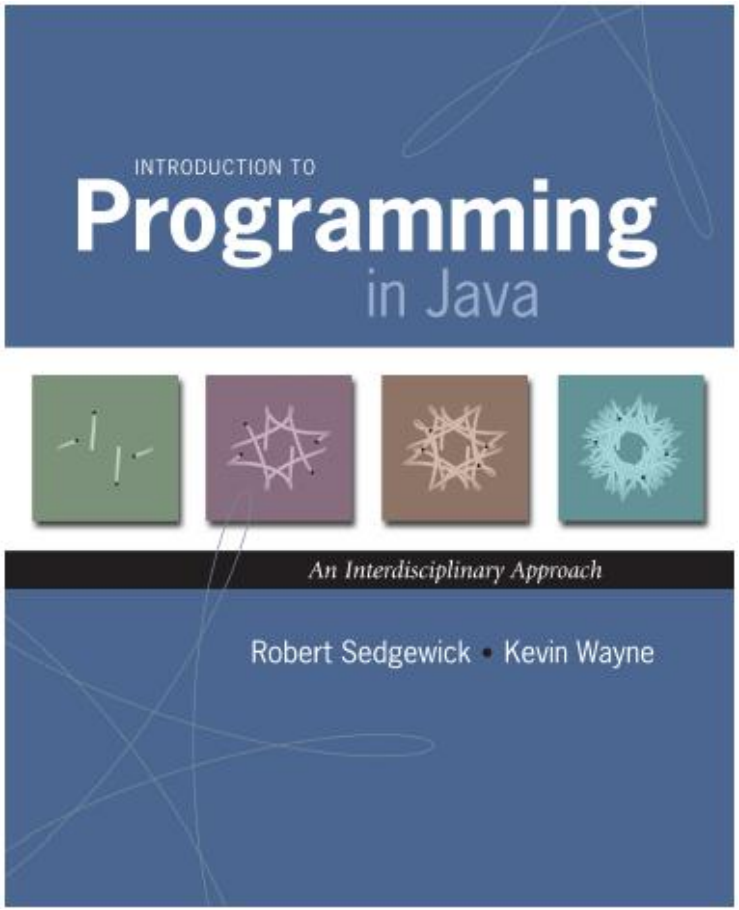
توابع و کتابخانه‌ها:
مسئله تراوش

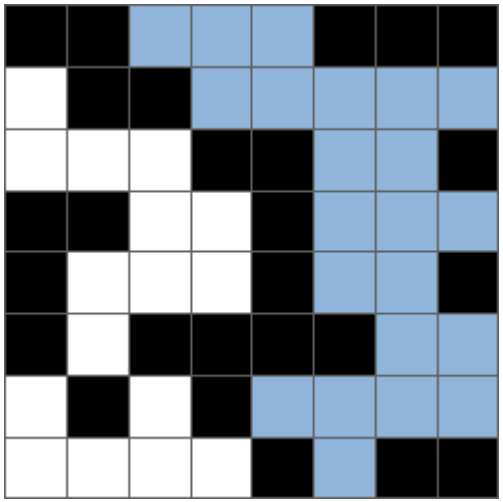
سید ناصر رضوی www.snrazavi.ir
۱۳۹۵

۴-۲ مطالعه موردی: مسئله تراوش



مطالعه موردی: مسئله تراوش

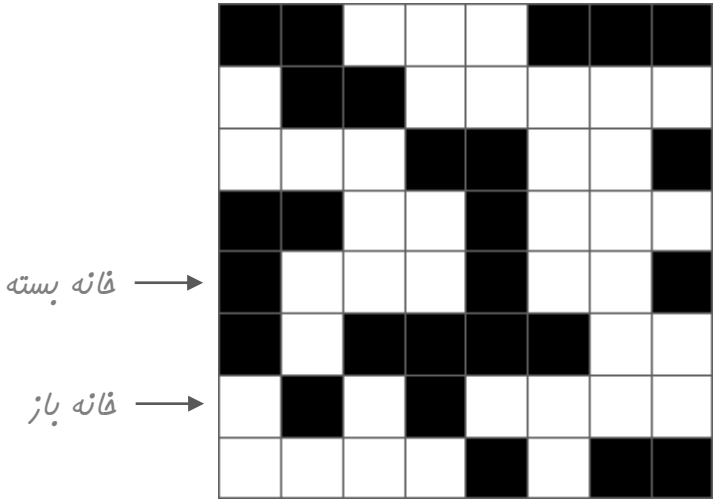
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟

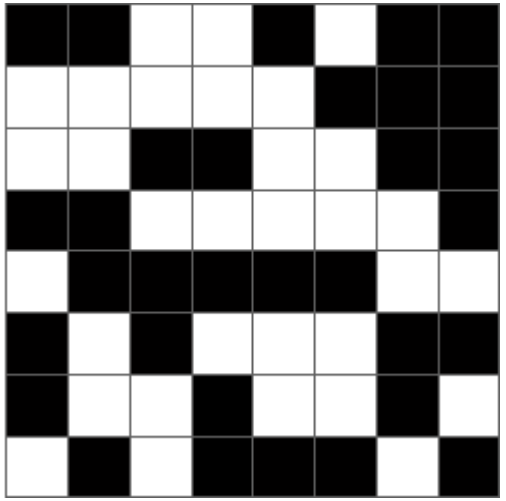


- کاربردها.
  - رنگ نگاری.
  - گسترش آتش در جنگل.
  - نشت گاز طبیعی از طریق سنگ‌های نیمه متخلخل.
  - جریان الکتریسیته در یک شبکه از مقاومت‌ها.
  - نفوذ گاز در فیلتر ماسک ضد گاز در معادن ذغال سنگ.
  - ...

مطالعه موردی: مسئله تراوش

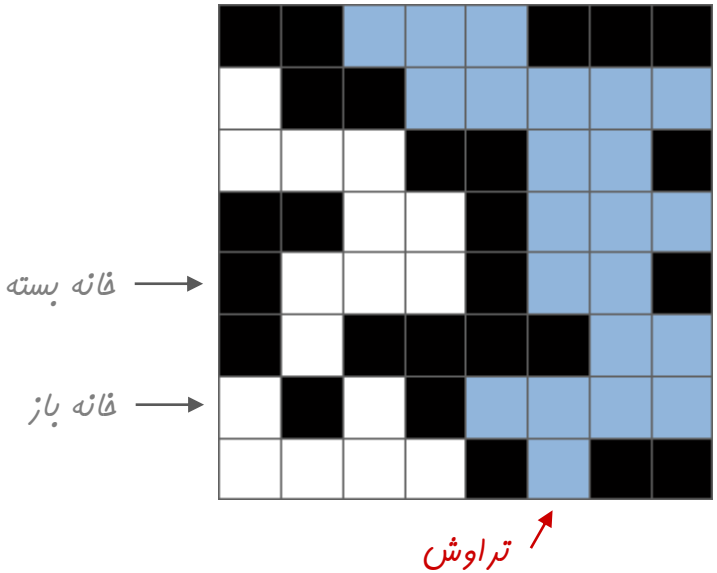
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟
- مدل انتزاعی.
  - یک جدول N در N از خانه‌ها.
  - هر خانه یا باز است و یا بسته.

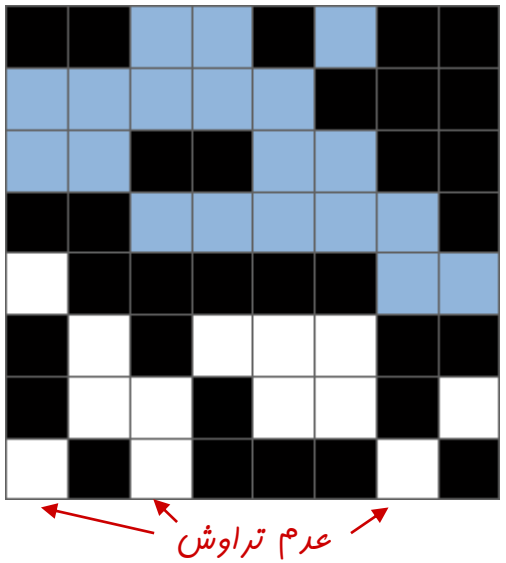




مطالعه موردی: مسئله تراوش

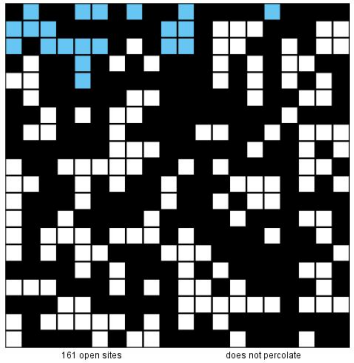
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟
- مدل انتزاعی.
  - یک جدول N در N از خانه‌ها.
  - هر خانه یا باز است و یا بسته.
  - یک خانه باز پر است اگر از طریق خانه‌های خالی به سطر بالایی متصل باشد.



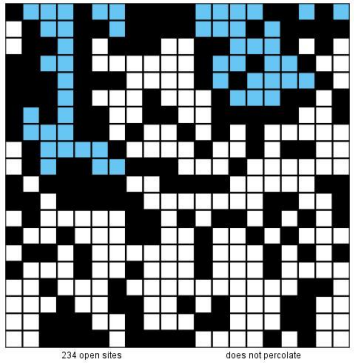


یک مسئله علمی

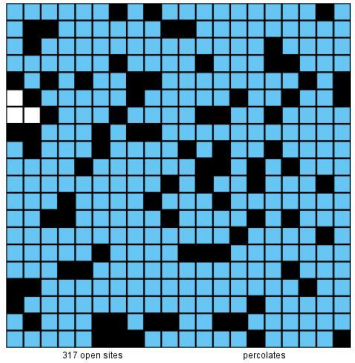
- تراوش تصادفی. با داشتن یک جدول N در N که هر خانه آن با احتمال p باز است، احتمال تراوش در سیستم چقدر است؟



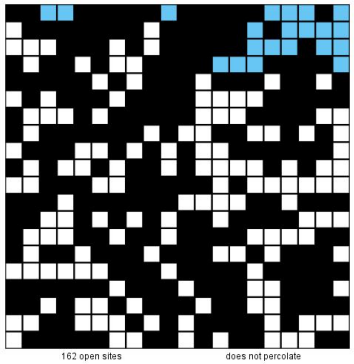
p = 0.4



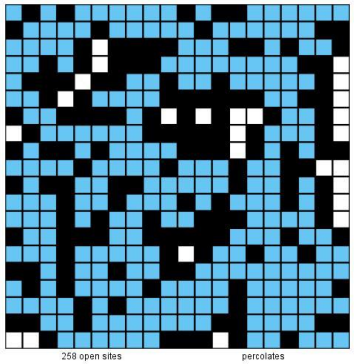
p = 0.6



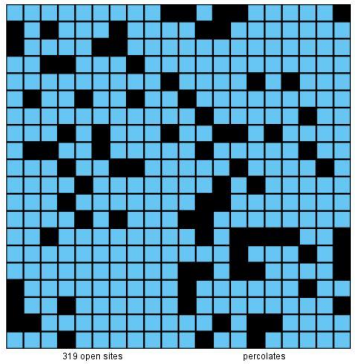
p = 0.8



عدم تراوش



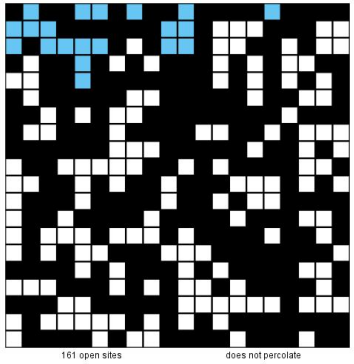
تراوش؟



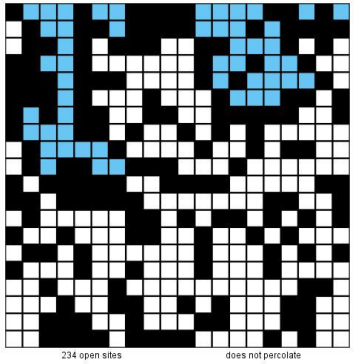
تراوش

یک مسئله علمی

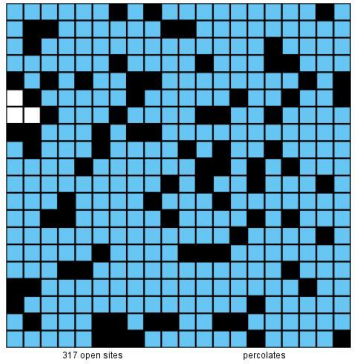
- تراوش تصادفی. با داشتن یک جدول N در N که هر خانه آن با احتمال p باز است، احتمال تراوش در سیستم چقدر است؟



p = 0.4



p = 0.6



p = 0.8

- ملاحظه. یک پرسش باز مشهور در فیزیک آماری.

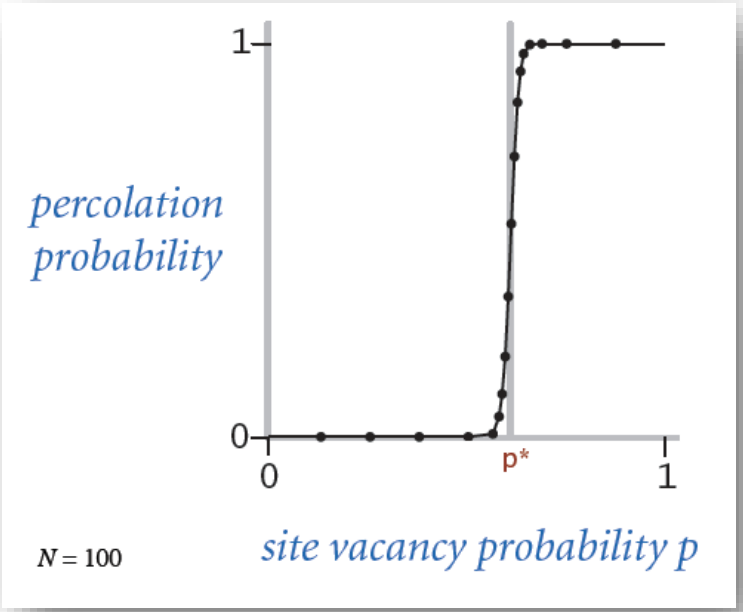
عدم وجود راه‌حل شناخته شده ریاضی

- راه‌حل. استفاده از یک رویکرد محاسباتی: شبیه‌سازی مونت کارلو.

تغییر فاز تراوش

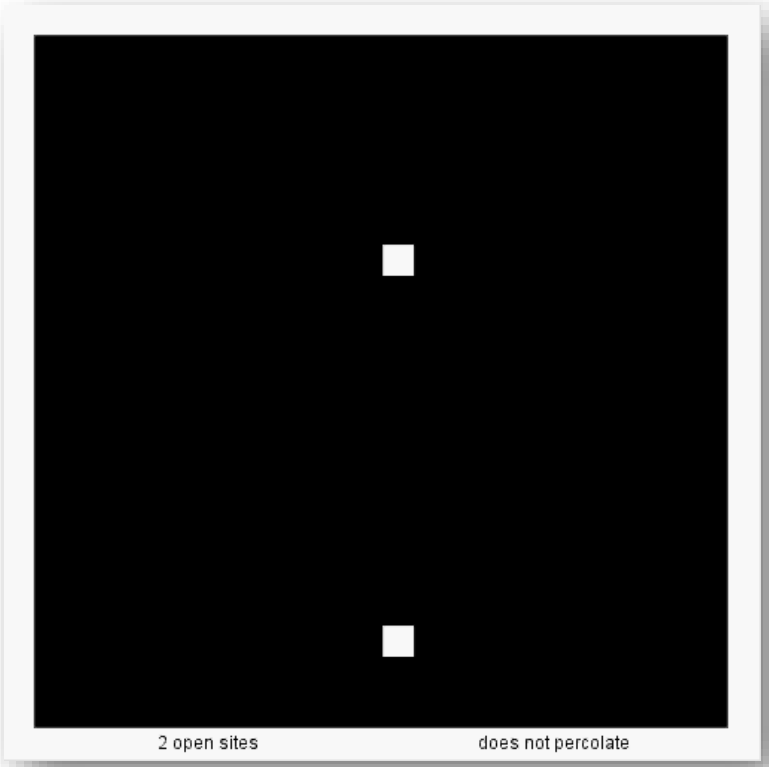
- به لحاظ نظری برای N های بزرگ، یک آستانه مانند p* وجود دارد:
  - p < p*: به احتمال نزدیک به یقین عدم تراوش
  - p > p*: به احتمال نزدیک به یقین تراوش

- س. مقدار p* چند است؟



شبیه سازی مونت-کارلو

- در ابتدا تمام خانه‌های جدول N در N بسته هستند.
- هر بار یک خانه‌ی تصادفی را باز کن تا زمانی که سطر اول و آخر به هم متصل شوند.
- درصد خانه‌های خالی تخمینی از مقدار p* است.



N = 20

خانه های پر

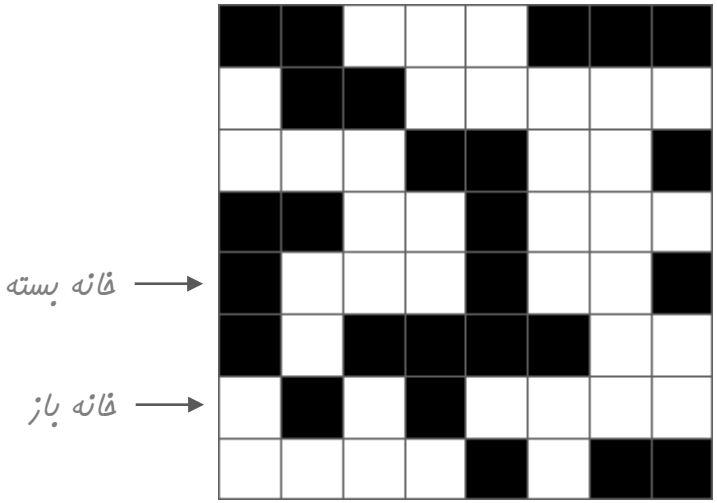
خانه های خالی

خانه های بسته

نمایش داده‌ها

- نمایش داده‌ها. از یک ماتریس N در N بولی برای نمایش خانه‌های باز و از یک ماتریس دیگر برای محاسبه خانه‌های پر استفاده کن.

- کتابخانه استاندارد `StdArrayIO`. یک کتابخانه به منظور پشتیبانی از عملیات خواندن و چاپ آرایه‌های یک بعدی و دو بعدی.




```
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
```

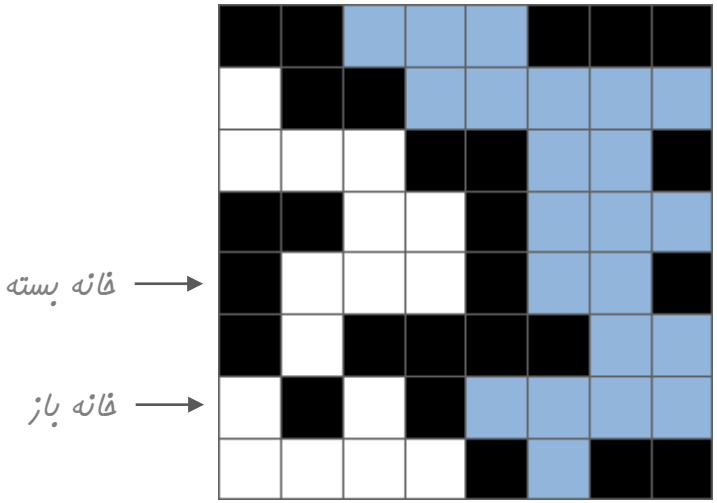
open [] []

0 به معنای بسته؛
1 به معنای باز

نمایش داده‌ها

- نمایش داده‌ها. از یک ماتریس N در N بولی برای نمایش خانه‌های باز و از یک ماتریس دیگر برای محاسبه خانه‌های پر استفاده کن.

- کتابخانه استاندارد StdArrayIO. یک کتابخانه به منظور پشتیبانی از عملیات خواندن و چاپ آرایه‌های یک بعدی و دو بعدی.




```
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
```

full[][]

0 به معنای خالی؛
1 به معنای پر

کتابخانه استاندارد StdArrayIO


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

مسئله تراوش

- رویکرد. با یک کد ساده شروع کنید. جزییات را بعداً پر کنید.

```
public class Percolation {

   // return boolean matrix representing full sites
   public static boolean[][] flow(boolean[][] open)

   // does the system percolate?
   public static boolean percolates(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;
   }

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

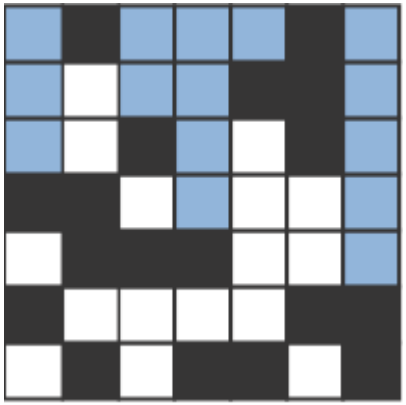
سیستم تراوش می‌کند اگر حداقل یکی از خانه‌های سطر آخر پر باشد

تراوش عمودی

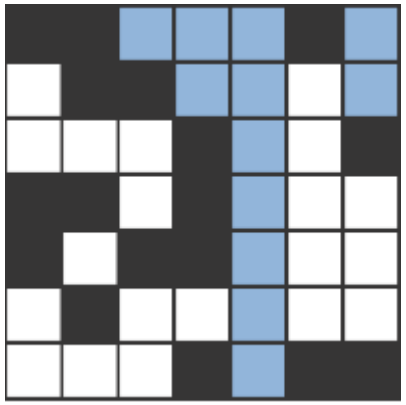
- گام بعدی. شروع با حل کردن یک نسخه ساده‌تر از مسئله.

- تراوش عمودی. آیا یک مسیر مستقیم شامل خانه‌های باز از سطر اول به سطر آخر وجود دارد؟

عدم تراوش عمودی



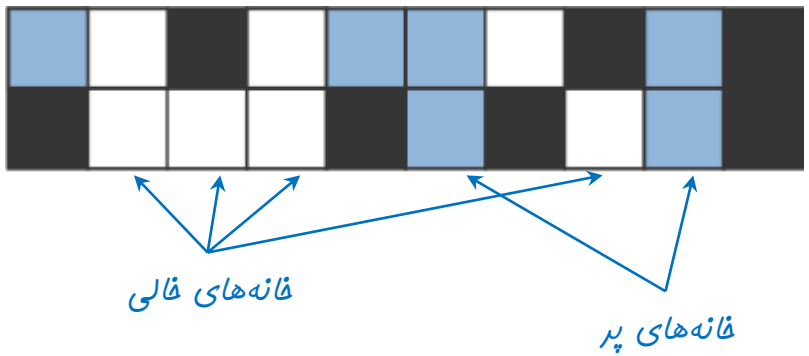
تراوش عمودی



تراوش عمودی

- پرسش. چگونه می‌توان پر بودن خانه `(i, j)` را تعیین نمود؟
- پاسخ. این خانه پر است اگر (۱) باز باشد و (۲) خانه `(i-1, j)` پر باشد.

- الگوریتم. بررسی سطرهای جدول از بالا به پایین.



تراوش عمودی

- پرسش. چگونه می‌توان پر بودن خانه (i , j) را تعیین نمود؟
- پاسخ. این خانه پر است اگر (۱) باز باشد و (۲) خانه (i-1, j) پر باشد.

- الگوریتم. بررسی سطرهای جدول از بالا به پایین.

```
public static boolean[][] flow(boolean[][] open) {

    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
        full[0][j] = open[0][j];

    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;
}
```

مقداردهی اولیه

یافتن خانه‌های پر

تراوش عمودی: آزمایش

- آزمایش. استفاده از ورودی و خروجی استاندارد برای آزمایش ورودی‌های کوچک.

```
% 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

% more testF.txt
5
1 0 1 0 0
1 0 1 1 1
1 1 1 0 1
1 0 0 0 1
0 0 0 1 1
```

```
% java VerticalPercolation < testT.txt
5
0 1 1 0 1
0 0 1 0 1
0 0 0 0 1
0 0 0 0 1
0 0 0 0 1
true

% java VerticalPercolation < testF.txt
5
1 0 1 0 0
1 0 1 0 0
1 0 1 0 0
1 0 0 0 0
0 0 0 0 0
false
```

تراوش عمودی: آزمایش

- آزمایش. افزودن یک متد کمکی برای تولید ورودی‌های تصادفی و به تصویر کشیدن خروجی با استفاده از ترسیم استاندارد.

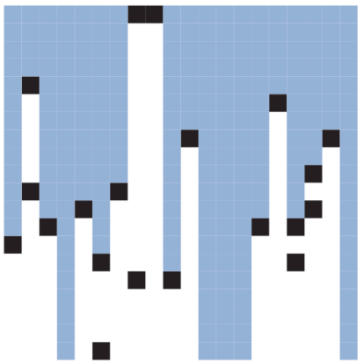
```
public class Percolation {
   ...

   // return a random N-by-N matrix; each cell true with prob p
   public static boolean[][] random(int N, double p) {
      boolean[][] a = new boolean[N][N];
      for (int i = 0; i < N; i++)
         for (int j = 0; j < N; j++)
            a[i][j] = StdRandom.bernoulli(p);
      return a;
   }

   // plot matrix to standard drawing
   public static void show(boolean[][] a, boolean foreground)
}
```

به تصویر کشیدن داده‌ها

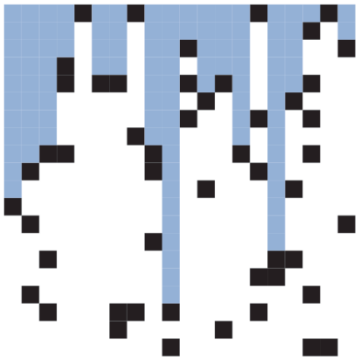
- به تصویر کشیدن داده‌ها. به منظور بررسی ورودی‌های بزرگ‌تر.

```
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);
    }
}
```

```
% java Visualize 20 .95 1
```



```
% java Visualize 20 .9 1
```



تخمین احتمال تراوش عمودی

- تحلیل. با داشتن مقادیر N و p، به تعداد T بار شبیه‌سازی کن و میانگین را گزارش کن.

```
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 / T;
   }

   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));
   }
}
```

تخمین احتمال تراوش عمودی

- تحلیل. با داشتن مقادیر N و p، به تعداد T بار شبیه‌سازی کن و میانگین را گزارش کن.

```
% java Estimate 20 .7 100000
 0.015768

% java Estimate 20 .8 100000
 0.206757

% java Estimate 20 .9 100000
 0.925191

% java Estimate 40 .9 100000
 0.448536
```

- زمان اجرا. متناسب با TN^2. ⟵ *حجم محاسبات بسیار زیاد!*
- مصرف حافظه. متناسب با N^2.

مسئله تراوش: راه‌حل بازگشتی

- تراوش. با داشتن یک جدول N در N، آیا مسیری شامل خانه‌های باز از سطر بالایی به سطر پایینی وجود دارد؟

- جستجوی اول-عمق. برای ملاقات تمام خانه‌های قابل دسترس از خانه `i-j`:
  - اگر خانه `i-j` هم اکنون به عنوان یک خانه قابل دسترس علامت گذاری شده، بازگشت کن.
  - اگر خانه `i-j` باز نیست، بازگشت کن.
  - خانه `i-j` را به عنوان یک خانه قابل دسترس علامت گذاری کن.
  - به صورت بازگشتی هر چهار همسایه خانه `i-j` را (در صورت وجود) ملاقات کن.

- راه‌حل مسئله تراوش.
  - جستجوی اول-عمق را از هر یک از خانه‌های سطر اول انجام بده.
  - بررسی کن آیا خانه‌ای در سطر آخر وجود دارد که به عنوان قابل دسترس علامت گذاری شده باشد.

مسئله تراوش: راه‌حل بازگشتی


```
public static boolean[][] flow(boolean[][] open) {
    int N = open.length;
    boolean[][] full = new boolean[N][N];
    for (int j = 0; j < N; j++)
        if (open[0][j]) flow(open, full, 0, j);
    return full;
}

public static void flow(boolean[][] open, boolean[][] full, int i, int j) {
    int N = full.length;
    if (i < 0 || i >= N || j < 0 || j >= N) return;
    if (!open[i][j]) return;
    if ( full[i][j]) return;
    full[i][j] = true;              // mark
    flow(open, full, i+1, j);       // down
    flow(open, full, i, j+1);       // right
    flow(open, full, i, j-1);       // left
    flow(open, full, i-1, j);       // up
}
```

تخمین احتمال تراوش

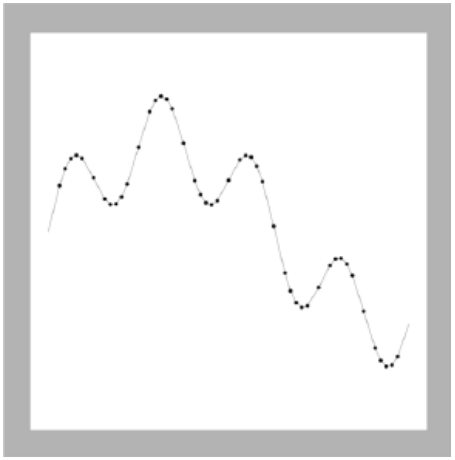
- تحلیل. با داشتن مقادیر N و p، به تعداد T بار شبیه‌سازی کن و میانگین را گزارش کن.

```
% 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
```

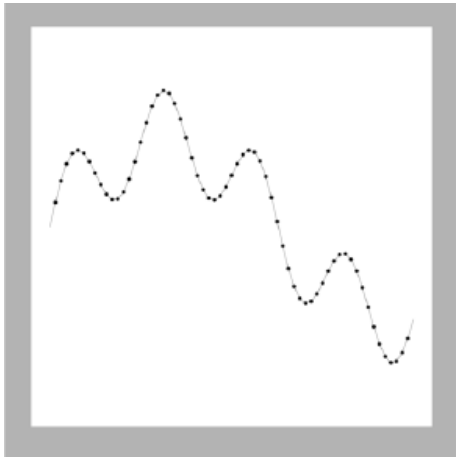
- زمان اجرا. هنوز متناسب با TN^2. ⟵ *حجم محاسبات بسیار زیاد!*
- مصرف حافظه. هنوز متناسب با N^2.

ترسیم تطبیقی

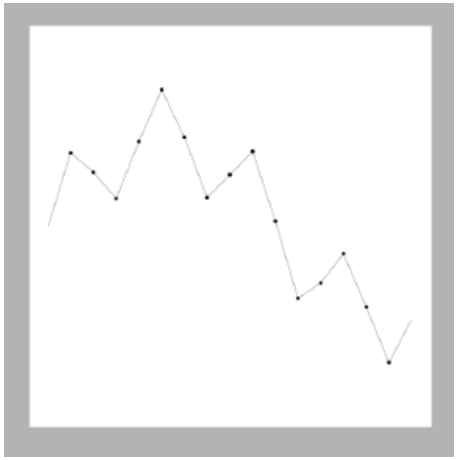
- ترسیم نتایج. ترسیم احتمال تراوش یک سیستم به عنوان تابعی از احتمال باز بودن خانه‌ها.
- تصمیمات طراحی.
  - چند مقدار p؟
  - کدام مقادیر p؟
  - چند آزمایش به ازای هر مقدار p؟



تعداد نقاط مناسب



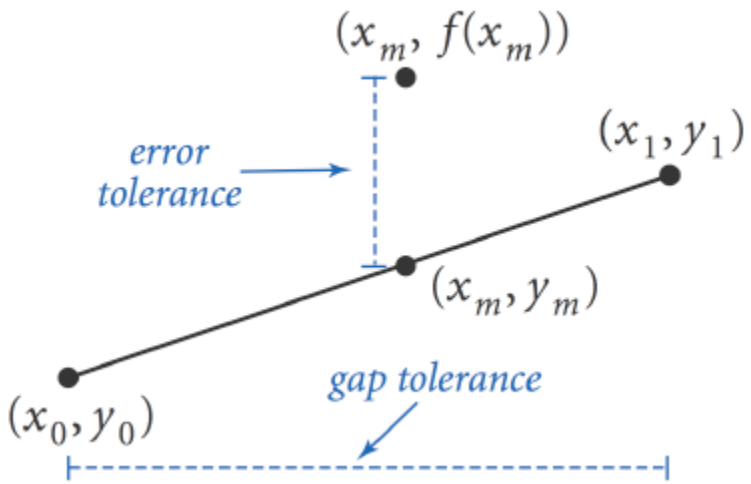
تعداد نقاط بسیار زیاد



تعداد نقاط بسیار کم

ترسیم تطبیقی برای مسئله تراوش

- ترسیم تطبیقی. برای ترسیم تابع f(x) در بازه $[x_0, x_1]$
  - اگر بازه به اندازه کافی کوچک است، توقف کن.
  - بازه را به دو قسمت مساوی تقسیم کن و $f(x_m)$ را محاسبه کن.
  - اگر مقدار $f(x_m)$ به مقدار $\frac{1}{2}(f(x_0) + f(x_1))$ نزدیک است، توقف کن.
  - تابع f(x) را به صورت بازگشتی در بازه $[x_0, x_m]$ ترسیم کن.
  - نقطه $(x_m, f(x_m))$ را ترسیم کن.
  - تابع f(x) را به صورت بازگشتی در بازه $[x_m, x_1]$ ترسیم کن.



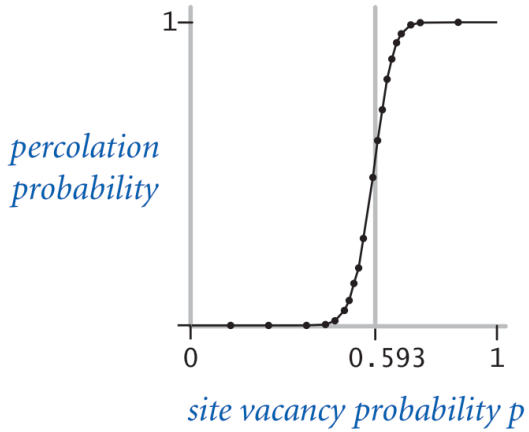
ترسیم تطبیقی برای مسئله تراوش


```
public class PercPlot {
    public static void curve(int N, double x0, double y0,
    public static void curve(int N, 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, .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);
    }
}
```

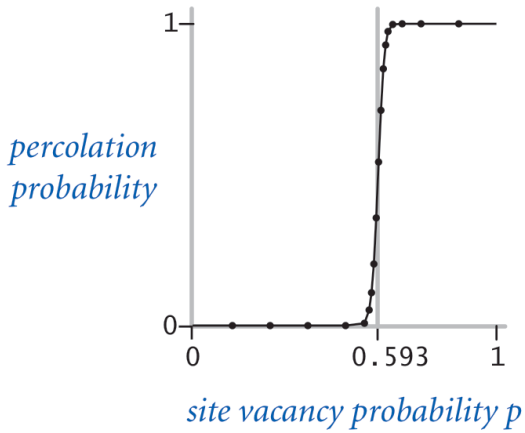
ترسیم تطبیقی

- ترسیم نتایج. ترسیم احتمال تراوش یک سیستم به عنوان تابعی از احتمال باز بودن خانه‌ها.

```
% java PercPlot 20
```

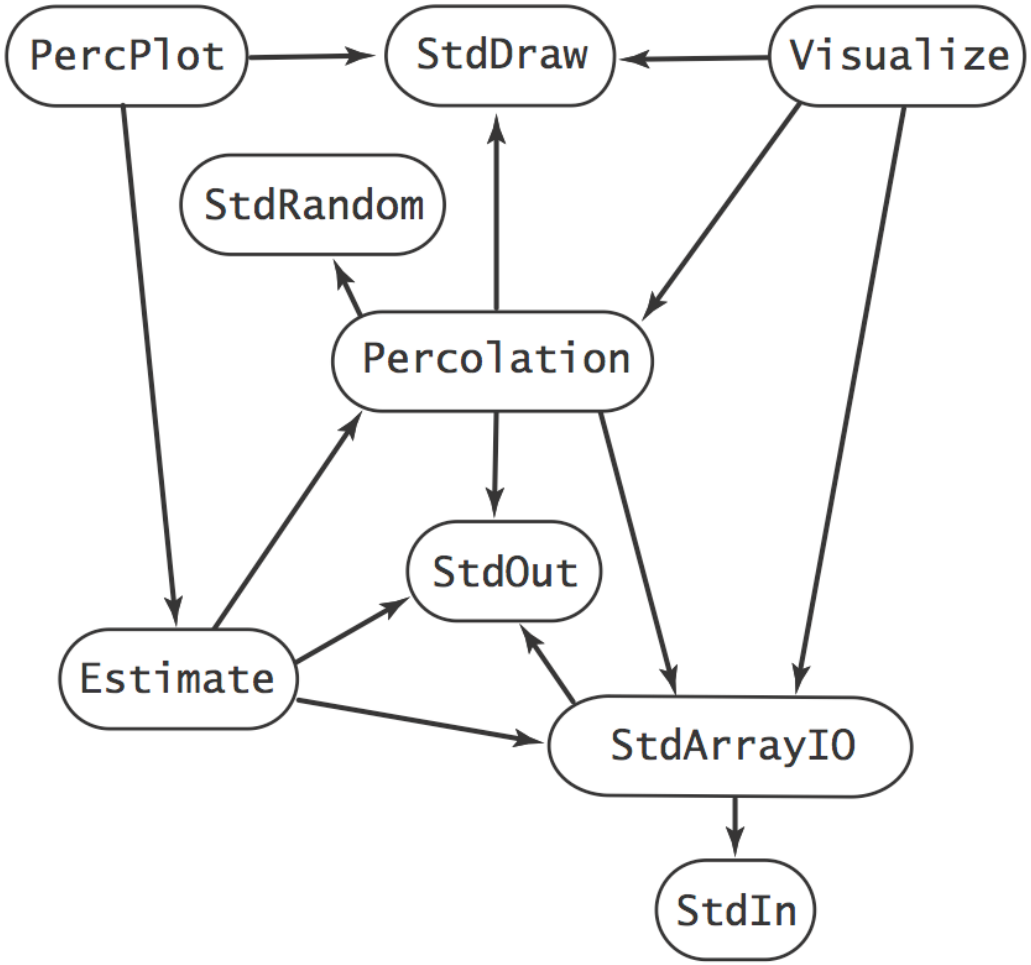



```
% java PercPlot 100
```



- تغییر فاز. اگر $p < 0.593$ باشد سیستم تقریباً هرگز تراوش نمی‌کند، اما اگر $p > 0.593$ باشد سیستم تقریباً همواره تراوش می‌کند.

گراف وابستگی



خلاصه

- همیشه برای خطا آماده باشد.
  - برنامه را بر روی ورودی‌های کوچک اجرا کنید.
- اندازه ماجول‌ها را تا حد امکان کوچک نگه دارید.
  - برای ساده‌سازی فرایند آزمایش و اشکال زدایی
- از روش توسعه تدریجی استفاده کنید.
  - هر ماجولی که می‌نویسید آن را اجرا و اشکال زدایی کنید.
- ابتدا یک مسئله ساده‌تر را حل کنید.
  - به عنوان اولین گام.
- به راه‌حل بازگشتی فکر کنید.
  - یک ابزار بسیار ارزشمند.
- کتابخانه‌های قابل استفاده بسازید.
  - مانند کتابخانه‌های استاندارد.