

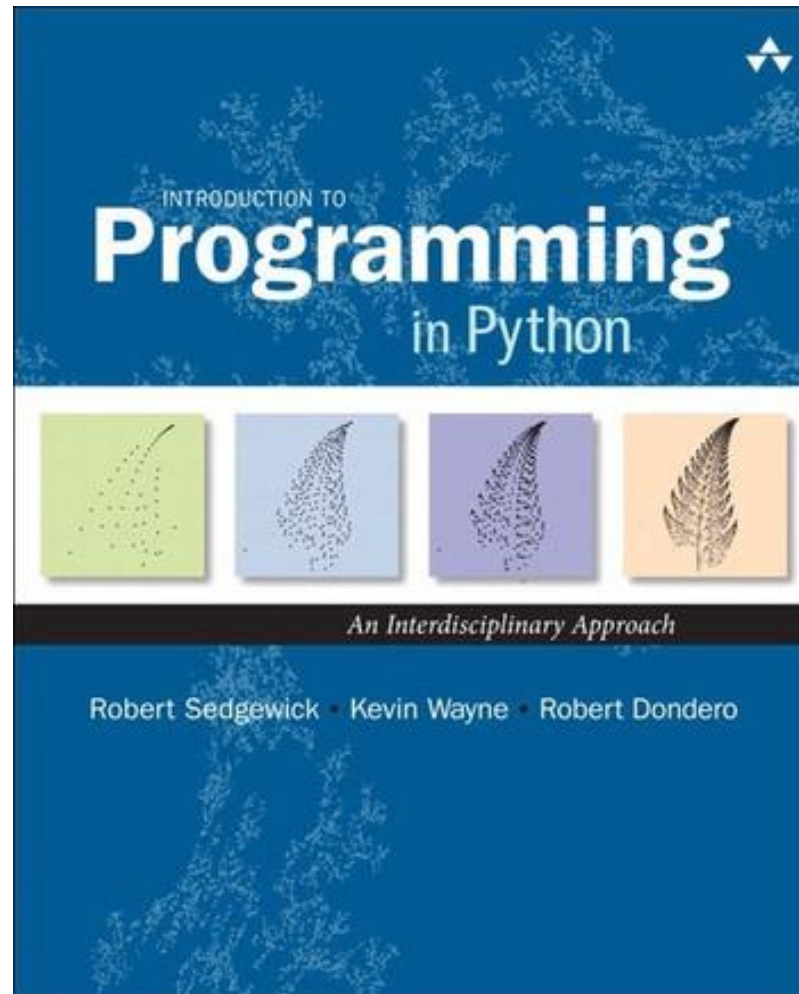
توابع و کتابخانه‌ها: توابع بازگشتی

سید ناصر رضوی www.snrazavi.ir

۱۳۹۸

۲-۳ توابع بازگشتی

۲



توابع بازگشتی

۳

□ تابع بازگشتی. تابعی که خودش را به صورت مستقیم یا غیرمستقیم فراخوانی می کند.



□ مزایای یادگیری توابع بازگشتی.

□ آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)

□ آشنایی با یک الگوی قدرتمند برنامه نویسی



□ رابطه نزدیک با استقرار ریاضی.

فاکتوریل

۴

□ محاسبه فاکتوریل n .

$$n! = 1 * 2 * \dots * (n - 1) * n$$

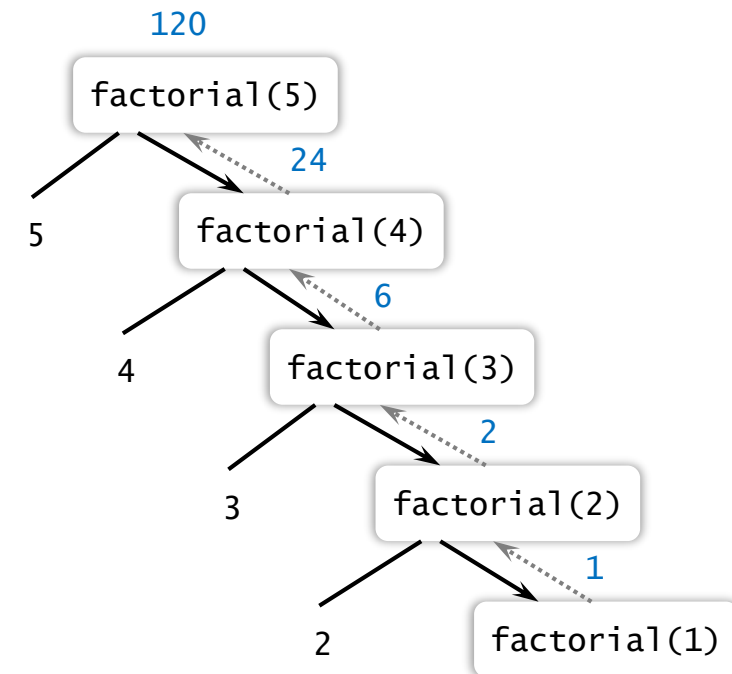
$$n! = \begin{cases} 1 & n = 1 \\ n \times (n - 1)! & n > 1 \end{cases}$$

```
def factorial(n):
```

```
    if n == 1: return 1 ← فتم بازگشت
```

```
    return n * factorial(n - 1)
```

↑
فراخوانی بازگشتی



فاکتوریل

۵

□ محاسبه فاکتوریل n .

$$n! = 1 * 2 * \dots * (n - 1) * n$$

$$n! = \begin{cases} 1 & n = 1 \\ n \times (n - 1)! & n > 1 \end{cases}$$

```
def factorial(n):  
    if n == 1: return 1  
    return n * factorial(n - 1)
```

فراخوانی بازگشتی

```
factorial(5)  
    factorial(4)  
        factorial(3)  
            factorial(2)  
                factorial(1)  
                    return 1  
                return 2*1 = 2  
            return 3*2 = 6  
        return 4*6 = 24  
    return 5*24 = 120
```

بزرگ‌ترین مقسوم علیه مشترک

۶

□ ب.م.م. یافتن بزرگ‌ترین عدد صحیحی که p و q بر آن بخش پذیرند.

□ مثال.

$$\gcd(4032, 1272) = 24.$$

$$4032 = 2^6 * 3^2 * 7^1$$

$$1272 = 2^3 * 3^1 * 53^1$$

$$\gcd = 2^3 * 3^1$$

□ کاربردها.

□ ساده‌سازی اعداد کسری

□ رمزنگاری RSA

بزرگ‌ترین مقسوم علیه مشترک

۷

□ الگوریتم اقلیدس. [۳۰۰ سال قبل از میلاد]

$$\gcd(p, q) = \begin{cases} p & q = 0 \\ \gcd(q, p \% q) & \text{otherwise} \end{cases}$$

$$\begin{aligned} \gcd(1440, 408) &= \gcd(408, 216) \\ &= \gcd(216, 192) \\ &= \gcd(192, 24) \\ &= \gcd(24, 0) \\ &= 24 \end{aligned}$$

$$1440 = 3 * 408 + 216$$


بزرگ‌ترین مقسوم علیه مشترک

۸

□ الگوریتم اقلیدس. [۳۰۰ سال قبل از میلاد]

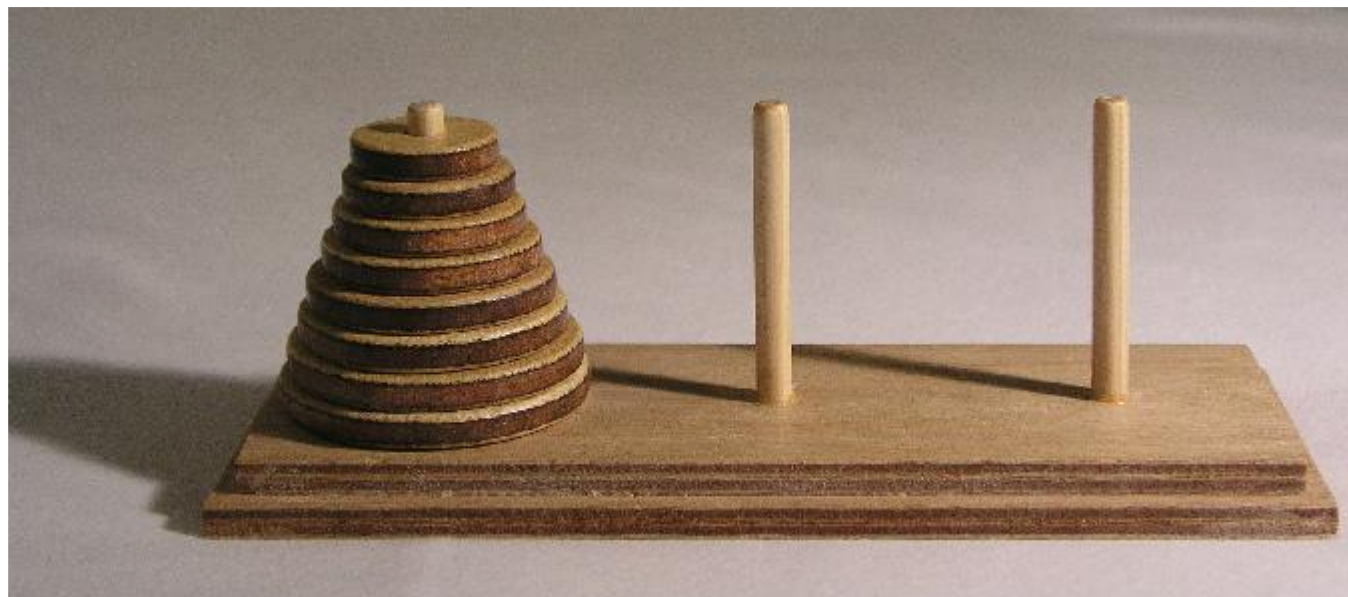
$$\text{gcd}(p, q) = \begin{cases} p & q = 0 \\ \text{gcd}(q, p \% q) & \text{otherwise} \end{cases}$$

```
def gcd(p, q):  
    if q == 0: return p  
    return gcd(q, p % q)
```

```
gcd(1440, 408)  
    gcd(408, 216)  
        gcd(216, 192)  
            gcd(192, 24)  
                gcd(24, 0)  
                    return 24  
                return 24  
            return 24  
        return 24  
    return 24
```

برج‌های هانوی

۹



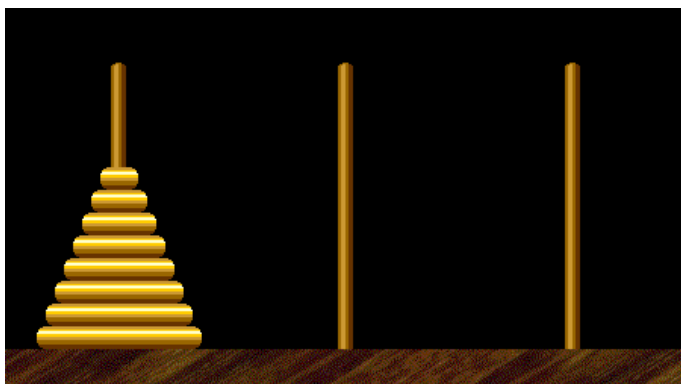
<http://en.wikipedia.org/wiki/Image:Hanoikle.in.jpg>

برج‌های هانوی

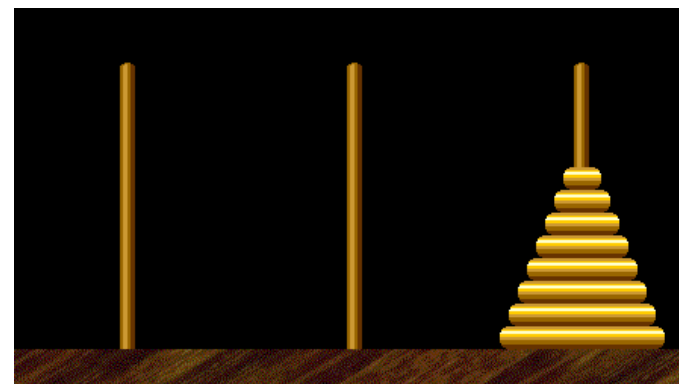
۱۰

- هدف. انتقال دیسک‌ها از میله سمت چپ به میله سمت راست با حفظ ترتیب.
- در هر حرکت تنها مجاز به انتقال یک دیسک هستیم.
- هیچ‌گاه مجاز نیستیم یک دیسک بزرگ‌تر را بر روی یک دیسک کوچک‌تر قرار دهیم.

شروع



پایان



افسانه برچه‌های هانوی

۱۱

□ س. دنیا چه زمانی به پایان می‌رسد؟

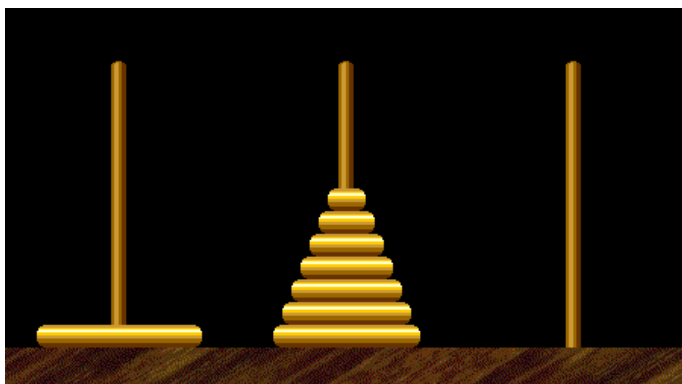
□ ۶۴ دیسک طلا بر روی سه میله الماس در ابتدای آفرینش.

□ با اتمام کار یک گروه خاص از کشیش‌ها که مسؤول انتقال این دیسک‌ها هستند، دنیا به پایان می‌رسد.

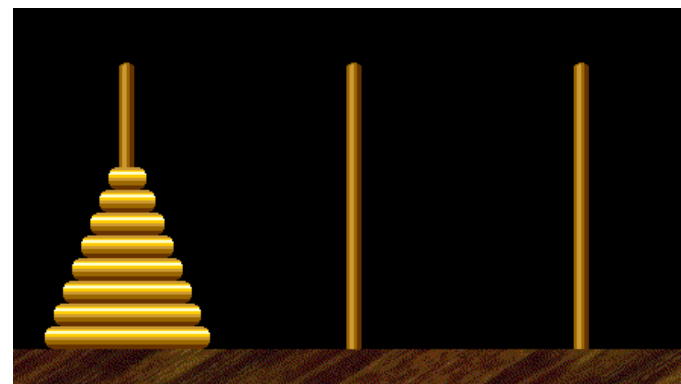
□ س. آیا می‌توان از کامپیوتر برای پیش‌بینی زمان پایان دنیا استفاده کرد؟

برج‌های هانوی: راه‌حل بازگشتی

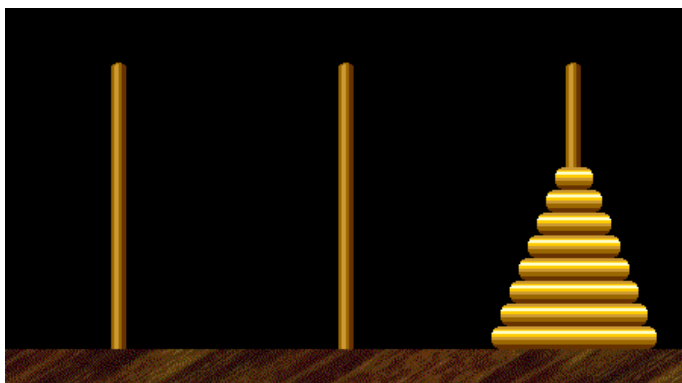
۱۲



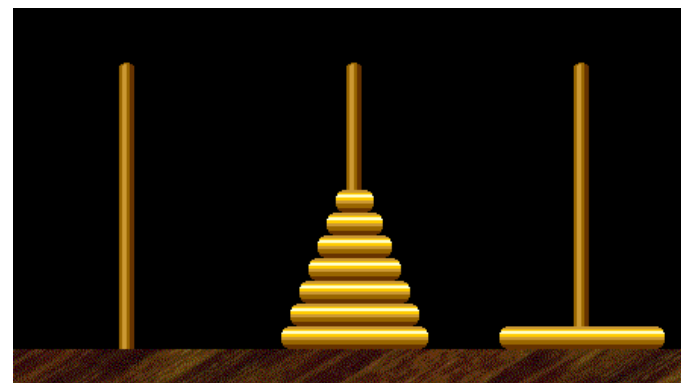
(۱) انتقال $n - 1$ دیسک به میله وسط



حالت شروع



(۳) انتقال $n - 1$ دیسک به میله راست



(۲) انتقال بزرگترین دیسک به میله راست

برچ‌های هانوی: راه‌حل بازگشتی

۱۳

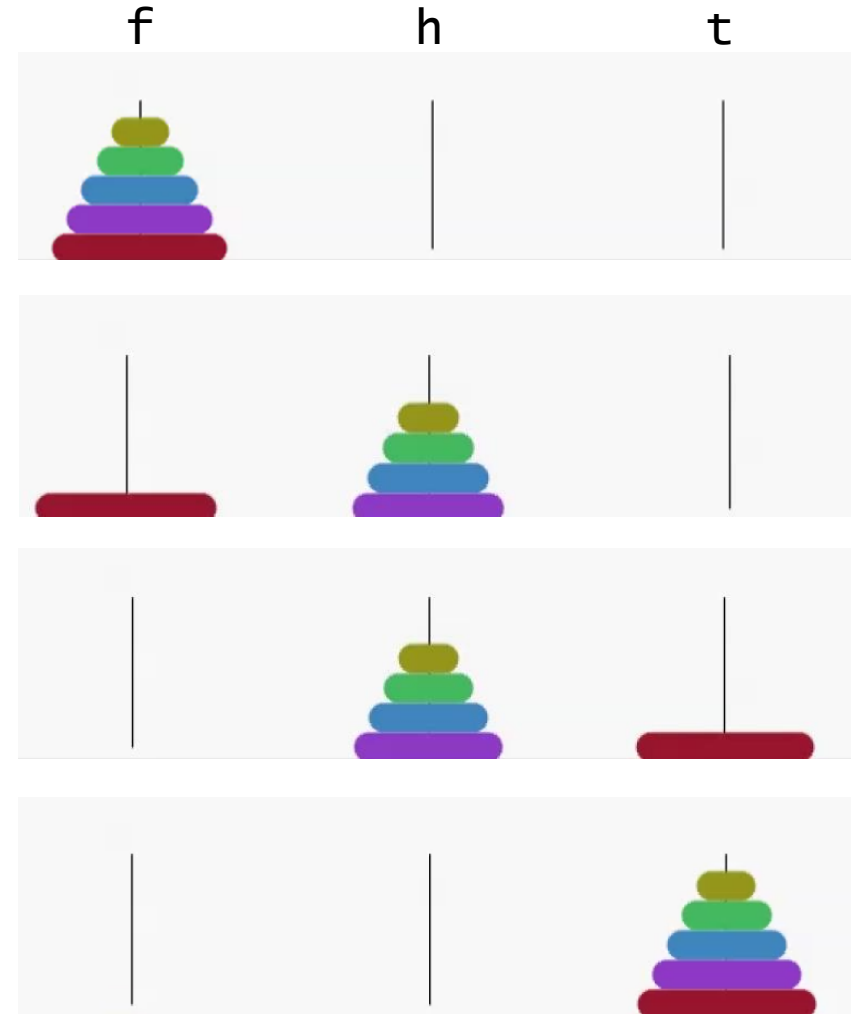
```
import sys

def moves(n, f, t, h):
    if n == 0:
        return

    moves(n - 1, f, h, t)
    print(str(f) + ' --> ' + str(t))
    moves(n - 1, h, t, f)

def main():
    n = int(sys.argv[1])
    moves(n, 1, 3, 2)

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



برج‌های هانوی: راه‌حل بازگشتی

۱۴

```
import sys

def moves(n, f, t, h):
    if n == 0:
        return

    moves(n - 1, f, h, t)
    print(str(f) + ' --> ' + str(t))
    moves(n - 1, h, t, f)

def main():
    n = int(sys.argv[1])
    moves(n, 1, 3, 2)

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

```
% python towersofhanoi.py 1
1 --> 3
```

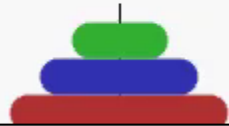
```
% python towersofhanoi.py 2
1 --> 2
1 --> 3
2 --> 3
```

```
% python towersofhanoi.py 3
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```

برج‌های هانوی: اجرای نمایشی

۱۵

$n = 3$



$n = 4$

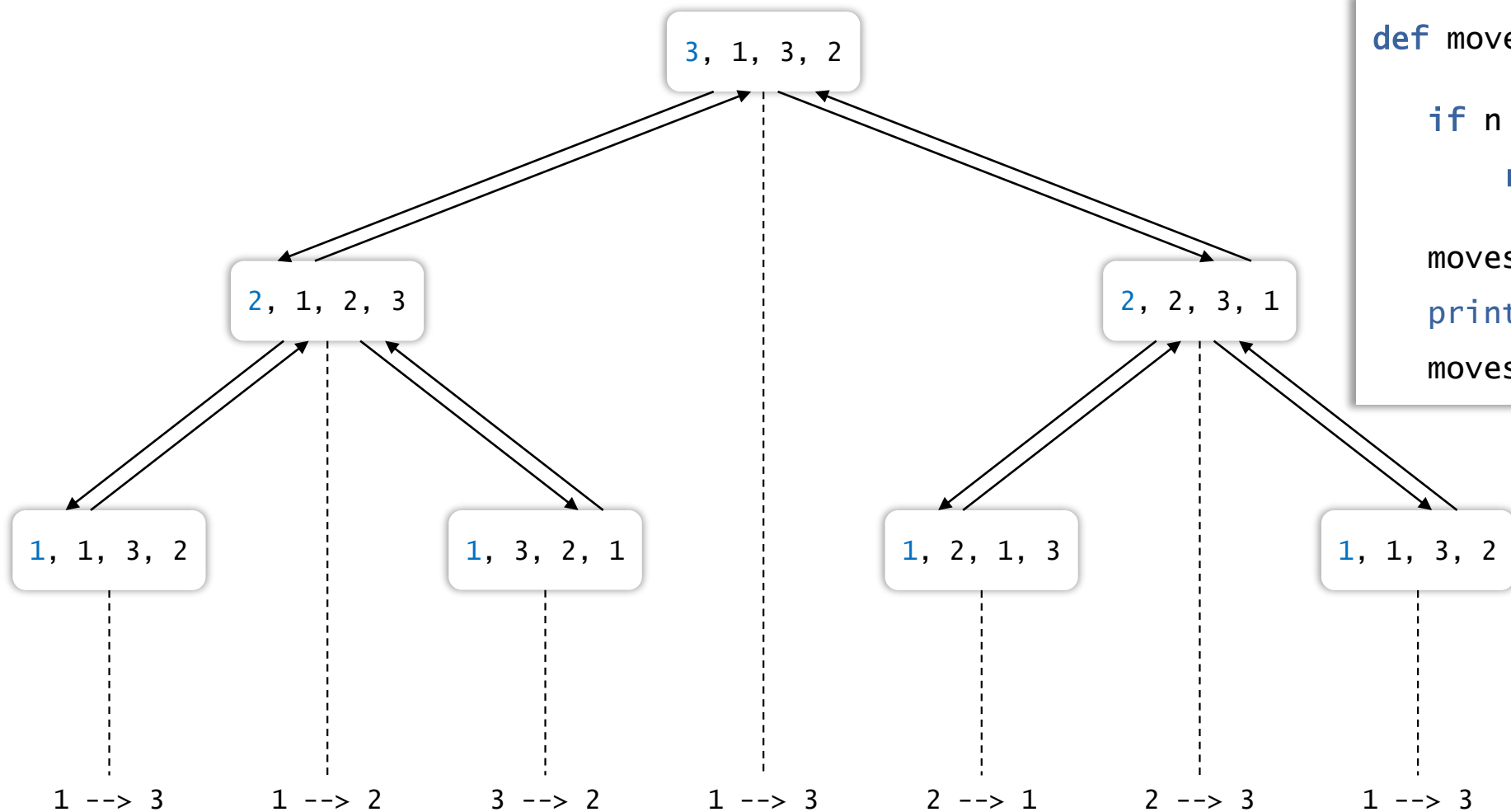


$n = 5$



برج‌های هانوی: راه‌حل بازگشتی

۱۶



```
def moves(n, f, t, h):
```

```
    if n == 0:
```

```
        return
```

```
    moves(n - 1, f, h, t)
```

```
    print(str(f) + ' --> ' + str(t))
```

```
    moves(n - 1, h, t, f)
```

برج‌های هانوی: ویژگی‌های راه‌حل

۱۷

□ تعداد جابه‌جایی‌های لازم برای انتقال n دیسک؟

□ برای یک مسئله با n دیسک به $2^n - 1$ جابه‌جایی نیاز است !!!

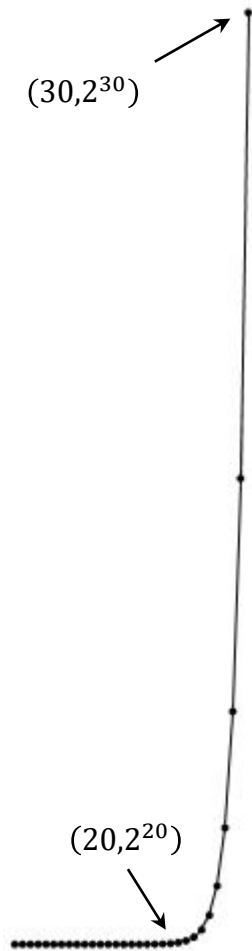
$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(n-1) + 1 & n > 1 \end{cases}$$

□ پس بالاخره سرنوشت دنیا چه می‌شود؟

□ با این فرض که هر جابه‌جایی تنها به یک ثانیه زمان نیاز داشته باشد، دنیا ۵۸۵ میلیارد سال پس از خلقت آن به پایان می‌رسد.

□ هنوز بیش از ۵۷۰ میلیارد سال باقیمانده است.

□ برای این که خیالتان راحت شود، این راه‌حل سریع‌ترین راه‌حل است.



کد گری

۱۸

□ **کد گری**. کدگذاری اعداد 0 تا $2^n - 1$ با دنباله‌ای از n بیت به گونه‌ای که هر عدد دقیقاً در یک بیت با عدد قبل از خود تفاوت دارد.

کد گری ۱ بیتی	کد گری ۲ بیتی	کد گری ۳ بیتی
0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

0	0	← کد گری ۱ بیتی
0	1	
1	1	← کد گری ۱ بیتی
1	0	(معکوس شده)

0	00	
0	01	← کد گری ۲ بیتی
0	11	
0	10	
1	10	
1	11	← کد گری ۲ بیتی
1	01	(معکوس شده)
1	00	

```
def graycode(n):
    if n == 0: return ['']
    code1 = graycode(n - 1)
    code2 = code1[::-1]

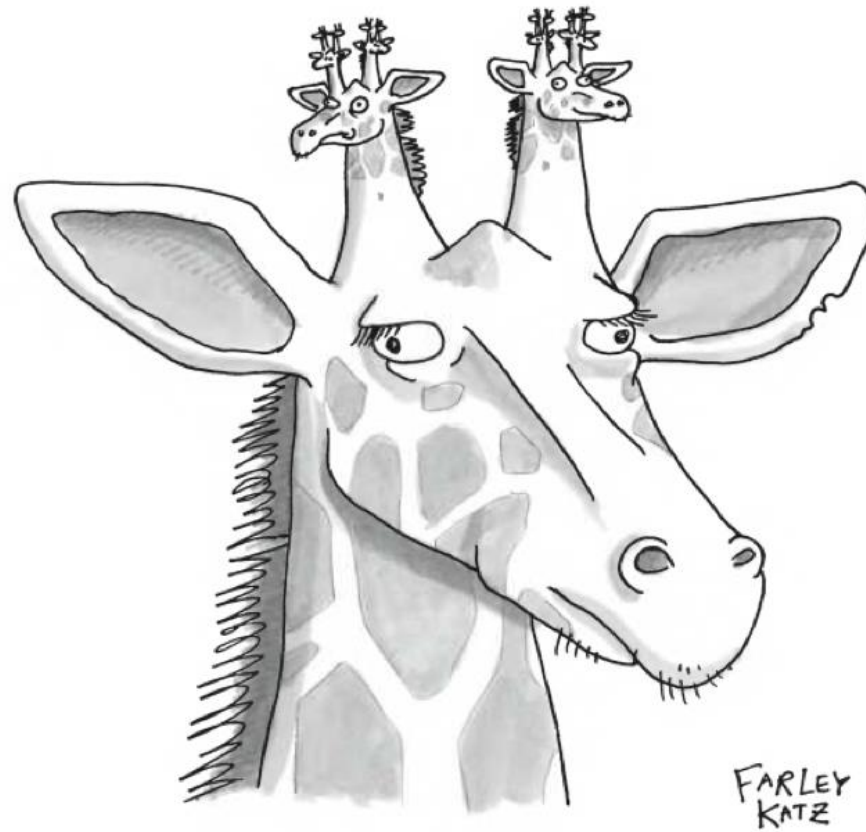
    for i in range(len(code1)):
        code1[i] = '0' + code1[i]
    for i in range(len(code2)):
        code2[i] = '1' + code2[i]

    return code1 + code2
```

0	00	
0	01	
0	11	← کد گری ۲ بیتی
0	10	
1	10	
1	11	← کد گری ۲ بیتی
1	01	(معکوس شده)
1	00	

ترسیم بازگشتی

۲۰



درخت H

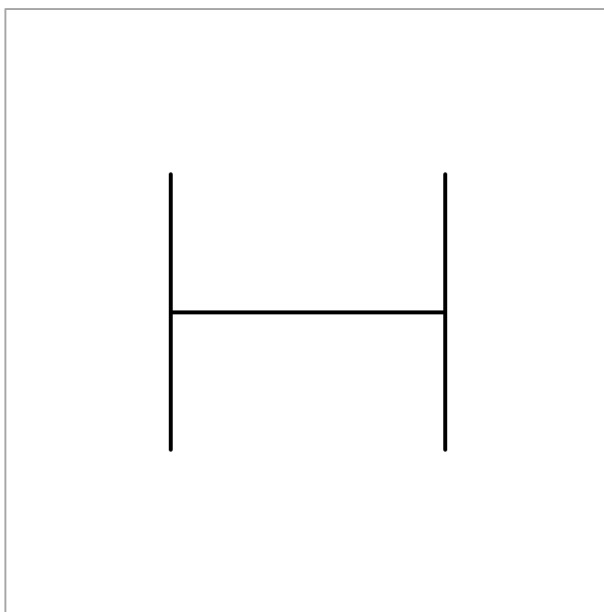
۲۱

□ درخت H از مرتبه n .

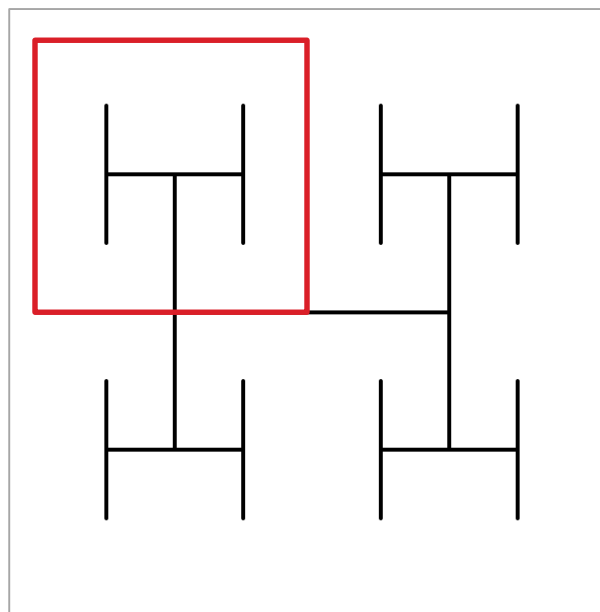
□ یک H ترسیم کنید.

□ به طور بازگشتی در هر یک از چهار نقطه انتهایی، یک درخت H از مرتبه $n - 1$ با اندازه نصف ترسیم کنید.

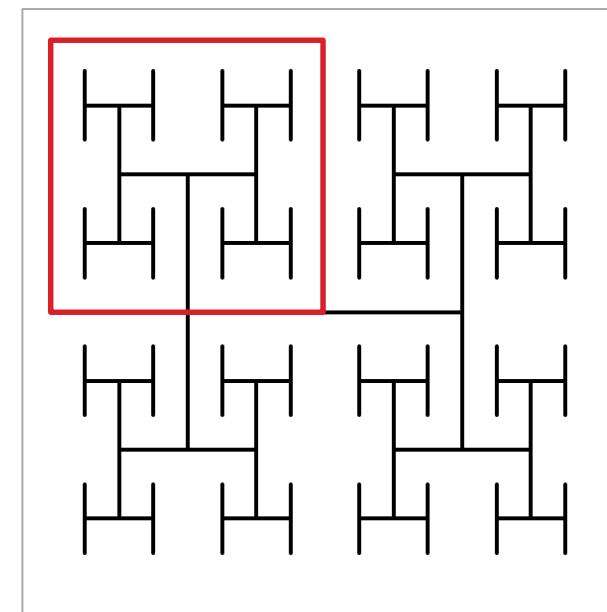
مرتبه ۱



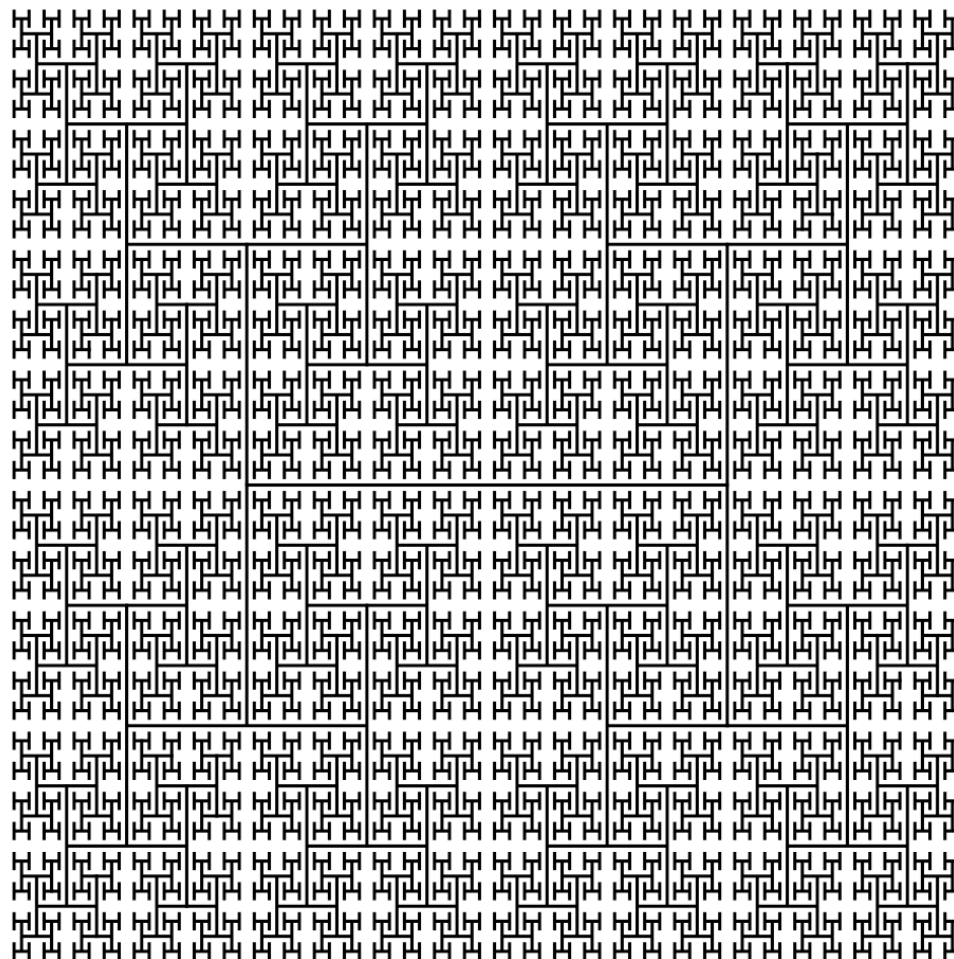
مرتبه ۲



مرتبه ۳



مرتبه ۴



کاربرد. یک مجموعه
بزرگ از مکان‌های دارای
فواصل منظم را به یک
منبع یکسان متصل کن.

ترسیم بازگشتی: درخت H

۲۳

```
def draw(n, sz, x, y):
    if n == 0: return

    x0, x1 = x - sz/2, x + sz/2
    y0, y1 = y - sz/2, y + sz/2

    stddraw.line(x0, y, x1, y)
    stddraw.line(x0, y0, x0, y1)
    stddraw.line(x1, y0, x1, y1)

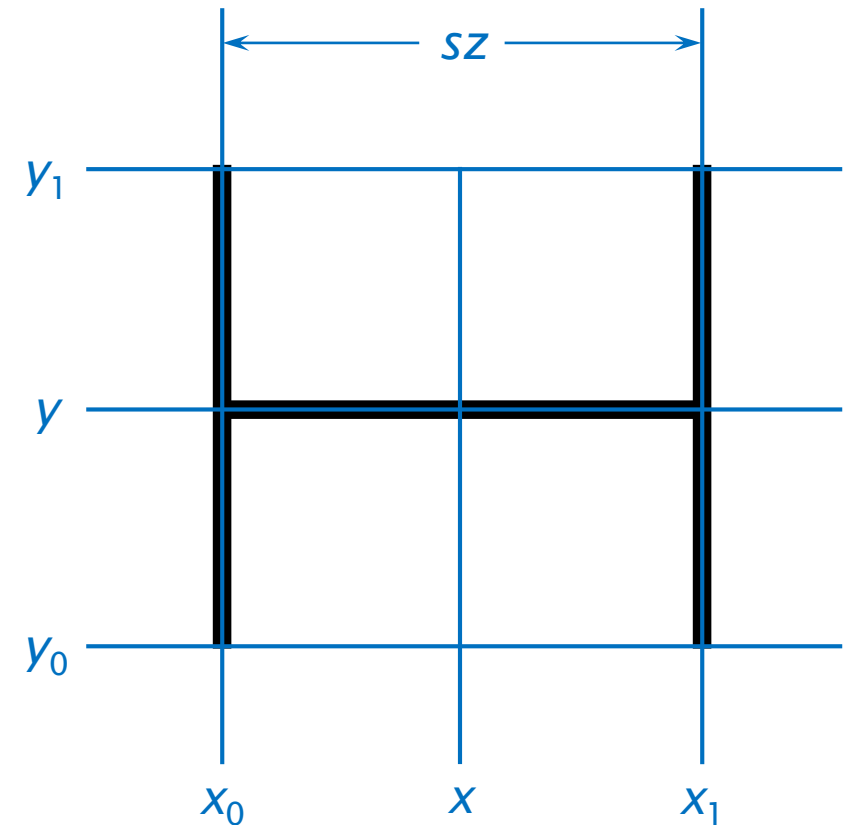
    draw(n - 1, sz/2, x0, y0)
    draw(n - 1, sz/2, x0, y1)
    draw(n - 1, sz/2, x1, y0)
    draw(n - 1, sz/2, x1, y1)

def main():
    n = int(sys.argv[1])
    stddraw.setPenRadius(0.0)
    draw(n, 0.5, 0.5, 0.5)
    stddraw.show()

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

ترسیم H، به مرکز (x, y)

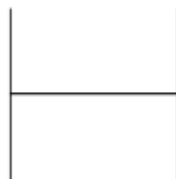
ترسیم چهار درخت H با اندازه نصف به صورت بازگشتی



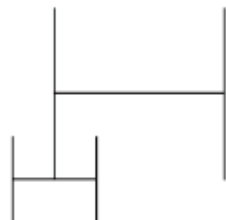
درخت H

۲۴

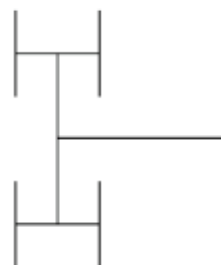
□ ترسیم درخت H به صورت انیمیشن. پس از رسم هر H کمی مکث کن.



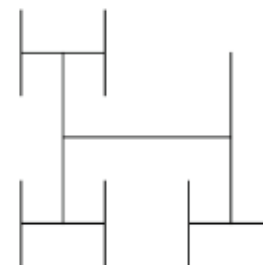
20%



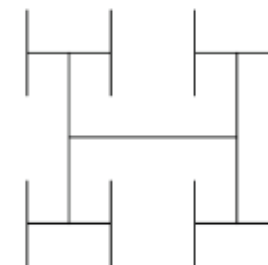
40%



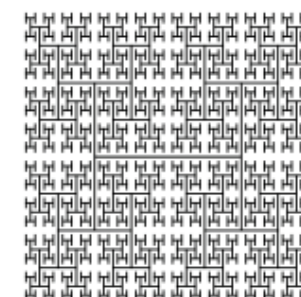
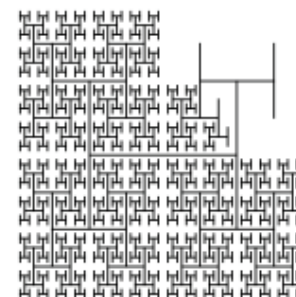
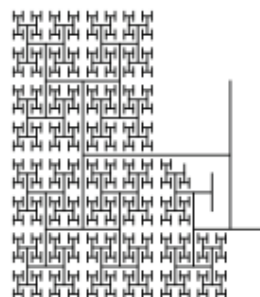
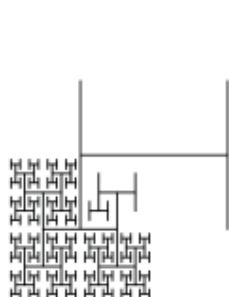
60%



80%



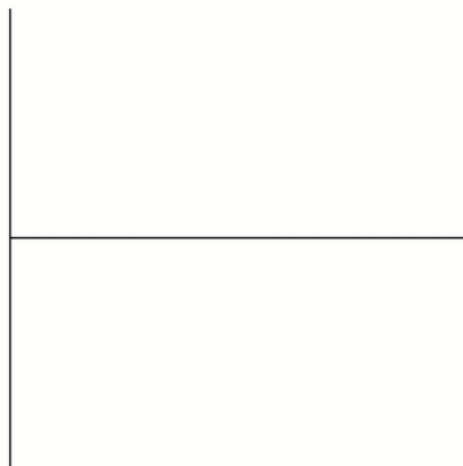
100%



درخت H

۲۵

□ ترسیم درخت H به صورت انیمیشن. پس از رسم هر H کمی مکث کن.



پل براونی

۲۶

□ پل براونی. فرایندی که می‌تواند بسیاری از پدیده‌ها را مدل‌سازی کند.

□ قیمت سهام

□ انتشار سیالات

□ سطوح ناهموار و دنداندار کوهستان‌ها و ابرها

□ ...

مدل پل براونی



یک کوه واقعی



قیمت یک سهام واقعی



□ روش جابه‌جایی نقطه میانی.

□ یک پاره‌خط از (x_0, y_0) به (x_1, y_1) در نظر بگیرید.

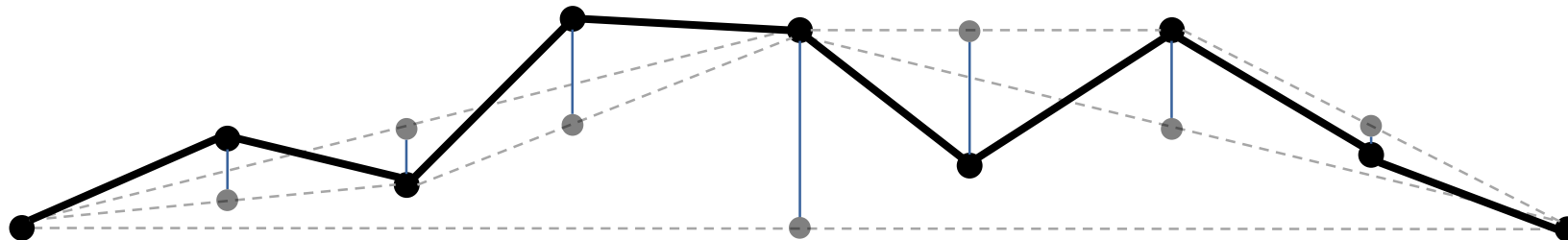
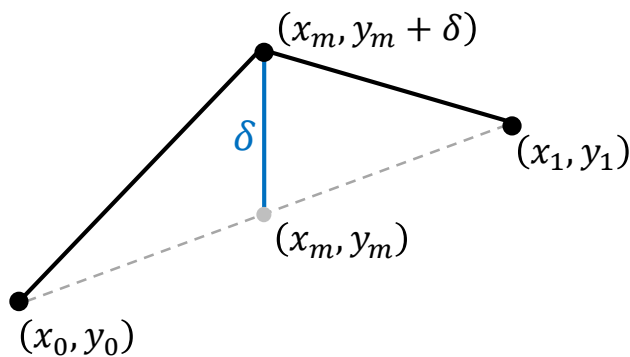
□ اگر پاره‌خطی به اندازه کافی کوتاه است، آن را ترسیم کن و برگرد.

□ پاره‌خط را از وسط به دو نیم تقسیم کن، در نقطه (x_m, y_m) .

□ مقدار δ را به صورت تصادفی از یک توزیع گاوسی انتخاب کن.

□ مقدار δ را به y_m اضافه کن.

□ مراحل بالا را به صورت بازگشتی بر روی دو پاره‌خط ایجاد شده تکرار کن.



پل براونی: پیاده سازی

۲۸

```
def curve(x0, y0, x1, y1, variance, scale_factor):
```

```
    # stop if interval is sufficiently small
```

```
    if x1 - x0 < .01:
```

```
        stddraw.line(x0, y0, x1, y1)
```

```
        return
```

```
    xm = (x0 + x1) / 2.0
```

```
    ym = (y0 + y1) / 2.0
```

```
    delta = stdrandom.gaussian(0, math.sqrt(variance))
```

```
    curve(x0, y0, xm, ym + delta, variance / scale_factor, scale_factor)
```

```
    curve(xm, ym + delta, x1, y1, variance / scale_factor, scale_factor)
```

```
def main():
```

```
    h = float(sys.argv[1])
```

```
    scale_factor = 2 ** (2.0 * h)
```

```
    stddraw.setPenRadius(0.0)
```

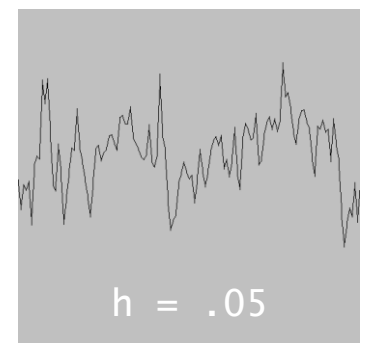
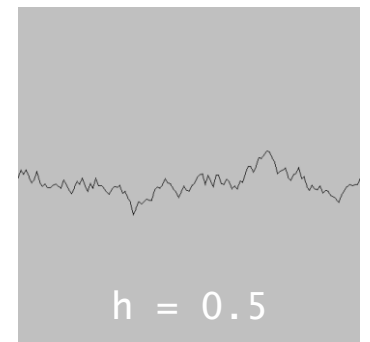
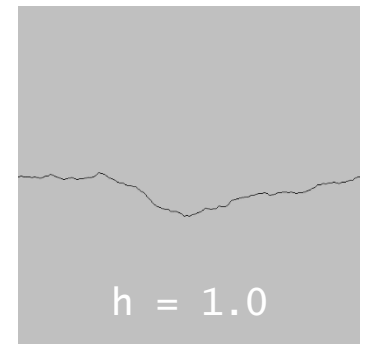
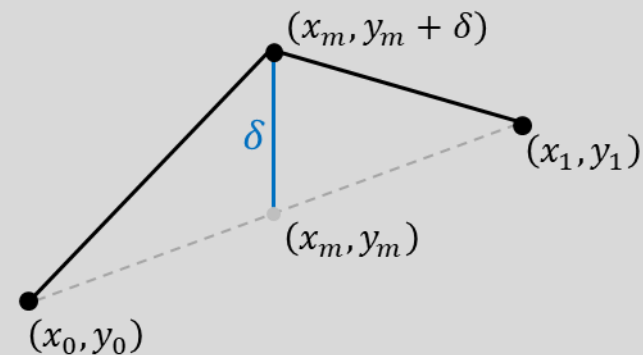
```
    stddraw.clear(stddraw.LIGHT_GRAY)
```

```
    curve(0.0, 0.5, 1.0, 0.5, .01, scale_factor)
```

```
    stddraw.show()
```

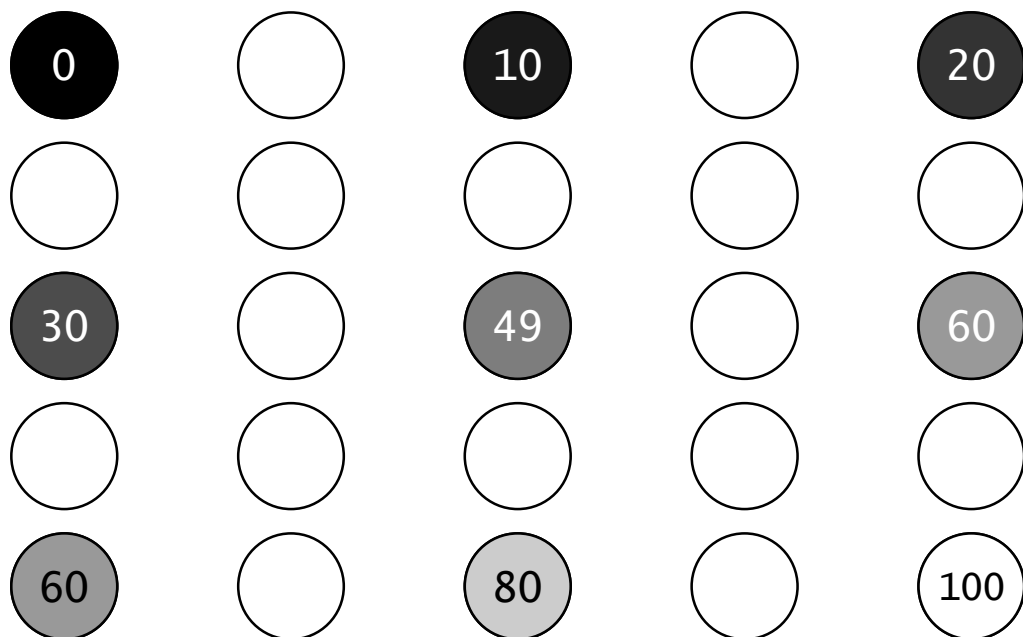
```
if __name__ == '__main__':
```

```
    main()
```



مدل براونی دو بعدی: ابرهای پلاسمایی

۲۹



□ روش جابه‌جایی نقطه میانی.

□ یک مستطیل به مرکز (x, y) در نظر بگیرید که پیکسل‌ها در چهار گوشه آن دارای مقدار هستند.

□ اگر مستطیل کوچک است، هیچ کاری انجام نده.

□ نقطه میانی هر ضلع مستطیل را با میانگین‌گیری از دو رأس مربوطه رنگ‌آمیزی کن.

□ مقدار δ را به صورت تصادفی از «توزیع گاوسی» انتخاب کن.

□ رنگ پیکسل مرکزی را برابر با میانگین رنگ چهار رأس به علاوه مقدار δ قرار بده.

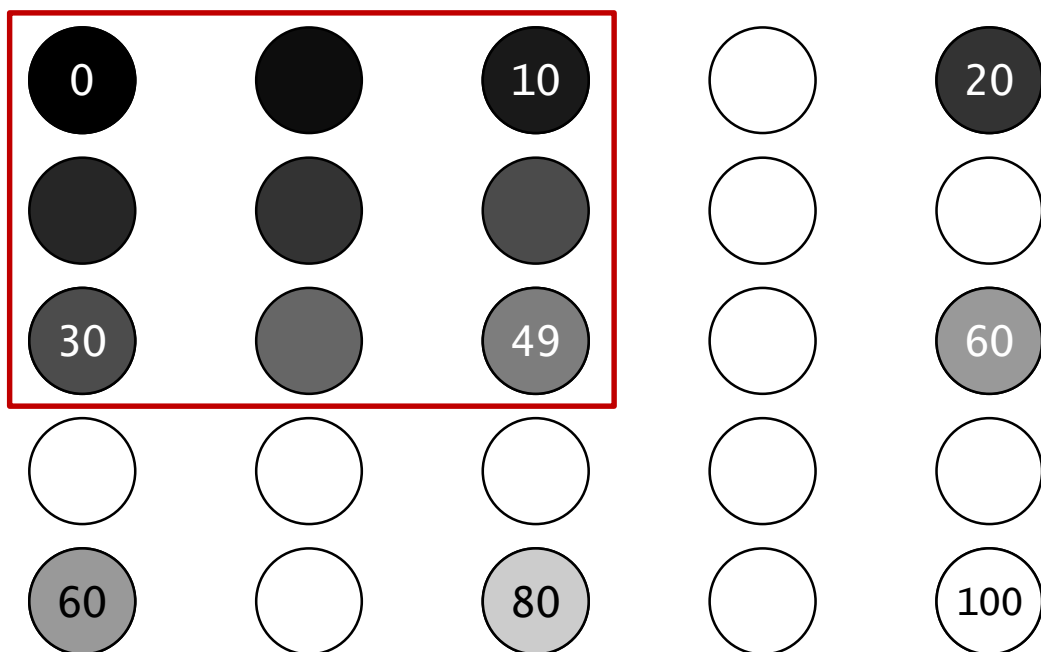
□ مراحل بالا را بر روی هر یک از چهار مستطیل کوچک‌تر تکرار کن.

مدل براونی دو بعدی: ابرهای پلاسمایی

۳۰

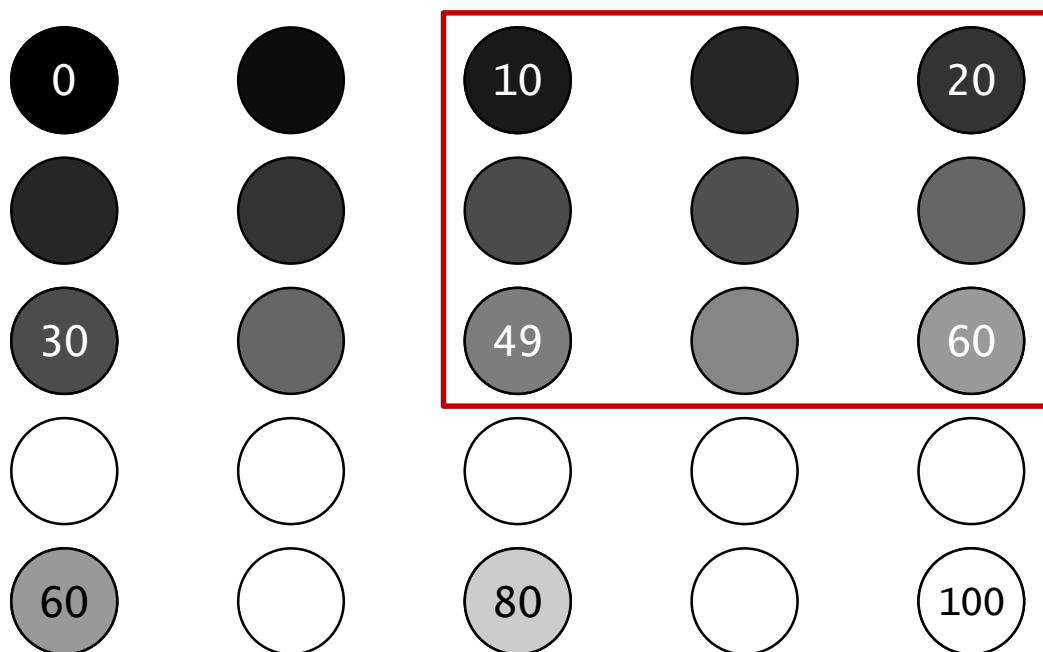
□ روش جابه‌جایی نقطه میانی.

- یک مستطیل به مرکز (x, y) در نظر بگیرید که پیکسل‌ها در چهار گوشه آن دارای مقدار هستند.
- اگر مستطیل کوچک است، هیچ کاری انجام نده.
- نقطه میانی هر ضلع مستطیل را با میانگین‌گیری از دو رأس مربوطه رنگ‌آمیزی کن.
- مقدار δ را به صورت تصادفی از «توزیع گاوسی» انتخاب کن.
- رنگ پیکسل مرکزی را برابر با میانگین رنگ چهار رأس به علاوه مقدار δ قرار بده.
- مراحل بالا را بر روی هر یک از چهار مستطیل کوچک‌تر تکرار کن.



مدل براونی دو بعدی: ابرهای پلاسمایی

۳۱



□ روش جابه‌جایی نقطه میانی.

□ یک مستطیل به مرکز (x, y) در نظر بگیرید که پیکسل‌ها در چهار گوشه آن دارای مقدار هستند.

□ اگر مستطیل کوچک است، هیچ کاری انجام نده.

□ نقطه میانی هر ضلع مستطیل را با میانگین‌گیری از دو رأس مربوطه رنگ‌آمیزی کن.

□ مقدار δ را به صورت تصادفی از «توزیع گاوسی» انتخاب کن.

□ رنگ پیکسل مرکزی را برابر با میانگین رنگ چهار رأس به علاوه مقدار δ قرار بده.

□ مراحل بالا را بر روی هر یک از چهار مستطیل کوچک‌تر تکرار کن.

مدل براونی دو بعدی: ابرهای پلاسمایی

۳۲

□ روش جابه‌جایی نقطه میانی.

□ یک مستطیل به مرکز (x, y) در نظر بگیرید که پیکسل‌ها در چهار گوشه آن دارای مقدار هستند.

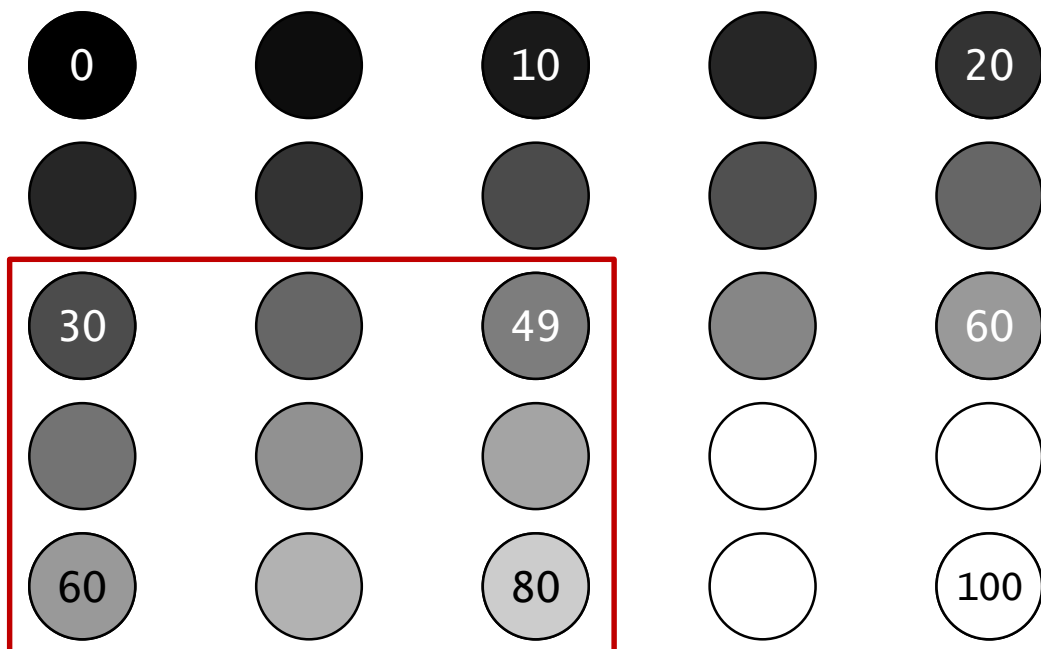
□ اگر مستطیل کوچک است، هیچ کاری انجام نده.

□ نقطه میانی هر ضلع مستطیل را با میانگین‌گیری از دو رأس مربوطه رنگ‌آمیزی کن.

□ مقدار δ را به صورت تصادفی از «توزیع گاوسی» انتخاب کن.

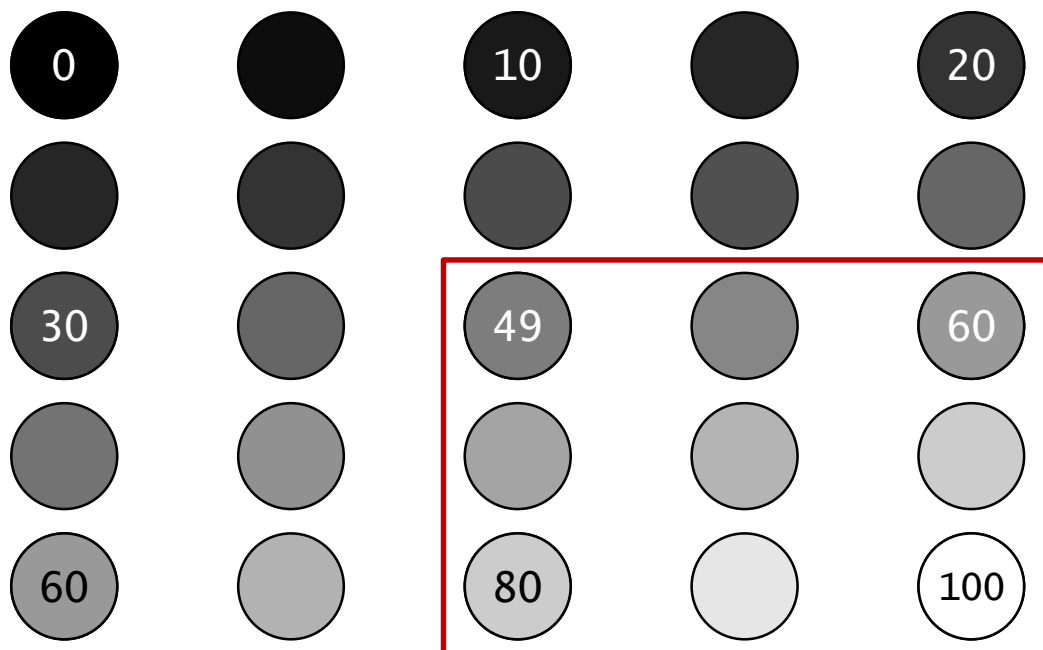
□ رنگ پیکسل مرکزی را برابر با میانگین رنگ چهار رأس به علاوه مقدار δ قرار بده.

□ مراحل بالا را بر روی هر یک از چهار مستطیل کوچک‌تر تکرار کن.



مدل براونی دو بعدی: ابرهای پلاسمایی

۳۳



□ روش جابه‌جایی نقطه میانی.

□ یک مستطیل به مرکز (x, y) در نظر بگیرید که پیکسل‌ها در چهار گوشه آن دارای مقدار هستند.

□ اگر مستطیل کوچک است، هیچ کاری انجام نده.

□ نقطه میانی هر ضلع مستطیل را با میانگین‌گیری از دو رأس مربوطه رنگ‌آمیزی کن.

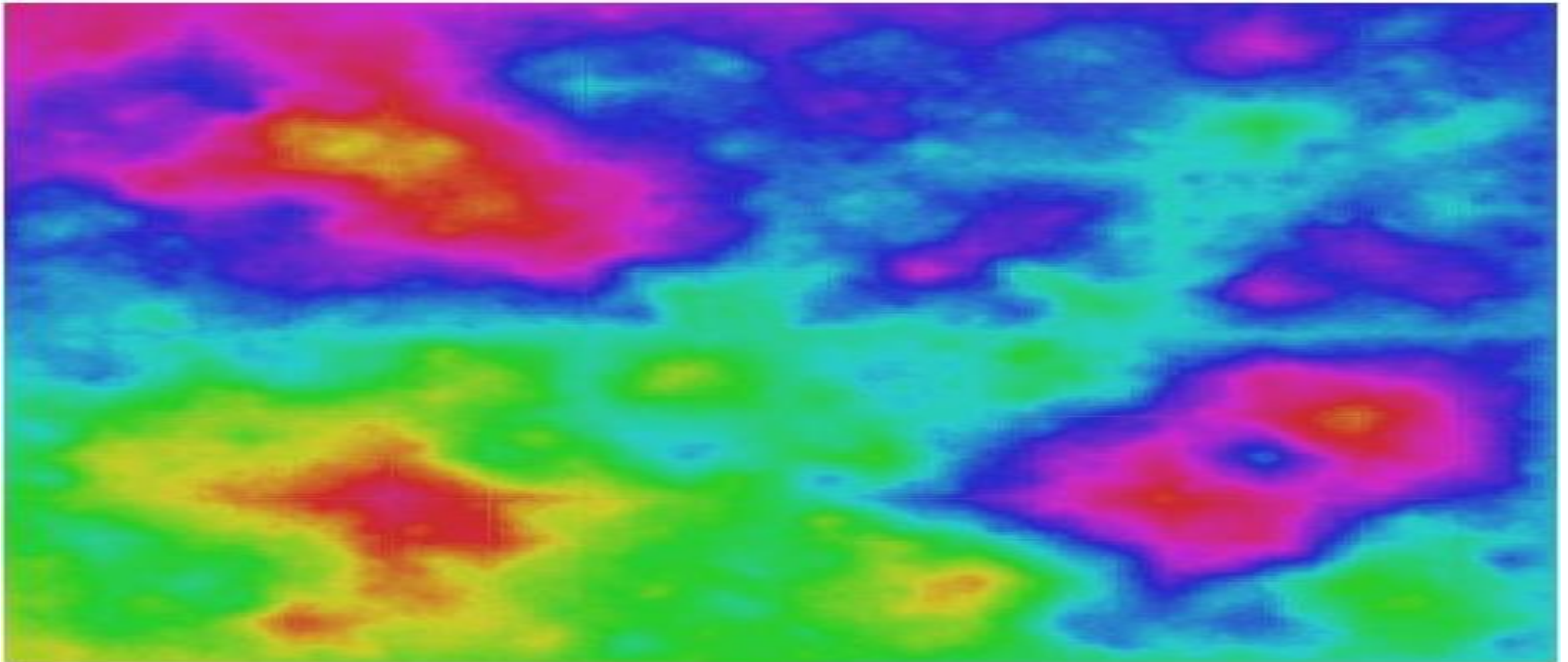
□ مقدار δ را به صورت تصادفی از «توزیع گاوسی» انتخاب کن.

□ رنگ پیکسل مرکزی را برابر با میانگین رنگ چهار رأس به علاوه مقدار δ قرار بده.

□ مراحل بالا را بر روی هر یک از چهار مستطیل کوچک‌تر تکرار کن.

ابر پلاسمایی

۳۴



ابر پلاسمایی

۳۵



یک منظره براونی

۳۶



برنامه‌نویسی با پایتون - توابع بازگشتی - سید ناصر رضوی - ۱۳۹۸

توابع بازگشتی: خطاهای متداول (۱)

۳۷

□ فراموش کردن حالت پایه (ختم بازگشت).

□ فراموش کردن حالت پایه، منجر به گیر افتادن در یک **حلقه بی‌نهایت** می‌شود.

```
def H(n):  
    return H(n-1) + 1.0/n
```



□ عدم تضمین همگرایی.

□ استفاده از فراخوانی‌های بازگشتی برای حل زیرمسائلی که **کوچک‌تر نیستند**.

```
def H(n):  
    if n == 1: return 1.0  
    return H(n) + 1.0/n
```



توابع بازگشتی: خطاهای متداول (۲)

۳۸

□ محاسبات تکراری.

□ یک تابع بازگشتی ساده نیز ممکن است به دلیل محاسبات تکراری، دارای پیچیدگی زمانی **نمایی** باشد!

```
def fib(n):  
    if n == 0: return 0  
    if n == 1: return 1  
    return fib(n-1) + fib(n-2)
```

$$F_n = F_{n-1} + F_{n-2}$$

```
fib(5)  
  fib(4)  
    fib(3)  
      fib(2)  
        fib(1)  
          return 1  
        fib(0)  
          return 0  
        return 1+0 = 1  
      fib(1)  
        return 1  
      return 1+1 = 2  
    fib(2)  
      fib(1)  
        return 1  
      fib(0)  
        return 0  
      return 1+0 = 1
```

محاسبات تکراری

اعداد فیبوناچی

۳۹



لئوناردو فیبوناچی
(۱۲۵۰ - ۱۱۷۰)

□ دنباله فیبوناچی. فرض کنید به ازای $n > 1$ تعریف کنیم $F_n = F_{n-1} + F_{n-2}$ و $F_0 = 0$ و $F_1 = 1$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
F_n	0	1	1	2	3	5	8	13	21	34	55	89	144	233	...

□ این دنباله پدیده‌های بسیاری را مدل‌سازی می‌کند و به طور گسترده‌ای در هنر و معماری یافت می‌شود.

□ مدلی برای نرخ تکثیر خرگوش‌ها

□ پوسته ملوانک (ناتیلوس)

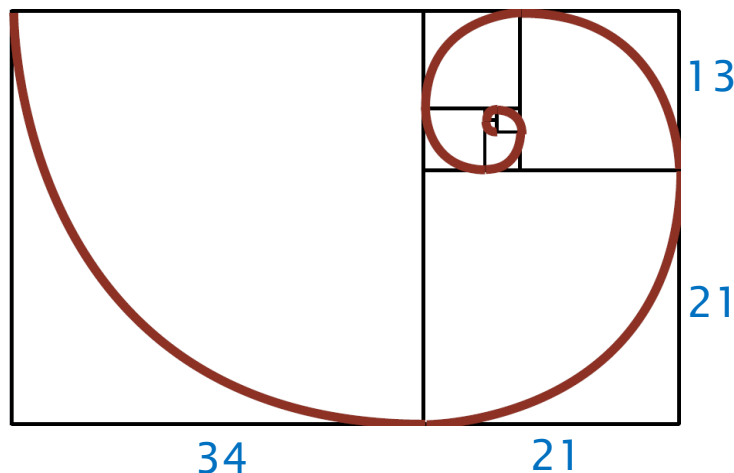
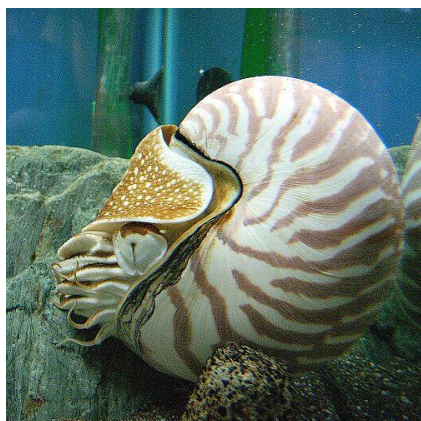
□ مونا لیزا

□ دانستنی‌ها.

□ نسبت F_n به F_{n-1} به مقدار ϕ میل می‌کند.

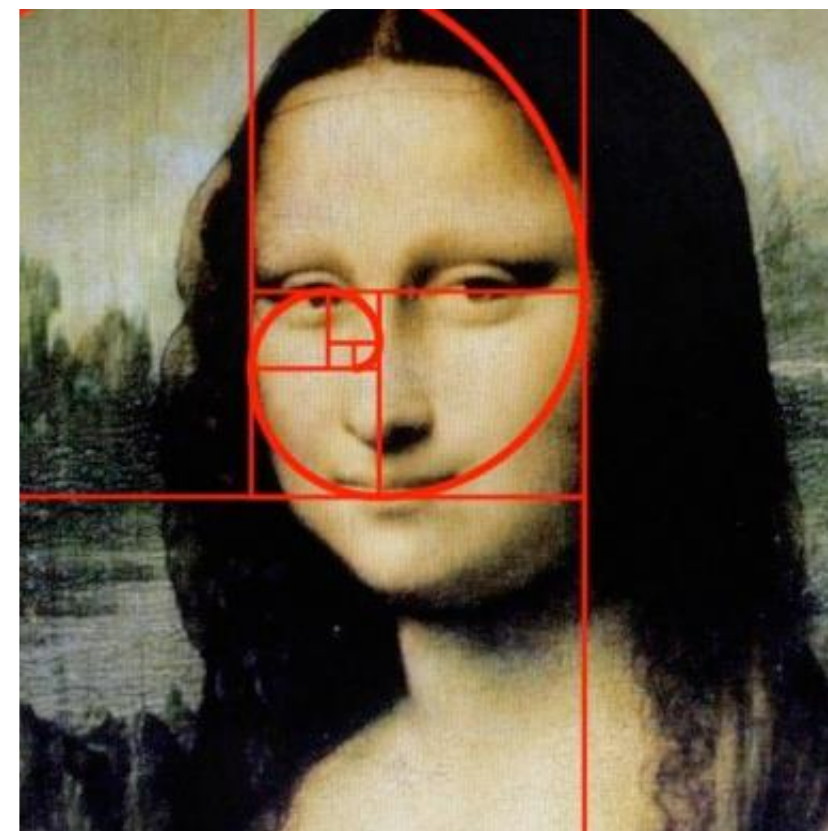
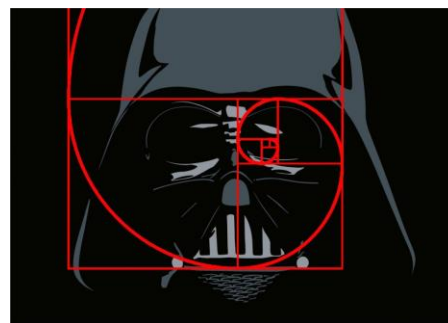
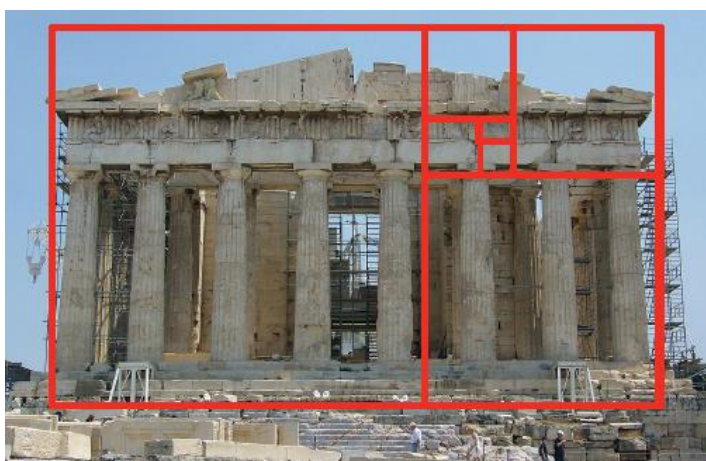
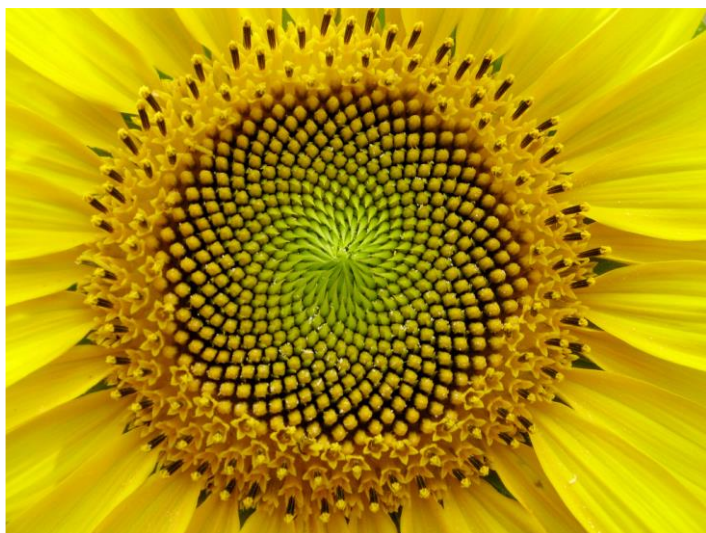
□ مقدار F_n نزدیک‌ترین عدد صحیح به $\phi^n / \sqrt{5}$ است.

نسبت طلایی F_n / F_{n-1}



نسبت طلایی در طبیعت

۴۰



محاسبه اعداد فیبوناچی

۴۱

□ پرسش. [یک فرد کنجکاو] مقدار دقیق F_{60} چقدر است؟

□ پاسخ. [یک برنامه‌نویس تازه‌کار] چند لحظه به من فرصت بده تا یک برنامه بازگشتی بنویسم.

```
import sys

def fib(n):
    if n == 0: return 0
    if n == 1: return 1
    return fib(n-1) + fib(n-2)

n = int(sys.argv[1])
print(fib(n))
```

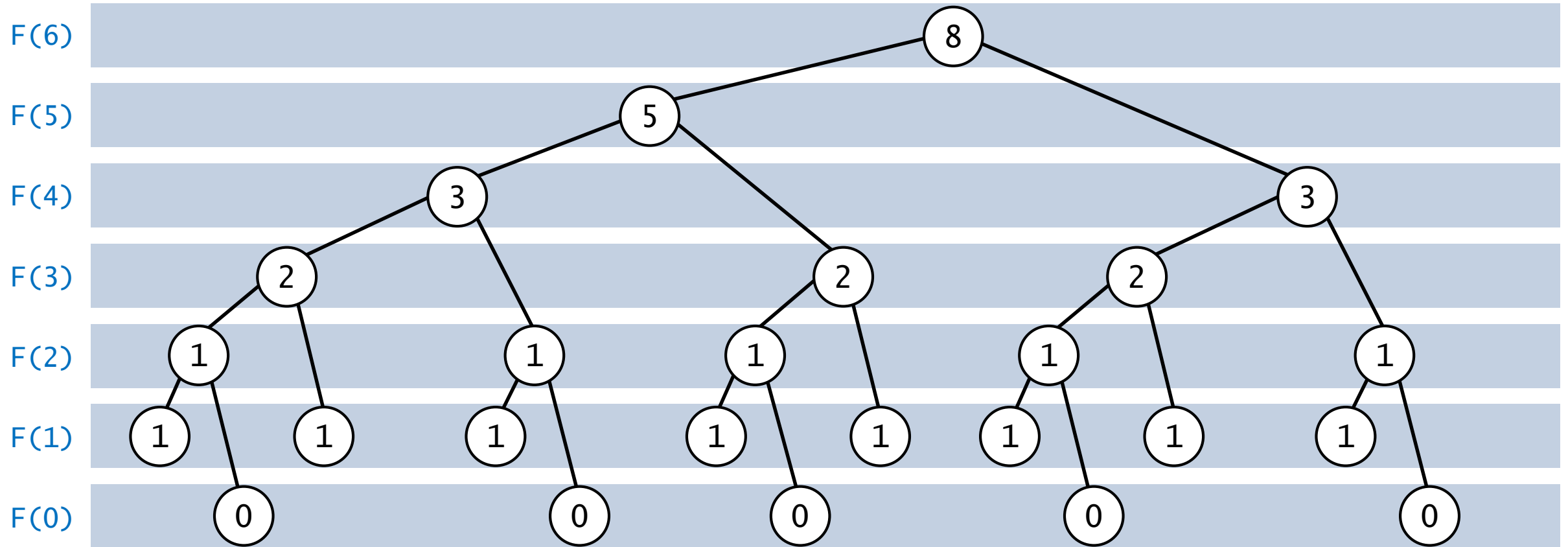
```
% python fibonacciR.py 5
5
% python fibonacciR.py 10
55
% python fibonacciR.py 12
144
% python fibonacciR.py 50
12586269025
% python fibonacciR.py 60
```

چند دقیقه طول
می‌کشد. چرا؟

آیا رایانه من ایرادی پیدا کرده است؟

درخت فراخوانی بازگشتی برای اعداد فیبوناچی

۴۲

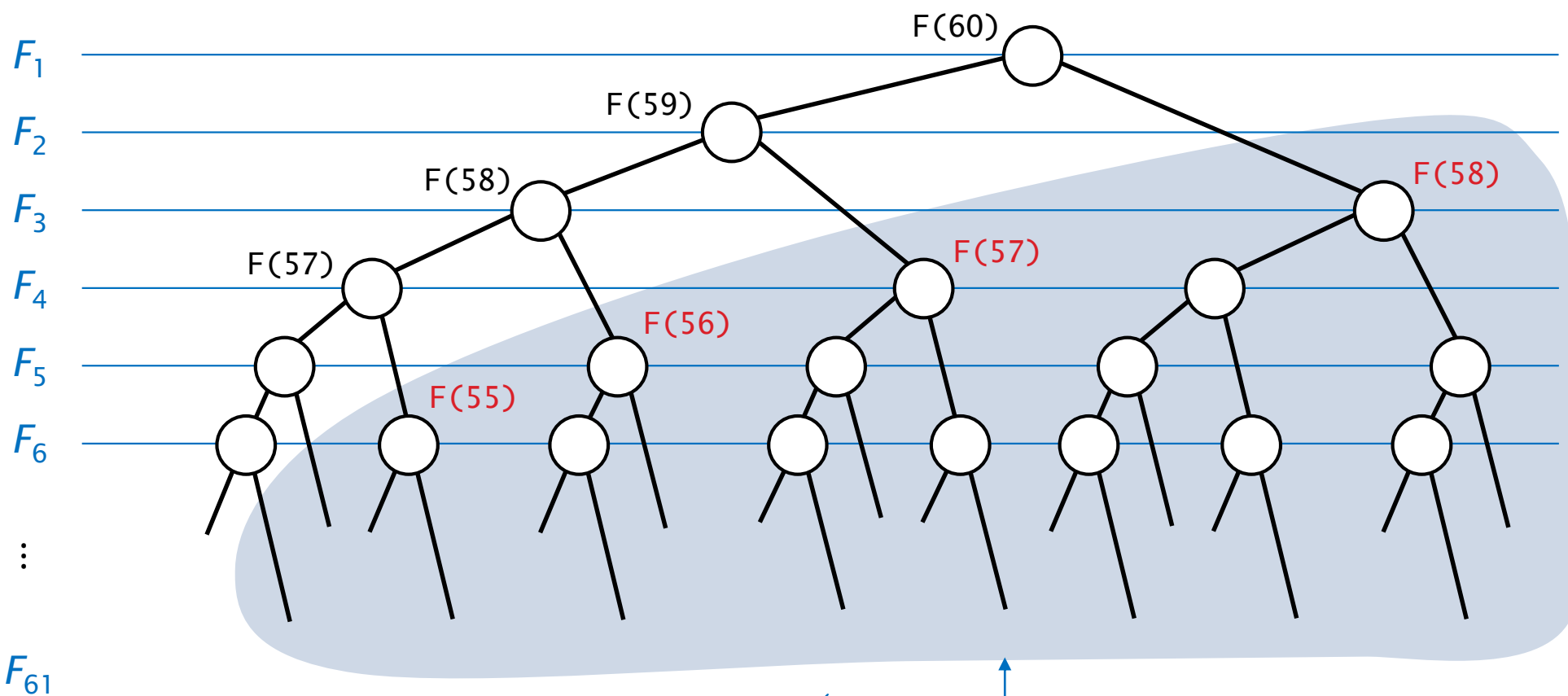


درخت فراخوانی بازگشتی برای اعداد فیبوناچی

۴۳

□ فرض کنید C_n بیانگر تعداد دفعات فراخوانی $F(n)$ در هنگام محاسبه $F(60)$ باشد.

n	C_n
60	1
59	1
58	2
57	3
56	5
55	8
⋮	⋮
0	$>2.5 \times 10^{12}$



اتلاف زمان به صورت نمایی به دلیل مناسبه تکراری این مقادیر

اجتناب از اتلاف زمانی

۴۴

```
import sys
```

```
memo = [0] * 200
```

```
def fib(n):
```

```
    if n == 0: return 0
```

```
    if n == 1: return 1
```

```
    if memo[n] == 0:
```

```
        memo[n] = fib(n-1) + fib(n-2)
```

```
    return memo[n]
```

```
n = int(sys.argv[1])
```

```
print(fib(n))
```

□ یادداشت‌برداری.

□ استفاده از یک آرایه به منظور به خاطر سپاری تمامی مقادیر محاسبه شده

□ اگر مقداری قبلاً محاسبه شده، آن را برگردان.

□ در غیر این صورت، آن مقدار را محاسبه کن، در آرایه ذخیره کن و سپس آن را برگردان.

```
% python fibonacciM.py 60
```

```
1548008755920
```

```
% python fibonacciM.py 80
```

```
23416728348467685
```

```
% python fibonacciM.py 100
```

```
354224848179261915075
```

برنامه‌ریزی پویا

۴۵



ریچارد بلمن ۱۹۲۰-۱۹۸۴

□ برنامه‌ریزی پویا.

- انجام محاسبات به شیوه «پایین به بالا»
- ابتدا زیرمسائل کوچک‌تر را حل و راه‌حل آنها را ذخیره کن.
- از این راه‌حل‌ها برای ایجاد راه‌حل‌های بزرگ‌تر استفاده کن.

```
def fib(n):  
    F = [0] * (n + 1)  
    F[0], F[1] = 0, 1  
    for i in range(2, n + 1):  
        F[i] = F[i - 1] + F[i - 2]  
    return F[n]
```

fib(5)

0	1	2	3	4	5
0	1	1	2	3	5

برنامه‌ریزی پویا

۴۶



امیر کاظمی
۱۳۸۴-۱۳۹۰

□ برنامه‌ریزی پویا.

- انجام محاسبات به شیوه «پایین به بالا»
- ابتدا زیرمسائل کوچک‌تر را حل و راه‌حل آنها را ذخیره کن.
- از این راه‌حل‌ها برای ایجاد راه‌حل‌های بزرگ‌تر استفاده کن.

```
def fib(n):  
    F = [0] * (n + 1)  
    F[0], F[1] = 0, 1  
    for i in range(2, n + 1):  
        F[i] = F[i - 1] + F[i - 2]  
    return F[n]
```

```
% python fibonacci.py 60  
1548008755920  
  
% python fibonacci.py 80  
23416728348467685  
  
% python fibonacci.py 100  
354224848179261915075
```

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۴۷

□ **تعریف.** یک زیردنباله از رشته S هر رشته‌ای است که با حذف تعدادی از کاراکترهای S به دست آید.

```
s = ggcaccacg
    cac          ggcaccacg
    gcaacg       ggcaccacg
    ggcaacg      ggcaccacg
    ggcacacg     ggcaccacg
    ...
```

$[2^n]$ زیردنباله در یک رشته به طول n

```
t = acggcggatacg
    gacg          acggcggatacg
    ggggg         acggcggatacg
    cggcgg        acggcggatacg
    ggcaacg       acggcggatacg
    ggggaacg      acggcggatacg
    ...
```

□ **تعریف.** طولانی‌ترین زیردنباله مشترک S و t طولانی‌ترین رشته‌ای است که زیردنباله‌ای از هر دو باشد.

□ **هدف.** یک الگوریتم کارا برای یافتن طولانی‌ترین زیردنباله مشترک و طول آن. ————— کاربردهای متعدد علمی

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۴۸

□ هدف. یک الگوریتم کارا برای یافتن طولانی‌ترین زیردنباله مشترک و طول آن.

□ رویکرد. ذخیره طولانی‌ترین زیردنباله مشترک $s[i..M]$ و $t[j..N]$ در $opt[i, j]$

Ex: $i = 6, j = 7$

```
s[6..9) = acg
t[7..12) = atacg
LCS(cg, tacg) = cg
LCS(acg, atacg) = acg
```

□ سه حالت:

□ اگر $i = M$ یا $j = N$

$opt[i][j] = 0$

□ اگر $s[i] = t[j]$

$opt[i][j] = opt[i+1][j+1] + 1$

□ در غیر این صورت

$opt[j][j] = \max(opt[i+1][j], opt[i][j+1])$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۴۹

□ هدف. یک الگوریتم کارا برای یافتن طولانی‌ترین زیردنباله مشترک و طول آن.

□ رویکرد. ذخیره طولانی‌ترین زیردنباله مشترک $s[i..M]$ و $t[j..N]$ در $opt[i, j]$

Ex: $i = 6, j = 4$

$s[6..9] = \text{acg}$

$t[4..12] = \text{cggatacgc}$

$\text{LCS}(\text{acg}, \text{ggatacgc}) = \text{acg}$

$\text{LCS}(\text{cg}, \text{cggatacgc}) = \text{cg}$

$\text{LCS}(\text{acg}, \text{cggatacgc}) = \text{acg}$

□ سه حالت:

□ اگر $i = M$ یا $j = N$

$opt[i][j] = 0$

□ اگر $s[i] = t[j]$

$opt[i][j] = opt[i+1][j+1] + 1$

□ در غیر این صورت

$opt[j][j] = \max(opt[i+1][j], opt[i][j+1])$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۰

	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	c	g	g	c	g	g	a	t	a	c	g	
0	g												0
1	g												0
2	c												0
3	a												0
4	c												0
5	c												0
6	a												0
7	c												0
8	g												0
9		0	0	0	0	0	0	0	0	0	0	0	0

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۱

	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	c	g	g	c	g	g	a	t	a	c	g	
0	g												0
1	g												0
2	c												0
3	a												0
4	c												0
5	c												0
6	a												0
7	c												0
8	g											1	0
9		0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \text{opt}[i+1][j+1] + 1$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۲

	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	c	g	g	c	g	g	a	t	a	c	g	
0	g												0
1	g												0
2	c												0
3	a												0
4	c												0
5	c												0
6	a												0
7	c												0
8	g										1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \max(\text{opt}[i+1][j], \text{opt}[i][j+1])$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۳

	0	1	2	3	4	5	6	7	8	9	10	11	12
	a	c	g	g	c	g	g	a	t	a	c	g	
0	g												0
1	g												0
2	c												0
3	a												0
4	c												0
5	c												0
6	a												0
7	c												0
8	g									1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \max(\text{opt}[i+1][j], \text{opt}[i][j+1])$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۴

		0	1	2	3	4	5	6	7	8	9	10	11	12
		a	c	g	g	c	g	g	a	t	a	c	g	
0	g													0
1	g													0
2	c													0
3	a													0
4	c													0
5	c													0
6	a													0
7	c												1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \max(\text{opt}[i+1][j], \text{opt}[i][j+1])$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۵

		0	1	2	3	4	5	6	7	8	9	10	11	12
		a	c	g	g	c	g	g	a	t	a	c	g	
0	g													0
1	g													0
2	c													0
3	a													0
4	c													0
5	c													0
6	a													0
7	c											2	1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \text{opt}[i+1][j+1] + 1$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۶

		0	1	2	3	4	5	6	7	8	9	10	11	12
		a	c	g	g	c	g	g	a	t	a	c	g	
0	g													0
1	g													0
2	c													0
3	a													0
4	c													0
5	c													0
6	a													0
7	c										2	2	1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0

$$\text{opt}[i][j] = \max(\text{opt}[i+1][j], \text{opt}[i][j+1])$$

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۷

		0	1	2	3	4	5	6	7	8	9	10	11	12
		a	c	g	g	c	g	g	a	t	a	c	g	
0	g	7	7	7	6	6	6	6	4	3	3	2	1	0
1	g	6	6	6	6	5	5	5	4	3	3	2	1	0
2	c	6	5	5	5	5	4	4	4	3	3	2	1	0
3	a	6	5	4	4	4	4	4	4	3	3	2	1	0
4	c	5	5	4	4	4	3	3	3	3	3	2	1	0
5	c	4	4	4	4	4	3	3	3	3	3	2	1	0
6	a	3	3	3	3	3	3	3	3	3	3	2	1	0
7	c	2	2	2	2	2	2	2	2	2	2	2	1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0

$opt[i][j] =$
 $\max(opt[i+1][j], opt[i][j+1])$

طولانی‌ترین زیر دنباله مشترک: پیاده‌سازی

۵۸

```
def LCS(s, t):  
    m, n = len(s), len(t)  
    opt = [[0] * (n+1) for i in range(m+1)]  
    for i in range(m - 1, -1, -1):  
        for j in range(n - 1, -1, -1):  
            if s[i] == t[j]:  
                opt[i][j] = opt[i+1][j+1] + 1  
            else:  
                opt[i][j] = max(opt[i+1][j], opt[i][j+1])  
    return opt[0][0]
```

```
% python lcs.py ggcaccacg acggcggatacg  
7
```

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۵۹

□ محاسبه طولانی‌ترین زیردنباله مشترک

		0	1	2	3	4	5	6	7	8	9	10	11	12	
		a	c	g	g	c	g	g	a	t	a	c	g		
0	g	7	← 7	← 7	↖ 6	6	6	6	4	3	3	2	1	0	
1	g	6	6	6	6	↖ 5	5	5	4	3	3	2	1	0	
2	c	6	5	5	5	5	↖ 4	4	4	3	3	2	1	0	
3	a	6	5	4	4	4	4	← 4	← 4	4	↖ 3	3	2	1	0
4	c	5	5	4	4	4	3	3	3	3	← 3	3	2	1	0
5	c	4	4	4	4	4	3	3	3	3	3	↖ 3	2	1	0
6	a	3	3	3	3	3	3	3	3	3	3	3	↖ 2	1	0
7	c	2	2	2	2	2	2	2	2	2	2	2	2	↖ 1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	1	↖ 0
9		0	0	0	0	0	0	0	0	0	0	0	0	0	0

برنامه‌ریزی پویا: طولانی‌ترین زیردنباله مشترک

۶۰

	0	1	2	3	4	5	6	7	8	9	10	11	12	
	a	c	g	g	c	g	g	a	t	a	c	g		
0	g	7 ←	7 ←	7	6	6	6	6	4	3	3	2	1	0
1	g	6	6	6	6	5	5	5	4	3	3	2	1	0
2	c	6	5	5	5	5	4	4	4	3	3	2	1	0
3	a	6	5	4	4	4	4	4	4	3	3	2	1	0
4	c	5	5	4	4	4	3	3	3	3	3	2	1	0
5	c	4	4	4	4	4	3	3	3	3	3	2	1	0
6	a	3	3	3	3	3	3	3	3	3	3	2	1	0
7	c	2	2	2	2	2	2	2	2	2	2	2	1	0
8	g	1	1	1	1	1	1	1	1	1	1	1	1	0
9		0	0	0	0	0	0	0	0	0	0	0	0	0

$s = \text{ggcaccacg}$

$t = \text{acggcggatacg}$

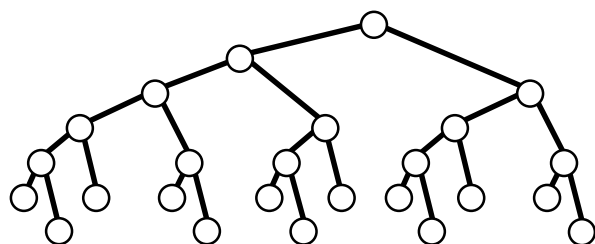
$\text{LCS}(s, t) = \text{ggcaacg}$

برنامه‌ریزی پویا و الگوریتم‌های بازگشتی

۶۱

□ دو رویکرد بسیار مفید برای حل مسائل با ترکیب راه‌حل‌های زیرمسائل کوچک‌تر.

□ چرا برنامه‌ریزی پویا و الگوریتم‌های بازگشتی؟



□ بیانگر یک شیوه جدید فکر کردن

□ الگوهای قدرتمند برنامه‌نویسی

□ درک طبیعت محاسبات

□ استفاده موفق در طول دهه‌ها



□ چگونه می‌توان یک تابع بازگشتی نوشت؟

□ حالت پایه، فراخوانی بازگشتی.

□ ردیابی اجرای یک تابع بازگشتی.

□ استفاده از شکل.

□ مزایای یادگیری توابع بازگشتی.

□ آشنایی با یک سبک جدید تفکر (تفکر بازگشتی)

□ آشنایی با یک الگوی قدرتمند برنامه نویسی

□ تقسیم و حل. یک راه حل ظریف و زیبا برای بسیاری از مسائل مهم.

□ برنامه‌ریزی پویا. اجتناب از محاسبات تکراری با حل زیرمسائل از پایین به بالا و ذخیره راه‌حل‌ها.