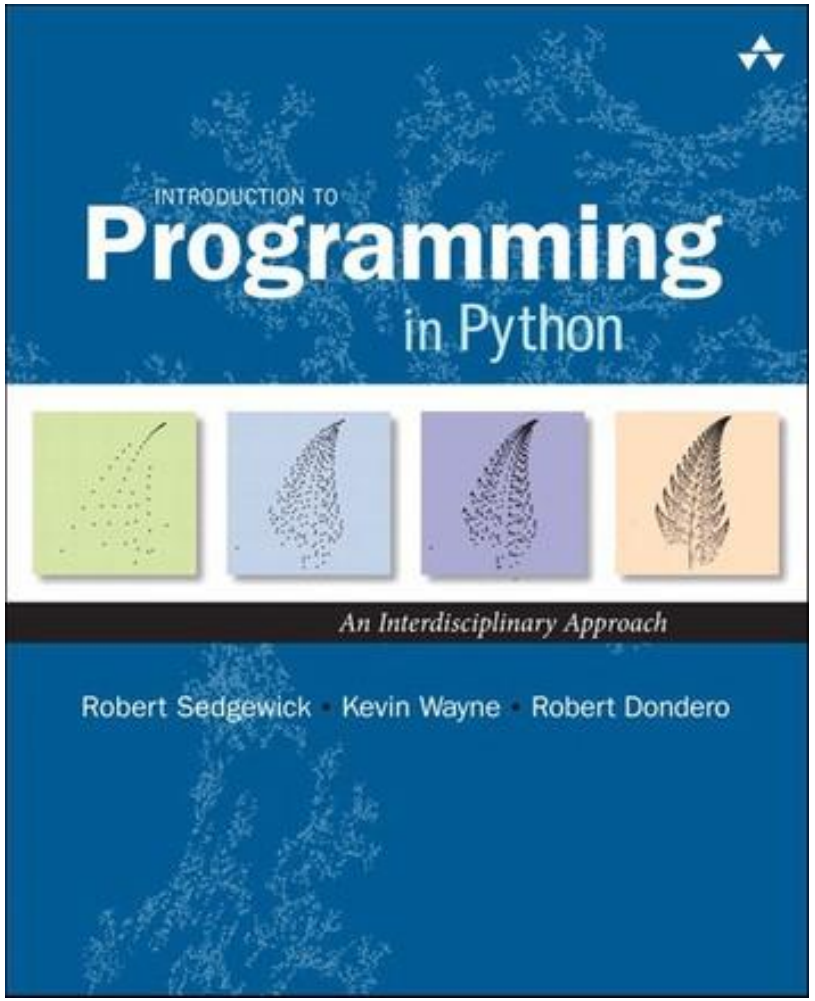
توابع و کتابخانه‌ها: مسئله تراوش

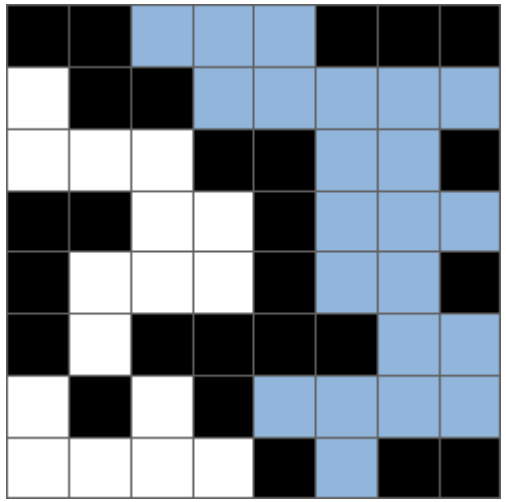
سید ناصر رضوی www.snrazavi.ir
۱۳۹۸

۴-۲ مطالعه موردی: مسئله تراوش



مطالعه موردی: مسئله تراوش

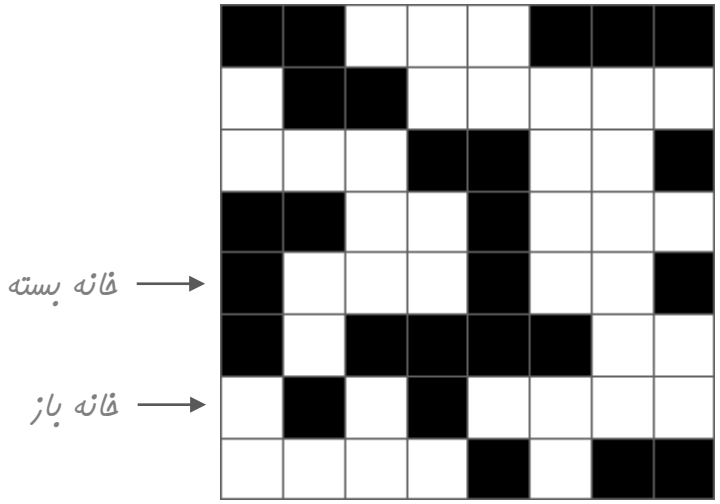
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟

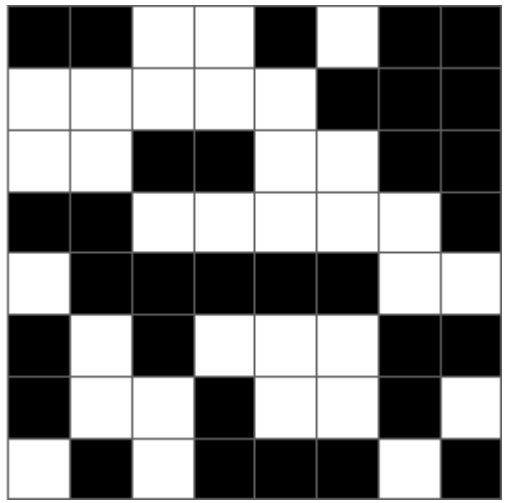


- کاربردها.
  - رنگ نگاری.
  - گسترش آتش در جنگل.
  - نشت گاز طبیعی از طریق سنگ‌های نیمه متخلخل.
  - جریان الکتریسیته در یک شبکه از مقاومت‌ها.
  - نفوذ گاز در فیلتر ماسک ضد گاز در معادن زغال‌سنگ.
  - ...

مطالعه موردی: مسئله تراوش

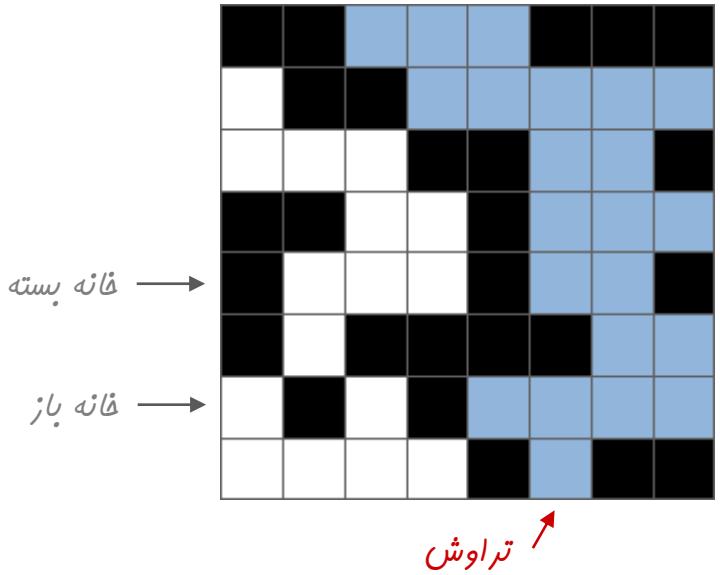
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟
- مدل انتزاعی.
  - یک جدول n در n از خانه‌ها.
  - هر خانه یا باز است و یا بسته.

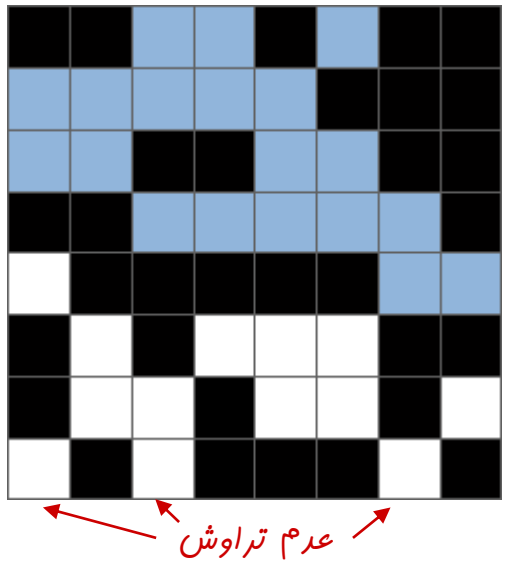




مطالعه موردی: مسئله تراوش

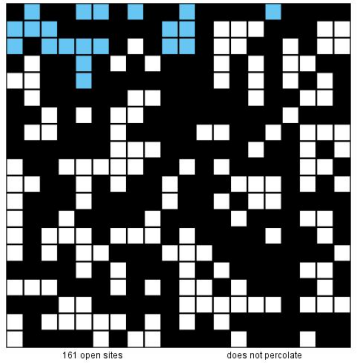
- تراوش. مقداری مایع بر روی سطح یک ماده متخلخل بریزید. آیا مایع به قسمت زیرین ماده خواهد رسید؟
- مدل انتزاعی.
  - یک جدول n در n از خانه‌ها.
  - هر خانه یا باز است و یا بسته.
  - یک خانه باز، پر است اگر از طریق خانه‌های خالی به سطر بالایی متصل باشد.



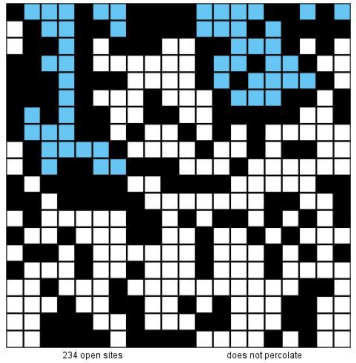


یک مسئله علمی

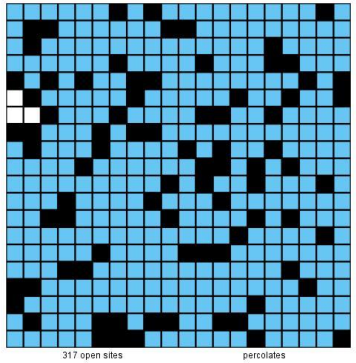
- تراوش تصادفی. با داشتن یک جدول n در n که هر خانه آن با احتمال p باز است، احتمال تراوش در سیستم چقدر است؟



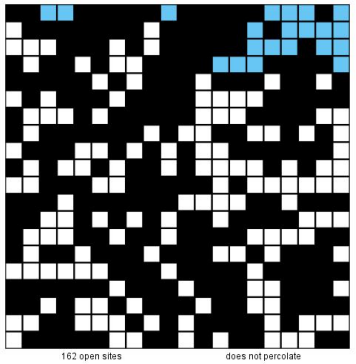
p = 0.4



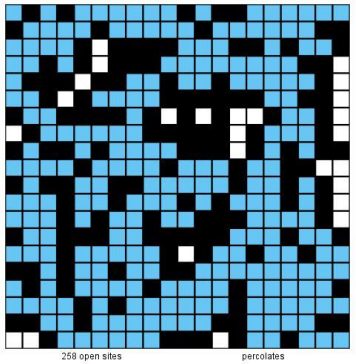
p = 0.6



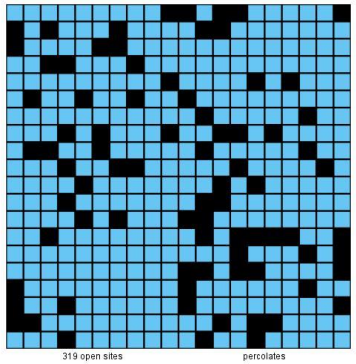
p = 0.8



عدم تراوش



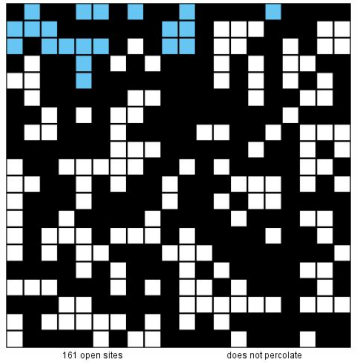
تراوش؟



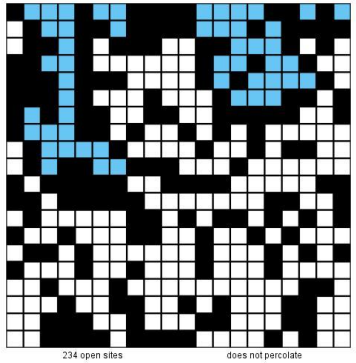
تراوش

یک مسئله علمی

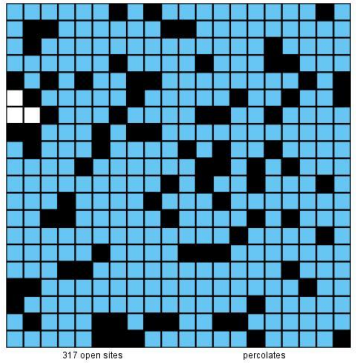
- تراوش تصادفی. با داشتن یک جدول n در n که هر خانه آن با احتمال p باز است، احتمال تراوش در سیستم چقدر است؟



p = 0.4



p = 0.6



p = 0.8

- ملاحظه. یک پرسش باز مشهور در فیزیک آماری.

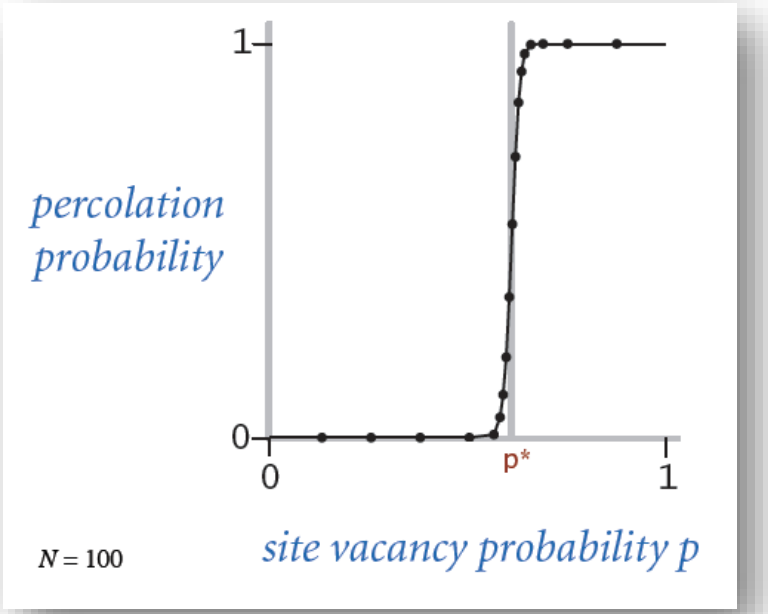
عدم وجود راه‌حل شناخته شده ریاضی

- راه‌حل. استفاده از یک رویکرد محاسباتی: شبیه‌سازی مونت‌کارلو.

تغییر فاز تراوش

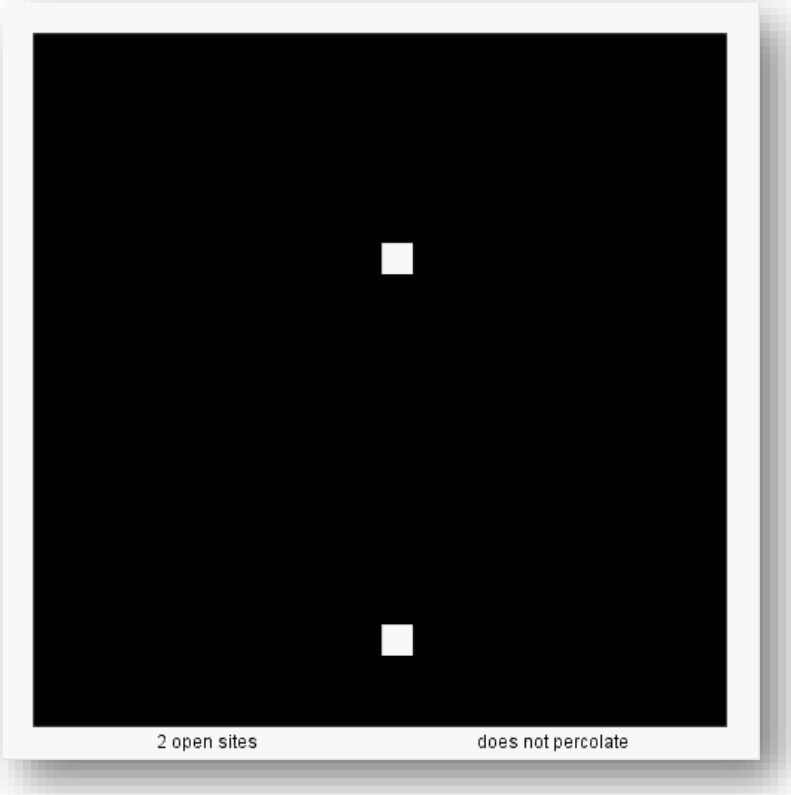
- به لحاظ نظری برای n های بزرگ، یک آستانه مانند p* وجود دارد:
  - p < p*: به احتمال نزدیک به یقین عدم تراوش
  - p > p*: به احتمال نزدیک به یقین تراوش

- پرسش. 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

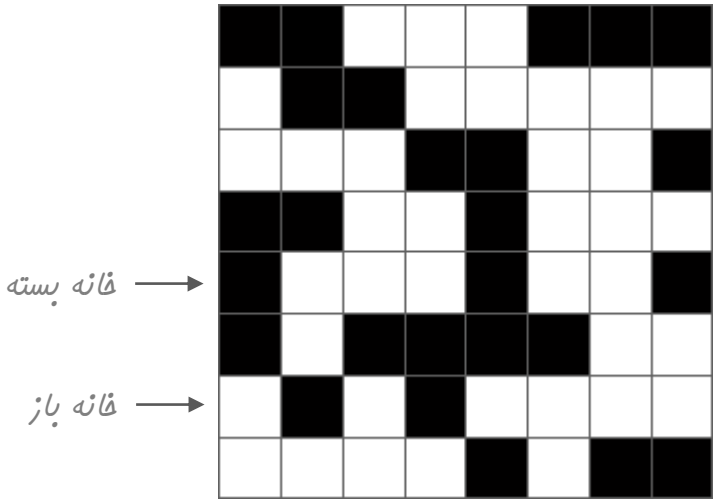
خانه های خالی

خانه های بسته

نمایش داده‌ها

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

- کتابخانه استاندارد `stdarray.py`. 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
```

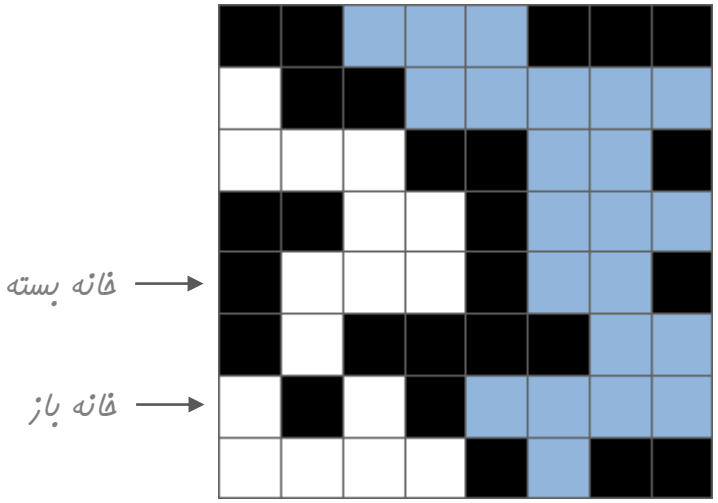
`is_open[][]`

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 بولی دیگر برای محاسبه خانه‌های پر استفاده کن.

- کتابخانه استاندارد `stdarray.py`. 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
```

is_full[][]

0 به معنای خالی؛
1 به معنای پر

کتابخانه استاندارد stdarray.py


```
def readBool1D():
    count = stdio.readInt()
    a = create1D(count, False)
    for i in range(count):
        a[i] = stdio.readBool()
    return a
```

```
def readBool2D():
    row_count = stdio.readInt()
    col_count = stdio.readInt()
    a = create2D(row_count, col_count, False)
    for row in range(row_count):
        for col in range(col_count):
            a[row][col] = stdio.readBool()
    return a
```

```
8
0 0 1 1 1 0 0 0
```

```
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
```

مسئله تراوش

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

```
def flow(is_open):
    pass
```

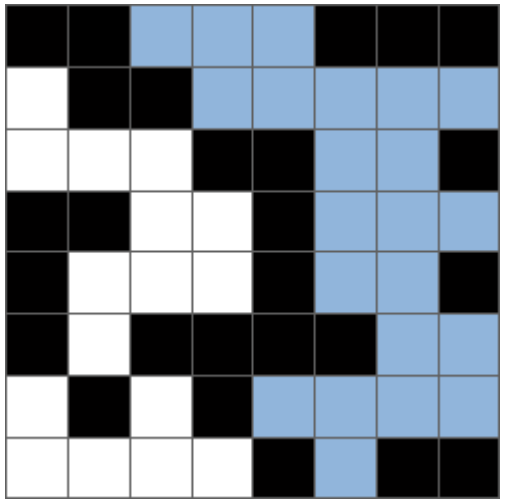
محاسبه خانه‌های پر

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

بررسی تراوش

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

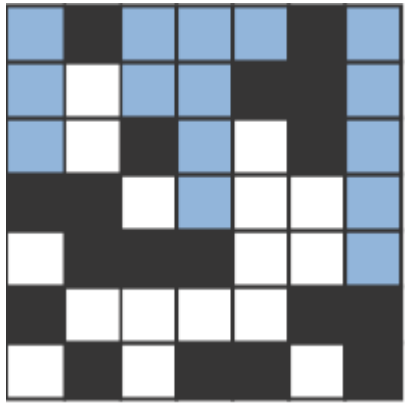
```
def main():
    is_open = stdarray.readBool2D()
    stdarray.write2D(flow(is_open))
    print(percolates(is_open))
```



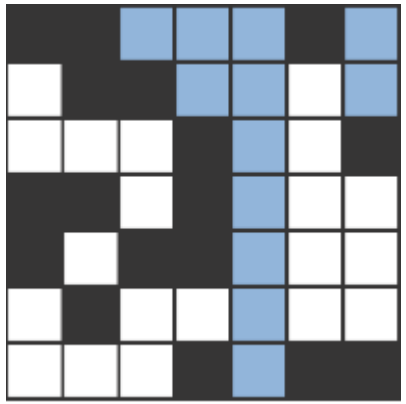
تراوش عمودی

- 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



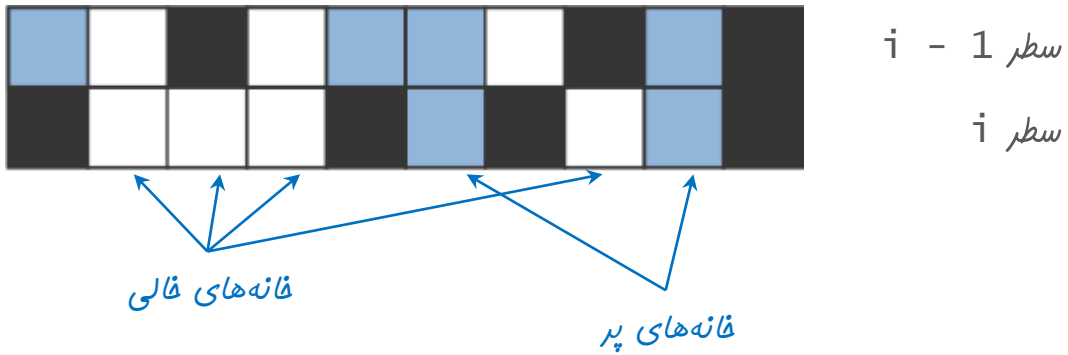
تراوش عمودی



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.

```
def flow(is_open):
    n = len(is_open)
    is_full = stdarray.create2D(n, n, False)
    for j in range(n):
        is_full[0][j] = is_open[0][j]

    for i in range(1, n):
        for j in range(n):
            is_full[i][j] = is_open[i][j] and is_full[i-1][j]

    return is_full
```

مقداردهی اولیه

یافتن خانه‌های پر

تراوش عمودی: آزمایش

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

```
% more test5.txt
5 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 test5F.txt
5 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
```

```
% python percolationv.py < test5.txt
5 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

% python percolationv.py < test5F.txt
5 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.

```
# return a random n-by-n matrix; each cell true with prob p
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
```

تراوش عمودی: آزمایش

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

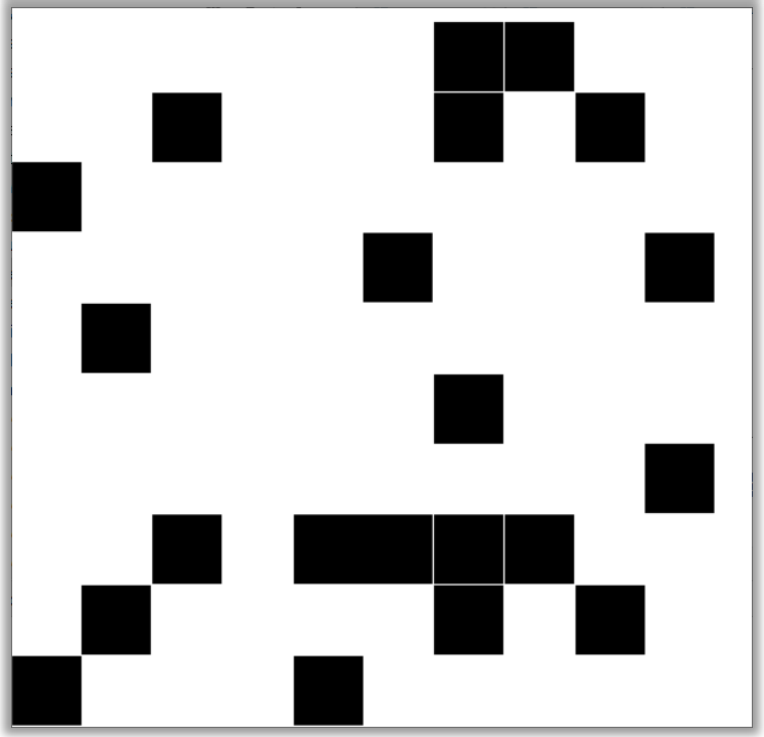
```
# plot matrix to standard drawing
def draw(a, which):
    n = len(a)
    stddraw.setXscale(-.5, n)
    stddraw.setYscale(-.5, n)
    for i in range(n):
        for j in range(n):
            if a[i][j] == which:
                stddraw.filledSquare(j, n-i-1, .5)
```

تراوش عمودی: آزمایش

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

```
# for testing
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()
```

```
% python percolationio.py 10 .8
```

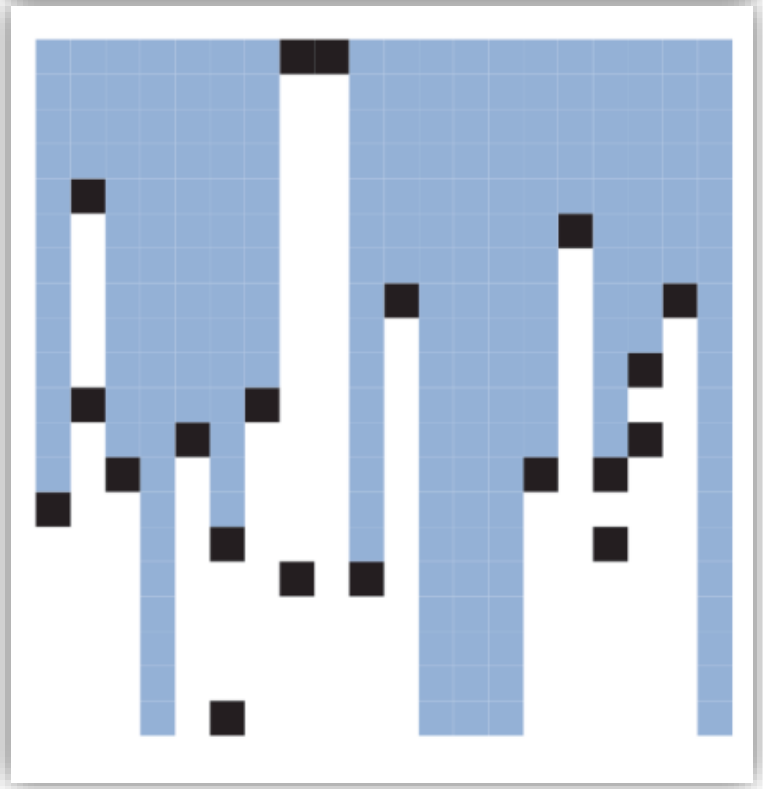


به تصویر کشیدن داده‌ها

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

```
def main():
    n = int(sys.argv[1])
    p = float(sys.argv[2])
    trials = int(sys.argv[3])
    for i in range(trials):
        is_open = percolationio.random(n, p)
        stddraw.clear()
        stddraw.setPenColor(stddraw.BLACK)
        percolationio.draw(is_open, False)
        full = percolationv.flow(is_open)
        stddraw.setPenColor(stddraw.BLUE)
        percolationio.draw(full, True)
        stddraw.show(1000)
    stddraw.show()

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

```
% python visualizev.py 20 .9 1
```



تخمین احتمال تراوش عمودی: شبیه‌سازی مونت‌کارلو

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

```
def evaluate(n, p, trials):
    count = 0
    for i in range(trials):
        is_open = percolationio.random(n, p)
        if percolationv.percolates(is_open)):
            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()
```

```
% python estimatev.py 20 .75 10
0.1

% python estimatev.py 20 .95 10
1.0

% python estimatev.py 20 .85 10
0.3

% python estimatev.py 20 .85 1000
0.543

% python estimatev.py 20 .85 1000
0.56

% python estimatev.py 40 .85 100
0.03
```

تخمین احتمال تراوش عمودی

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

```
% python estimatev.py 20 .75 10
0.1

% python estimatev.py 20 .95 10
1.0

% python estimatev.py 20 .85 10
0.3

% python estimatev.py 20 .85 1000
0.543

% python estimatev.py 20 .85 1000
0.56

% python estimatev.py 40 .85 100
0.03
```

- زمان اجرا. متناسب با n^2 * trials. ← 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 علامت‌گذاری شده، بازگشت کن.
  - اگر خانه `i-j` باز نیست، بازگشت کن.
  - خانه `i-j` را به عنوان یک خانه قابل دسترس علامت‌گذاری 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 علامت‌گذاری شده باشد.

مسئله تراوش: راه‌حل بازگشتی


```
def flow(is_open):
    n = len(is_open)
    is_full = stdarray.create2D(n, n, False)
    for j in range(n):
        _flow(is_open, is_full, 0, j)
    return is_full
```

```
def _flow(is_open, is_full, i, j):
    n = len(is_full)
    if i < 0 or i >= n or j < 0 or j >= n: return
    if not is_open[i][j]: return
    if is_full[i][j]: return
    is_full[i][j] = True                # mark
    _flow(is_open, is_full, i+1, j)   # down
    _flow(is_open, is_full, i, j+1)   # right
    _flow(is_open, is_full, i, j-1)   # left
    _flow(is_open, is_full, i-1, j)   # up
```

```
% python percolation.py < test5.txt
5 5
0 1 1 0 1
0 0 1 1 1
0 0 0 1 1
0 0 0 0 1
0 1 1 1 1
True
```

تخمین احتمال تراوش

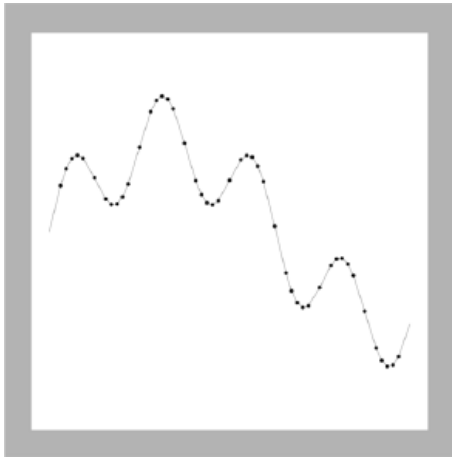
- تحلیل. با داشتن مقادیر n و p، به تعداد trials بار شبیه‌سازی کن و میانگین را گزارش کن.

```
% python estimate.py 20 .55 100
0.25
% python estimate.py 20 .65 100
0.88
% python estimate.py 20 .65 100
0.89
% python estimate.py 40 .55 1000
0.094
```

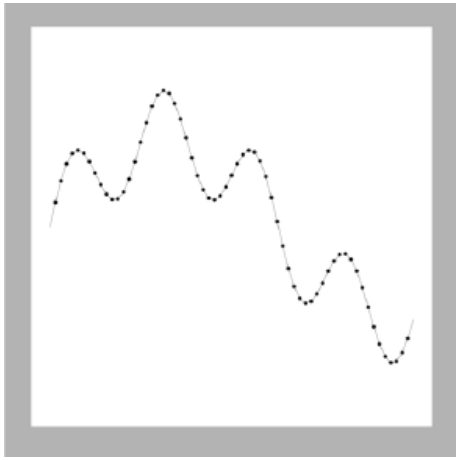
- زمان اجرا. هنوز متناسب با n^2 * trials. ← *حجم محاسبات بسیار زیاد!*
- مصرف حافظه. هنوز متناسب با n^2.

ترسیم تطبیقی

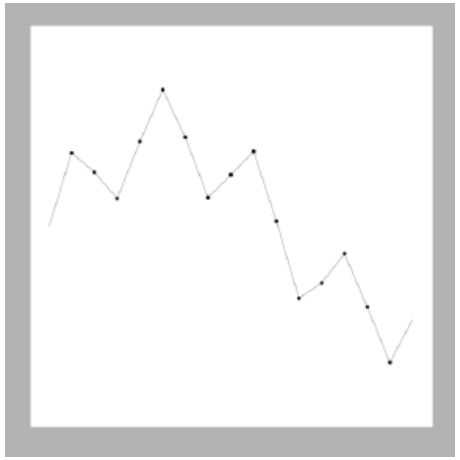
- ترسیم نتایج. ترسیم احتمال تراوش یک سیستم به عنوان تابعی از احتمال باز بودن خانه‌ها.
- تصمیمات طراحی.
  - چند مقدار p؟
  - کدام مقادیر p؟
  - چند آزمایش به ازای هر مقدار p؟



تعداد نقاط مناسب



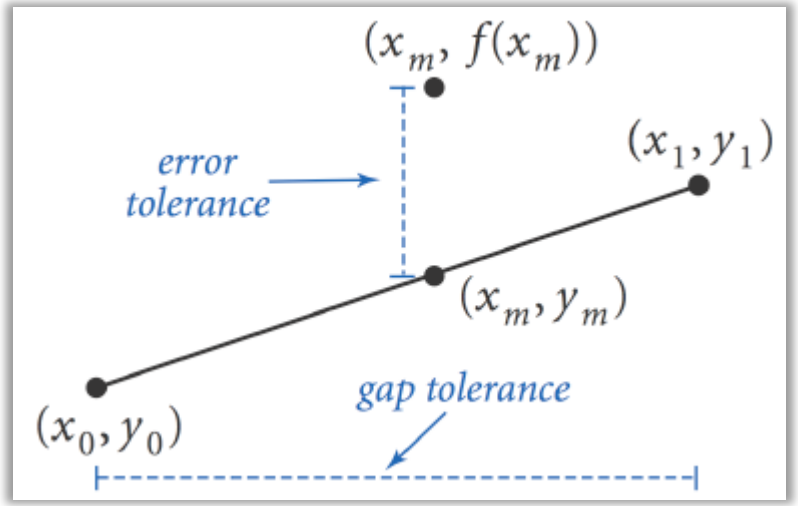
تعداد نقاط بسیار زیاد



تعداد نقاط بسیار کم

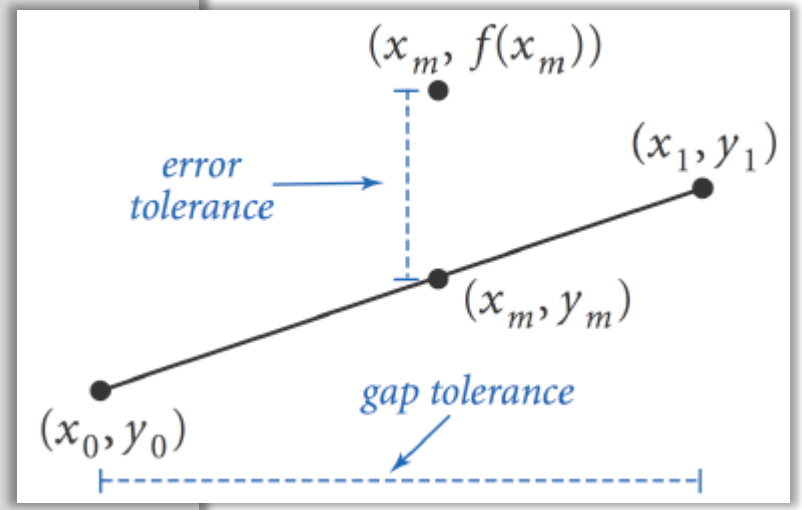
ترسیم تطبیقی برای مسئله تراوش

- 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

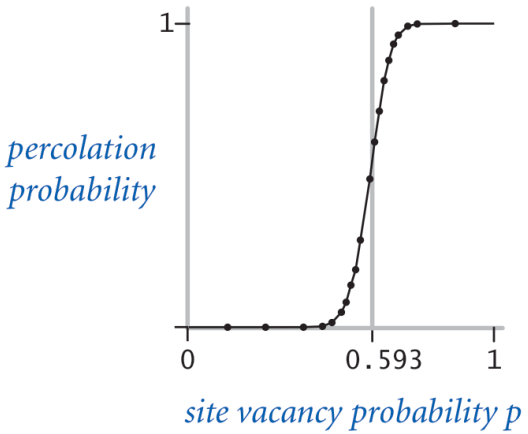

```
def curve(n, x0, y0, x1, y1, trials=10000, gap=.01, err=.0025):
    xm = (x0 + x1) / 2.0
    ym = (y0 + y1) / 2.0
    fxm = estimate.evaluate(n, xm, trials);
    if (x1 - x0 < gap) or (abs(ym - fxm) < err):
        stddraw.line(x0, y0, x1, y1)
        stddraw.show(0.0)
        return
    curve(n,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

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



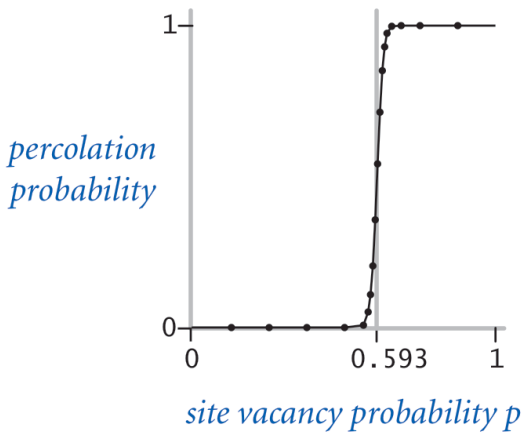
ترسیم تطبیقی

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

```
% python percplot.py 20
```

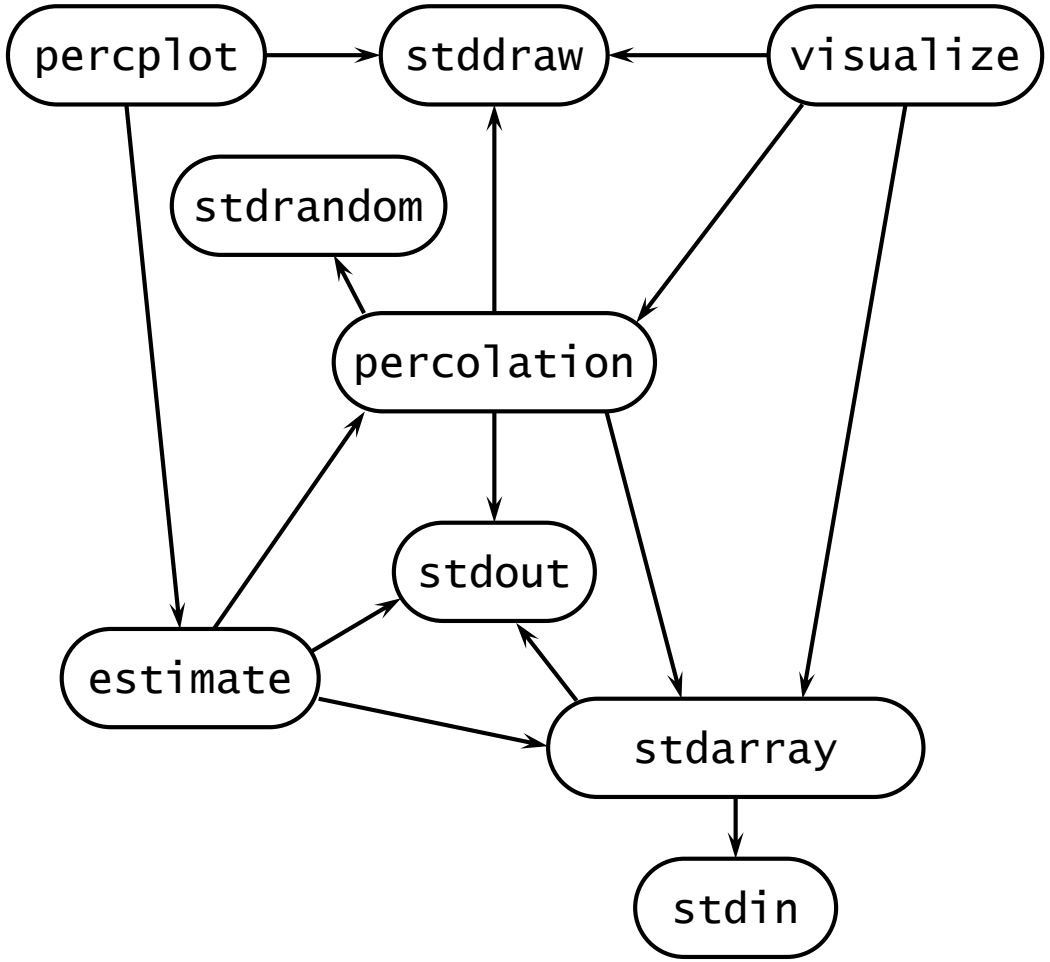



```
% python percplot.py 50
```



- 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 اشکال‌زدایی
- از روش توسعه تدریجی استفاده کنید.
  - هر ماجولی که می‌نویسید آن را اجرا و اشکال‌زدایی 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.