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

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

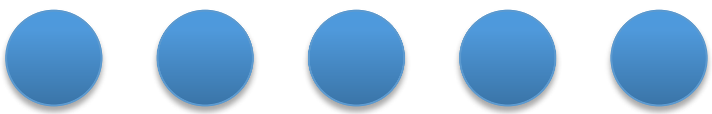
Today's Contents: Recursion

- 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

A B C D E



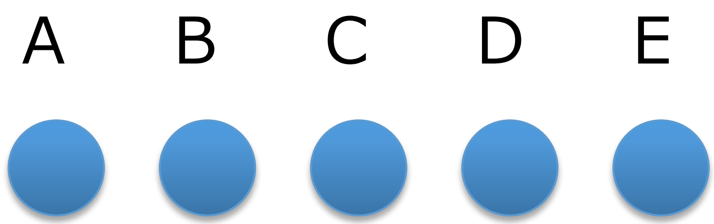
Ex. choose 2 items out of the 5 elements

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

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

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 ${}_nC_k$

```ruby
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

$c[i][i] = 1$

i=3

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

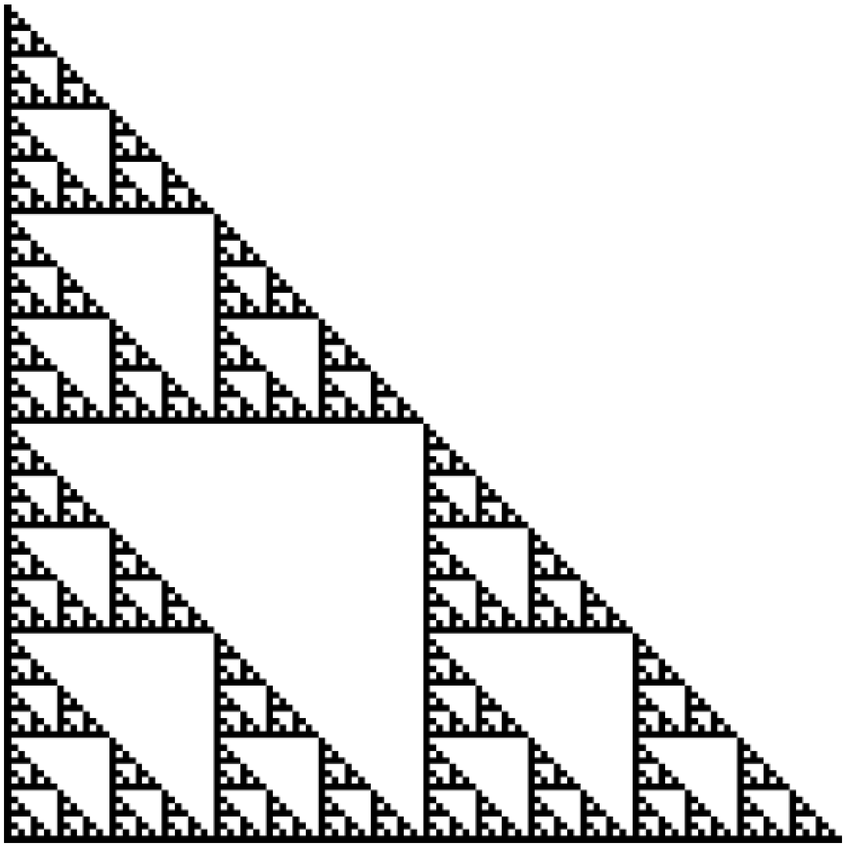
  - Compare the computation times for two functions when n and k are large
    - *combination*(n,100) & *combination_loop*(n,100) for n=100, 200, ⋯, 1000
      - Press Ctrl+C (and Return) to force-quit

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



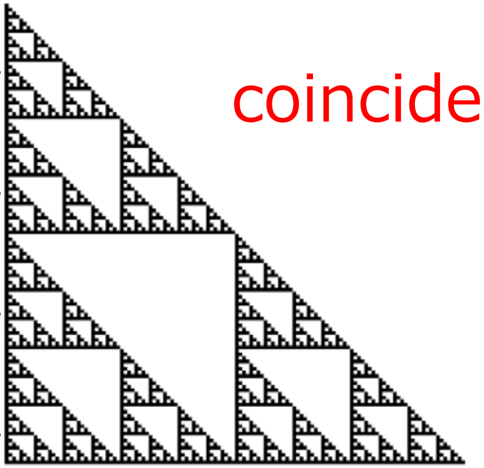
#Combinations

$n \setminus k$	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

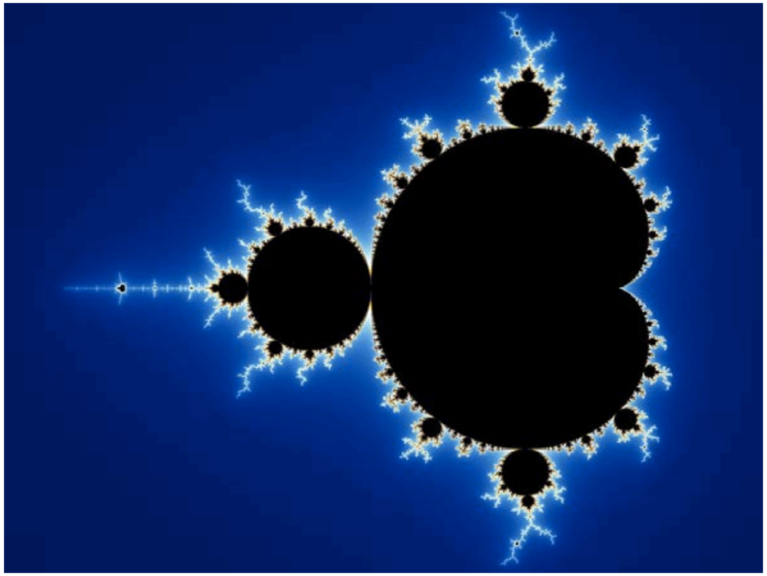
```
load ("./make2d.rb")
def sierpinski_loop(n)
    c = make2d(n+1,n+1)
    for i in 0..n
        c[i][0] = [ ? ]
        for j in 1..(i-1)
            [ ? ]
        end
        c[i][i] = [ ? ]
    end
    c
end
```

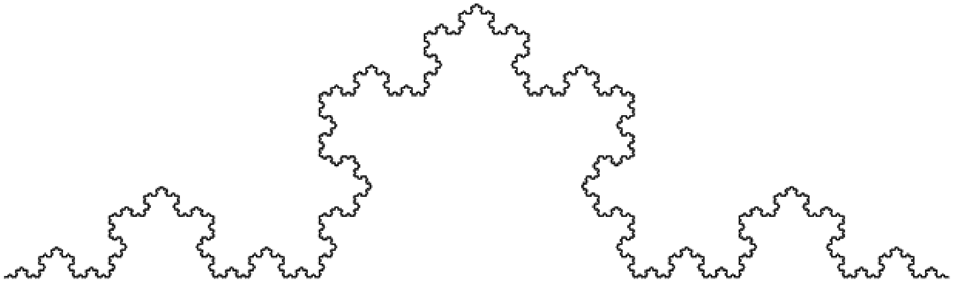
Add a few lines to flip 0 and 1:
for each entry in c
    c[i][j] = 1 – c[i][j]

Cf) Fractal

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

➢ Images with self-similarity

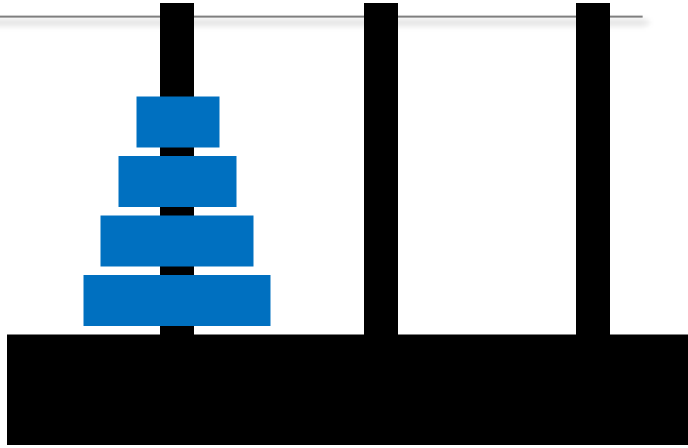




Today’s Contents: Recursion

- 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.

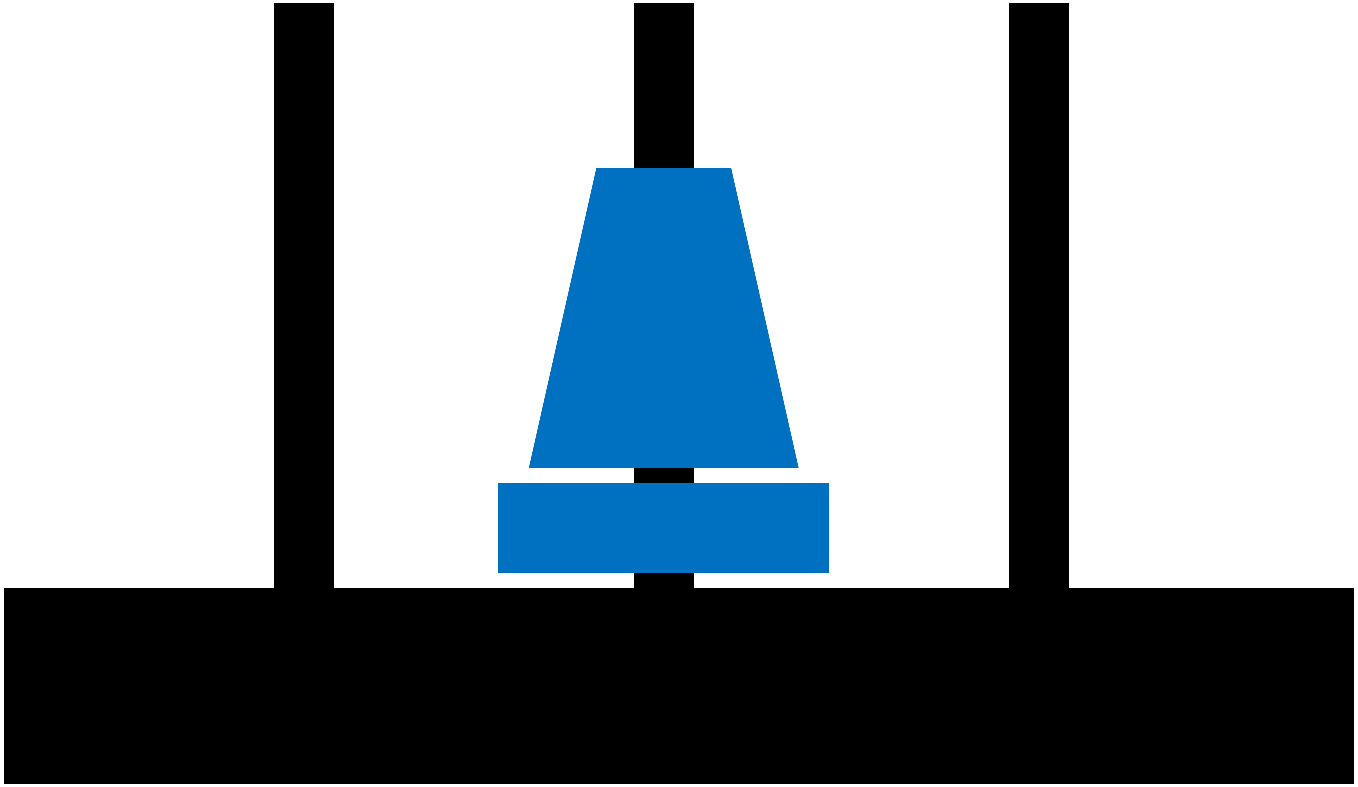
Tower of Hanoi: #disc = 2

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

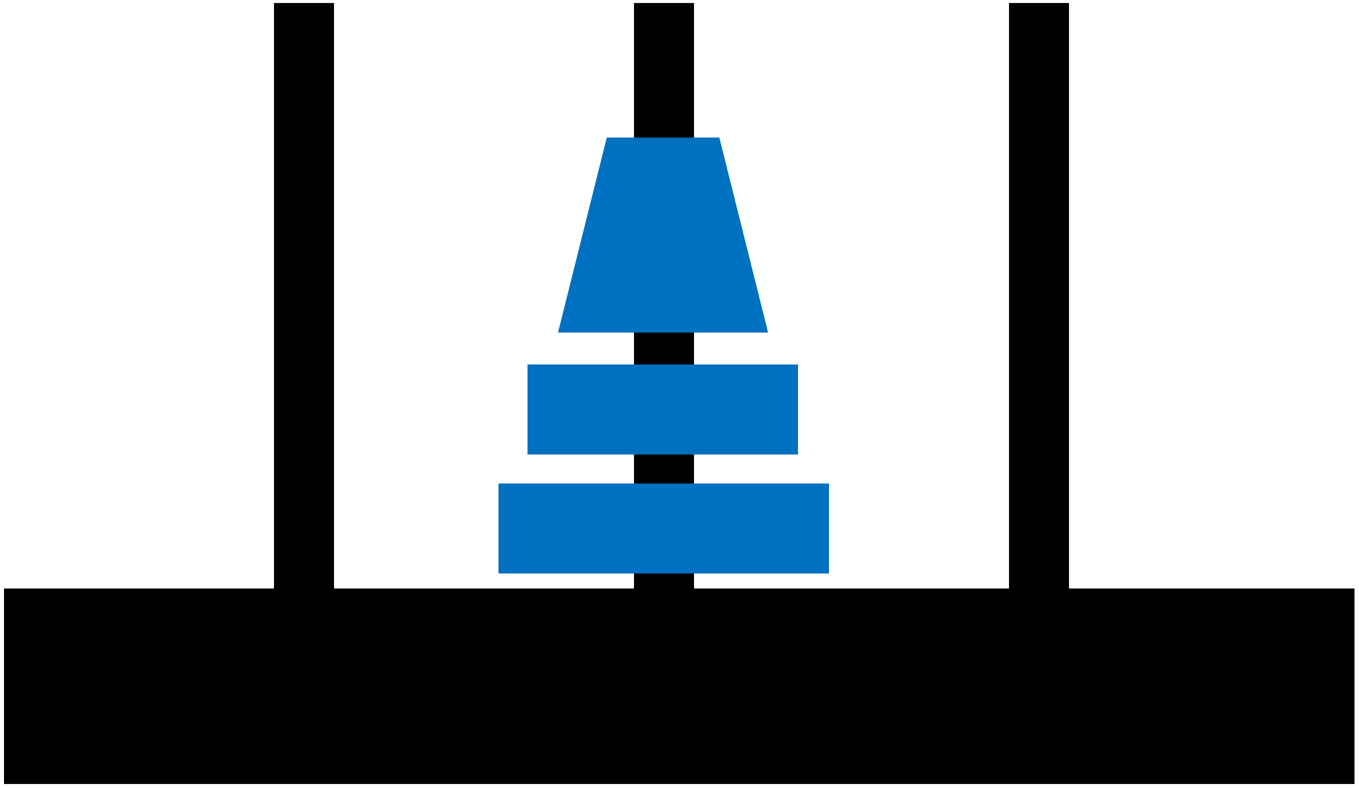
5 move

Tower of Hanoi

6 move

Tower of Hanoi: #disc = 3

7 moves: known to be minimum



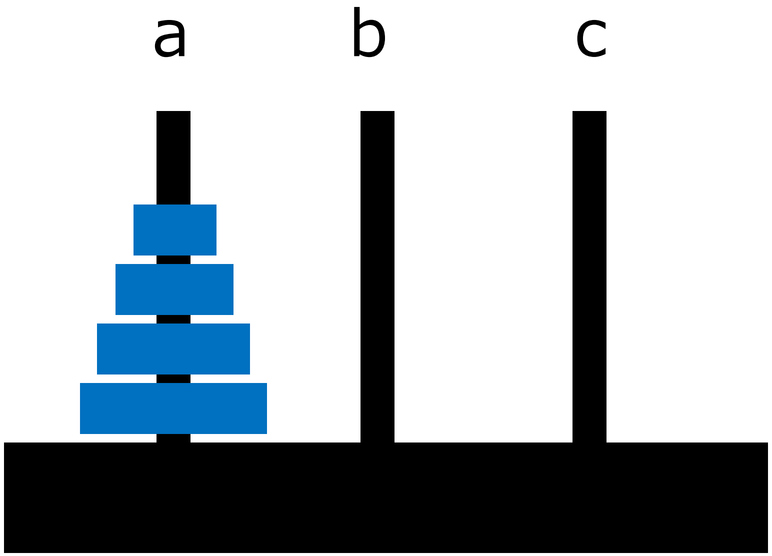
Exercises 2-1

- 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