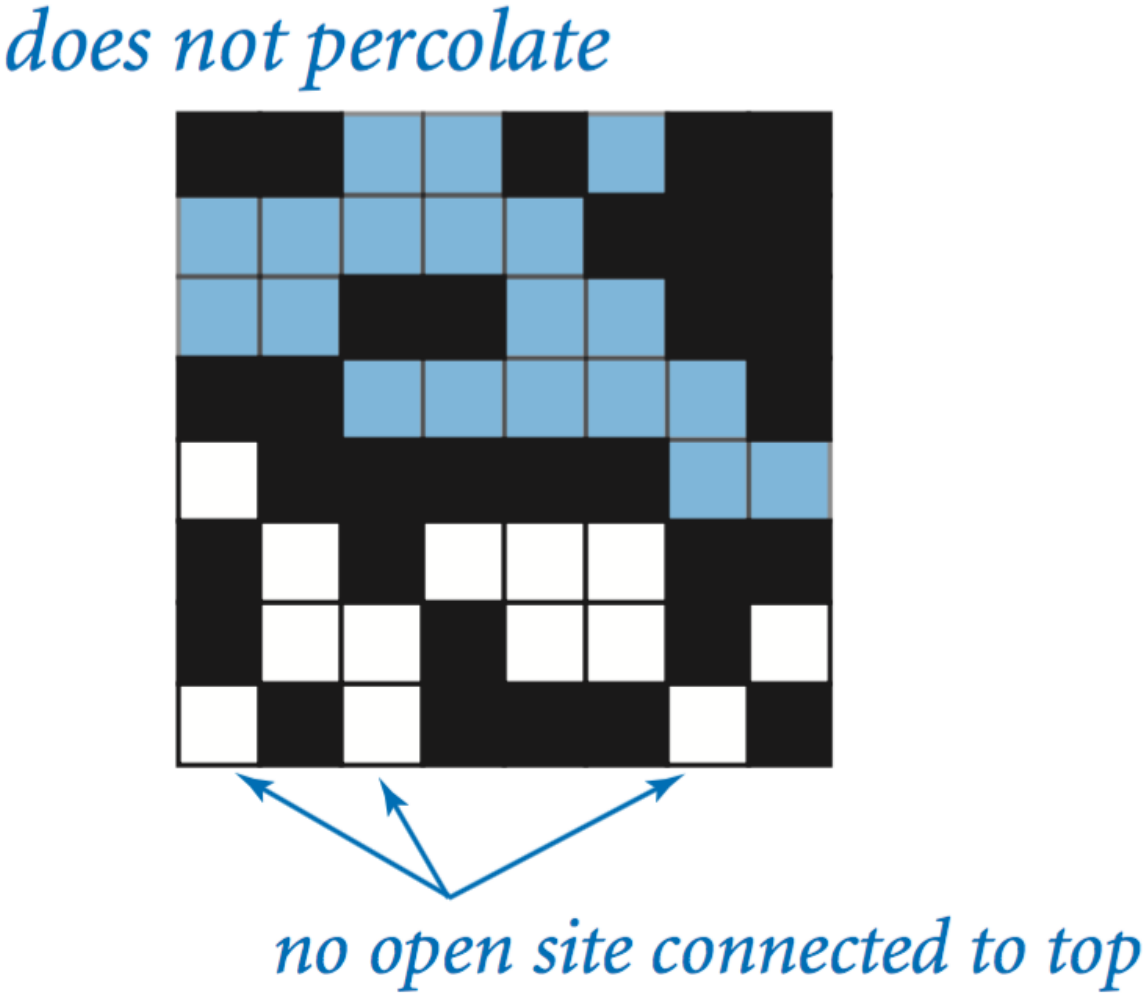
前言

前段时间在coursera上注册了两门课，Algorithms, Part I，Algorithms, Part II，这教授（Robert Sedgewick）讲得确实不错，不敢说甩国内老师一条街，起码比国内老师讲得要通俗易懂得多，配合ppt和作业，只要你认真做完，收获会非常大。对于学生，空闲时间比较多，同时上两门也OK；对于上班族，个人还是建议先上完Part I，再去上Part-II，一周同时上两门课，还要做两次作业，确实有点吃不消（本人作死，同时上了两门，下班回来做作业经常做到一两点，哭死）。

问题

发完牢骚，进入正题。第一周的作业是Percolation，在一个N*N的格子中，每个格子有三种状态，blocked, open, full。一开始所有的格子都是blocked状态，随机打开任意一个格子（格子状态变为open，如果格子能通过上下左右连接到顶部，则把格子状态改为full），如果格子中的上下两端能通过格子连接，那么我们就可以说整个系统处于渗透状态。问题更具体描述，可以查看：http://coursera.cs.princeton.edu/algs4/assignments/percolation.html


percolates
blocked site
full open site
empty open site
open site connected to top



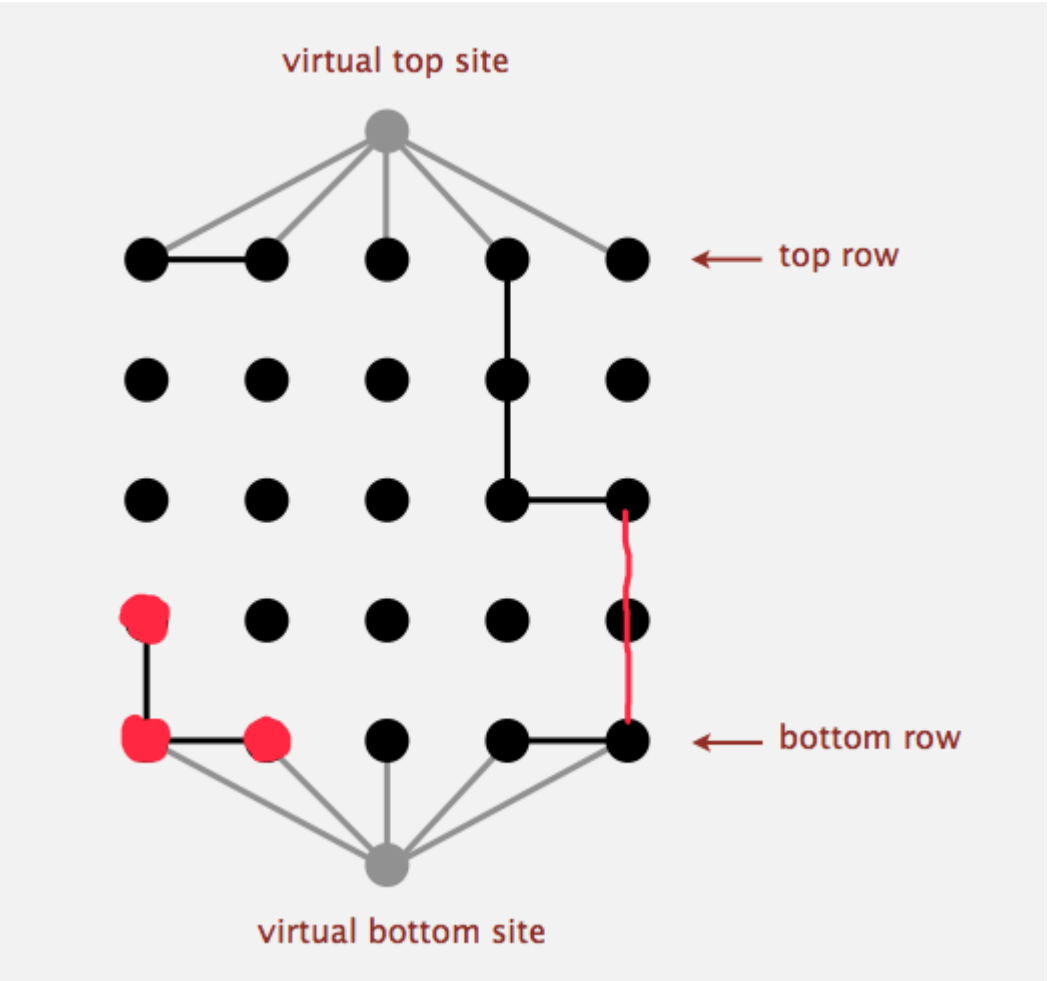
对于给定的N，实现下方API：

```
public class Percolation {
   public Percolation(int n)                // create n-by-n grid, with all sites bl
   public void open(int row, int col)    // open site (row, col) if it is not open a
   public boolean isOpen(int row, int col)  // is site (row, col) open?
   public boolean isFull(int row, int col)  // is site (row, col) full?
   public     int numberOfOpenSites()       // number of open sites
   public boolean percolates()              // does the system percolate?
   public static void main(String[] args)   // test client (optional)
}
```

实现思路

这道题考察的就是使用并查集解决实际问题的能力。每打开一个格子，我们直接把上下左右四个格子和当前格子合并即可。用一个二维数组记录当前格子(row, col)是否被打开，同时用openedCount来记录打开格子的个数。那么如何实现isFull()呢？如果暴力枚举的话，需要O(N)的

事件复杂度，对于percolates()更是需要O(N*N)的复杂度。有没有更好的方案呢？一个比较tricky的方法是在头部和尾部分别添加两个虚拟的节点。那么对于percolates()我们只需要用并查集判断isConnected(top, bottom)即可，时间复杂度降为O(logN)。对于isFull()判断isConnected(top, currentNodeIndex)，这样有没有问题？当然有，拿下图为例，左下角的三个红点应该是open状态，但是由于增加了虚拟的bottom节点，导致这三个节点变成了full状态。我能想到的解决方案是建立两个并查集，一个只添加top虚拟节点，另一个添加top和bottom节点，这样问题就迎刃而解了。



代码实现

Percolation源码如下：

```
import edu.princeton.cs.algs4.WeightedQuickUnionUF;
public class Percolation {
    private WeightedQuickUnionUF uf1;
    private WeightedQuickUnionUF uf2;
    private boolean[][] opened;
    private int openedCount;
    private int BOTTOM;
```

```
    private int TOP;
    private int N;

    // create n-by-n grid, with all sites blocked
    public Percolation(int n) {
        validateParam(n <= 0);
        N = n;
        BOTTOM = N * N + 2;
        TOP = N * N + 1;
        uf1 = new WeightedQuickUnionUF((N+2) * (N+2) + 2);
        uf2 = new WeightedQuickUnionUF((N+2) * (N+2) + 1);
        opened = new boolean[N+1][N+1];
        for (int i=1; i<=N; i++)
            for (int j=1; j<=N; j++)
                opened[i][j] = false;
        openedCount = 0;
    }
    // open site (row, col) if it is not open already
    public void open(int row, int col) {
        validateParam(row < 1 || row > N || col < 1 || col > N);
        if (!opened[row][col]) {
            opened[row][col] = true;
            int index = xyTo1D(row, col);
            if (row == N) uf1.union(BOTTOM, index);
            if (row == 1) {
                uf1.union(TOP, index);
                uf2.union(TOP, index);
            }
            if (row-1 >= 1 && opened[row-1][col]) {
                uf1.union( index, xyTo1D(row-1, col) );
                uf2.union( index, xyTo1D(row-1, col) );
            }
            if (row+1 <= N && opened[row+1][col]) {
                uf1.union( index, xyTo1D(row+1, col) );
                uf2.union( index, xyTo1D(row+1, col) );
            }
            if (col-1 >= 1 && opened[row][col-1]) {
                uf1.union( index, xyTo1D(row, col-1) );
                uf2.union( index, xyTo1D(row, col-1) );
            }
            if (col+1 <= N && opened[row][col+1]) {
                uf1.union( index, xyTo1D(row, col+1) );
                uf2.union( index, xyTo1D(row, col+1) );
            }
            openedCount++;
        }
```

```
    }
    // is site (row, col) open?
    public boolean isOpen(int row, int col) {
        validateParam(row < 1 || row > N || col < 1 || col > N);
        return opened[row][col];
    }
    // is site (row, col) full?
    public boolean isFull(int row, int col) {
        validateParam(row < 1 || row > N || col < 1 || col > N);
        return uf2.connected(TOP, xyTo1D(row, col) );
    }
    // number of open sites
    public int numberOfOpenSites() {
        return openedCount;
    }
    // does the system percolate?
    public boolean percolates() {
        return uf1.connected(TOP, BOTTOM);
    }
    private int xyTo1D(int row, int col) {
        return (row-1) * N + col;
    }
    private void validateParam(boolean invalid) {
        if (invalid)
            throw new IllegalArgumentException("Data invalid! Please check your inp
    }
    // test client (optional)
    public static void main(String[] args) {
    }
}
```

当然了，题目也要求增加统计API来计算渗透概率临界值p*

```
public class PercolationStats {
  // perform trials independent experiments on an n-by-n grid
   public PercolationStats(int n, int trials)
   public double mean()      // sample mean of percolation threshold
   public double stddev()   // sample standard deviation of percolation threshold
   public double confidenceLo() // low  endpoint of 95% confidence interval
   public double confidenceHi() // high endpoint of 95% confidence interval
   public static void main(String[] args)        // test client (described below)
}
```

这个是so easy的事，直接看assigments中的定义，把代码敲上即可。

```
import edu.princeton.cs.algs4.StdRandom;
import edu.princeton.cs.algs4.StdStats;
public class PercolationStats {
    private double[] probability;
    private double stddev = 0.0f;
    private double mean = 0.0f;
    // perform trials independent experiments on an n-by-n grid
    public PercolationStats(int n, int trials) {
        validateParam(n<=0 || trials <=0 );
        probability = new double[trials];
        for (int i=0 ; i<trials; i++) {
            Percolation percolation = new Percolation(n);
            while (!percolation.percolates()) {
                int x= StdRandom.uniform(n) + 1;
                int y= StdRandom.uniform(n) + 1;
                percolation.open(x, y);
            }
            probability[i] = (double) percolation.numberOfOpenSites() / (n*n);
        }
    }
    // sample mean of percolation threshold
    public double mean() {
        if (Double.compare(mean, 0.0f) == 0) {
            mean = StdStats.mean(probability);
        }
        return mean;
    }
    // sample standard deviation of percolation threshold
    public double stddev() {
        if (Double.compare(stddev, 0.0f) == 0) {
            stddev = StdStats.stddev(probability);
        }
        return stddev;
    }
    // low  endpoint of 95% confidence interval
    public double confidenceLo() {
        return mean() - 1.96d * stddev() / Math.sqrt(probability.length);
    }
    // high endpoint of 95% confidence interval
    public double confidenceHi() {
        return mean() + 1.96d * stddev() / Math.sqrt(probability.length);
    }
    private void validateParam(boolean invalid) {
        if (invalid)
```

```
            throw new IllegalArgumentException("Data invalid! Please check your inp
    }
    // test client (described below)
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        int times = Integer.parseInt(args[1]);
        PercolationStats stats = new PercolationStats(n, times);
        System.out.println("mean                    = " + stats.mean());
        System.out.println("stddev                  = " + stats.stddev());
        System.out.println("95% confidence interval = ["
                + stats.confidenceLo()
                + ", "
                + stats.confidenceHi()
                + "]");
    }
}
```

总结

渗透模型有很多应用（参考：Percolation Models）： 如森林火灾模型（当森林密度超过该阈值时，就会发生火灾），银行渗透模型，国家倒闭模型等。

PS:

第一次提交，只得了83分，原因是xyTo1D函数设计得有点问题，第二次提交得了99分，原因如下：

```
Test 1: count calls to StdStats.mean() and StdStats.stddev()
  * n =  20, trials =  10
    - calls StdStats.mean() the wrong number of times
    - number of student   calls to StdStats.mean() = 3
    - number of reference calls to StdStats.mean() = 1
==> FAILED
```

统计API没有做缓存，后来又改了一版，终于得到了100分。然而比较可惜的是，并没有得到bonus。

```
Estimated student memory = 17.00 n^2 + 105.00 n + 392.00   (R^2 = 1.000)
Test 2 (bonus): check that total memory <= 11 n^2 + 128 n + 1024 bytes
```

```
    -  failed memory test for n = 64
==> FAILED
```

对于当前的算法，暂时没有办法把n^2前面的系数减少，如果你有更好地算法，欢迎和我讨论。

打赏

Coursera # Percolation

Coursera公开课–Algorithm–WordNet

关于头条被整改的一点想法