投票システムを準備してください

関数から計算へ

課題実施状況の確認

1. 前回の課題（画像）を提出した
2. 前回の課題は完成したが提出できなかった
3. 前回の課題は完成できなかった
4. 前回の課題は手がけていない

課題実施状況の確認

1. 独自画像の作成は容易だった
2. カラーの独自画像作成が難しかった
3. モノクロの独自画像作成も難しかった
4. そもそも作業しなかった

課題実施状況の確認

1. プログラム（平均画像）は容易だった
2. カラーのプログラムが難しかった
3. モノクロのプログラムも難しかった
4. そもそも作業しなかった

課題実施状況の確認

1. 課題提出の方法はページの説明でわかった
2. 画像の取り込みが難しかった
3. 圧縮ファイルの作成が難しかった
4. 両方ともに難しかった
5. そもそも作業しなかった

練習

- x が y で割り切れることを判定する(true, false を返す)関数divisible(x,y) を定義せよ。
  （ヒント） 剰余の演算子「%」を用いると良い。

関数を定義したら投票

1. 定義した

練習

- x が y で割り切れることを判定する関数 divisible(x,y) を定義せよ。

```
def divisible(x,y)
    x % y == 0
end
```

divisible.rb

```
irb(main):004:0> divisible(6,2)
=> true
irb(main):005:0> divisible(6,3)
=> true
irb(main):006:0> divisible(5,2)
=> false
```

繰り返しで条件を満たす値を探す

- kの約数でn以下(1..n)の最大の値

```
load ("./divisible.rb”)
def gd_loop(k,n)
   while !divisible(k,n)
      n = n-1
   end
   n
end
```

gd_loop.rb *

```
irb(main):004:0> gd_loop(6,5)
=> 3
irb(main):005:0> gd_loop(98,30)
=> 14
irb(main):006:0> gd_loop(98,97)
=> 49
irb(main):007:0> gd_loop(47,46)
=> 1
```

4.4　定義のまとめ

条件を満たすまでの繰り返し: 式 が成り立つ間 命令$_1$ から 命令$_n$ を毎回実行する繰り返しは次のように書く。

```
while 式
    命令1
    ⋮
    命令n
end
```

わからない？

1. わかった
2. def
3. while !divisible(k,n)
4. n = n-1
5. そのほか 具体的に書く

gd_loop.rb を授業のページからダウンロードする

繰り返しによる反復計算の定義 --- 代入による変数の更新

```
irb(main):003:0> weight = 104.0
=> 104.0
irb(main):004:0> weight = weight-10
=> 94.0
irb(main):005:0> weight
=> 94.0
irb(main):006:0> weight = weight-10
=> 84.0
```

繰り返しによる総和の定義

```
def sum_loop(n)
    s = 0
    for i in 1..n
        s = s+i
    end
    s
end
```

sum_loop.rb *

```
irb(main):004:0> sum_loop(3)
=> 6
irb(main):005:0> sum_loop(10)
=> 55
irb(main):006:0> 1+2+3+4+5+
irb(main):007:0* 6+7+8+9+10
=> 55
```

繰り返しによる約数の和

```
load ("./divisible.rb”)
def sod_loop(k,n)
    s = 0
    for i in 1..n
        if divisible(k,i)
            s = s+i
        end
    end
    s
end
```

sod_loop.rb

```
irb(main):004:0> sod_loop(10,9)
=> 8
irb(main):005:0> 5+2+1
=> 8
irb(main):006:0> sod_loop(28,27)
=> 28
irb(main):007:0> 14+7+4+2+1
⇒28
irb(main):008:0> sod_loop(29,28)
=> 1
```

再帰による反復計算の定義

```
def sum(n)
   if n >= 2
      sum(n-1) + n
   else
      1
   end
end
```

sum.rb *

$$\mathrm{sum}(n) = \begin{cases} \mathrm{sum}(n-1) + n & (n \geq 2) \\ 1 & (n = 1) \end{cases}$$

```
irb(main):005:0> sum(10)
=> 55
irb(main):006:0> 1+2+3+4+5+
irb(main):007:0* 6+7+8+9+10
=> 55
```

約数の和(再帰)

```
load ("./divisible.rb”)
def sod(k,n)
   if n >= 2
      if divisible(k,n)
         sod(k,n-1) + n
      else
         sod(k,n-1)
      end
   else
      1
   end
end
```

sod.rb *

$$\mathrm{sod}(k,n) = \begin{cases} \mathrm{sod}(k,n-1)+n & (n \geq 2 \text{ かつ } n \text{ が } k \text{ の約数}) \\ \mathrm{sod}(k,n-1)+0 & (n \geq 2 \text{ かつ } n \text{ が } k \text{ の約数でない}) \\ 1 & (n = 1) \end{cases}$$

```
irb(main):006:0> sod(28,27)
=> 28
irb(main):007:0> 14+7+4+2+1
=> 28
irb(main):008:0> sod(29,28)
=> 1
```

条件を満たす値を探す(再帰)

```
load ("./divisible.rb")
def gd(k,n)
    if divisible(k,n)
        n
    else
        gd(k,n-1)
    end
end
```

gd.rb *

$$\mathrm{gd}(k,n) = \begin{cases} n & (n \text{ が } k \text{ の約数}) \\ \mathrm{gd}(k,n-1) & (n \text{ が } k \text{ の約数でない}) \end{cases}$$

```
irb(main):005:0> gd(98,30)
=> 14
irb(main):006:0> gd(98,97)
=> 49
irb(main):007:0> gd(47,46)
=> 1
```

わかった？

1. わかった
2. gdがわからない
3. sodがわからない
4. sumがわからない

組合せ数の計算

- パスカルの三角形

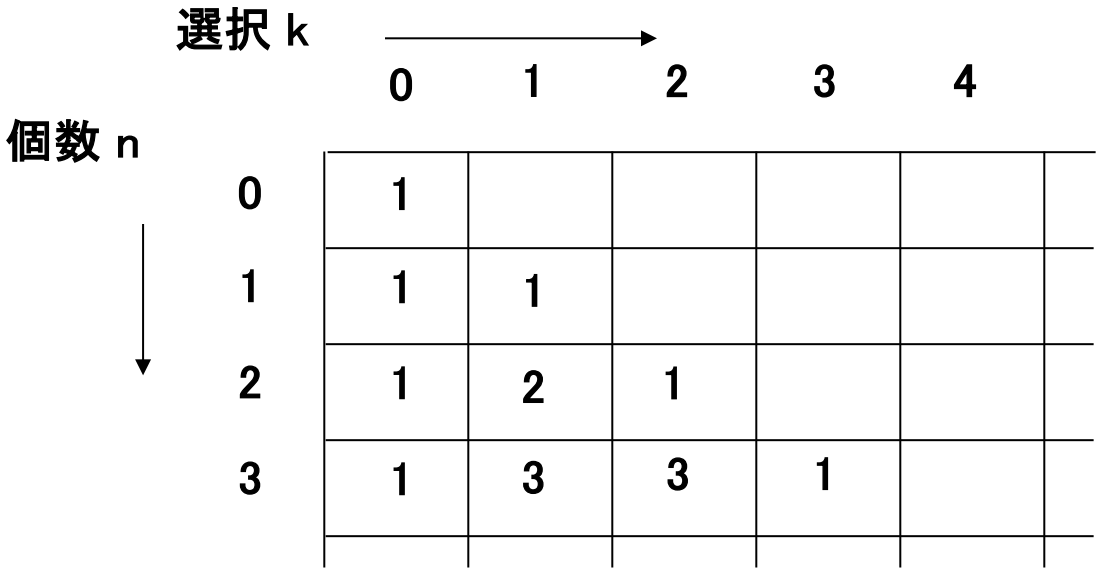
  combination(n, k)

  = combination(n−1, k−1) + combination(n−1, k)

  n個から k個を選択する

  = n−1個から k−1個を選択して n個目も選択

  + n−1個から k個を選択して n個目は選択せず



再帰を使った組合せ数の計算

$n \setminus k$	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

表 4.1: 組み合わせ数 ${}_nC_k$ の値

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2							
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1						
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2					
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3							
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1						
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3					
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3				
4							
5							
6							

n ＼ k	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4							
5							
6							

練習

- n 個から k 個を選ぶ組合せ数 ${}_nC_k$ を求める combination(n,k) を定義せよ。(教科書 58ページの練習4.2)

$$
{}_nC_k = \begin{cases} 0 & (k > n \text{ のとき}) \\ 1 & (k = 0 \text{ のとき}) \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & (\text{それ以外}) \end{cases}
$$

- 素数とは1 と自分自身しか約数がない数である。上で定義した関数 sod や gd を使って n が素数のときに true, そうでないときに false となるような関数 prime(n) を定義せよ。(教科書 57ページの練習4.1，59ページの練習4.3)

練習

- 教科書 58ページの練習4.2

$$
{}_nC_k = \begin{cases} 0 & (k > n \text{ のとき}) \\ 1 & (k = 0 \text{ のとき}) \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & (\text{それ以外}) \end{cases}
$$

練習4.2ができたら投票

1. 練習4.2ができた

- 続いて、教科書 57ページの練習4.1，59ページの練習4.3

```
def combination(n,k)
    if k > n
        0
    else
        if k == 0
            1
        else
            combination(n-1,k-1) + combination(n-1,k)
        end
    end
end
```

combination.rb

進捗状況の確認

1. 練習4.3まで終わった
2. 練習4.1まで終わった
3. 練習4.2だけ終わった
4. 練習4.2もできていない
5. それ以外，具体的に書く

配列と繰り返しを使った組合せ数の計算

$n \setminus k$	0	1	2	3	4	5	6
0	1						
1	1	1					
2	1	2	1				
3	1	3	3	1			
4	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	20	15	6	1

表 4.1: 組み合わせ数 ${}_nC_k$ の値

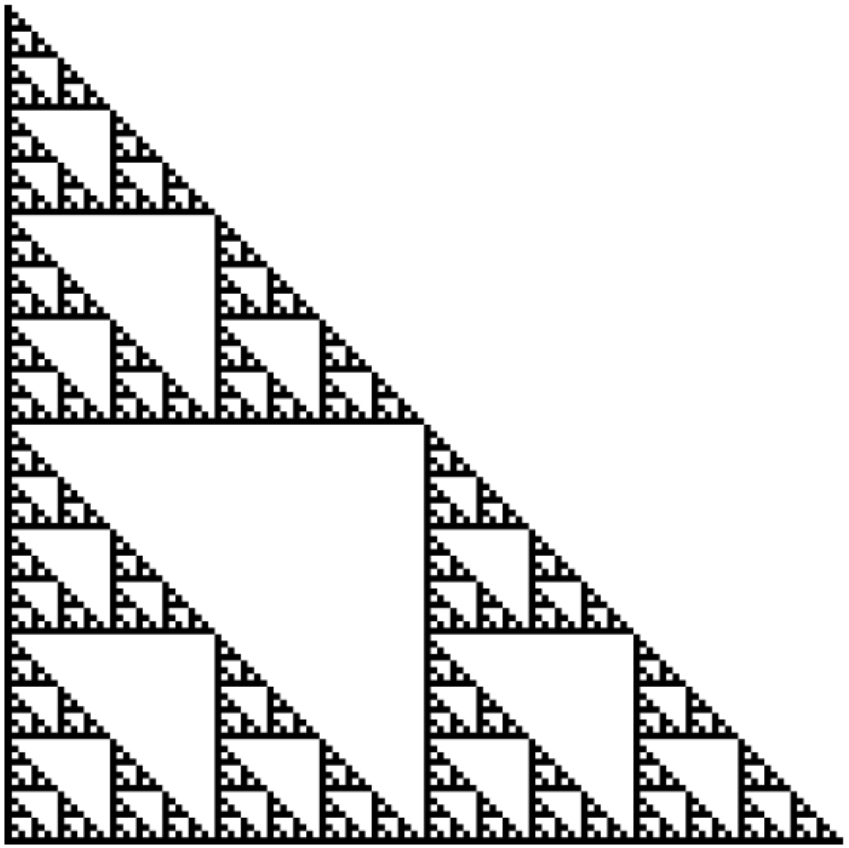
```
load ("./make2d.rb")

def combination_loop(n,k)
    c = make2d(n+1,n+1)
    for i in 0..n
        c[i ][0] = 1
        for j in 1..(i-1)
            c[i][j] = c[i-1][j-1] + c[i-1][j]
        end
        c[i][i] = 1
    end
    c[n][k]
end
```

combination_loop.rb

練習4.15(教科書 72ページ)

- Sierpinski の三角形
  - 大きさ n*n で、i 行 j 列目が ${}_iC_j$ を 2 で割った余りになっているような配列を作る関数 sierpinski2d_loop(n) を定義せよ。（見易さのために白黒を逆にしている。）



Cantor Set

- 区間 [0,1] の真ん中の 1/3 を除く。
- 残った二つの区間に対して、同じくことを行う。
- 同じことを無限に繰り返す。
- 残った点の集合が Cantor set。
- Cantor set の測度（長さ）は 0。
- しかし、濃度（点の数）は、もとの区間 [0,1] の実数に等しい。

Cantor Set

```
def cantor(n)
  a = make1d(3**n)
  subcantor(a, n, 0)
  a
end

def subcantor(a, n, x)
  if n==0
    a[x] = 1
  else
    subcantor(a, n-1, x)
    subcantor(a, n-1, x+2*3**(n-1))
  end
end
```

大きさ 3**n の配列を作成。
すべて 0 で初期化されている。

再帰的な手続き subcantor を
座標 0 に対して呼び出す。

配列 a を返す。

座標 x を起点として、
n階の Cantor set を
配列 a に設定する。

n が 0 (最小)のときは、
分割せずに、1 を設定。

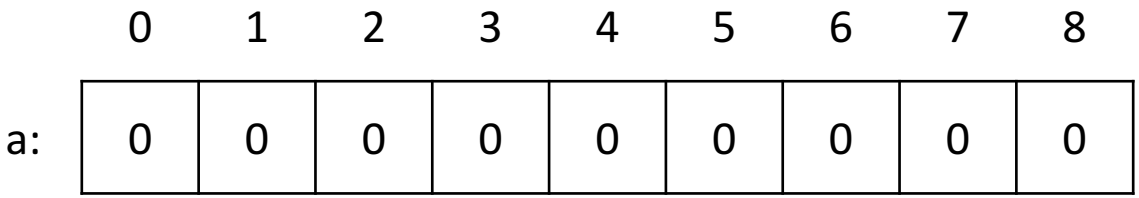
再帰呼び出し。起点
(最初の1/3)に注意。

再帰呼び出し。起点(最後の1/3)に注意。
2*3**(n-1) = (2/3) * 3**n と考えると良い。

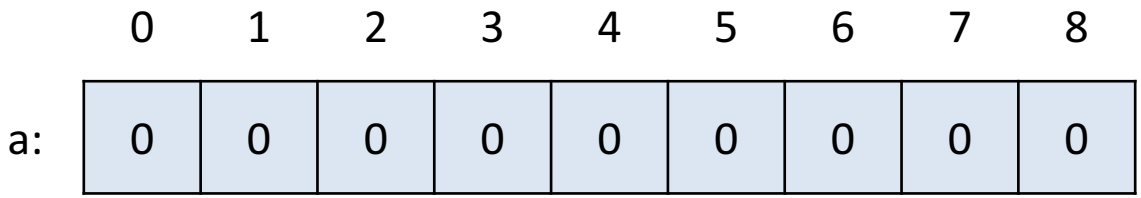
わからない？

1. わかった
2. make1d(3**n)
3. subcantor(a, n, 0)
4. if n == 0
5. subcantor(a, n-1, x)
6. subcantor(a, n-1, x+2*3**(n-1))

cantor.rb を授業のページからダウンロードする

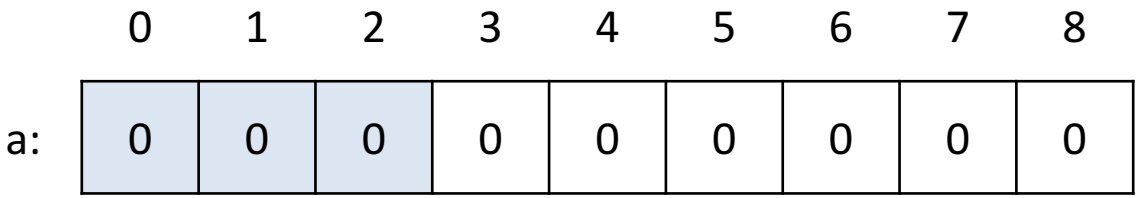


cantor(2)

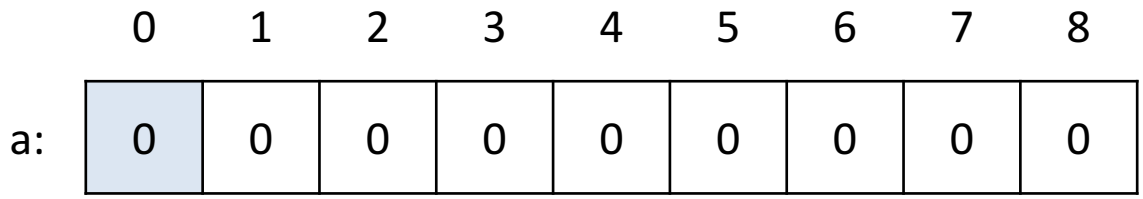


cantor(2)

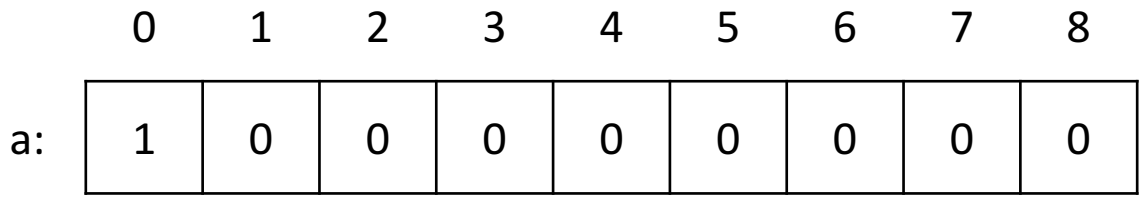
subcantor(a,2,0)



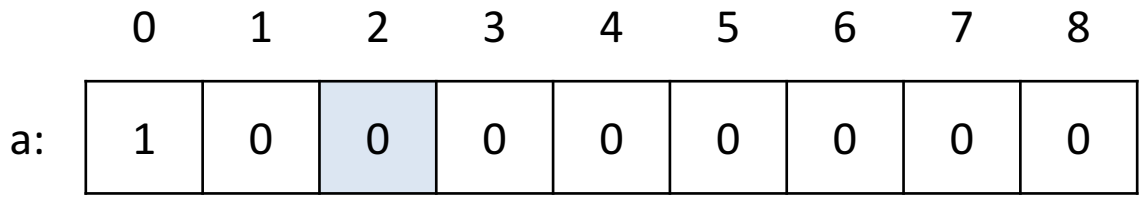

```
cantor(2)
   subcantor(a,2,0)
      subcantor(a,1,0)
```




```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
```



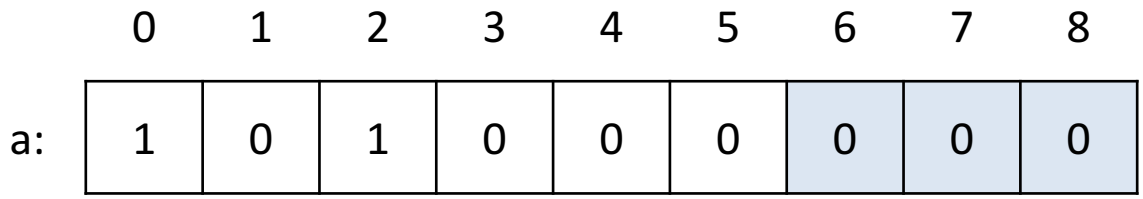

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
```



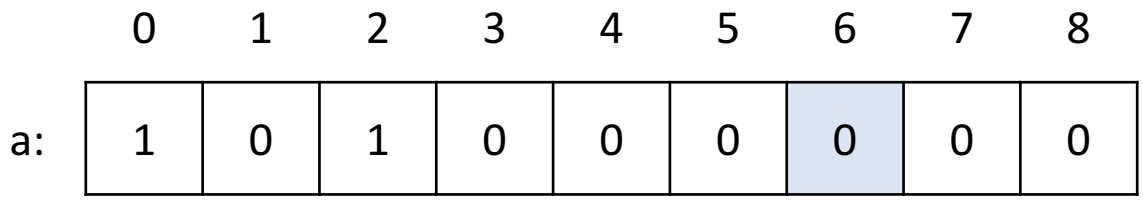

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
            subcantor(a,0,0+2*3**0)
```

	0	1	2	3	4	5	6	7	8
a:	1	0	1	0	0	0	0	0	0

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
            subcantor(a,0,0+2*3**0)
                a[2] = 1
```



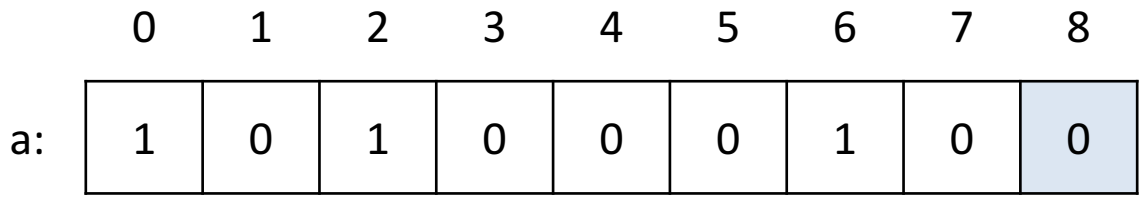

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
            subcantor(a,0,0+2*3**0)
                a[2] = 1
        subcantor(a,1,0+2*3**1)
```



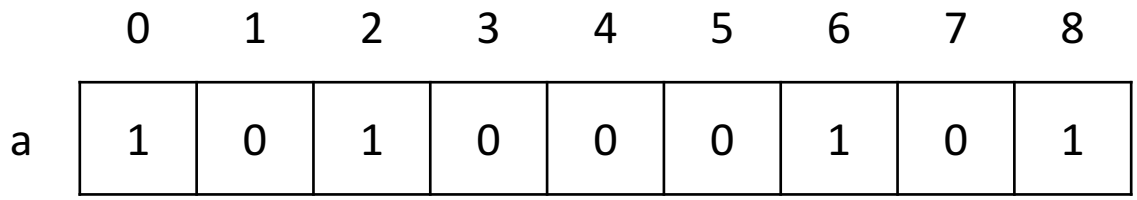

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
            subcantor(a,0,0+2*3**0)
                a[2] = 1
        subcantor(a,1,0+2*3**1)
            subcantor(a,0,6)
```

	0	1	2	3	4	5	6	7	8
a:	1	0	1	0	0	0	1	0	0

```
cantor(2)
    subcantor(a,2,0)
        subcantor(a,1,0)
            subcantor(a,0,0)
                a[0] = 1
            subcantor(a,0,0+2*3**0)
                a[2] = 1
        subcantor(a,1,0+2*3**1)
            subcantor(a,0,6)
                a[6] = 1
```




```
cantor(2)
   subcantor(a,2,0)
      subcantor(a,1,0)
         subcantor(a,0,0)
            a[0] = 1
         subcantor(a,0,0+2*3**0)
            a[2] = 1
      subcantor(a,1,0+2*3**1)
         subcantor(a,0,6)
            a[6] = 1
         subcantor(a,0,6+2*3**0)
```




```
cantor(2)
   subcantor(a,2,0)
      subcantor(a,1,0)
         subcantor(a,0,0)
            a[0] = 1
         subcantor(a,0,0+2*3**0)
            a[2] = 1
      subcantor(a,1,0+2*3**1)
         subcantor(a,0,6)
            a[6] = 1
         subcantor(a,0,6+2*3**0)
            a[8] = 1
```

Cantor Set の表示

- 試しに isrb で Cantor Set を表示してみる

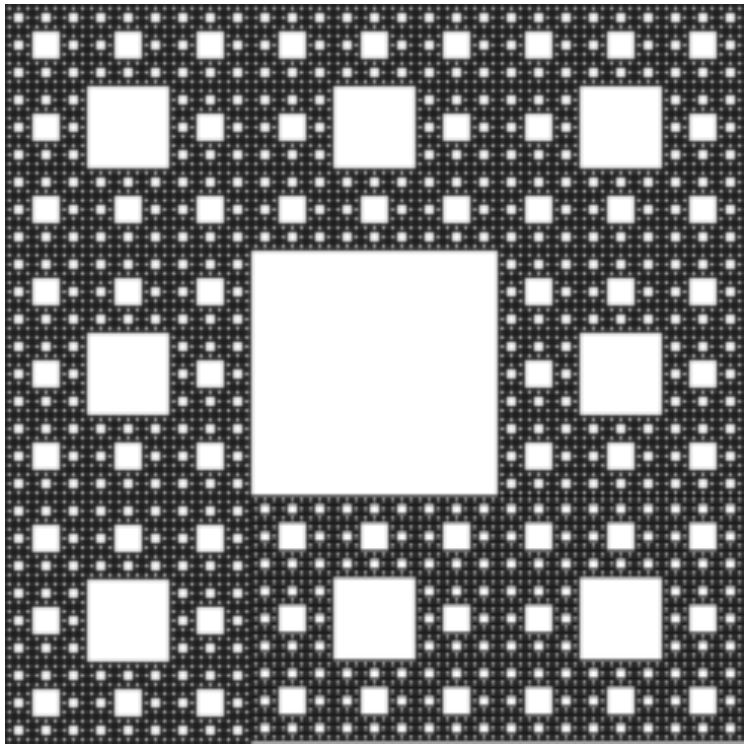
```
isrb(main):001:0> load("./cantor.rb")
true
isrb(main):002:0> show([cantor(1)])
[[1, 0, 1]]
isrb(main):003:0> show([cantor(2)])
[[1, 0, 1, 0, 0, 0, 1, 0, 1]]
isrb(main):004:0> show(cantor(3))
[[1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0, 1, 0, 1]]
isrb(main):005:0
```

1 次元配列を2次元配列に変更するために，[]で囲っている．実際には1次元配列のままでも表示できる．

練習

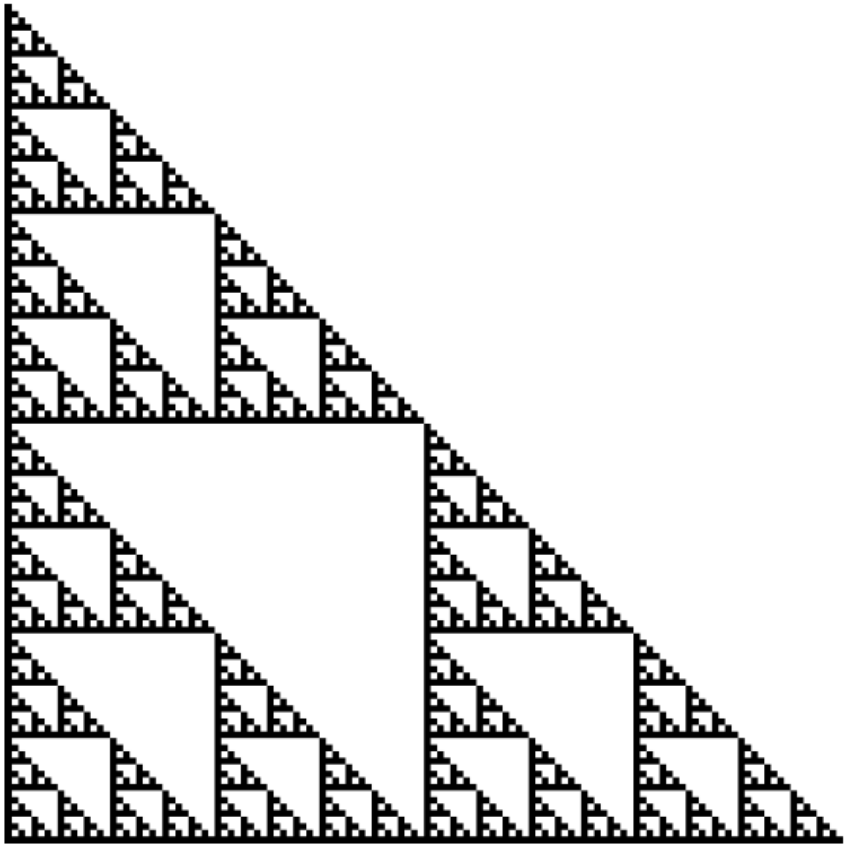
Cantor set の 2 次元版。
cf. Cantor dust

- Sierpinski のカーペット
  - n 次のカーペットは、縦横が 3**n で、n-1 階のカーペットを 8 枚敷き詰めて作られる。真ん中は空いている。0 階のカーペットは、縦横 1 の黒い正方形とする。（実際のプログラムでは、白黒が反転する。）



練習

- Sierpinski の三角形
  - 関数 sierpinski2d(n) を再帰的に定義せよ。n 階の三角形は、縦横が 2**n で、n-1 階のの三角形を 3 枚敷き詰めて作られる。真ん中は空いている。（見易さのために白黒を逆にしている。）



進捗状況の確認

1. Sierpinski のカーペットができた（できた時点で投票してください）

Sierpinski のカーペットができた人は、
Sierpinski の三角形に挑戦してみてください。