Information Science 7: Repetition and Recursion II: Combinations and Tower of Hanoi

Naonori Kakimura
垣村尚徳
kakimura@global.c.u-tokyo.ac.jp

PEAK
Programs in English at Komaba

Review:Recursion

- Recursion
  - Define a function using the function itself

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

  - Simpler description
    - no "…", no loop

Ex) sum(3)=sum(2)+3
=(sum(1)+2)+3
=(1+2)+3

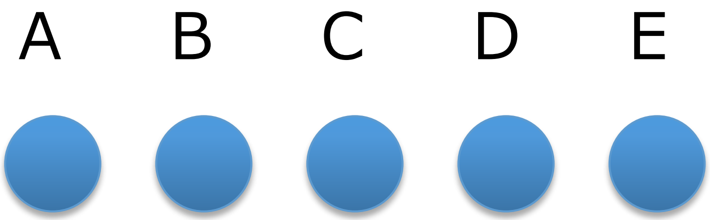
- Sometimes it takes much more time to compute than using repetition
  - Observe how it works today

Today's Contents: Recursion

- Review
  - Summation
- Number of Combinations
  - Using recursion
  - Using repetition
- Exercises
  - Sierpinski triangle
  - Tower of Hanoi

Combination Number ${}_nC_k$

- the number of combinations when we choose k items out of n items



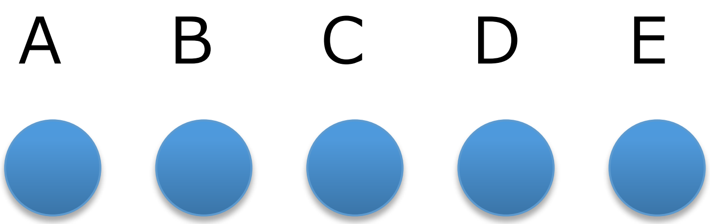
Ex. choose 2 items out of the 5 elements

AB	BC	CD
AC	BD	CE
AD	BE	DE
AE		

10 possibilities

Combination Number ${}_nC_k$

- the number of combinations when we choose k items out of n items



Ex. choose 2 items out of the 5 elements

Ans. First item: 5 possibilities (one out of (A, B, C, D, E))

2nd item: 4 possibilities (one other than the 1st one)

Reduce "double counting" (both AB & BA are counted)

Ans = 5*4/2=10

Combination Number ${}_nC_k$

- the number of combinations when we choose k items out of n items

Choose k items out of n elements

choosing k elements

$$ {}_nC_k = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k(k-1)(k-2)\cdots 2\cdot 1} $$

notation

reduce “double counting” due to the ordering

$$ = \frac{n!}{k!(n-k)!} $$

Recursive Definition of Combinations

- Cf) Exercise 5-11

choosing k items out of n items

$$
{}_nC_k = \begin{cases} 0 & (\text{when } k > n) \\ 1 & (\text{when } k = 0) \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & (\text{otherwise}) \end{cases}
$$

choosing k-1 items out of the first n-1 items
(the case the last one is chosen)

choosing k items out of the first n-1 items
(the case the last one is not chosen)

Today's Contents: Recursion

- Review
  - Summation
- Number of Combinations
  - Using recursion
    - natural implementation using recursion
  - Using repetition
- Exercises
  - Sierpinski triangle
  - Tower of Hanoi

```ruby
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
```

$$
{}_nC_k = \begin{cases} 0 & \text{(when k > n)} \\ 1 & \text{(when k = 0)} \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & \text{(otherwise)} \end{cases}
$$

combination.rb

Today's Contents: Recursion

- Review
  - Summation

- Number of Combinations
  - Using recursion
  - Using repetition
    - Faster implementation

- Exercises
  - Sierpinski triangle
  - Tower of Hanoi

Toward Faster Computation using Repetition

- Relationship

The upper-right part is 0

The 1st column is 1

$n \setminus k$	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

The value of ${}_nC_k$

Equations in the Table

$$ {}_nC_k = \begin{cases} 0 & \text{(when k > n)} \\ 1 & \text{(when k = 0)} \\ {}_{n-1}C_{k-1} + {}_{n-1}C_k & \text{(otherwise)} \end{cases} $$

… The upper-right part is 0

… The 1st column is 1

Relationship btw 2 lines

a row is determined by the row above

$n \setminus k$	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

The value of ${}_nC_k$

Concept: Suffices to Obtain The Table

- Make an (n+1)×(n+1) array
- Fill in the entries from i=0 to n

when n=6

the next row is determined by the last row

$n \setminus k$	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

The value of ${}_nC_k$

Concept: Suffices to Obtain The Table

- Make an (n+1)×(n+1) array
- Fill in the entries from i=0 to n

when n=6

the next row is determined by the last row

$n \setminus k$	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

The value of ${}_nC_k$

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

def combination_loop(n,k)
  c = make2d(n+1,n+1)
  for i in 0..n            # for each row, do the following
    c[i][0] = 1                      # 1st column
    for j in 1..(i-1)      # 2nd to (i-1)th column
      c[i][j] = c[i-1][j-1] + c[i-1][j]
    end
    c[i][i] = 1                      # i-th column
  end
  c[n][k]
end
```

combination_loop.rb

When we call combination_loop(6,3)

Make a 2-dimansional array c with 7×7

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

Fill in the first row

i=0

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

Case k>n is omitted in the table

Fill in the second row

i=1

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

c[i][0]=1
c[i][i]=1

Fill in the 3rd row

i=2

c[i][0]=1

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

Fill in the 3rd row

i=2

$c[i][j] = c[i-1][j-1] + c[i-1][j]$

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

Fill in the 3rd row

i=2

c[i][i]=1

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

Fill in the 4th row

i=3

$c[i][0]=1$

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

Fill in the 4th row

i=3

$c[i][j] = c[i-1][j-1] + c[i-1][j]$

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

Fill in the 4th row

i=3

$$c[i][j] = c[i-1][j-1] + c[i-1][j]$$

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							

Fill in the 4th row

i=3

$c[i][i] = 1$

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							

In the End

i=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	1	4	6	4	1		
5	1	5	10	10	5	1	
6	1	6	15	**20**	15	6	1

$_6C_3$ = c[6][3]

Exercise during Session (need not submit)

- Confirm that
  - Two functions *combination* and *combination_loop* return the same values for some k and n‘s

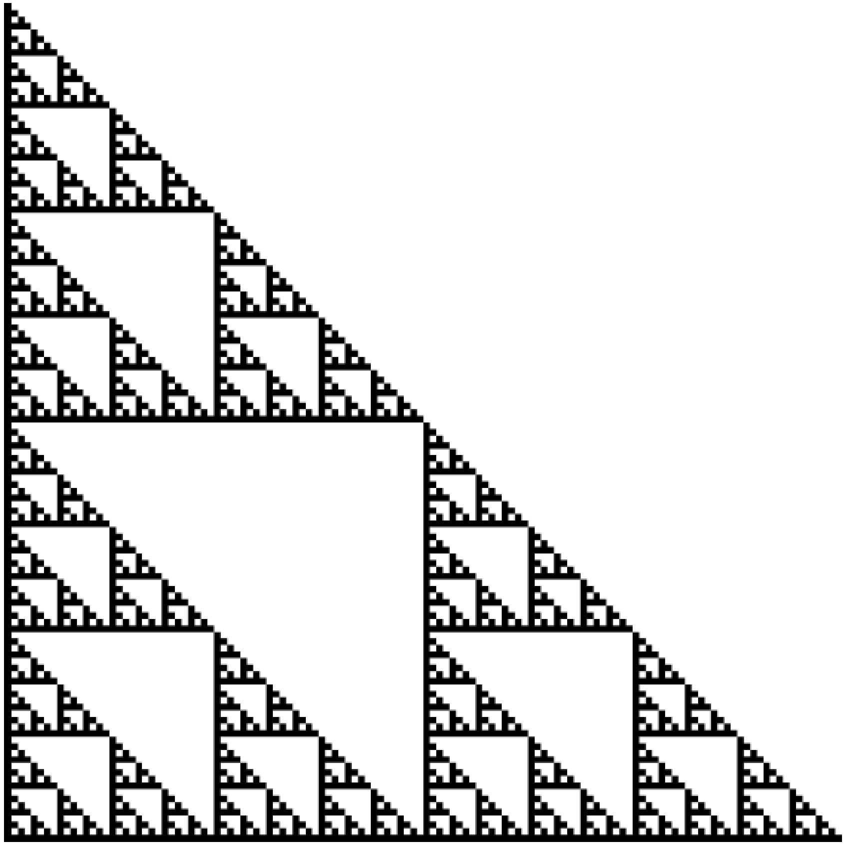
  - Compare the computation times for two functions when n and k are large
    - *combination*(n,100) & *combination_loop*(n,100) for n=100, 200, ···, 1000
      - Press Ctrl+C (and Return) to force-quit

Today's Contents: Recursion

- Review
  - Summation
- Number of Combinations
  - Using recursion
  - Using repetition
- Exercises
  - Sierpinski triangle
  - Tower of Hanoi

Exercise: Sierpinski Triangle

- Draw the following image using isrb



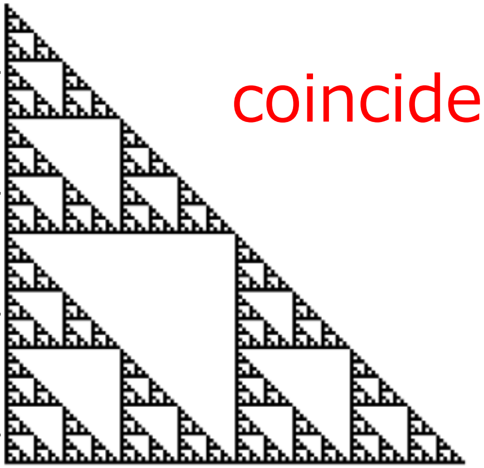
#Combinations

$n \setminus k$	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

The value of ${}_nC_k$

Remainders of n choose k when divided by 2

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



coincidence

1 1 1 1 1 1 1 1

1 0 0 0 0 0 0 0 1

Exercise1: Run sierpinski_loop(128)

- Modifying combination_loop, make a function that makes the image

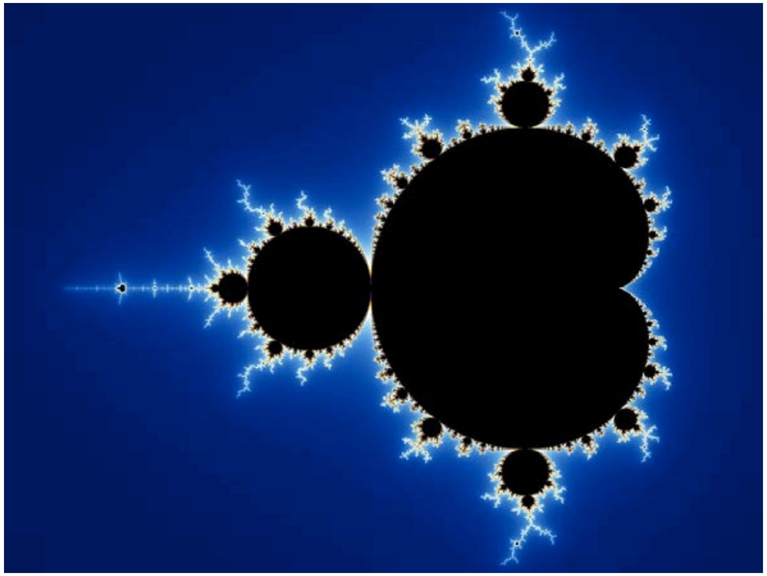
```
load ("./make2d.rb")
def sierpinski_loop(n)
    c = make2d(n+1,n+1)
    for i in 0..n
        c[i][0] = [ ? ]
        for j in 1..(i-1)
            [ ? ]
        end
        c[i][i] = [ ? ]
    end
    c
end
```

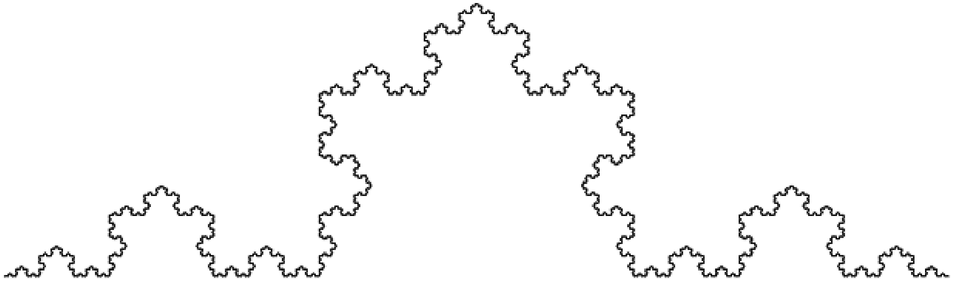
Add a few lines to flip 0 and 1:
for each entry in c
    c[i][j] = 1 – c[i][j]

Cf) Fractal

- http://en.wikipedia.org/wiki/Fractal

➢ Images with self-similarity

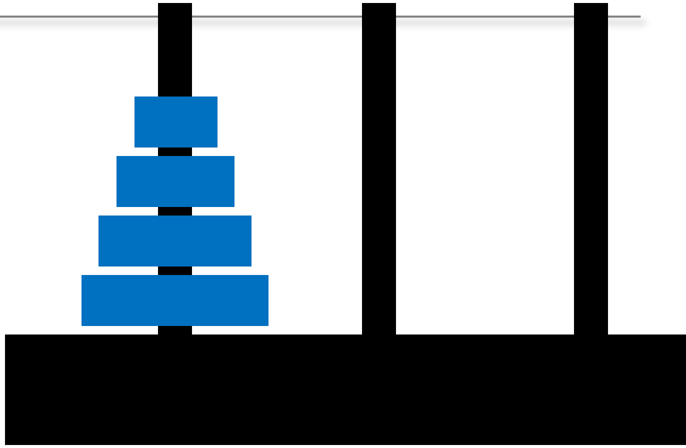




Today’s Contents: Recursion

- Review
  - Summation
- Number of Combinations
  - Using recursion
  - Using repetition
- Exercises
  - Sierpinski triangle
  - Tower of Hanoi

Exercise: Tower of Hanoi



- You can play at
  - http://www.mathsisfun.com/games/tower-of-hanoi-2.html
- Goal:
  - move all the discs from the left peg to the middle one
- Rule:
  - Only one disc may be moved at a time.
  - A disc can be placed either on an empty peg or on top of a larger disc.
- Try to move all the discs using the smallest number of moves possible.

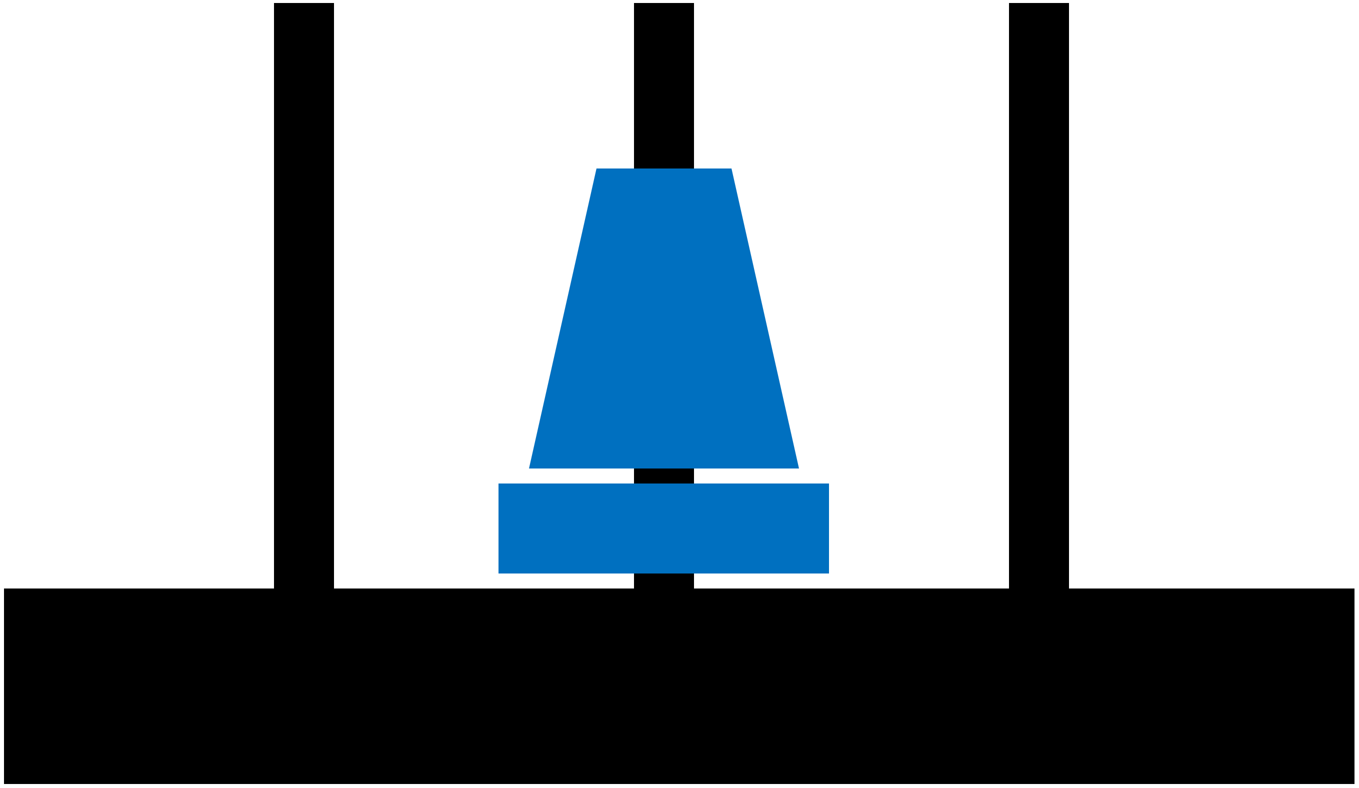
Tower of Hanoi: #disc = 2

Tower of Hanoi: #disc = 2

Tower of Hanoi: #disc = 2

Tower of Hanoi: #disc = 2

3 moves are enough



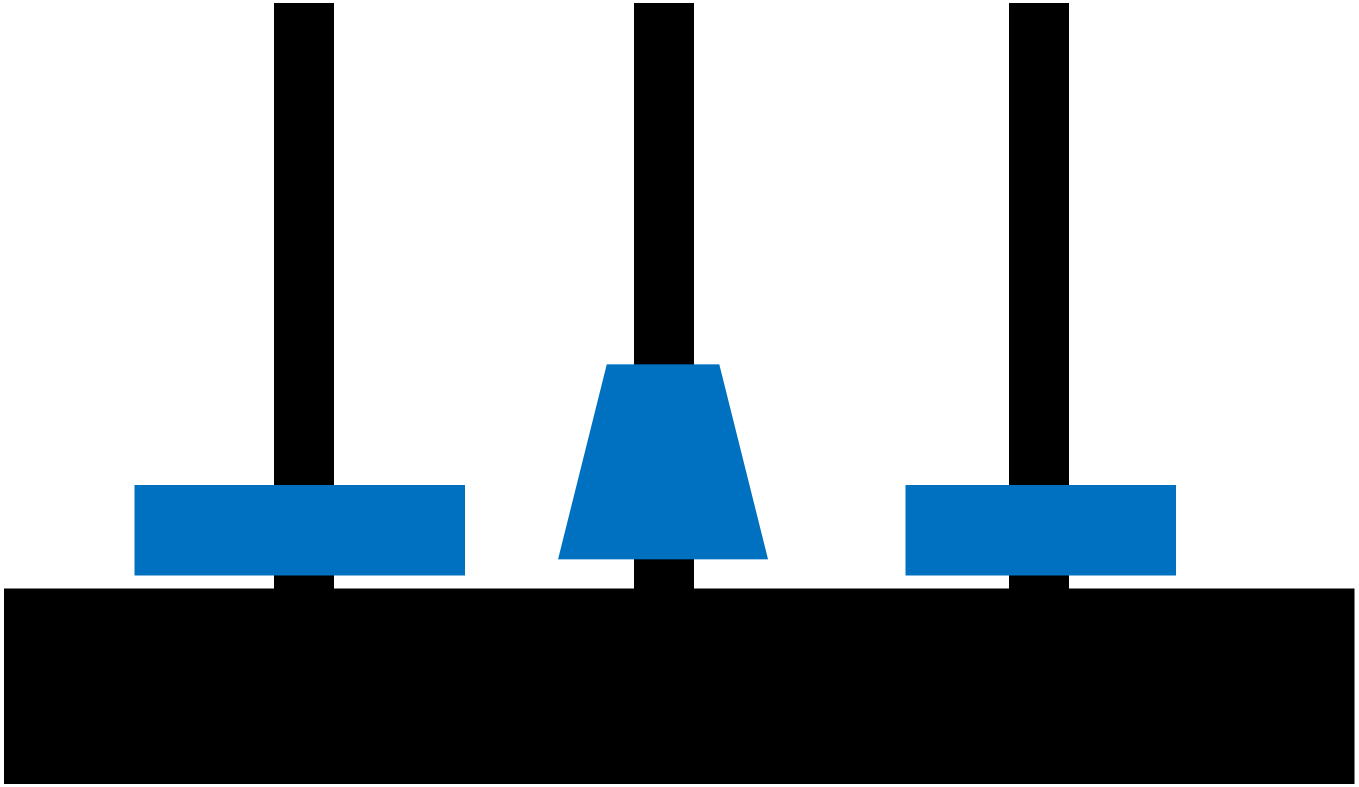
Tower of Hanoi: #disc = 3

Tower of Hanoi: #disc = 3

1 move

Tower of Hanoi: #disc = 3

2 move



Tower of Hanoi: #disc = 3

3 move

Tower of Hanoi: #disc = 3

4 move

Tower of Hanoi

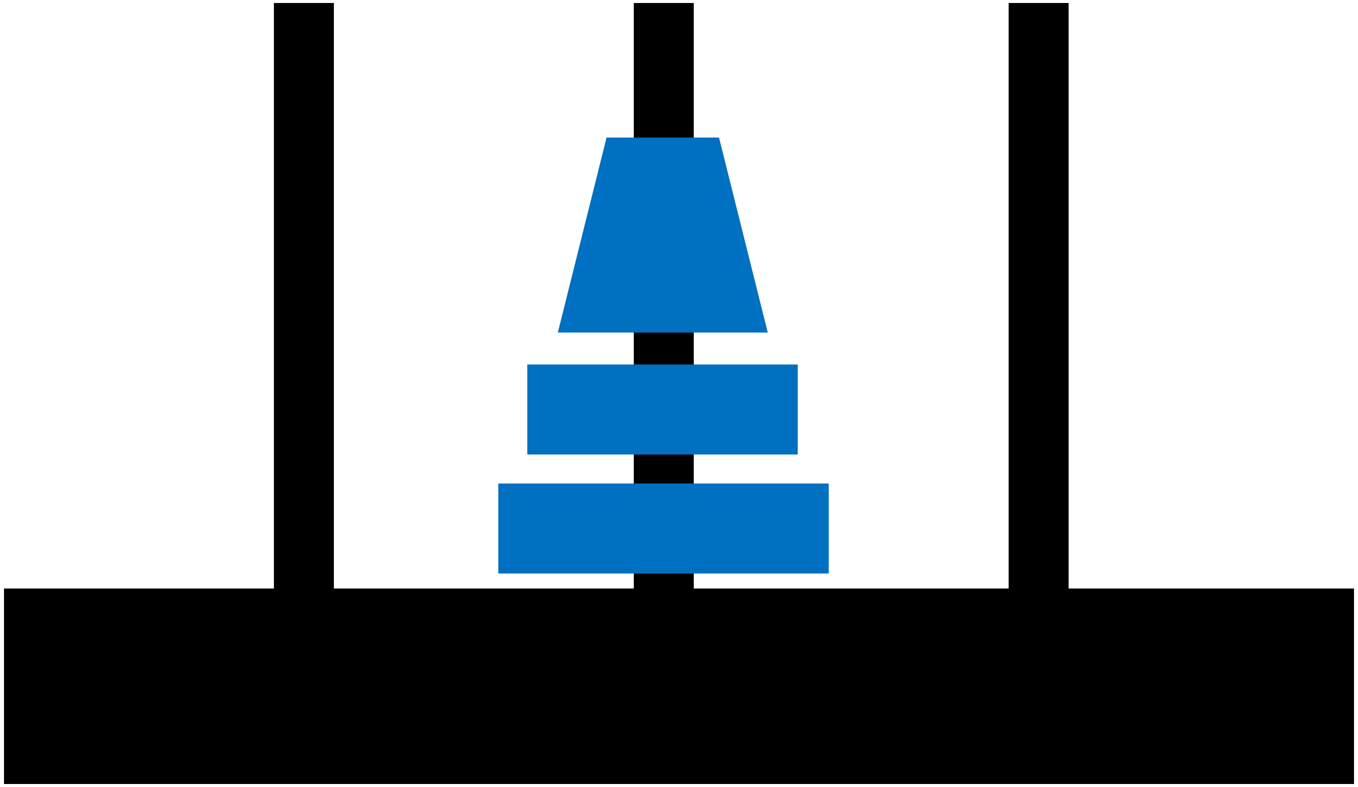
5 move

Tower of Hanoi

6 move

Tower of Hanoi: #disc = 3

7 moves: known to be minimum



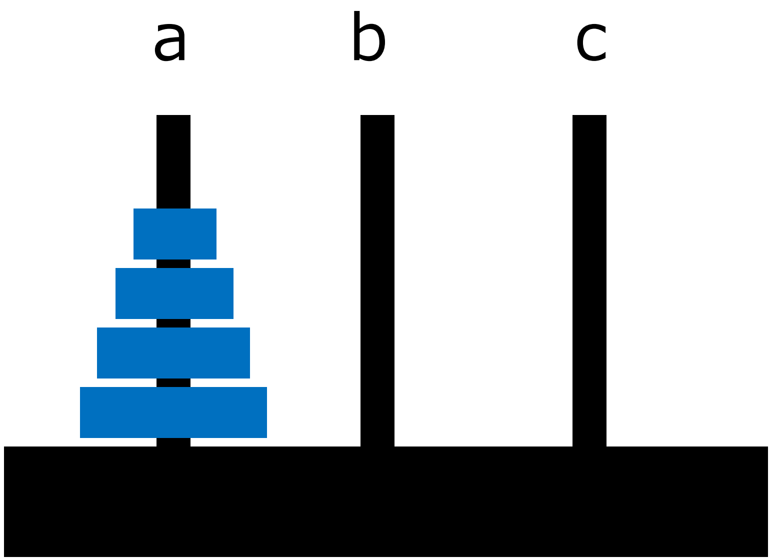
Exercises 2-1

- Try to solve the puzzle when n=4
  - By hand or
  - At
    - http://www.mathsisfun.com/games/tower-of-hanoi-2.html
    - http://www.softschools.com/games/logic_games/tower_of_hanoi/
- What is the minimum number necessary to move?
  - Explain briefly how you can obtain

Defining Ruby Functions

- Hanoi_times(n, a, b, c)
  - Minimum # times of moving disks when we do Hanoi(n, a, b, c)

- Hanoi(n, a, b, c)
  - Describe a procedure to move n disks from a to b



Exercise2-2:Complete the Programs


```
def hanoi_times(n)
  if n==0
      0
  else
     hanoi_times(____) + 1 + hanoi_times(____)
  end
end
```

Exercise2-3:Complete the Program


```
def hanoi(n, a, b, c)
    if n==1
        print “Move from “, a, “to ”, b, “¥n”
    else
        hanoi(____, __, __, __)
        print “Move from “, a , " to “, b , “¥n"
        hanoi(____, __, __, __)
    end
end
```

Examples of Outputs


```
>> hanoi(3,"a","b","c")
Move from a to b
Move from a to c
Move from b to c
Move from a to b
Move from c to a
Move from c to b
Move from a to b
```

You need "" in the parameter because a,b,c are characters. Otherwise they are variables

Exercise2-4:Comfirmation

- Try hanoi(4,"a","b","c") and compare your answer in Exercise2-1

- Hints are put at the end of the slides

Today's Exercise

- Solve
  - Sierpinski Triangle
    - Execute it on isrb2
    - submit an image and Ruby programs
  - Solve Tower of Hanoi (Exercise 2- 1--4)
    - Solve when n=4
    - Fill in the blanks and submit Ruby program
    - Confirm it works

  - If you have time
    - Optional quizzes of the last week
    - Past exam 2013, Problem 2 (a)—(d)

Deadline

- By **Nov. 23 (Wed) 23:59**
  - Through ITC-LMS
    - Don’t forget to send the outcome images

- Next week
  - How to estimate computational time
  - How to evaluate the performance of programs

Tower of Hanoi: #disc = n

Tower of Hanoi: #disc = n

Tower of Hanoi: #disc = n

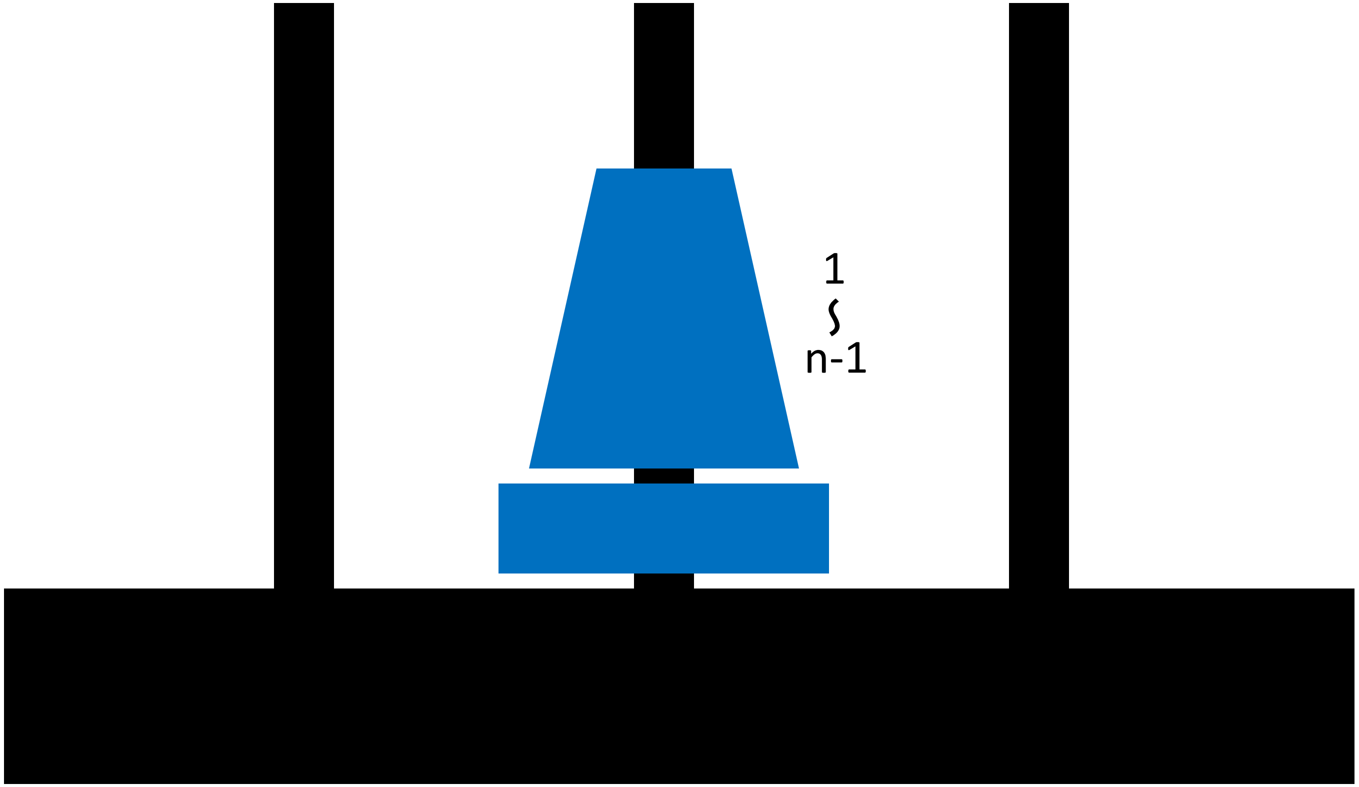
Move the first n-1 discs to the right

Tower of Hanoi

1 step to move the largest disc to the middle

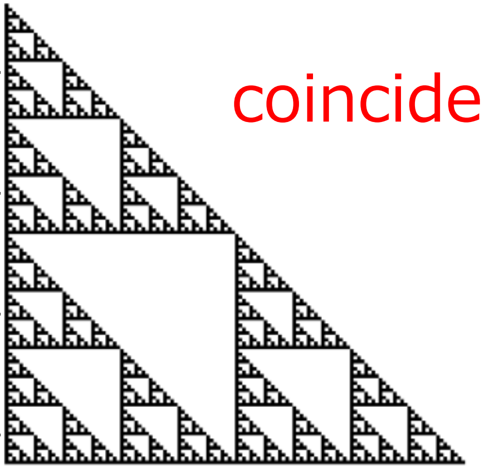
Tower of Hanoi

Move the first n-1 discs to the middle



About Sierpinski Triangle

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



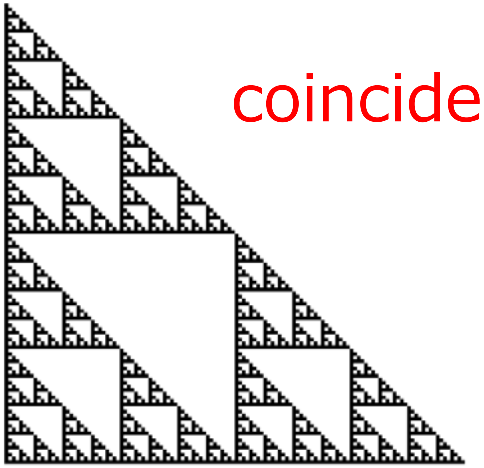
coincidence

1 1 1 1 1 1 1 1

1 0 0 0 0 0 0 0 1

About Sierpinski Triangle

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



coincidence

1 1 1 1 1 1 1 **1**

1 0 0 0 0 0 0 0 **1**

About Sierpinski Triangle


```
def sierpinski_loop(n)
  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])%2
    end
    c[i][i] = 1
  end
  for i in 0..n
    for j in 0..n
      c[i][j]=1-c[i][j]
    end
  end
  c
end
```

Another Solution: Using combination(,)


```
def sierpinski_loop(n)
  c = make2d(n+1,n+1)
  for i in 0..n
    c[i][0] = 1
    for j in 1..(i-1)
      c[i][j] =
        if !even(combination_loop(i,j)) then 1
        else 0
      end
    end
    c[i][i] = 1
  end
  c
end
```

Computational time becomes large (later in exercises)

Compute combination_loop(i, j).
If it is even, return 1, o/w, return 0

Remarks on Exercises

- Submit complete programs
  - not only the corresponding part, to check the correctness easily by copy&paste
  - not only answers, but images,
    - to give a partial point
    - You can write what you did

- Exercises' programs can be made by modifying sample programs

Remark

- We have more exercises
  - Downloadable from CFIVE
  - Some are same as in the sessions
  - Some are interesting
  - Some may be useful for better understanding

Ans1: Run sierpinski_loop(128)

- Modifying combination_loop, make a function that makes the image

```
load ("./make2d.rb")
def sierpinski_loop(n)
    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])%2
        end
        c[i][i] = 1
    end
    c
end
```

For flipping 0 and 1, we need:

```
for each entry in c
    c[i][j] = 1 – c[i][j]
```

Answer


```
def hanoi(n, a, b, c)
    if n==1
        print "Move from ", a, " to ", b, "¥n"
    else
        hanoi(n-1, a, c, b)
        print "Move from ", a, " to ", b, "¥n"
        hanoi(n-1, c, b, a)
    end
end
```

Answer


```
def hanoi_times(n)
    if n==0
        0
    else
        hanoi_times(n-1) + 1 + hanoi_times(n-1)
    end
end
```

Converting to binary – basic

- Arrange the remainder when divided by 2 in reverse

25 → 11001

12	…	1
6	…	0
3	…	0
1	…	1
0	…	1

Converting to binary – repetition


```
def binary_loop(n)
    s = ""
    while n >= 1
        s = s + (n%2).to_s()
        n = n / 2
    end
    s
end
```

```
irb(main):007:0> binary_loop(25)
=> "10011"
```

Converting to Binary – recursion


```
def binary(n)
    if n < 2
        n.to_s()
    else
        binary(n/2) + (n%2).to_s()
    end
end
```

```
irb(main):009:0> binary(25)
=> "11001"
```

2進数への変換 – 反復版


```
def binary_good(n)
    s = ""
    while n >= 1
        s = (n%2).to_s() + s
        n = n / 2
    end
    s
end
```

```
irb(main):008:0> binary_loop(25)
=> "11001"
```

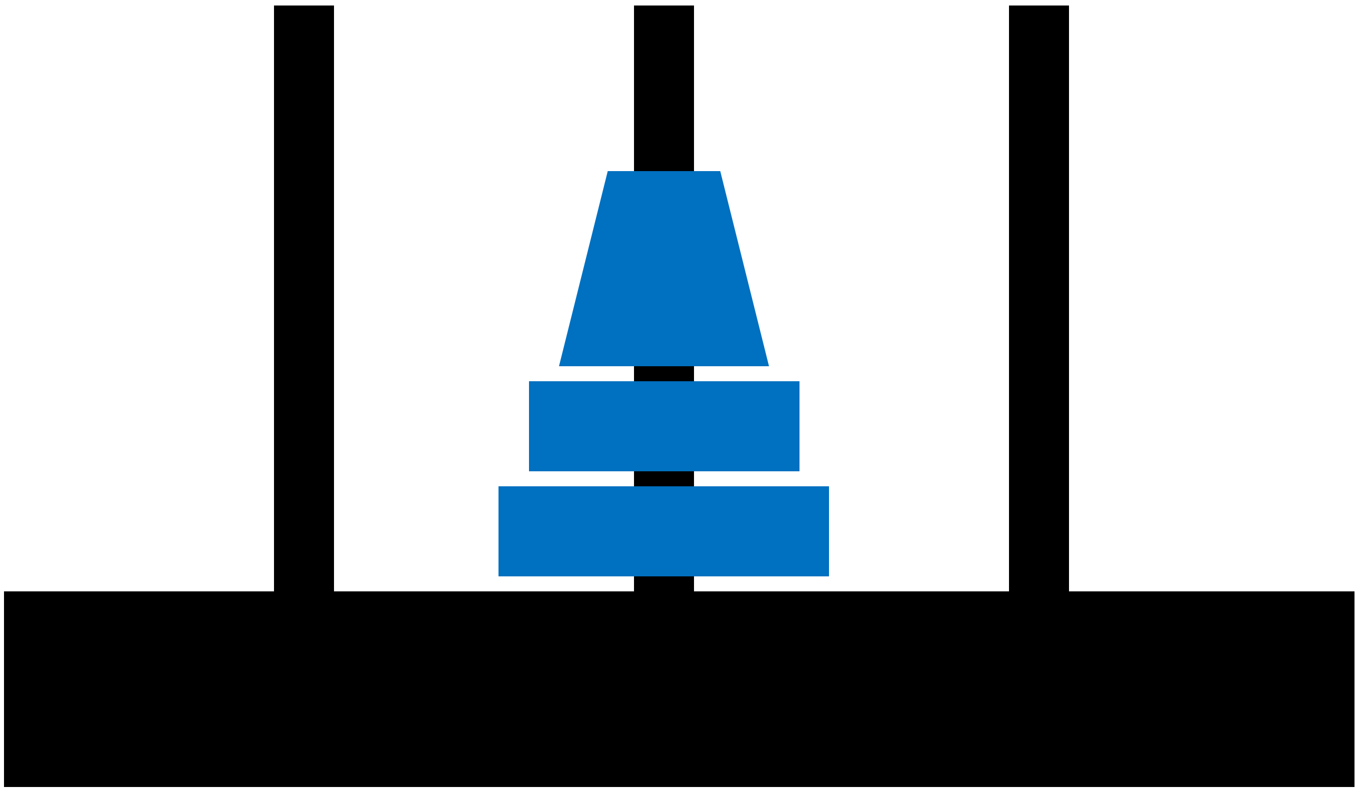
Tower of Hanoi

Tower of Hanoi

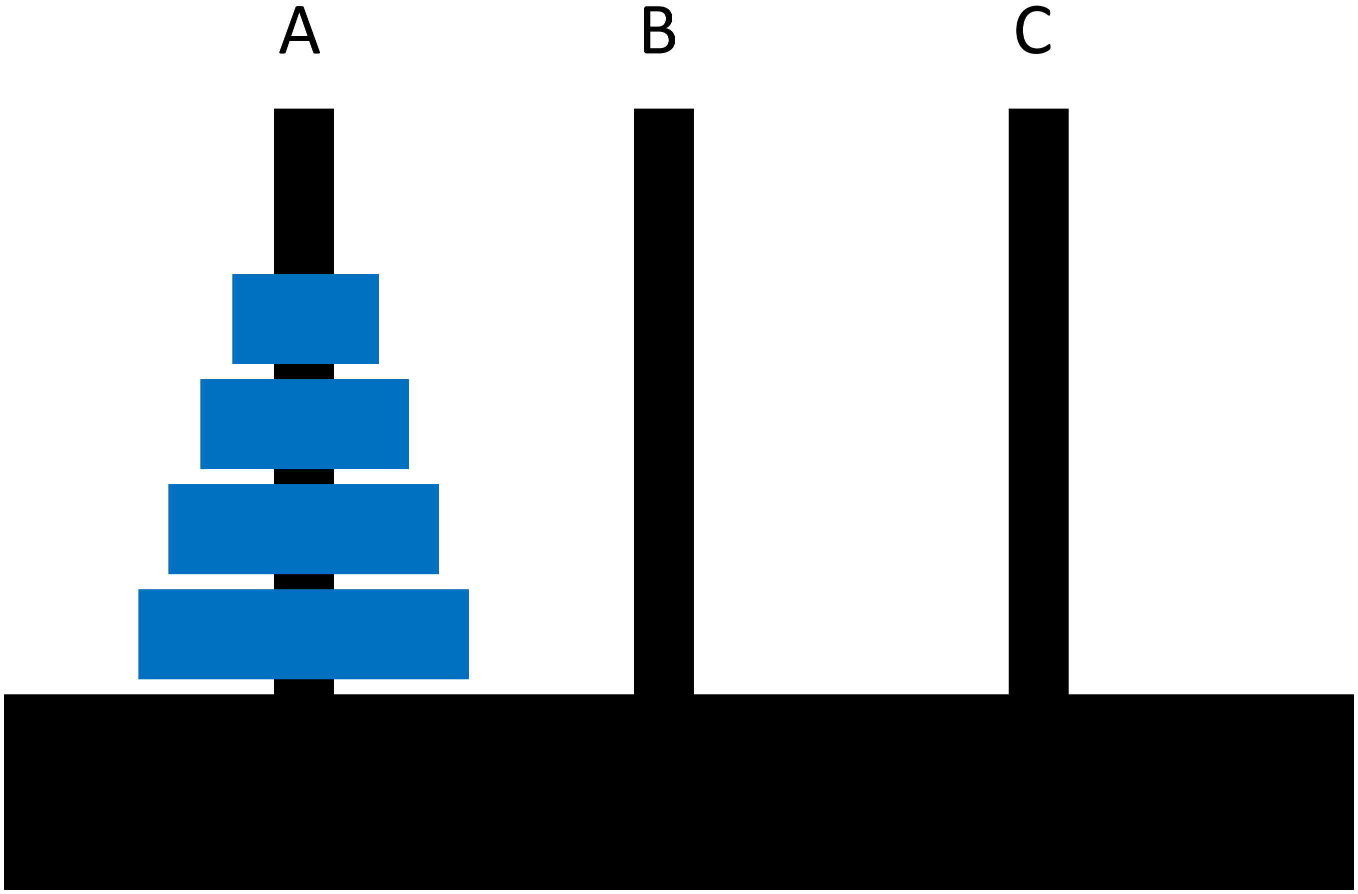
Tower of Hanoi

Tower of Hanoi

Tower of Hanoi



Tower of Hanoi



レポート

- 課題
  - 再帰で絵を描いてみよう
    - Sierpinski のカーペット・三角形
  - 再帰でハノイの塔の移動回数を解いてみよう
  - 再帰でハノイの塔を解いてみよう
- 提出先

  CFIVE
- 〆切

  12月 1日 授業開始前

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
```

Make an array with size 3**n
All entries are 0

Call subprocedure subcantor
for the point 0

Return a

Set to the array a the
Cantor set of order n
around x

If n is 0, it is just 1

Recursive calling.
Notice its origin

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

cantor(2)

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

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

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

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

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

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

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

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

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

	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
```

	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
    subcantor(a,1,0+2*3**1)
```

	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
    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
```

	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
      subcantor(a,0,6+2*3**0)
```

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

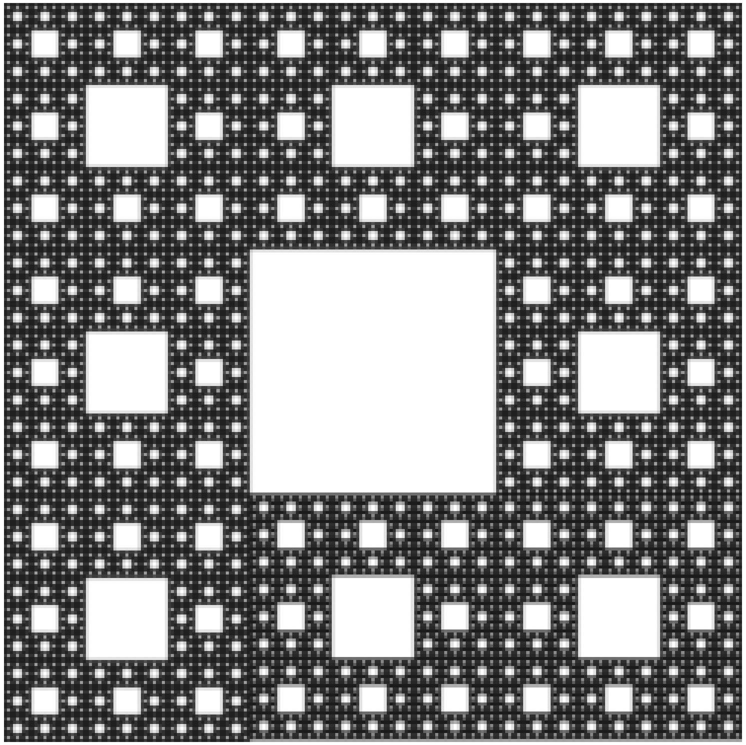
- Delete 1/3 of interval [0,1]
- For the remaining two interval, do the same.
- Repeat the same thing infinite times
- The remaining set is called the Cantor set.

- It is known that
- しらべる
- the measure(length) of the Cantor set is 0.
- しかし、濃度（点の数）は、もとの区間 [0,1] の実数に等しい。

Exercise: Sierpinski Carpet

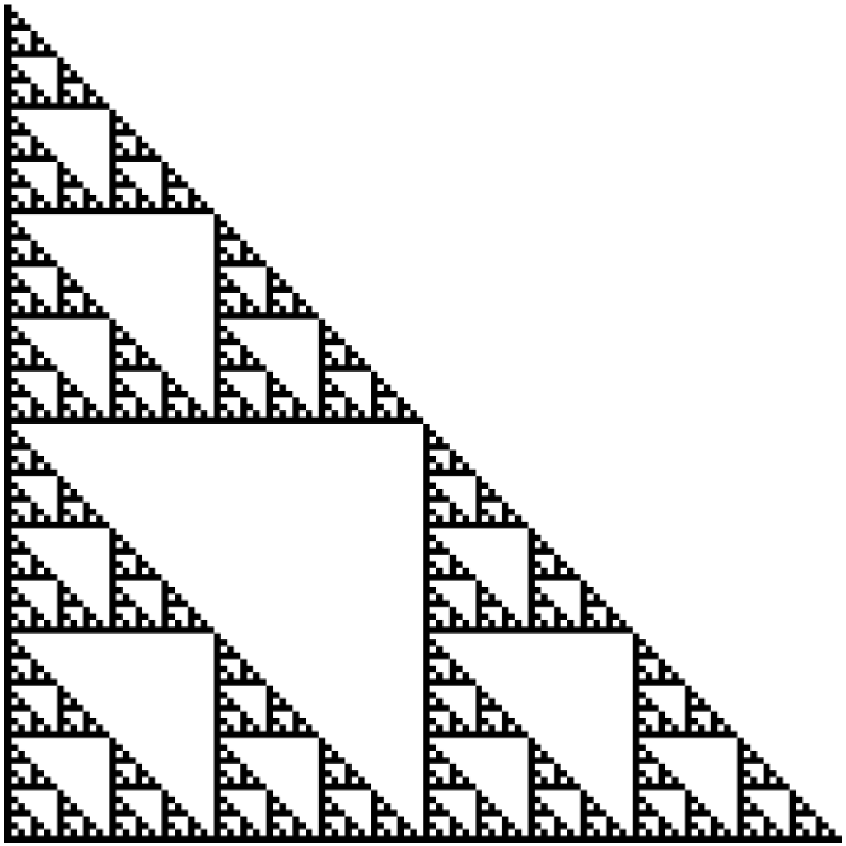
Cantor set の 2 次元版。
cf. Cantor dust

- Sierpinski のカーペット
  - n 次のカーペットは、縦横が 3**n で、n-1 次ののカーペットを 8 枚敷き詰めて作られる。真ん中は空いている。0 次のカーペットは、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 次の三角形は、縦横が 2**n で、n-1 次のの三角形を 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。）



Difference between Variables and Functions

- When we use a function, do not forget to specify parameters

- Let Hanoi(n) be a function that computes the minimum number of steps to move n discs to the right. Describe a recursive relation of the function.
- Express Hanoi(n) not recursively only using n.
- Make the function Hanoi(n), and compute Hanoi(4), Hanoi(5), Hanoi(6), and Hanoi(64).