The Royal Institution of Great Britain

The play within the play

Computer Science Masterclass

Gustavo Lau

The play within the play

Computer Science Masterclass

Gustavo Lau

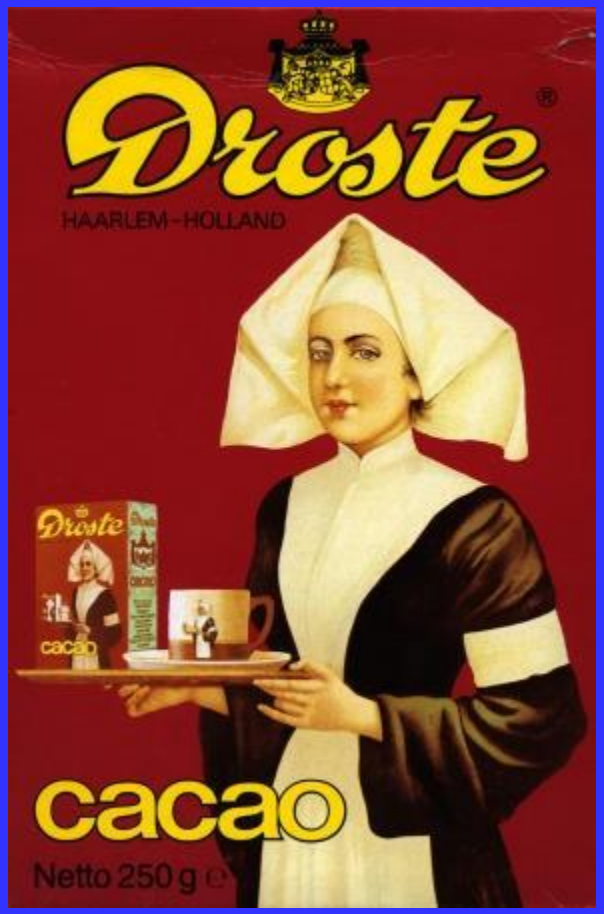
Self-reference - Mise en abyme

Velázquez,
Las Meninas

Self-reference

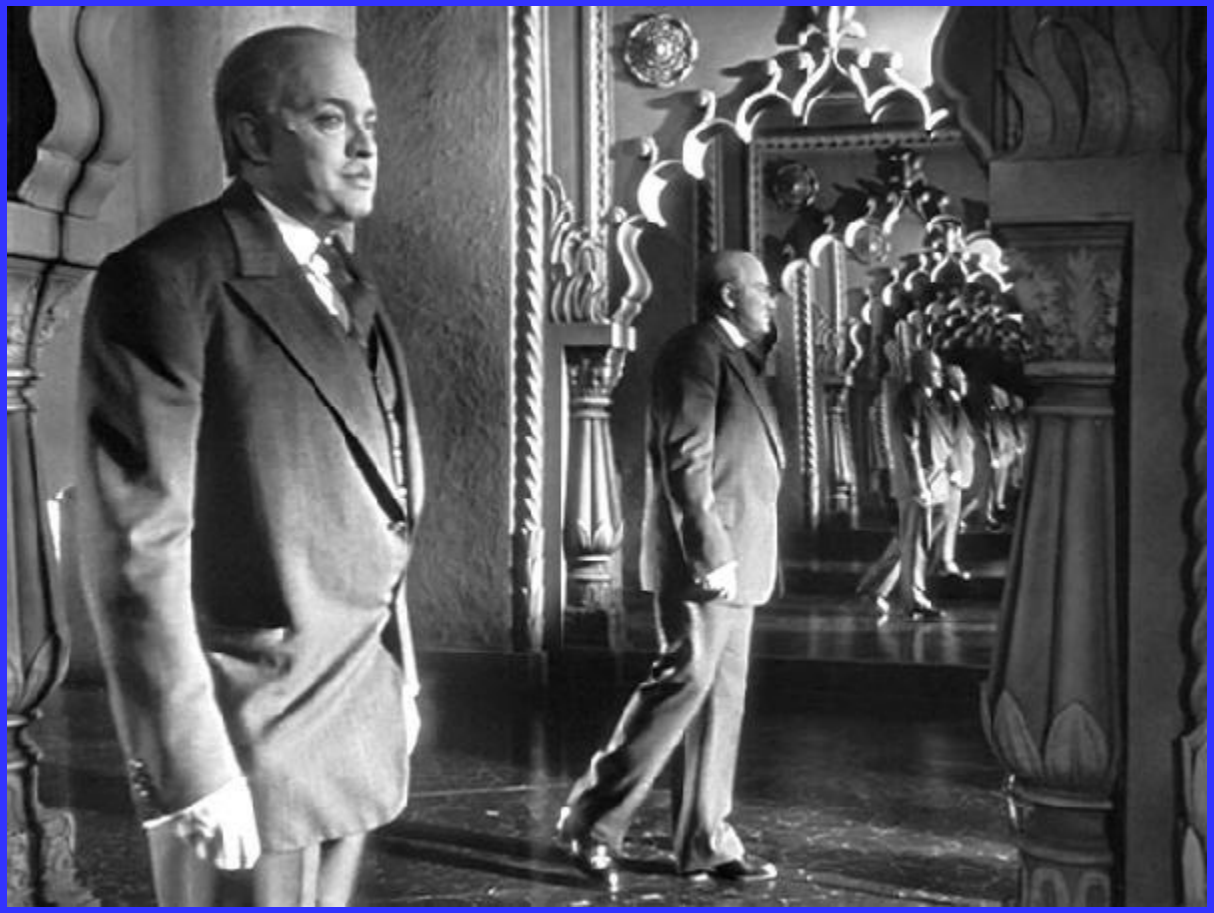
Russian or Matryoshka dolls

Self-reference



en.wikipedia.org/wiki/Droste_effect

Self-reference



Orson Welles, *Citizen Kane*

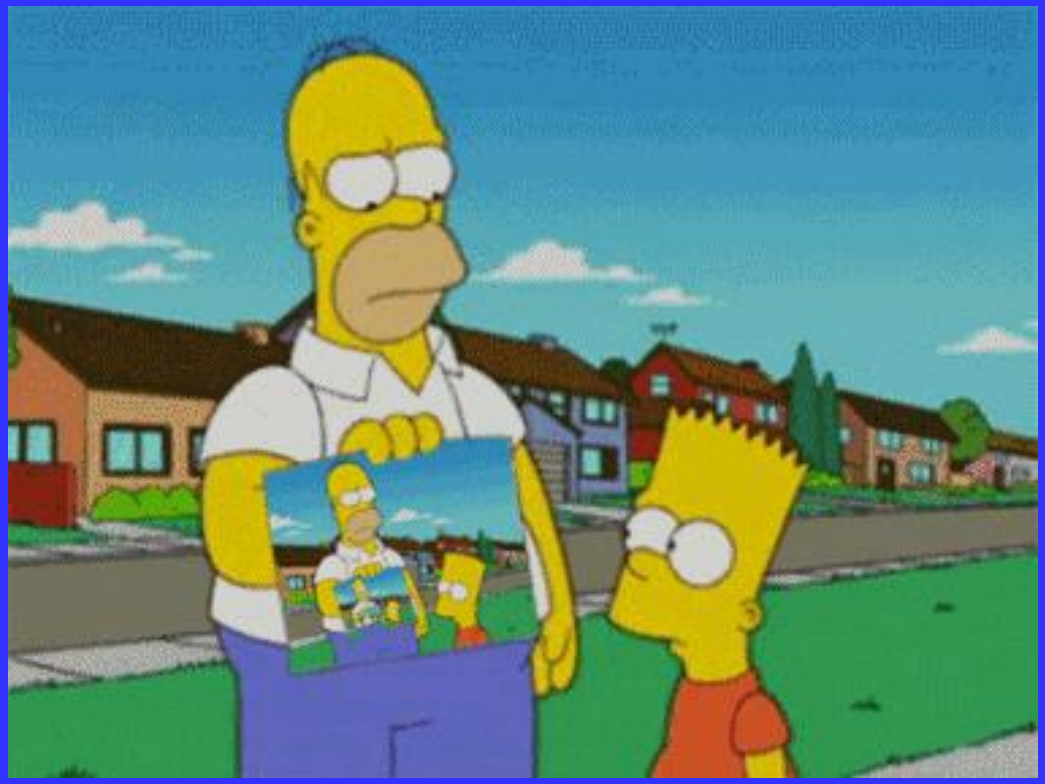
Self-reference

Google “recursive painting”

Self-reference

Wil Wheaton (Big Bang Theory), recursive t shirt

Self-reference



Google “simpsons recursion gif”

Self-reference

Model Villages of Bourton-on-the-Water, Gloucestershire

Self-reference

Google “recursive photo” images.

BTW, if you google recursion you get a recursive answer.

Worksheet 1: Self-reference

Induction

Consider the following:

$1^2 - 1 + 41 = 41$ is a prime

$2^2 - 2 + 41 = 43$ is a prime

$3^2 - 3 + 41 = 47$ is a prime

…

$30^2 - 30 + 41 = 911$, 911 is a prime

What could we conclude?

$n^2 - n + 41$ is a prime for all n

But $41^2 - 41 + 41 = 41^2$ is not prime.

Therefore that was not a conclusion, it was just a conjecture and it is false.

Mathematical induction

Consider the following equalities:

$1 = 1^2$

$1 + 3 = 2^2$

$1 + 3 + 5 = 3^2$

$1 + 3 + 5 + 7 = 4^2$

What could we conjecture?

That the sum of the first n odd numbers adds up to the n-th square number.

Mathematical induction

What is the formula for the sequence of the odd numbers?

Or how do we write the n-th odd number algebraically?

n	n-th odd number
1	1
2	3
3	5
…	
n	2n - 1

Mathematical induction

Consider the following equalities:

$1 = 1^2$

$1 + 3 = 2^2$

$1 + 3 + 5 = 3^2$

$1 + 3 + 5 + 7 = 4^2$

How do we write our conjecture algebraically?

Sum of the first n odd numbers n-th square number

$$1 + 3 + \ldots + (2n-1) = n^2$$

let's call this statement S(n).

How can we prove that S(n) is true for all n?

Mathematical induction

How can we prove that S(n) is true for all n?

We know that S(1), S(2), S(3) and S(4) are true.

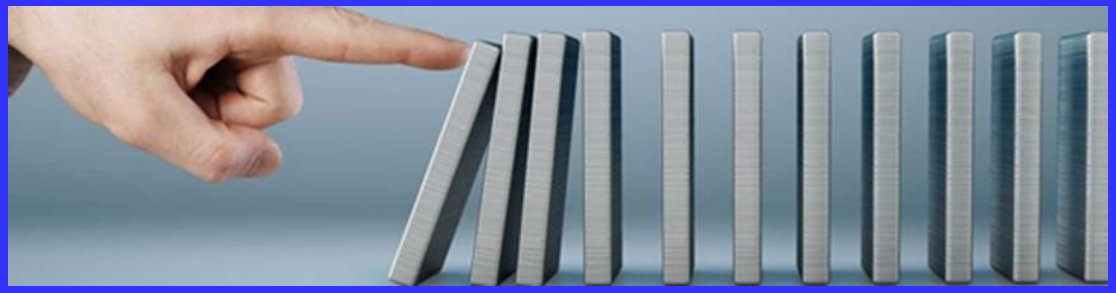
We will prove that:

$S(4) \Rightarrow S(5)$, if S(4) is true then S(5) is also true

$S(5) \Rightarrow S(6)$, if S(5) then S(6)

$S(6) \Rightarrow S(7)$, S(6) implies S(7)

In general we need to prove $S(k) \Rightarrow S(k+1)$ so that we get a domino effect:



Mathematical induction

How can we prove that S(n) is true for all n?

Let's assume that S(k) is true for a particular k:

$1 + 3 + \ldots + (2k-1) = k^2$

If we add the next odd number, 2k+1, to both sides we get:

$1 + 3 + \ldots + (2k-1) + (2k+1) = k^2 + 2k + 1$

$1 + 3 + \ldots + (2k-1) + (2(k+1)-1) = (k + 1)^2$

Then S(k+1) is true.

Therefore we have proved that if S(k) is true then S(k+1) is also true, that is $S(k) \Rightarrow S(k+1)$.

Mathematical induction

To prove by mathematical induction that a statement S(n) is true for all natural numbers you prove that:

1. S(1), or S(0), is true. This is called the **base case**.

2. If S(k) is true then S(k+1) is also true. This is called the **inductive step**.

If you take Further Maths A-level you'll study this there.

Computer Science recursion

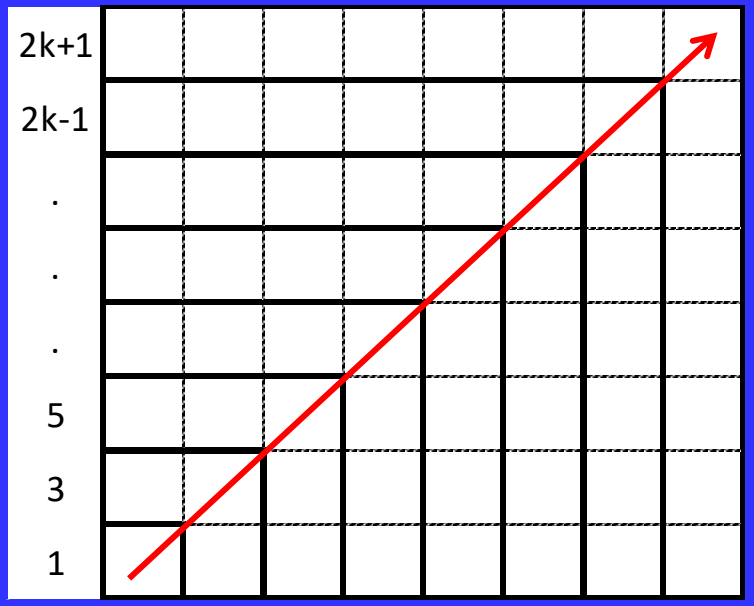
Wikipedia: Recursion in computer science is a method where **the solution to a problem depends on solutions to smaller instances of the same problem**. The approach can be applied to many types of problems, and **recursion is one of the central ideas of computer science**.

Maths induction and CS recursion

How is mathematical induction related to computer science recursion?

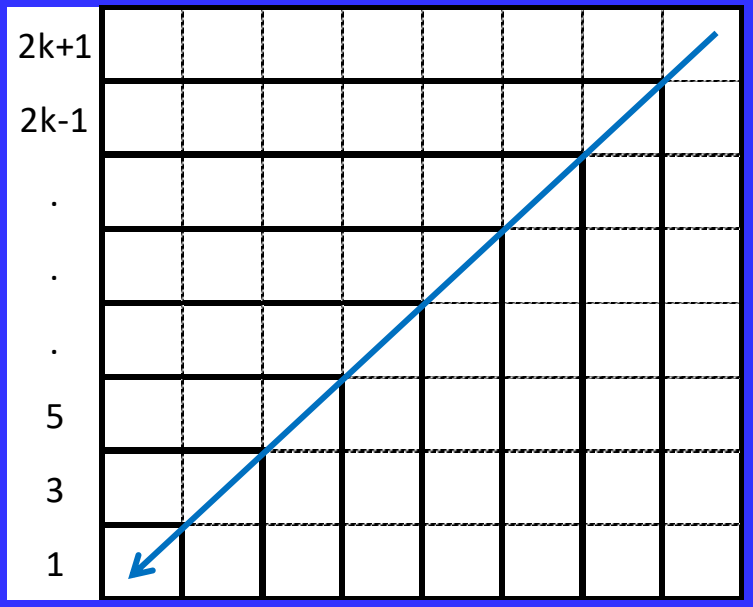
Mathematical induction:

From small case to bigger case.

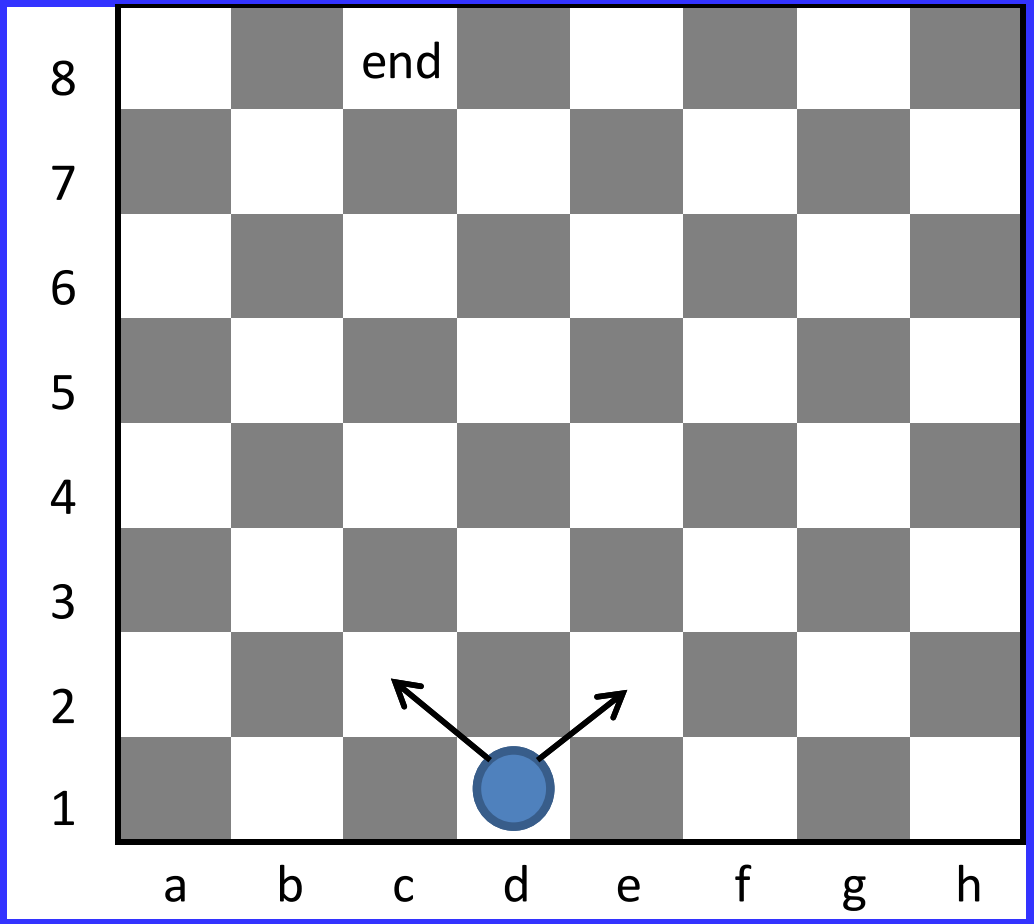


Computer Science recursion:

From big case to smaller case.



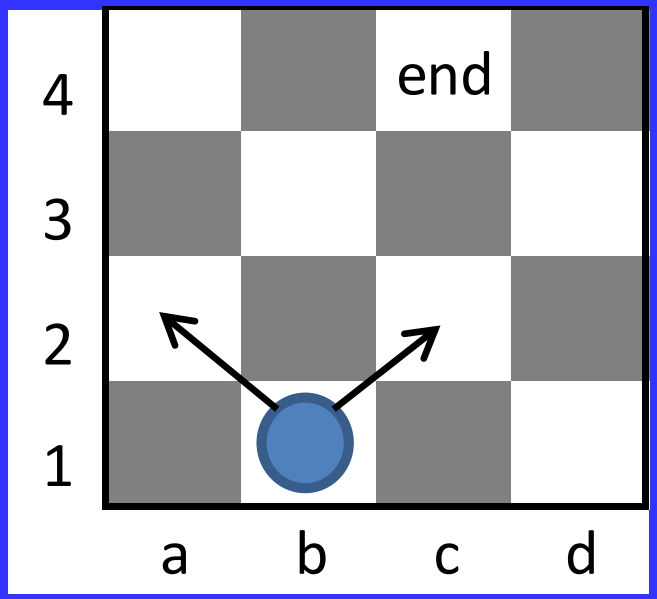
Worksheet 2
Draughts path counting



Draughts path counting

Maths problem solving principle:

Can you think of a smaller similar problem?

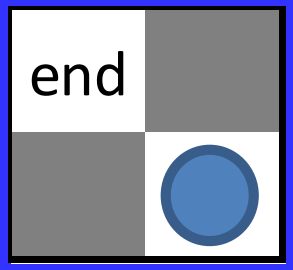


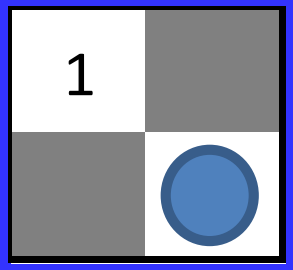
Draughts path counting

Maths problem solving principle:

Can you think of the smallest case?

More in general, can you think of an extreme case?





How many subsets does the empty set have?

We could call any of them the base case.

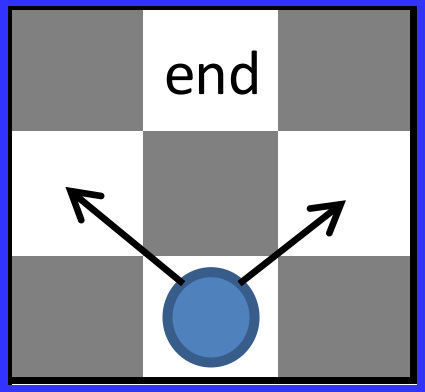
Draughts path counting

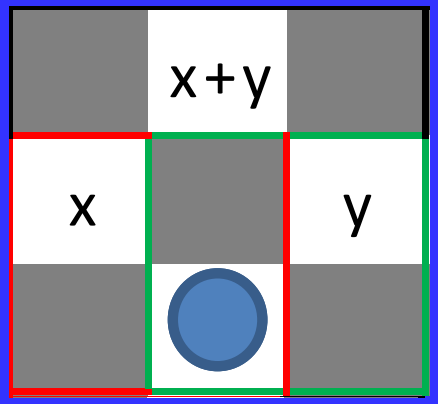
Maths problem solving technique:

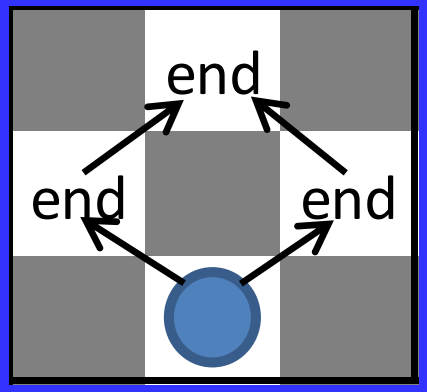
Can you think of the next smallest problem?

Divide and conquer principle:

Can you **decompose** the problem into similar smaller problems?



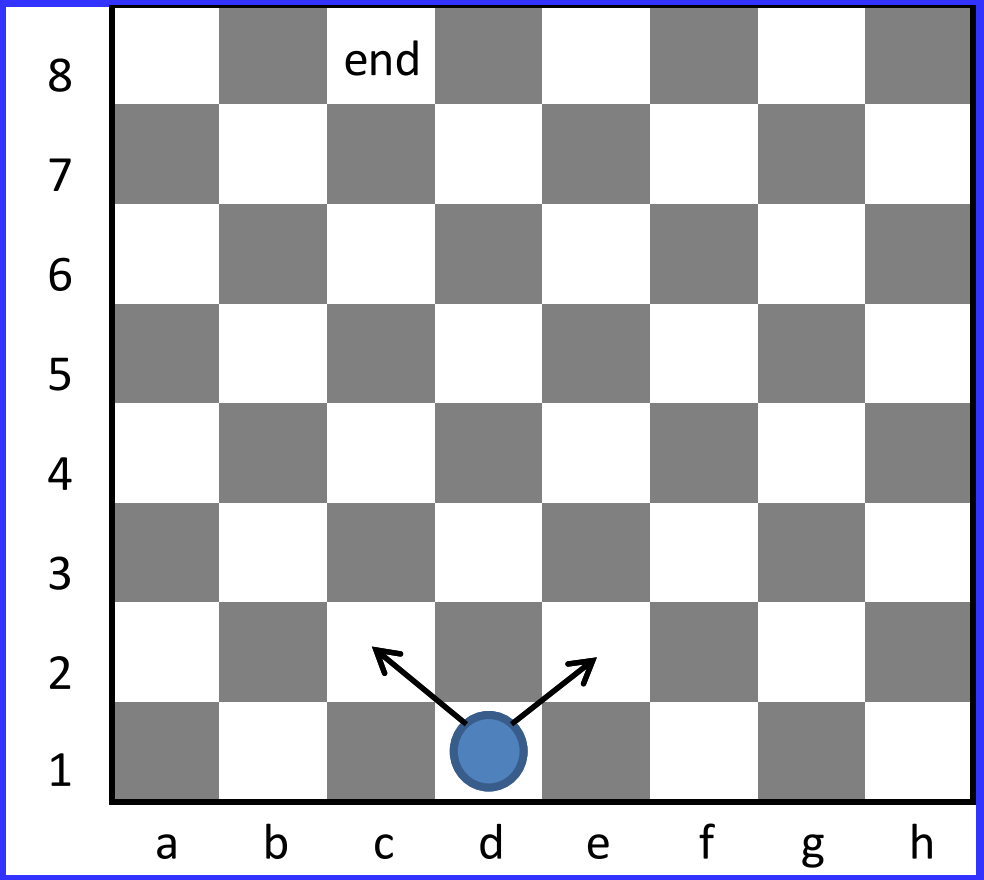




We could call this the inductive step.

Draughts path counting

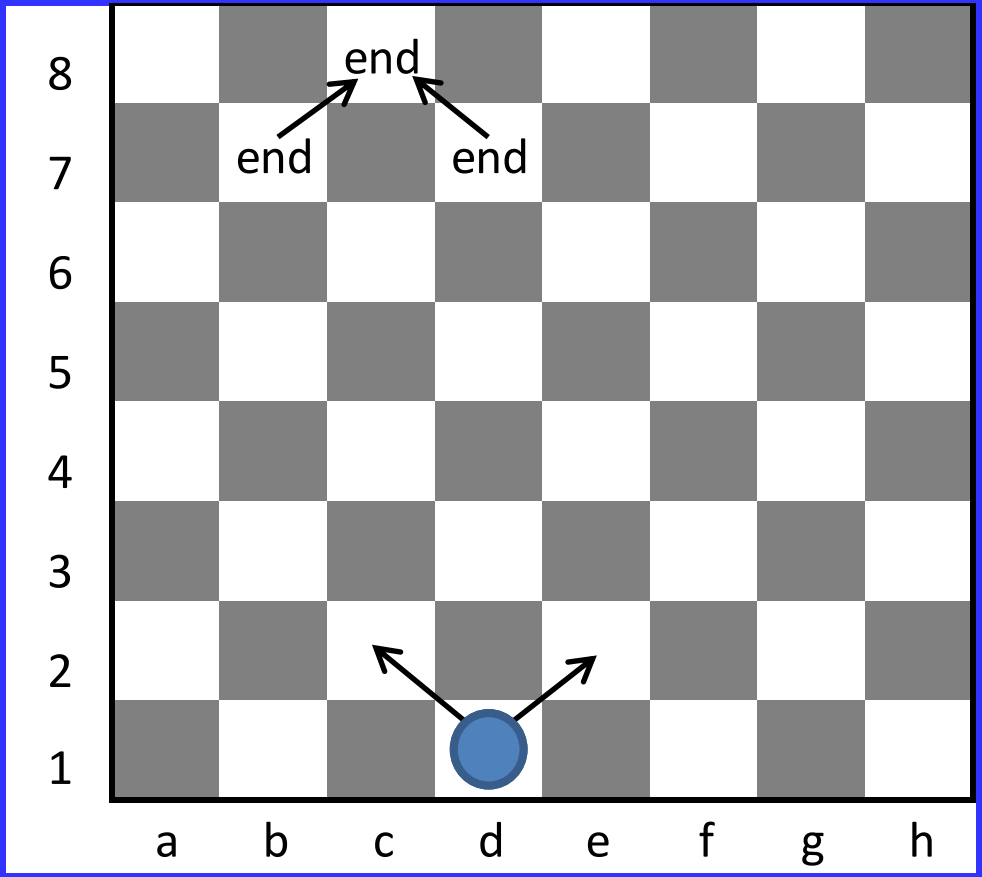
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

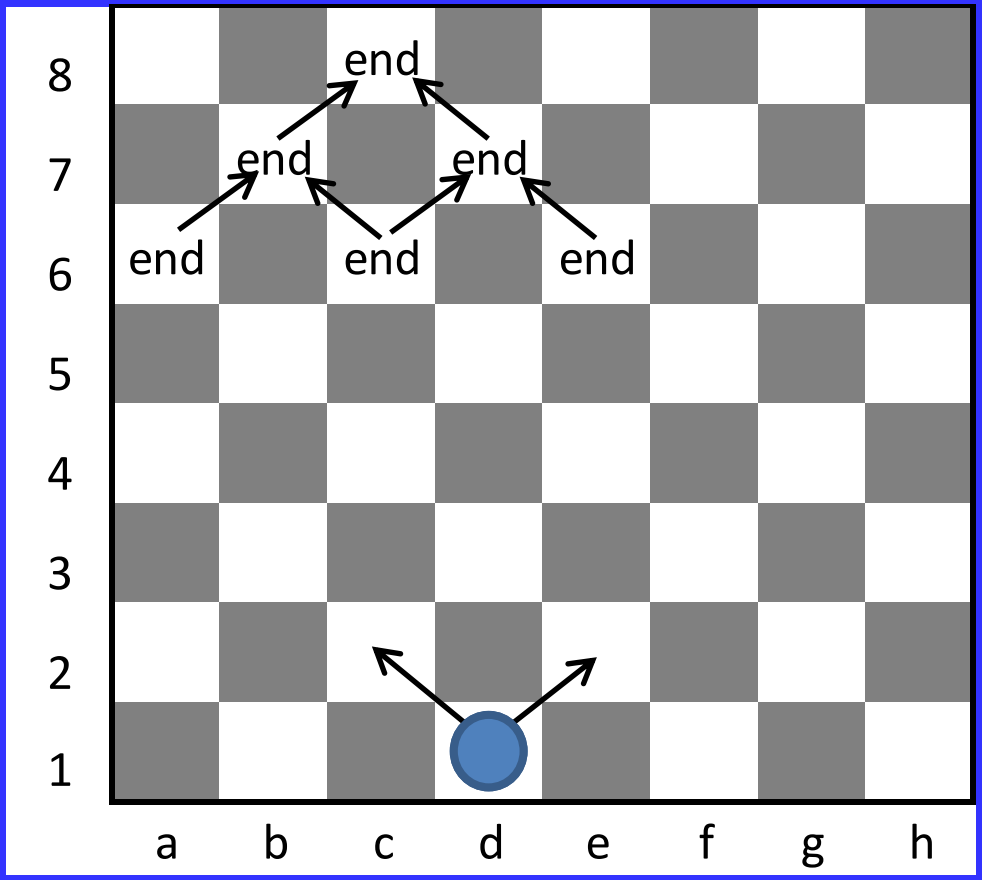
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

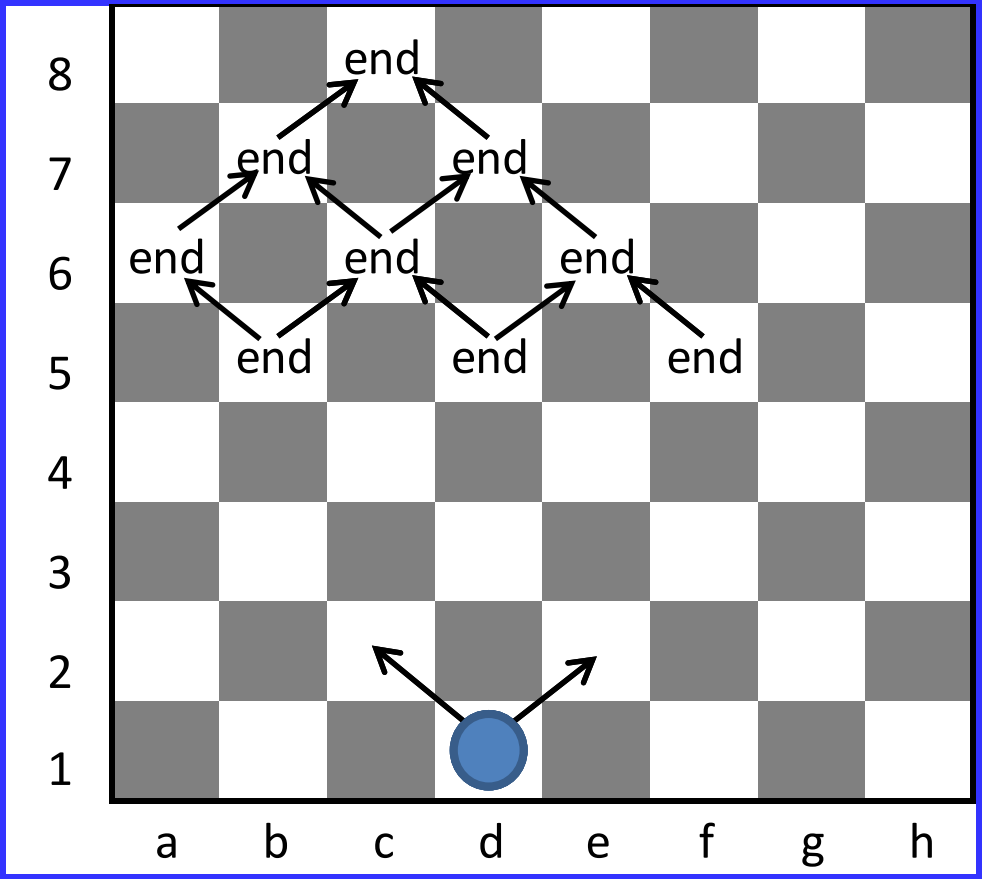
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

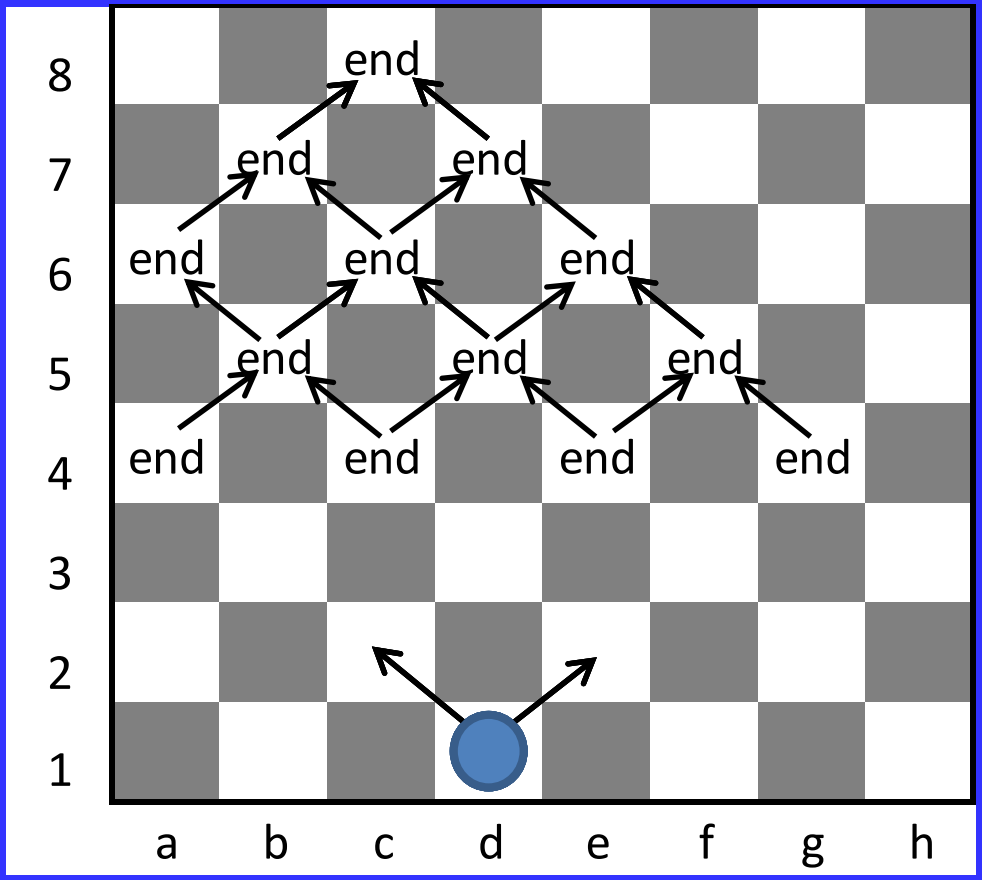
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

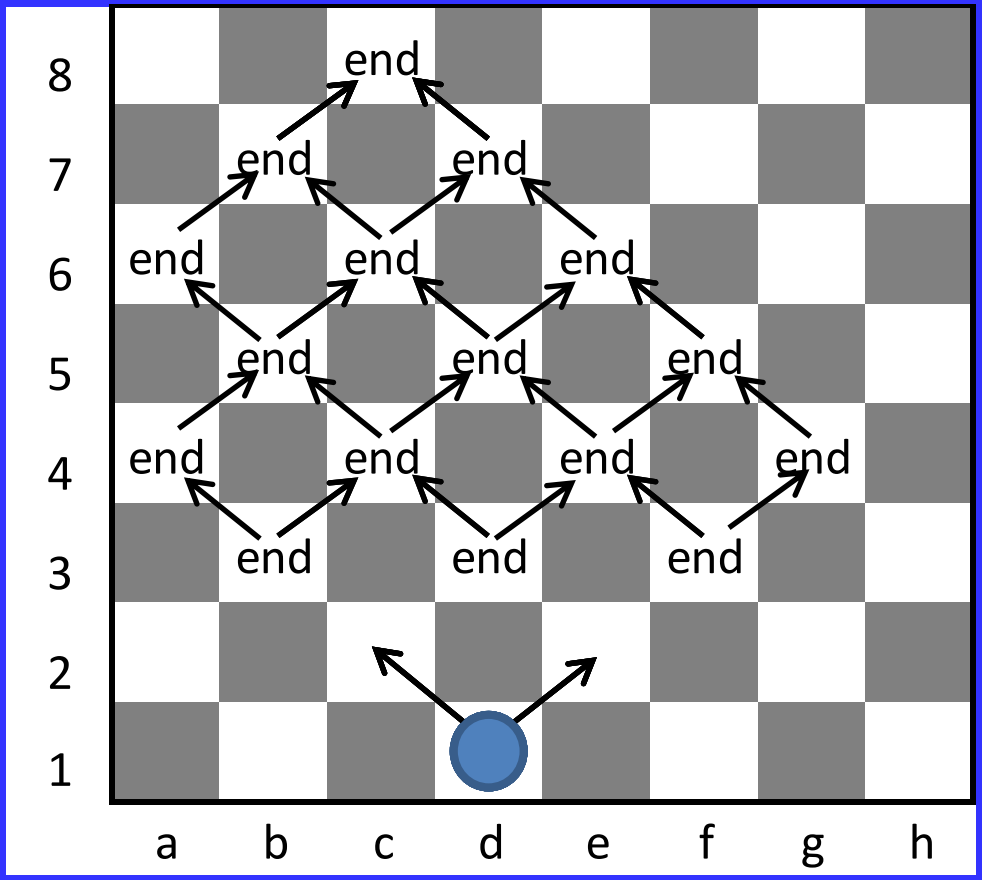
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

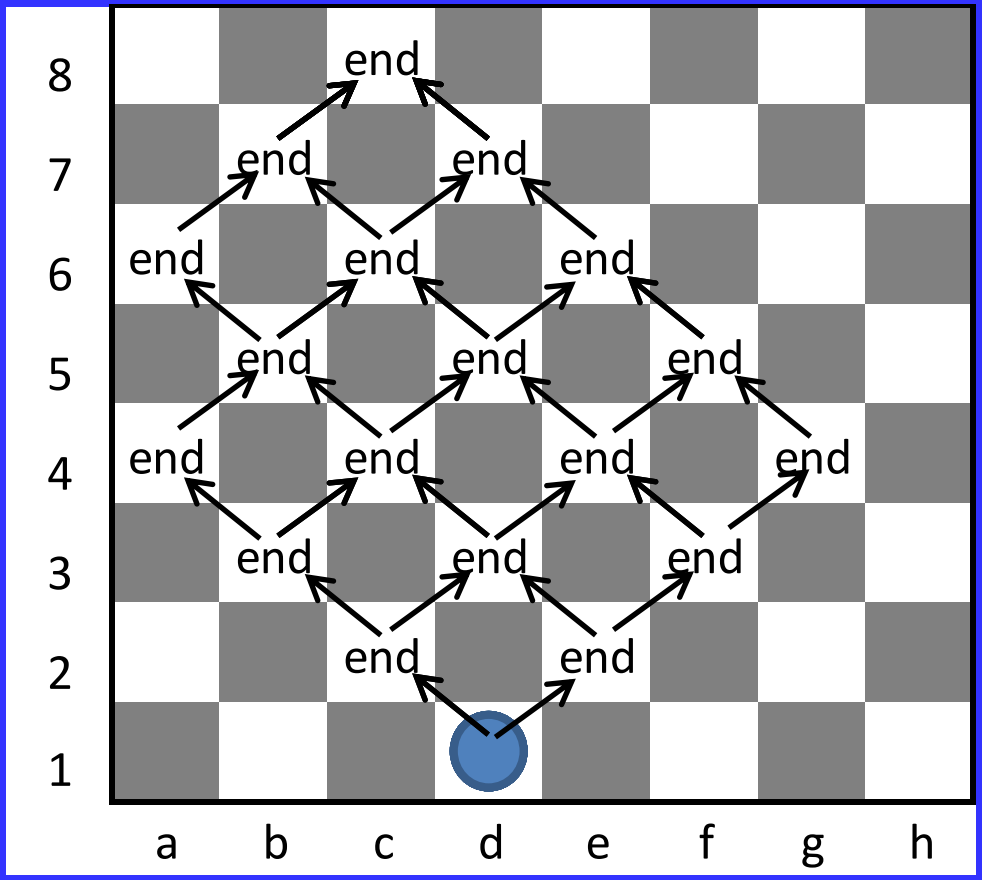
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

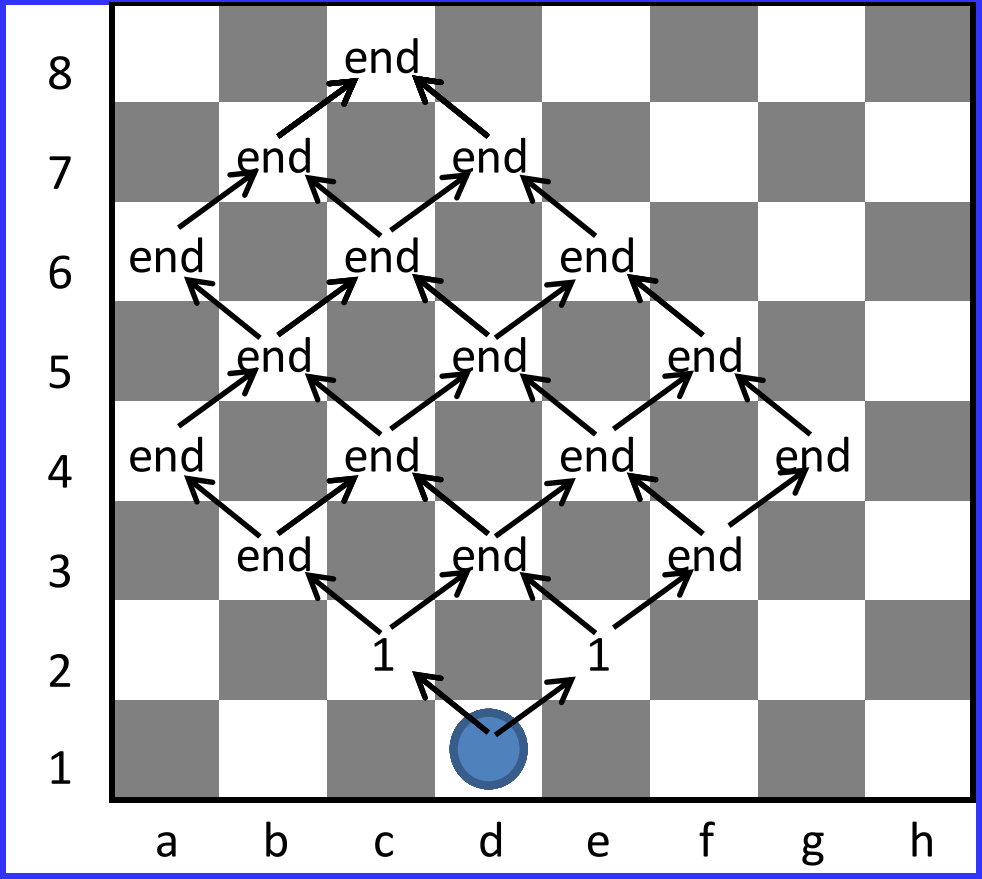
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

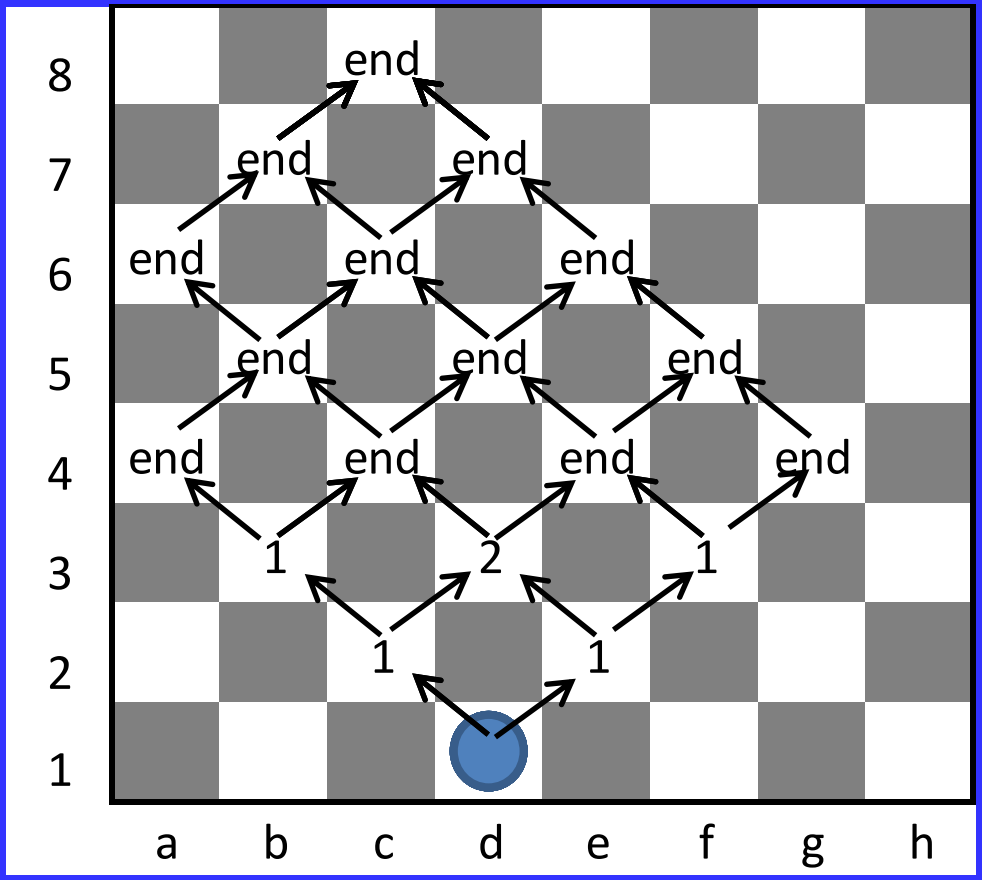
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

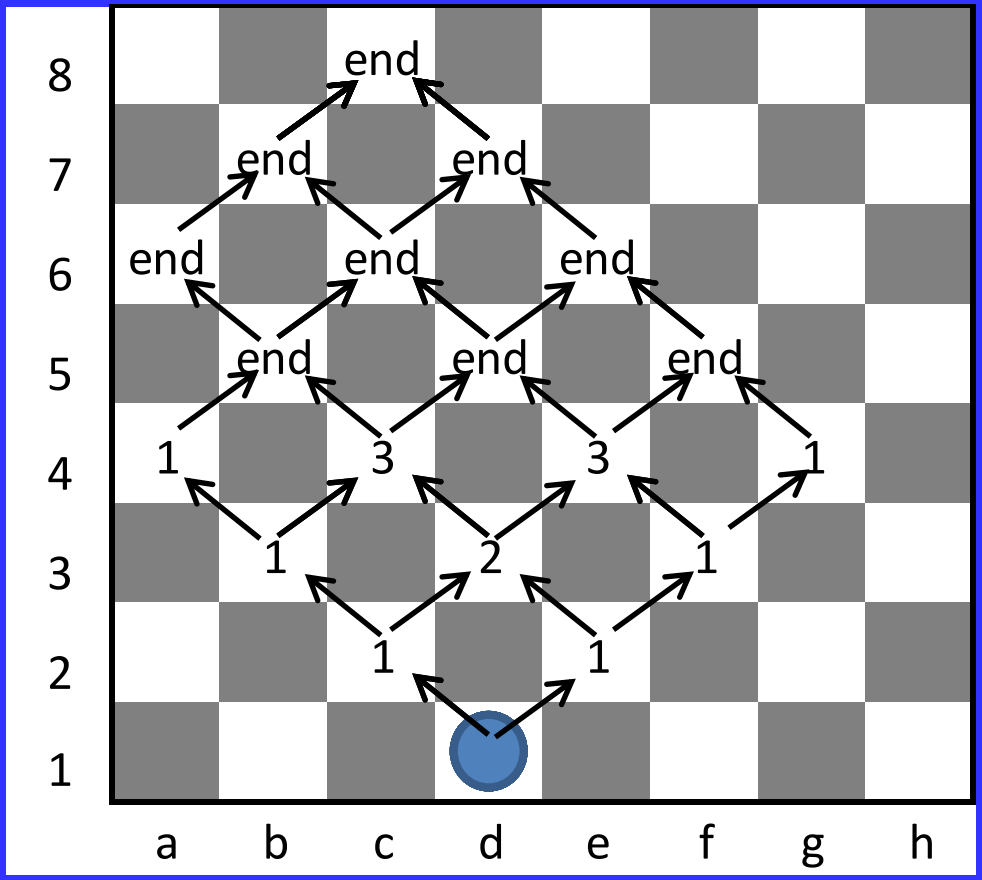
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

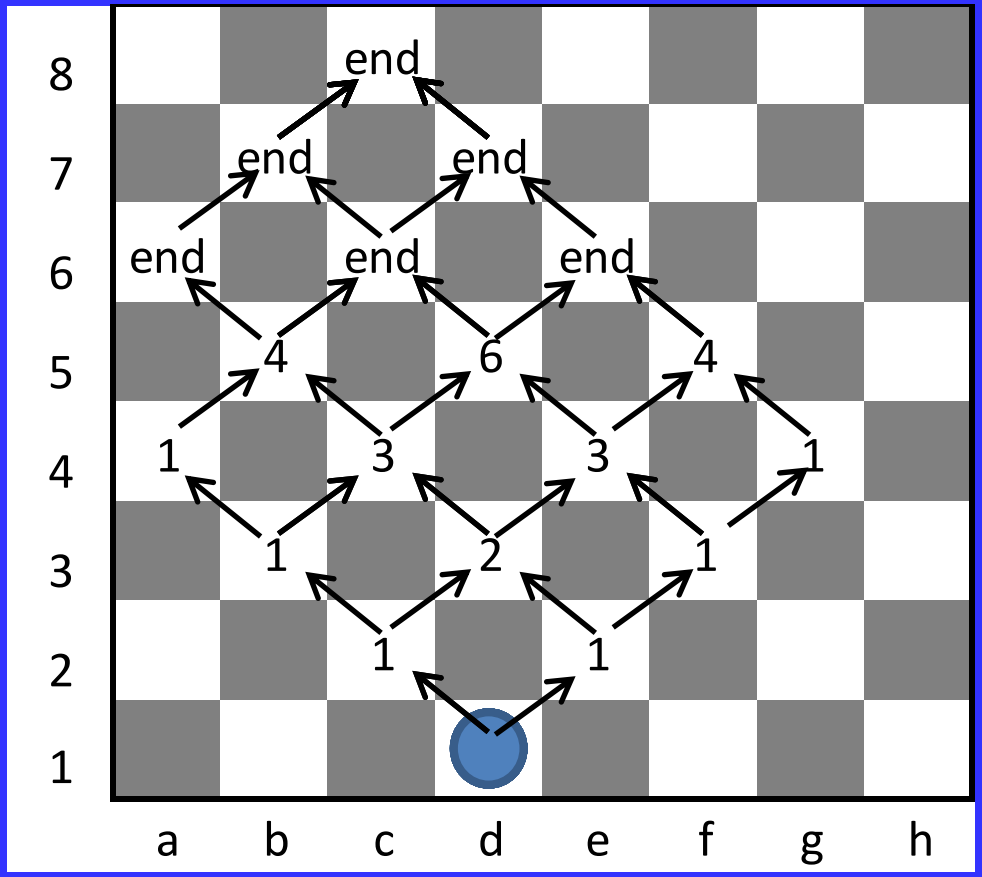
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

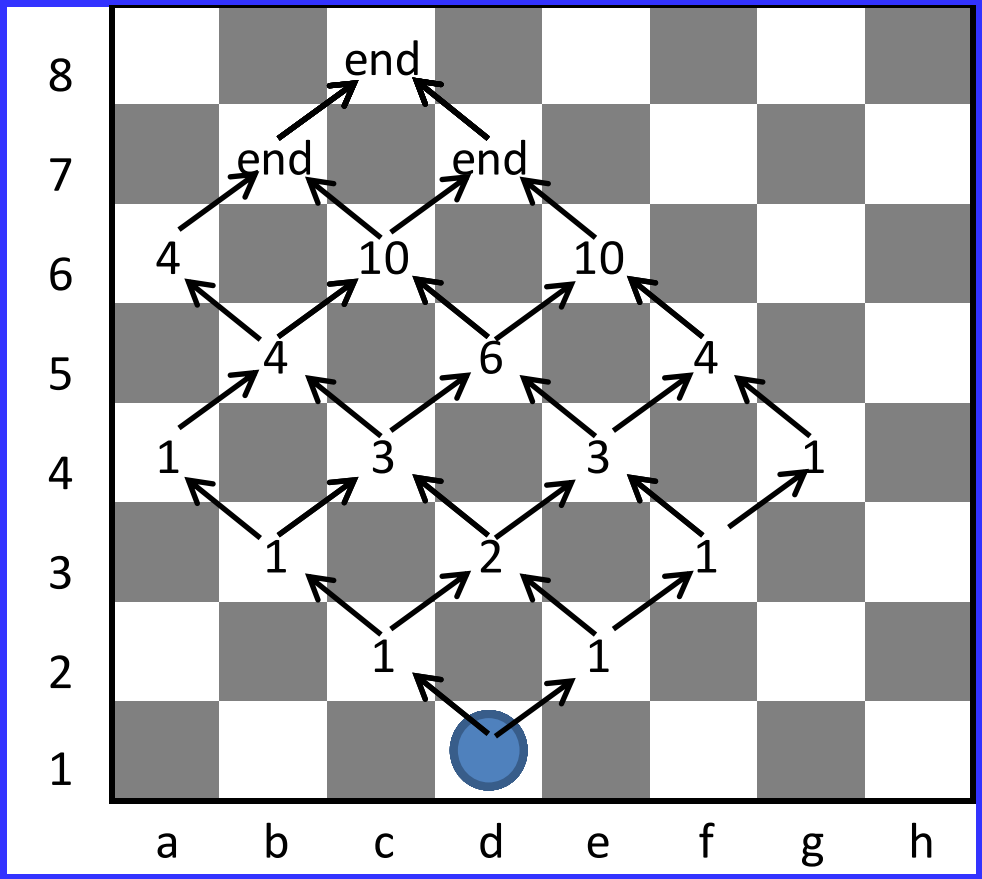
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

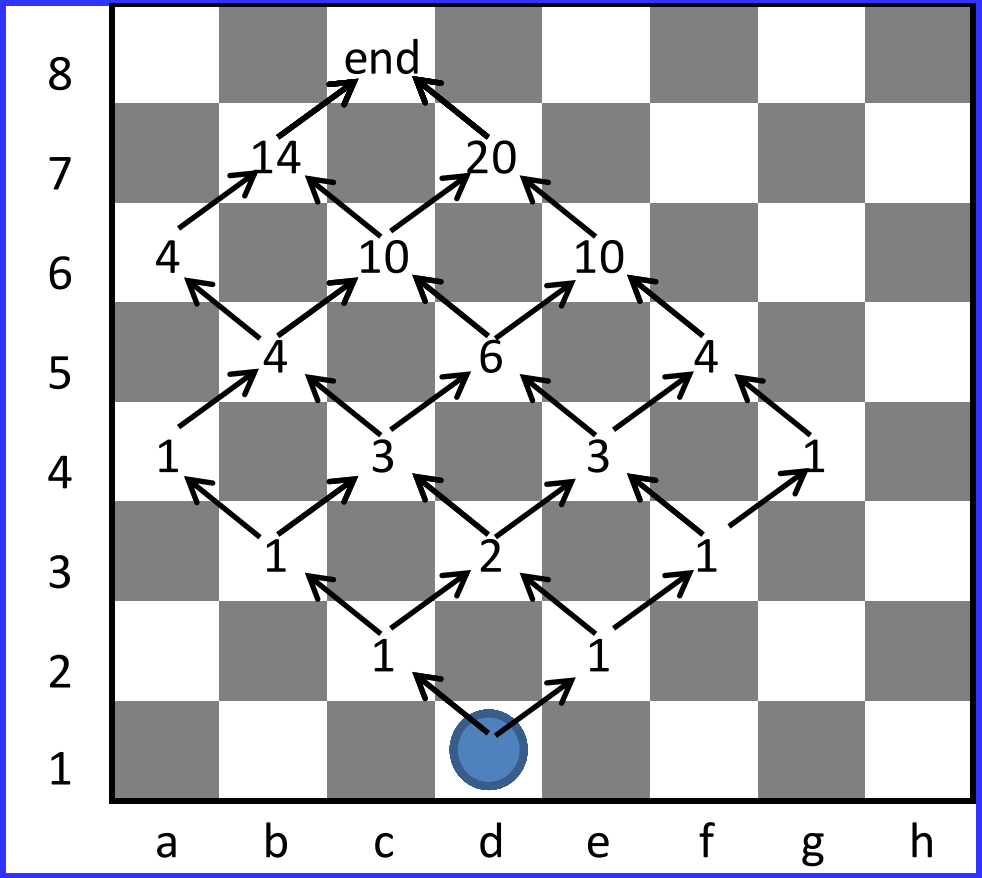
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

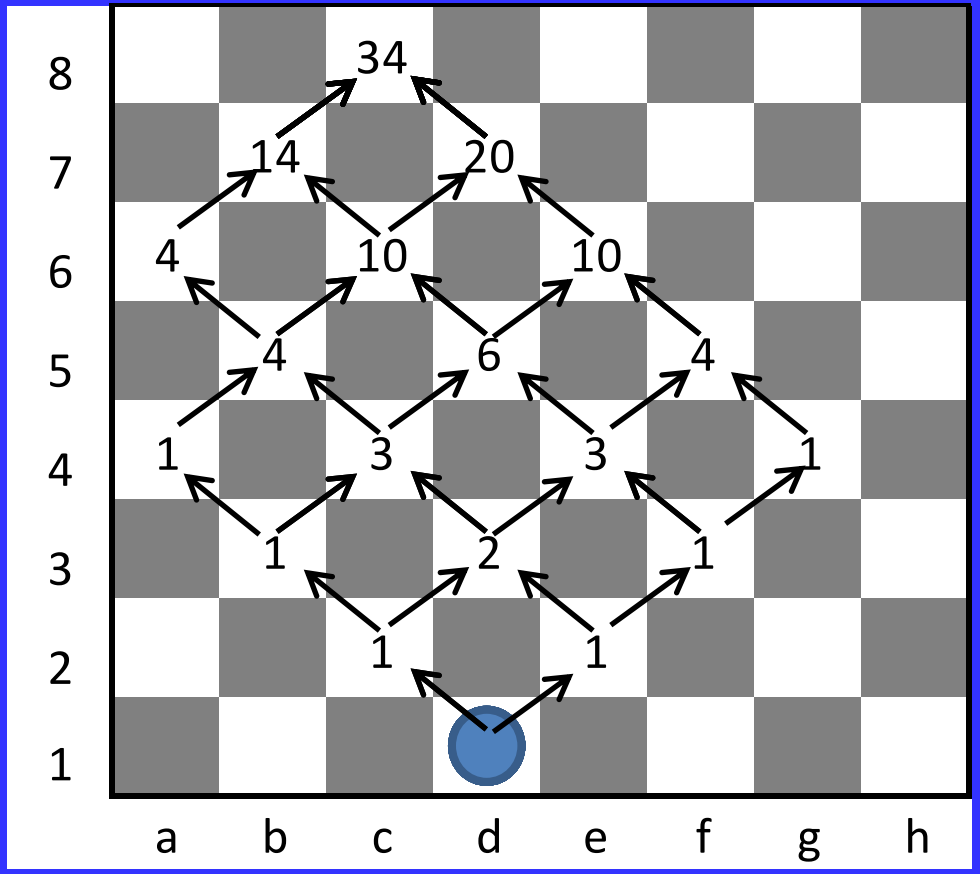
Now let's look at the original problem:



Can you decompose it into similar smaller problems?

Draughts path counting

Now let's look at the original problem:

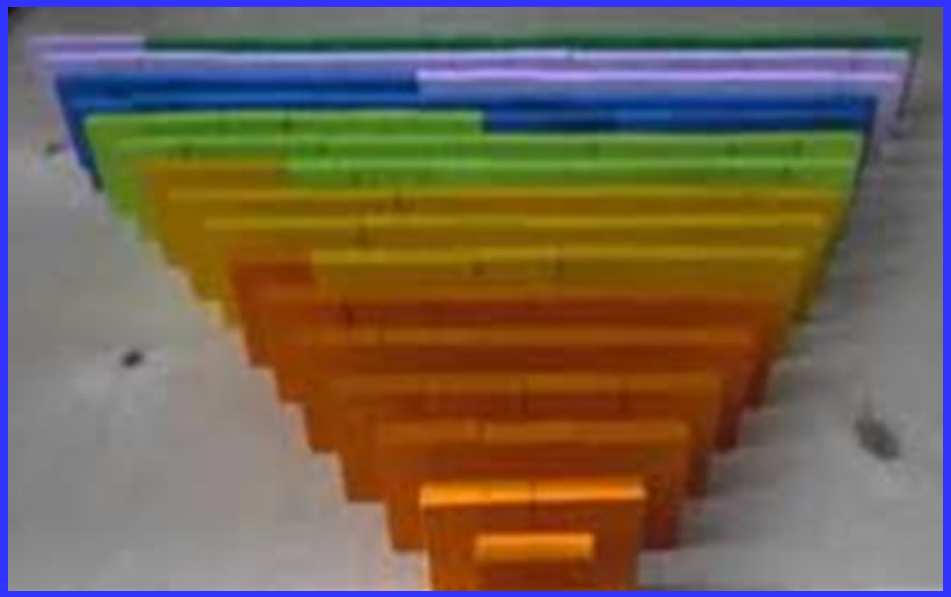


Can you decompose it into similar smaller problems?

Draughts path counting

This is a more interesting recursion because the problem decomposes not just into one but into two smaller problems.

Instead of a simple domino effect, we could say that we get a chain reaction:



Recursive play

Bjork, *Bachelorette*
directed by Michel Gondry:

https://www.youtube.com/watch?v=pzUB1D5XQ_w

Recursive play

Wikipedia: Recursion is the process a procedure goes through when one of the steps of the procedure involves invoking the procedure itself. A procedure that goes through recursion is said to be 'recursive'.

Recursive play

How would we write a play like *Bachelorette* that has as many recursive levels as the stage capacity allows?

Bachelorette

1) Things done before the play within the play
2) The play within the play
3) Things done after the play within the play

But how to specify how many times to do it?

Bachelorette(n)

- If n > 0
  - Things done before play
  - Bachelorette(n-1)
  - Things done after play

Bachelorette(Stage capacity)

Recursive play

Wikipedia: To understand recursion, one must recognize **the distinction between a procedure and the running of a procedure**.

A procedure is a set of steps based on a set of rules. The running of a procedure involves actually following the rules and performing the steps. An analogy: a procedure is like a written recipe; running a procedure is like actually preparing the meal.

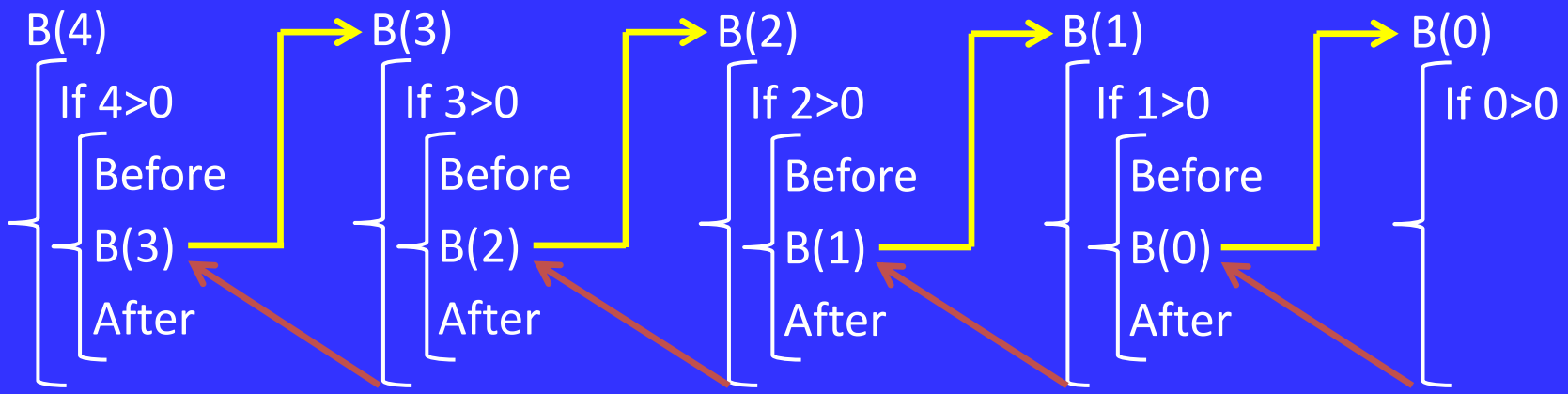
An algorithm is like a play; running an algorithm is like a performance.

Recursive play

Bachelorette(n)

- If n > 0
  - Things done before the play within the play
  - Bachelorette(n-1)
  - Things done after the play within the play

Bachelorette(4)

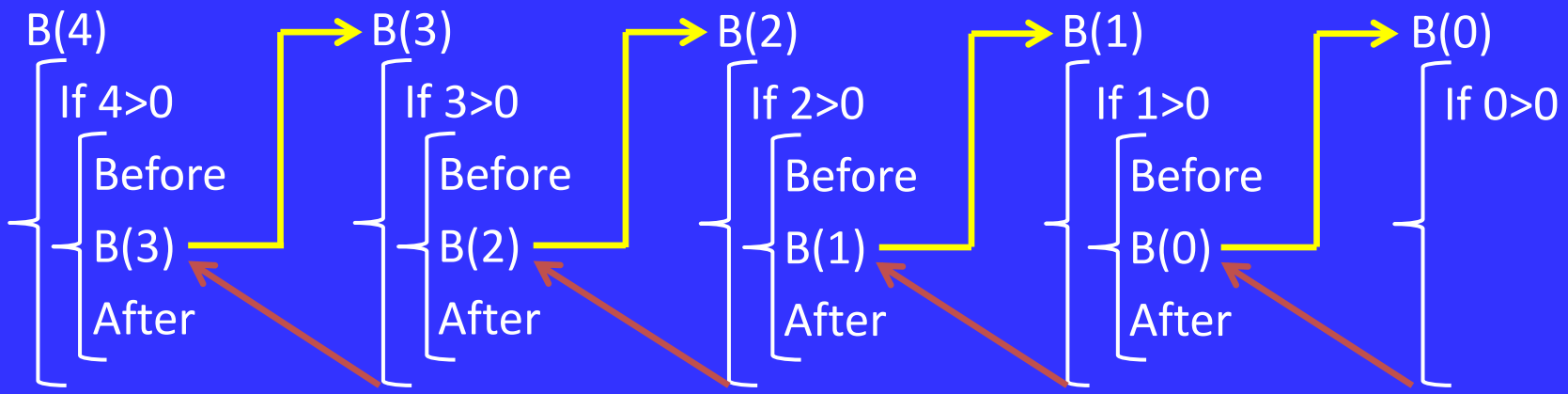


Worksheet 3
Recursive play writing

Bachelorette(n)

- If n > 0
  - Things done before the play within the play
  - Bachelorette(n-1)
  - Things done after the play within the play

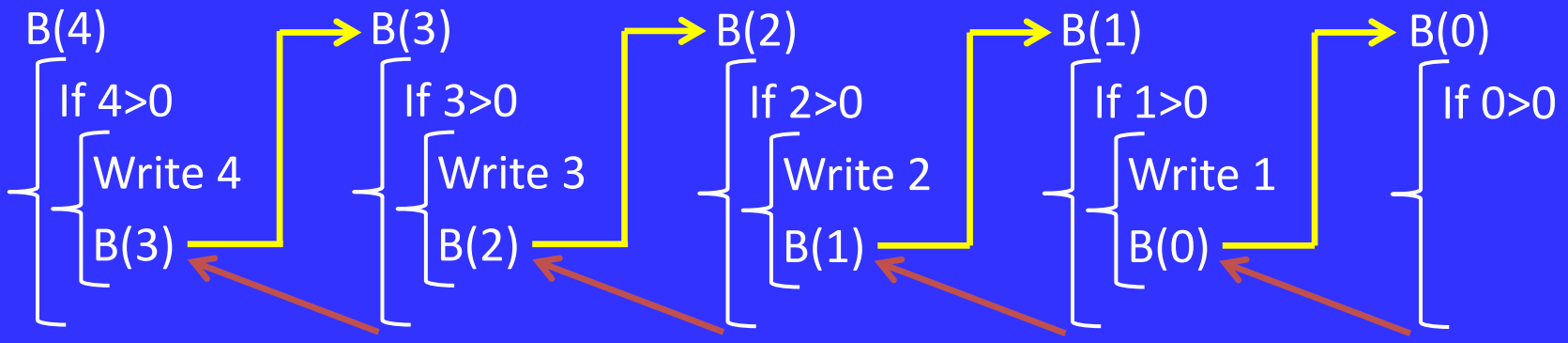
Bachelorette(4)



Recursive play

```
B(n)
    If n > 0
        Write n in the blackboard
        B(n-1)
B(4)
```

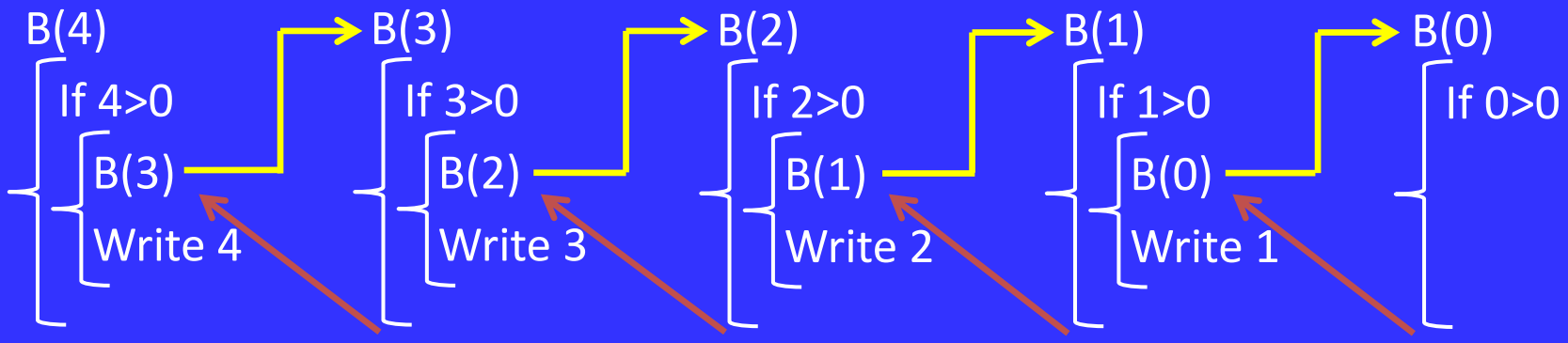
```
4
3
2
1
```



Recursive play

```
B(n)
    If n > 0
        B(n-1)
        Write n in the blackboard
B(4)
```

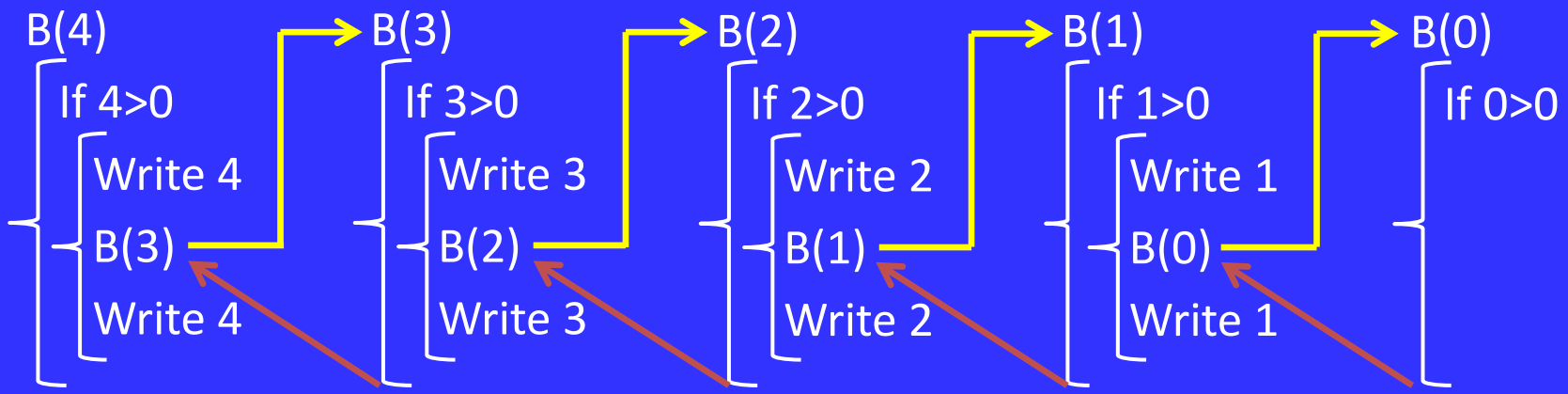
```
1
2
3
4
```



Recursive play

```
B(n)
    If n > 0
        Write n in the blackboard
        B(n-1)
        Write n in the blackboard
B(4)
```

```
4
3
2
1
1
2
3
4
```



Algorithms

In Computer Science we don’t write plays we write programs, but before programs we write algorithms.

A computer program has a lot of details and a syntax more suited to machines than to humans. Before writing that it is a very good practice to write down how the program is going to solve the problem. That is what is called the algorithm.

Algorithms

The word *algorithm* is derived from the name of Muḥammad ibn Mūsā al-Khwārizmī,
a Persian mathematician.

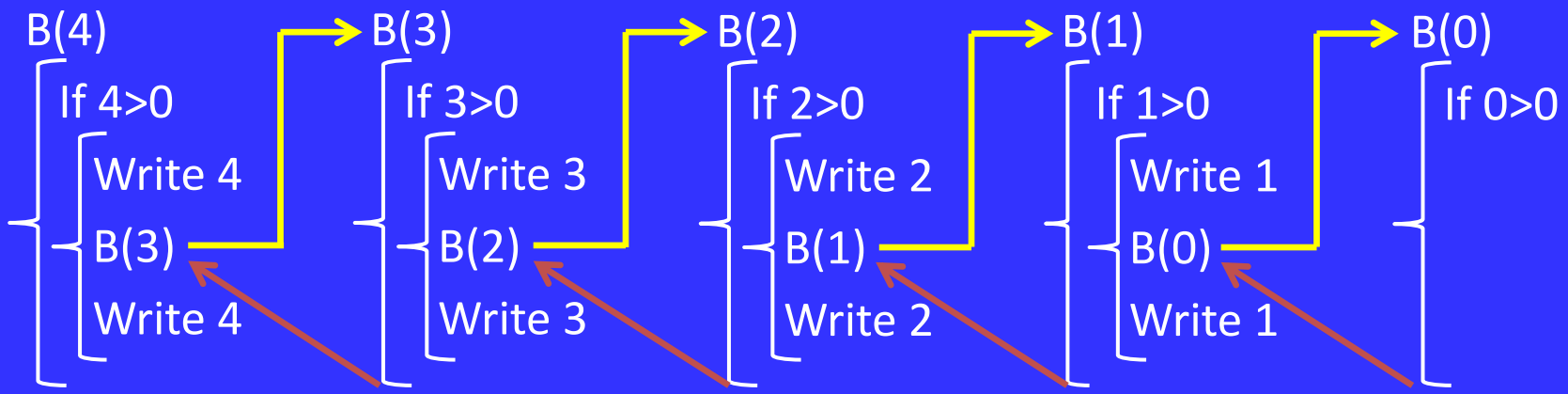
He is considered one of the “fathers of algebra” because in 820 AD he wrote arguably the first algebra book, *Al-Kitab al-Jabr wa-l-Muqabala*.

The word *algebra* is derived from the word *al-Jabr* in the title of that book.

Worksheet 4
Algorithms

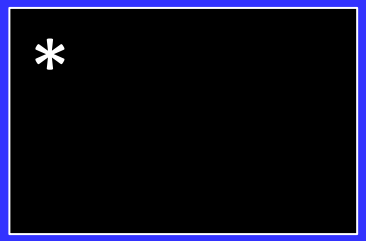
```
B(n)
    If n > 0
        Write n in the blackboard
        B(n-1)
        Write n in the blackboard
B(4)
```

```
4
3
2
1
1
2
3
4
```



Algorithms

```
P(n)
    If n = 1
        Write *
    else
        P(n-1)
        Write * n times
        P(n-1)
P(1)
```

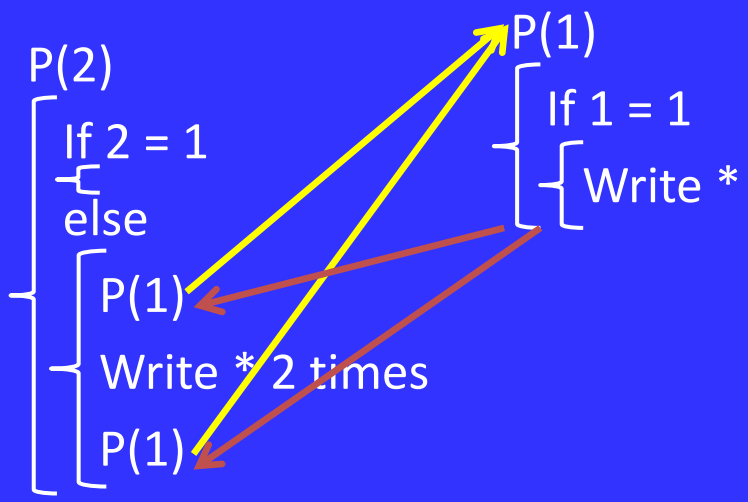



```
P(1)
    If 1 = 1
        Write *
```

Algorithms

```
P(n)
    If n = 1
        Write *
    else
        P(n-1)
        Write * n times
        P(n-1)
P(2)
```

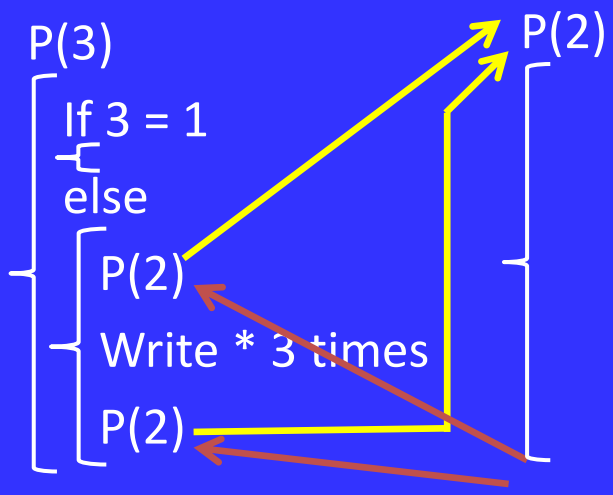
```
*
**
*
```



Algorithms

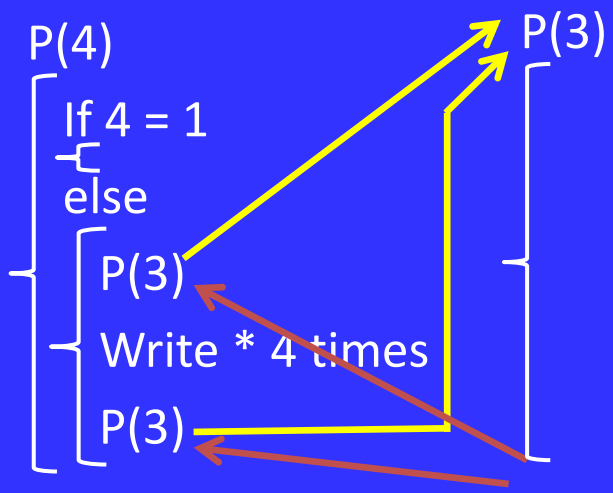
```
P(n)
    If n = 1
        Write *
    else
        P(n-1)
        Write * n times
        P(n-1)
P(3)
```

```
*
**
*
***
*
**
*
```



Algorithms

```
P(n)
    If n = 1
        Write *
    else
        P(n-1)
        Write * n times
        P(n-1)
P(4)
```



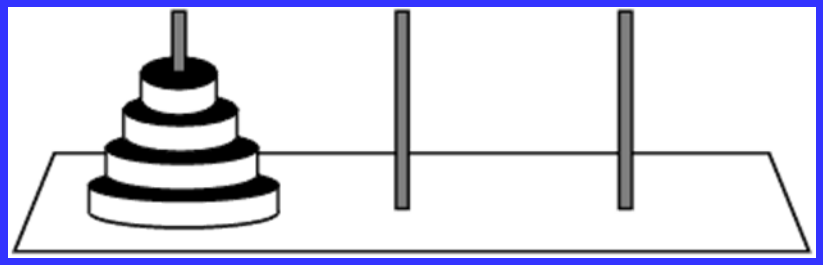

```
*
**
*
***
*
**
*
****
*
**
*
***
*
**
*
```

Rise of the Planet of the Apes

Directed by Rupert Wyatt. Twentieth Century Fox, 2011:

https://www.youtube.com/watch?v=5-9d0qYk2s4

Tower of Hanoi



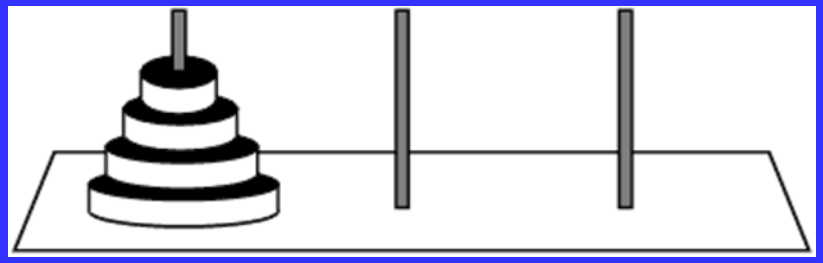
The legend:

In an Indian temple there is a large room with three posts in it surrounded by 64 golden disks. Brahmin priests, acting out the command of an ancient prophecy, have been moving these disks.

According to the legend, when the last move of the puzzle will be completed, the world will end.

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them roughly 585 billion years or more than 40 times the age of the universe.

Break - Worksheet 5
Tower of Hanoi



The objective: to move the entire stack from rod 1 to rod 3.

The rules:

a. Only one disk can be moved at a time.

b. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

c. No disk may be placed on top of a smaller disk.

Tower of Hanoi

Can you think of the smallest case?

What is the size of the problem?

The number of disks.

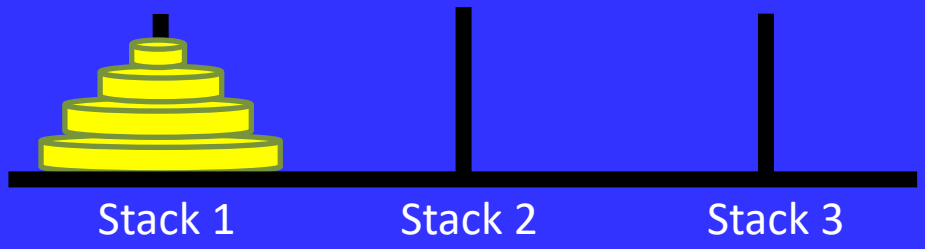
Therefore the smallest size of the problem is 1, just one disk to move from stack 1 to stack 3.

The solution is trivial: move the disk from stack 1 to stack 3.

Tower of Hanoi

Divide and conquer principle:

Can you **decompose** the problem into similar smaller problems?

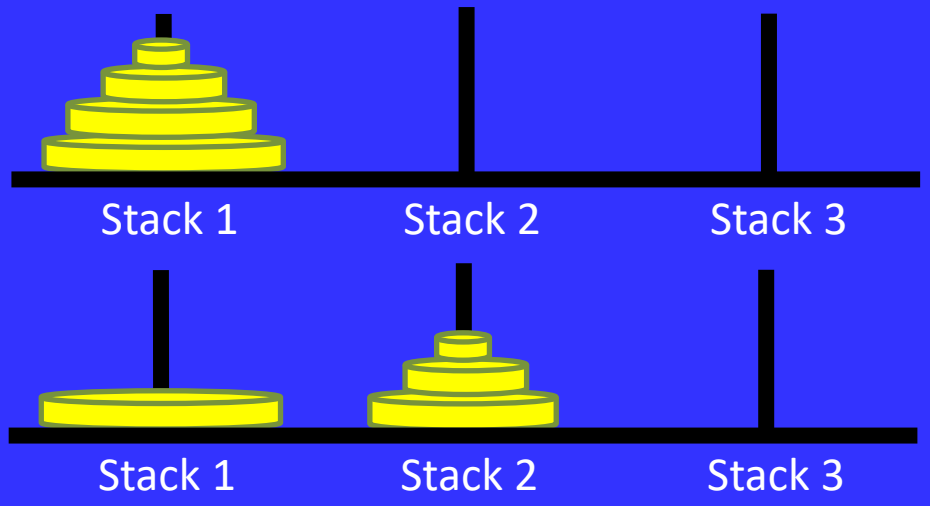


To move 4 disks from stack 1 to stack 3:

Tower of Hanoi

Divide and conquer principle:

Can you **decompose** the problem into similar smaller problems?



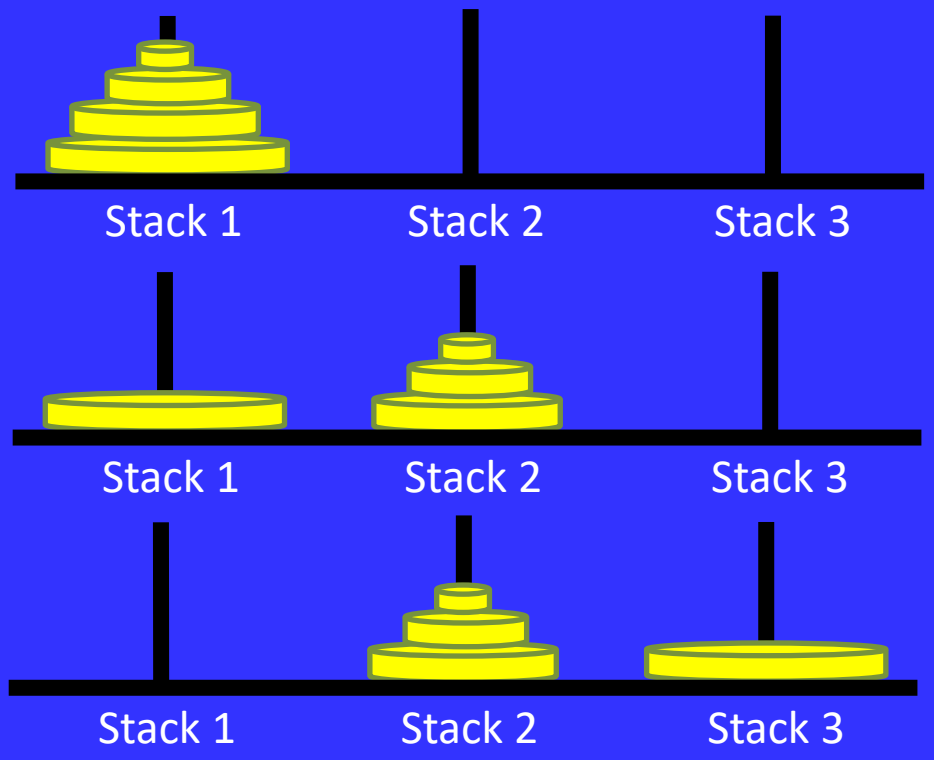
To move 4 disks from stack 1 to stack 3:

a) Use the solution for 3 to move 3 disks from stack 1 to stack 2 (smaller problem)

Tower of Hanoi

Divide and conquer principle:

Can you **decompose** the problem into similar smaller problems?



To move 4 disks from stack 1 to stack 3:

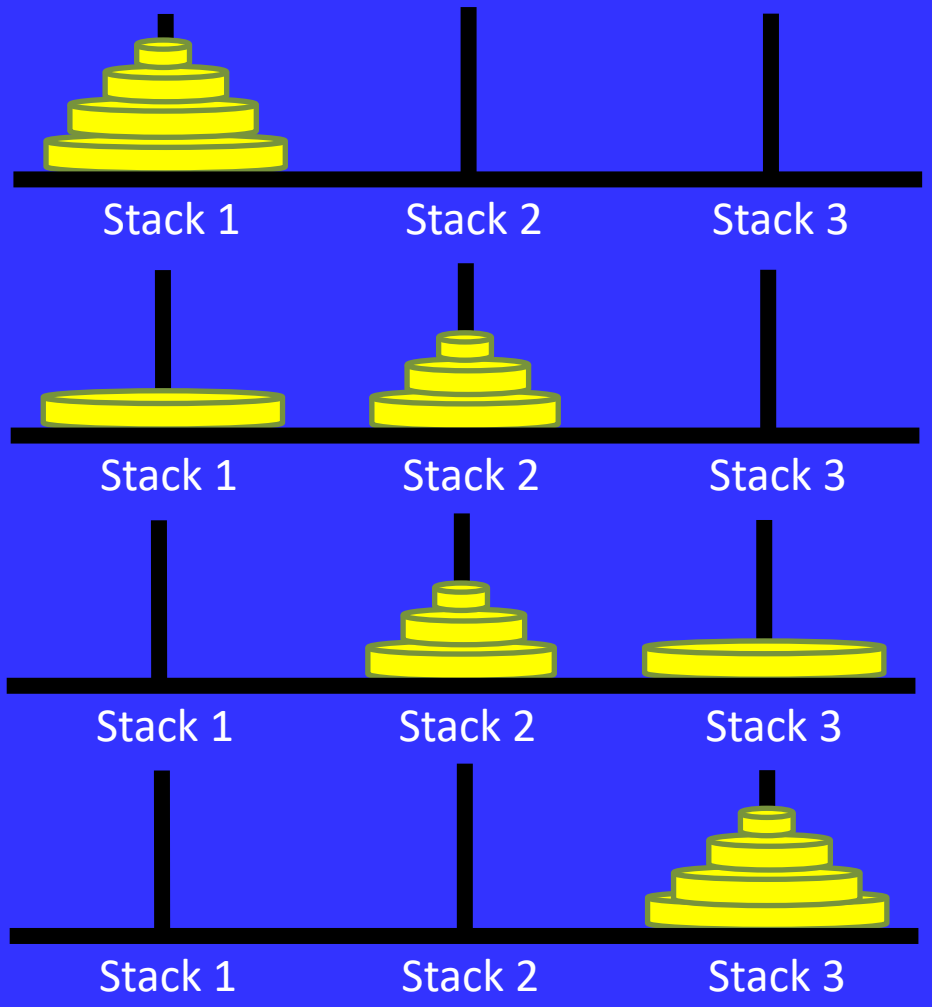
a) Use the solution for 3 to move 3 disks from stack 1 to stack 2 (smaller problem)

b) Move the biggest disk from stack 1 to stack 3

Tower of Hanoi

Divide and conquer principle:

Can you **decompose** the problem into similar smaller problems?



To move 4 disks from stack 1 to stack 3:

a) Use the solution for 3 to move 3 disks from stack 1 to stack 2 (smaller problem)

b) Move the biggest disk from stack 1 to stack 3

c) Use the solution for 3 to move 3 disks from stack 2 to stack 3 (smaller problem)

Tower of Hanoi

Hanoi(n, origin, destination)

- If n = 1
  - Move disk from origin to destination
- else
  - Hanoi(n-1, origin, other stack)
  - Move disk from origin to destination
  - Hanoi(n-1, other stack, destination)

How to compute the number of the other stack as a function of origin number and destination number?

The stacks, 1, 2 and 3, add up to 6.

The formula is: other stack = 6 – origin – destination.

Tower of Hanoi

Algorithm:

```
Hanoi(n, origin, destination)
    If n = 1
        Move disk from origin to destination
    else
        Hanoi(n-1, origin, 6 – origin – destination)
        Move disk from origin to destination
        Hanoi(n-1, 6 – origin – destination, destination)
```

Python program:

```
def hanoi(n,origin,destination):
        if n == 1:
                print "move disk from", origin, "to", destination
        else:
                hanoi(n-1,origin,6-origin-destination)
                print "move disk from", origin, "to", destination
                hanoi(n-1,6-origin-destination,destination)
```

Tower of Hanoi

Try it the Python program:
Please login to Google drive

Username: theplaywithintheplay

Password: gladesmore2017

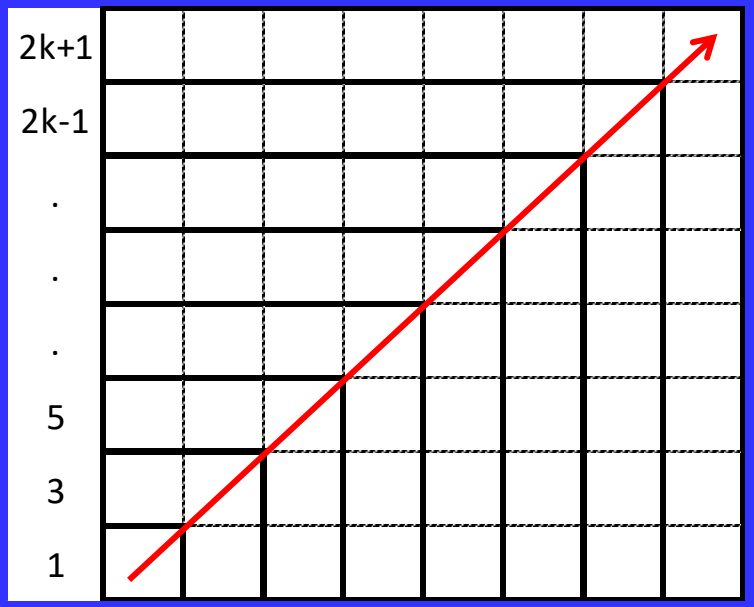
Go to Google Drive and download the file 50_hanoi.py, open it and press F5 to run it.
To call the function type for example: hanoi(3,1,3)

You can press Arrow Up or Alt-p one or more times to bring a previous command, then you can edit it and run it.

Maths induction and CS recursion

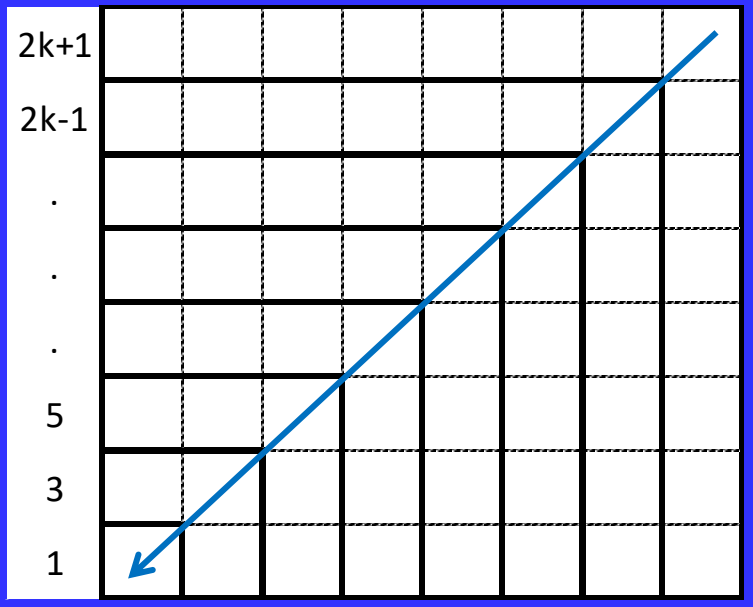
Mathematical induction:

From small case to bigger case.



Computer Science recursion:

From big case to smaller case.

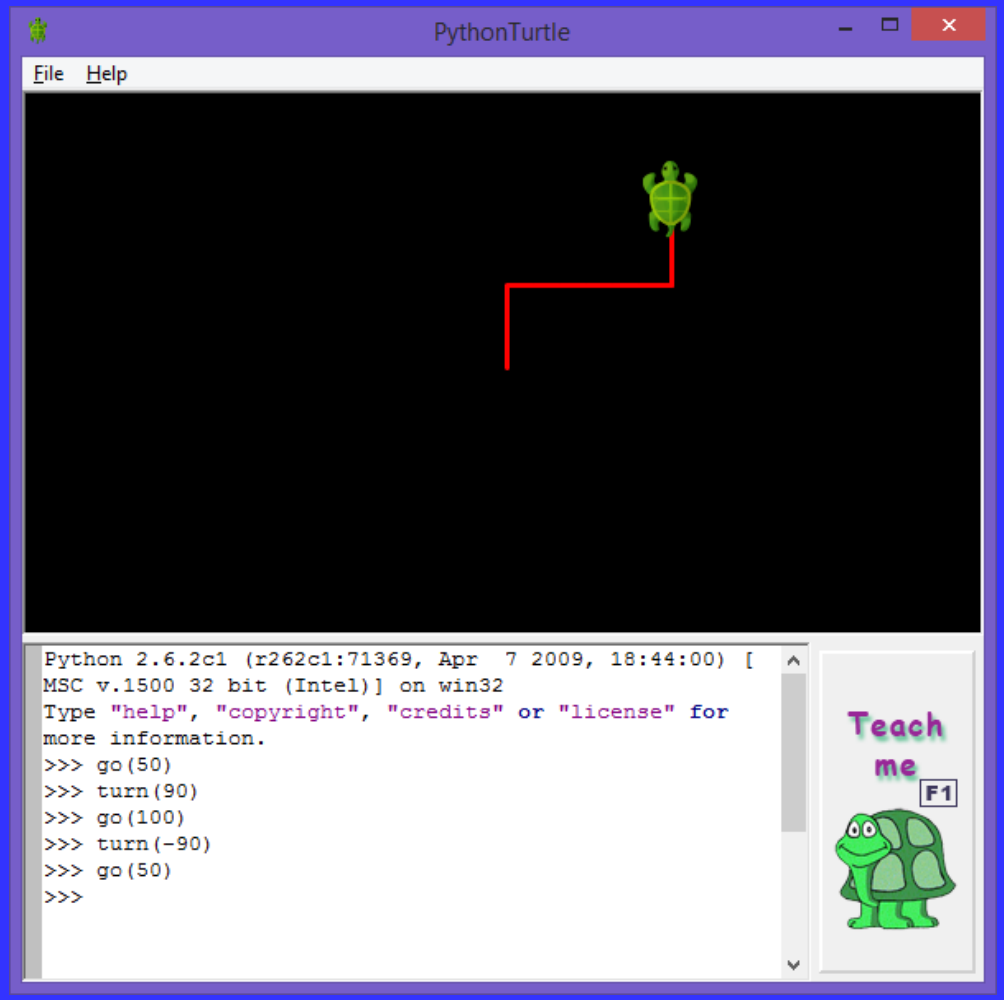


A generalization of mathematical induction, called structural induction, is used in Computer Science. It can have many base cases and many inductive steps.

Turtle programming

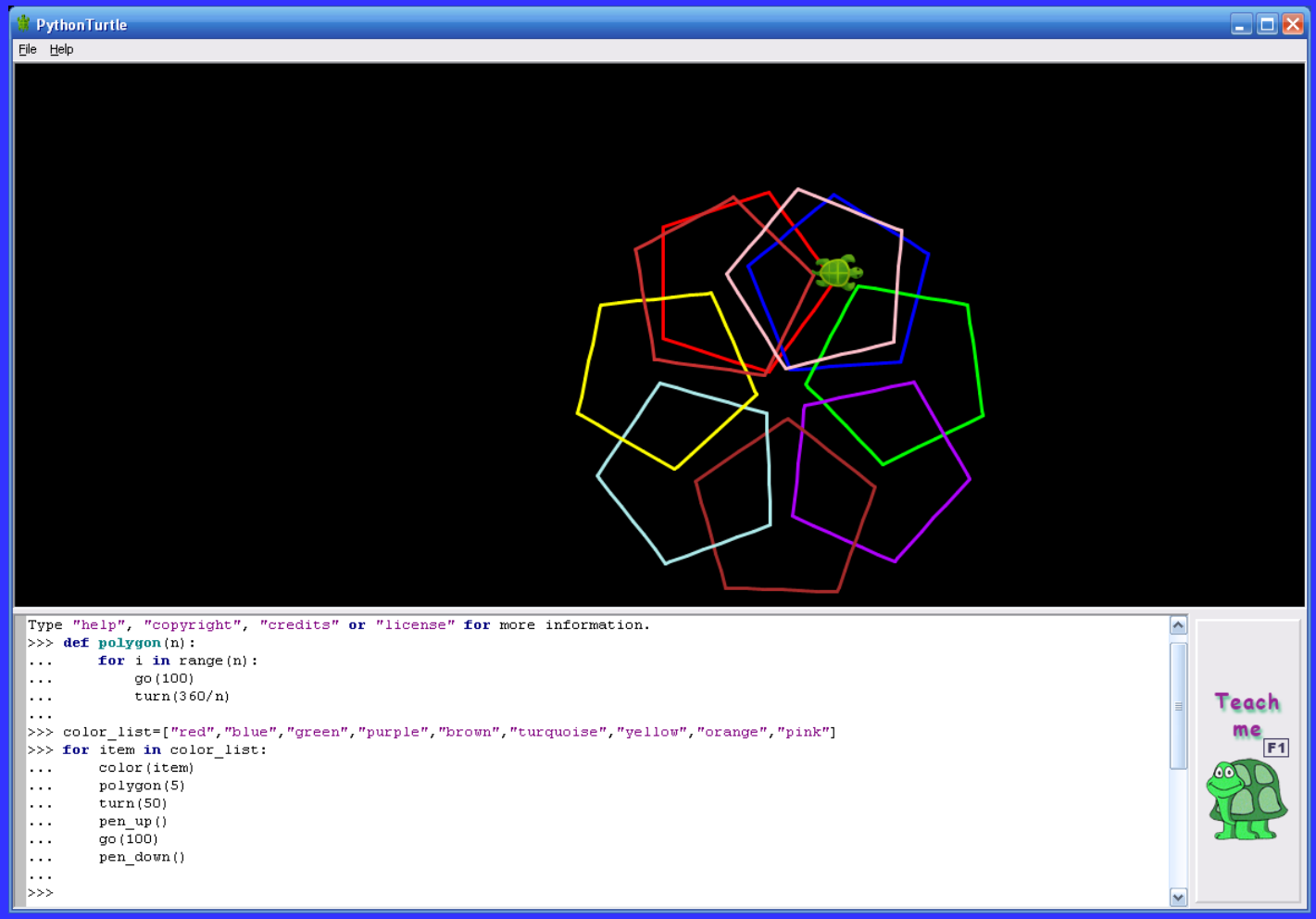
Seymour Papert et al, Logo Turtle (1969)

Turtle programming



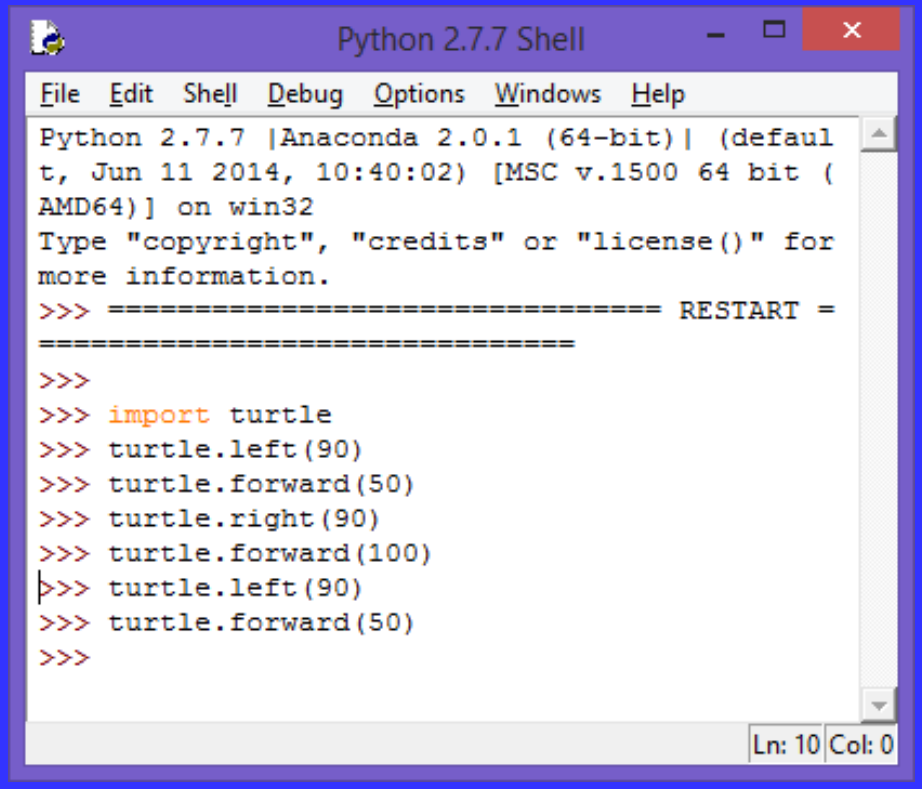
pythonturtle.org

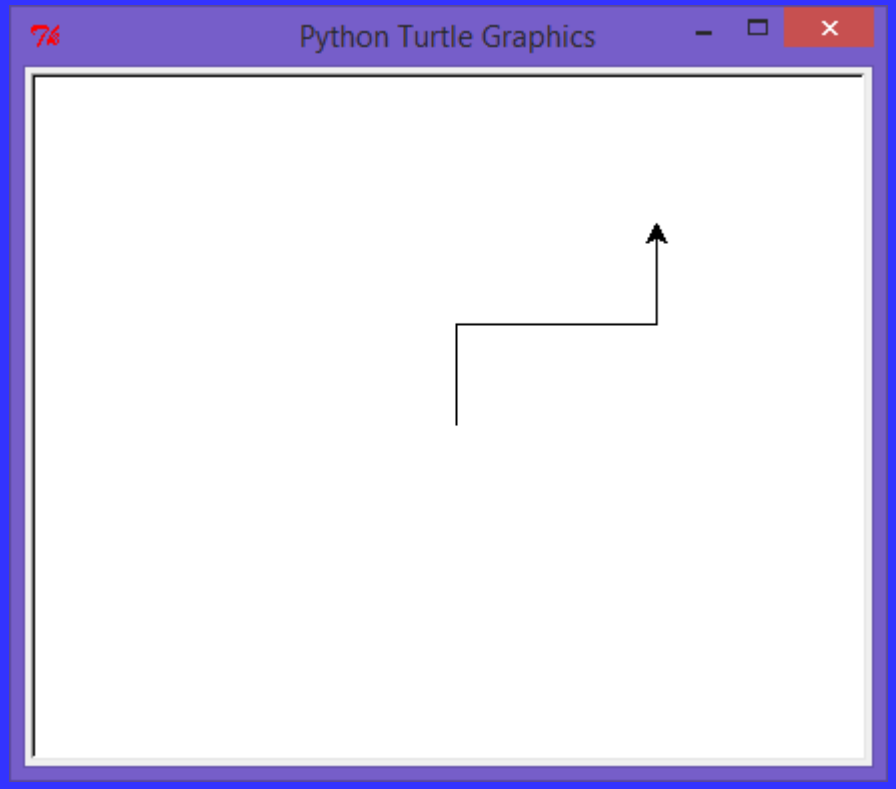
Turtle programming



pythonturtle.org

Turtle programming





Python Shell and Python's turtle library

Turtle programming

Some of the instructions that you can give to the turtle are:

forward(distance)	backward(distance)
left(angle)	right(angle)
penup()	pendown()
if then else	speed(n)
def	

Demo

Worksheet 6
Turtle programming

Some of the instructions that you can give to the turtle are:

forward(distance)	backward(distance)
left(angle)	right(angle)
penup()	pendown()
if then else	speed(n)
def	

Try it: In the window called Shell type “import turtle”, return and for example turtle.forward(100), turtle.left(90).

Worksheet 6
Turtle programming

Solution to 6.1): Use Ctrl-O to open the file 61_pol_solution.py and run it (F5). Advice: close the window of the previous file.

Solution to 6.2): Try pol(10,6,60)

Solution to 6.3): Open the file 63_ManyPolygons_solution.py and run it (F5).

Before running ManyPolygons2(100,8) instead of turtle.reset() run rs(), that resets the turtle and makes it move at its maximum speed.

Worksheet 7
Turtle programming

Solution to 7.1): 71_polback.py

```
def polBack(length, angle, n):
        if n > 0:
                turtle.forward(length)
                turtle.left(angle)
                polBack(length, angle, n-1)
                turtle.right(angle)
                turtle.backward(length)
```

Worksheet 7
Turtle programming

Solution to 7.2): File 72_spiral_solution.py.

Solution to 7.3): File 73_a_b_figures_solution.py.

Run the examples there that are commented out (lines that start with #).

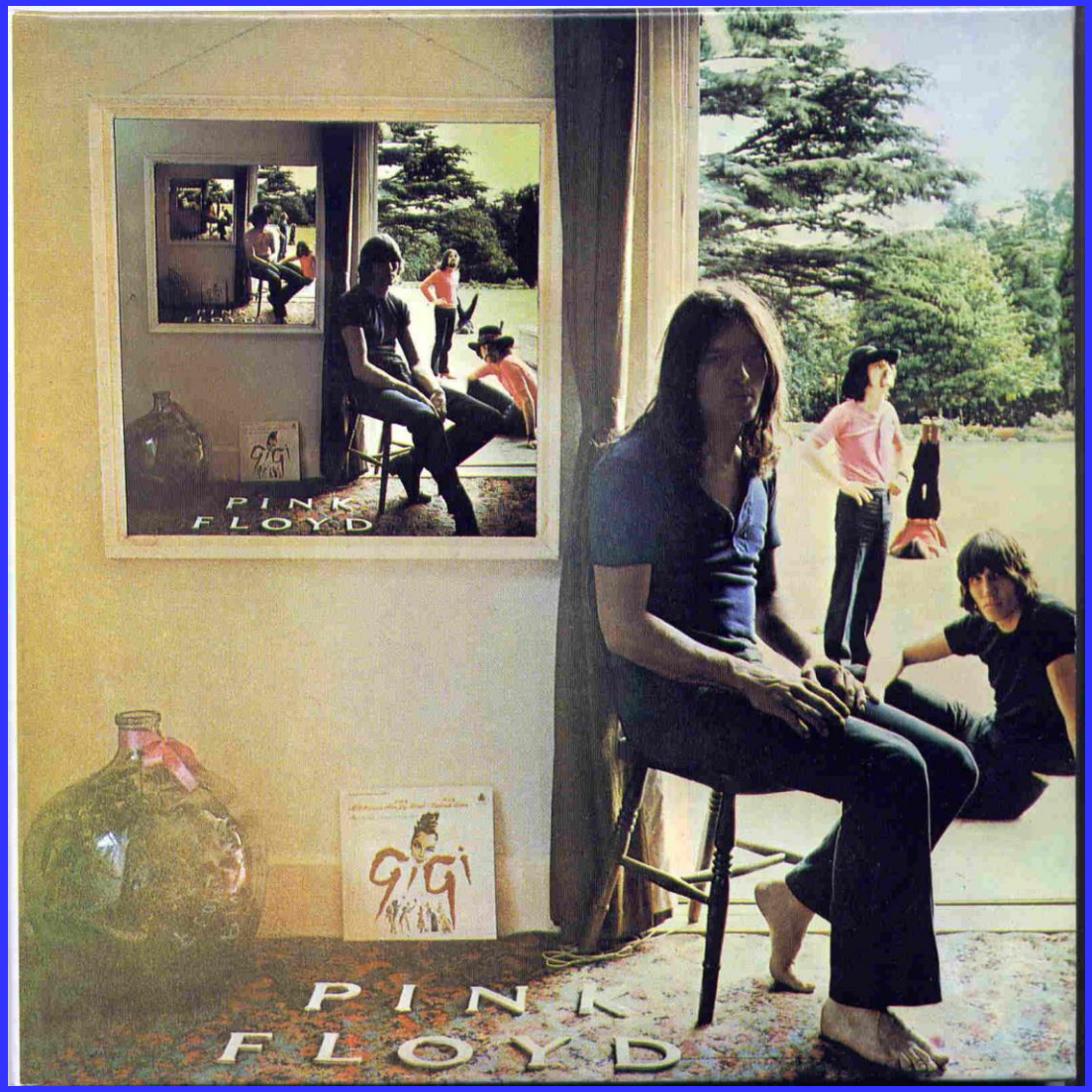
Self-similarity

A **self-similar** object is exactly or approximately similar to a part of itself.

Self-similarity

Spirals are the simplest self-similar objects.

Self-similarity



Pink Floyd, *Ummagumma*

Self-similarity

Taken from www.andrewlipson.com

Fractals

The Mandelbrot set:
https://vimeo.com/6644398

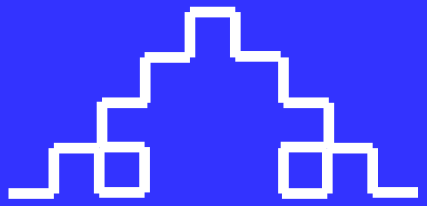
Fractals - Koch curve variant

We can obtain a Koch curve variant with the following procedure.

Start with a segment:

and replace it with five segments of 1/3 of the original size:

and replace each of them with four segments in the same way:



and then repeat this process again and again.

Fractals - Koch curve variant

Here is a description of this curve using a L-system (taken from http://en.wikipedia.org/wiki/L-system):

start: F

rule: F → F+F−F−F+F

Here, F means "draw forward", + means "turn left 90°",

and − means "turn right 90°". The shrinking factor is 1/3.

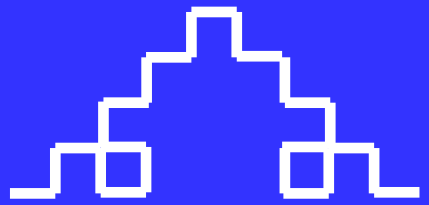
n = 0:

F 

n = 1:

F+F−F−F+F 

n = 2:



F+F−F−F+F + F+F−F−F+F − F+F−F−F+F − F+F−F−F+F + F+F−F−F+F

Fractals - Koch curve variant

start: F

rule: F → F+F−F−F+F

F means “draw forward”

\+ means “turn left 90°”

− means “turn right 90°”

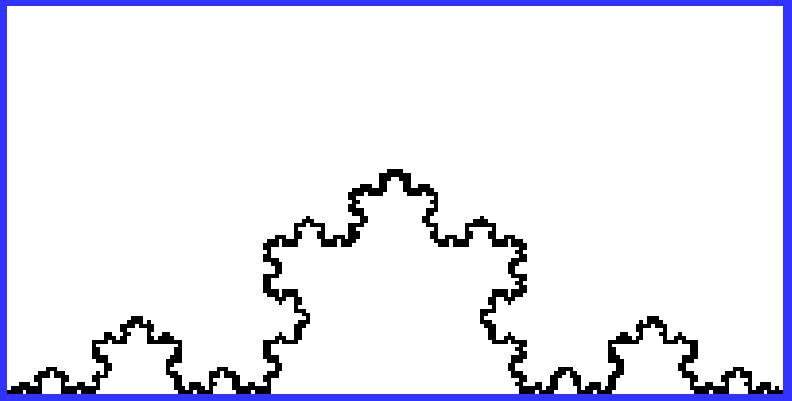
Shrinking factor = 1/3

Here is a program that draws this curve:

```
def koch(length, n):
    if n == 0:
        turtle.forward(length)
    else:
        koch(length/3,n-1)
        turtle.left(90)
        koch(length/3,n-1)
        turtle.right(90)
        koch(length/3,n-1)
        turtle.right(90)
        koch(length/3,n-1)
        turtle.left(90)
        koch(length/3,n-1)
```

Worksheet 8
Drawing fractals

Koch snowflake



Fractals - Koch snowflake

Here is a description of this curve using a L-system (taken from http://en.wikipedia.org/wiki/Koch_snowflake):

start: F

rule: F → F+F−F+F

Here, F means "draw forward", + means “turn left 60°”

and − means “turn right 120°”. The shrinking factor is 1/3.

n = 0:

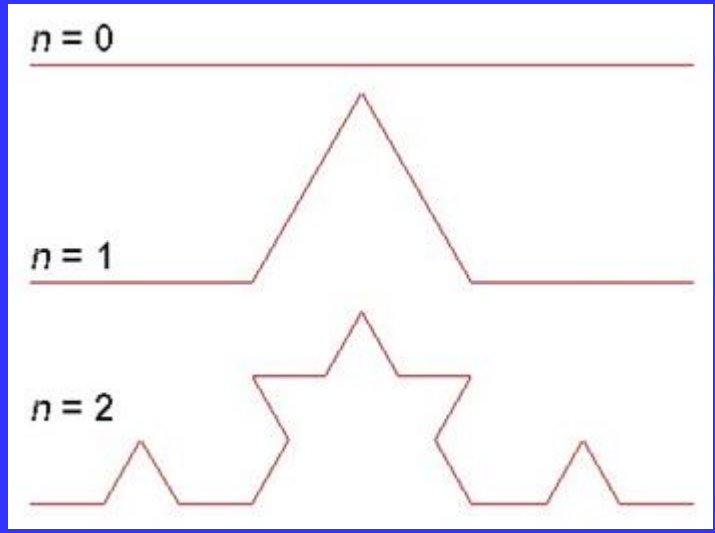
F

n = 1:

F+F−F+F

n = 2:

F+F−F+F + F+F−F+F − F+F−F+F + F+F−F+F



Fractals - Koch snowflake

start: F

rule: F → F+F−F+F

F means “draw forward”

\+ means “turn left 60°”

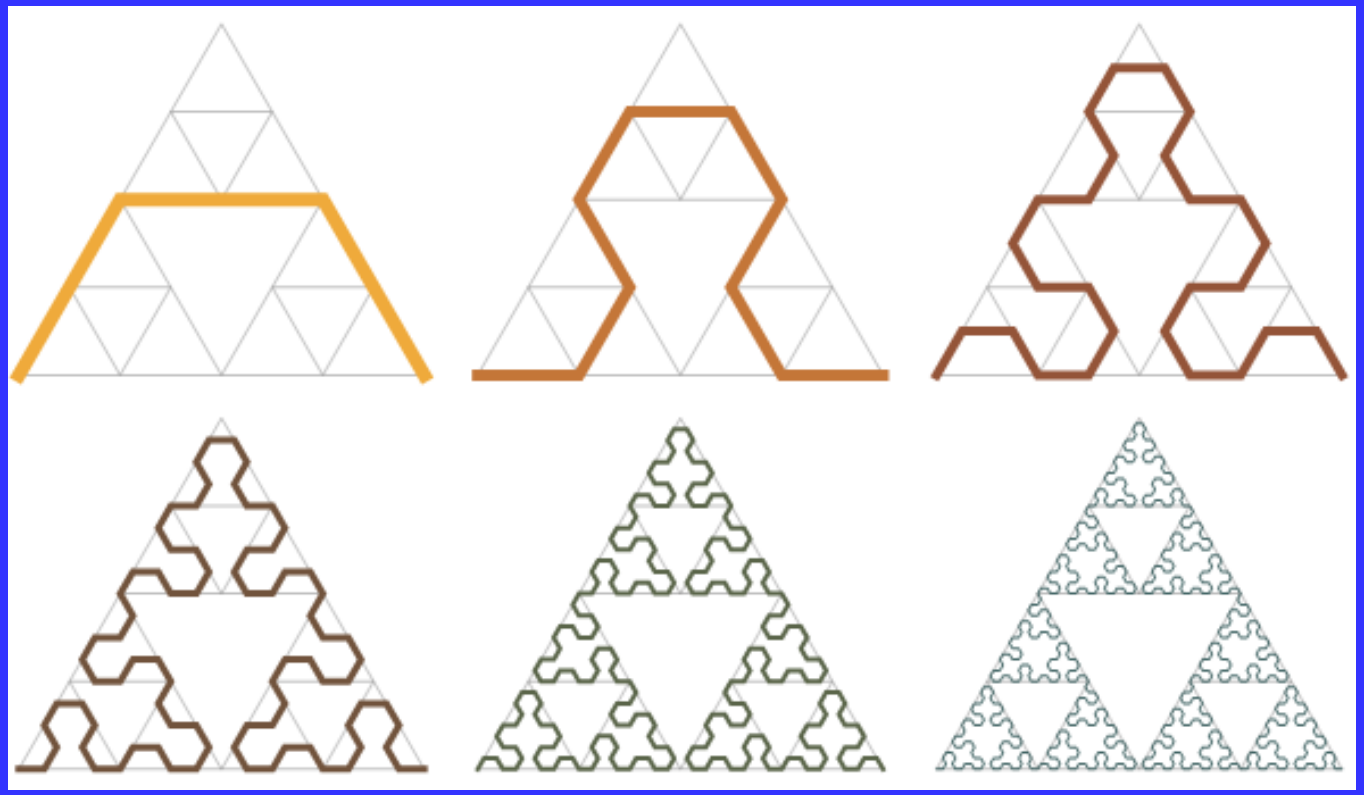
− means “turn right 120°”

Shrinking factor = 1/3

Here is a program that draws this curve:

```
def KochSnowflake1 (length, n):
        if n == 0:
                turtle.forward(length)
        else:
                koch(length/3,n-1)
                turtle.left(60)
                koch(length/3,n-1)
                turtle.right(120)
                koch(length/3,n-1)
                 turtle.left(60)
                koch(length/3,n-1)
```

Fractals - Sierpinski triangle



Fractals

The Sierpinski triangle

Fractals – Sierpinski triangle

start: A

rules: A → B−A−B

B → A+B+A

A and B both mean “draw forward”

+ means “turn left 60°”

− means “turn right 60°”

Shrinking factor = 1/2

Here is a program that draws this curve:

```
def sierpinski(variable, length, n):
    if n == 0:
        turtle.forward(length)
    else:
        if variable == 'A':
            sierpinski('B',length/2, n-1)
            turtle.right(60)
            sierpinski('A',length/2, n-1)
            turtle.right(60)
            sierpinski('B',length/2, n-1)
        else:
            sierpinski('A',length/2, n-1)
            turtle.left(60)
            sierpinski('B',length/2, n-1)
            turtle.left(60)
            sierpinski('A',length/2, n-1)
```

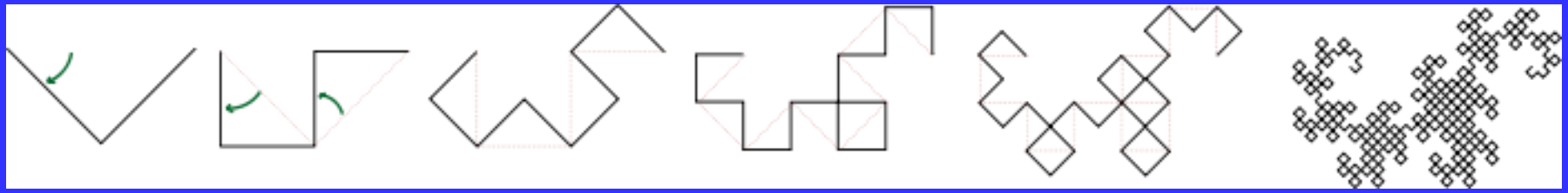
Fractals – Sierpinski triangle

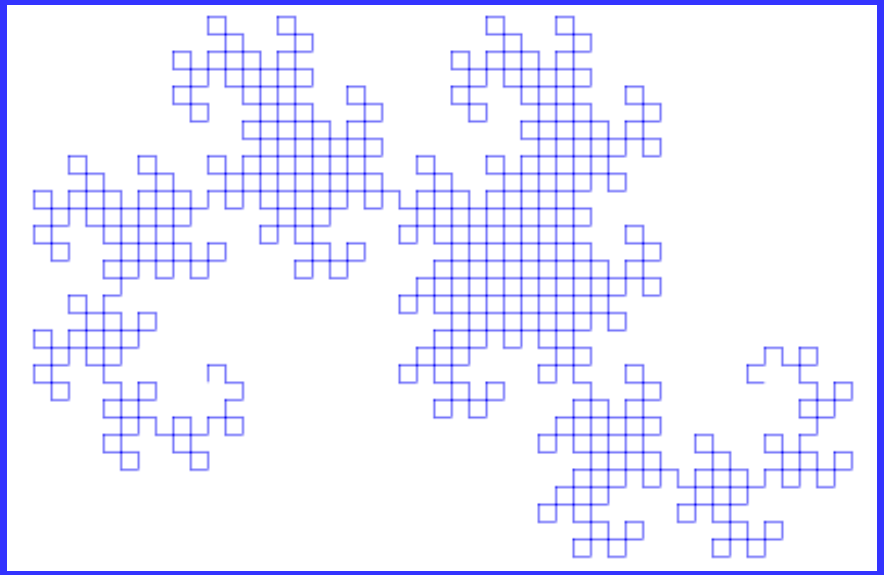
Open the file 80_Sierpinski.py and run the examples that are commented out (lines that start with #).

If you have time open also the file 90_Dragon.py and run the examples there.

Fractals – Dragon curve

Starting from a base segment, replace each segment by 2 segments with a right angle and with a rotation of 45° alternatively to the right and to the left:





Fractals – Dragon curve

start: FX

rules: X → X+YF

Y → FX-Y

F means "draw forward”

- means "turn left 90°“

+ means "turn right 90°".

Here is a program that draws this curve:

```
def dragon(variable, length, n):
    if n == 0:
        turtle.forward(length)
    else:
        if variable == 'X':
            dragon('X',length,n-1)
            turtle.right(90)
            dragon('Y',length,n-1)
            turtle.forward(length)
        else:
            turtle.forward(length)
            dragon('X',length,n-1)
            turtle.left(90)
            dragon('Y',length,n-1)
```

Fractals – Dragon curve

Open the file 90_Dragon.py and run the examples there.

Recursion vs. Iteration

Recursion

```
def pol(length, angle, n):
    if n > 0:
        turtle.forward(length)
        turtle.left(angle)
        pol(length, angle, n-1)
```

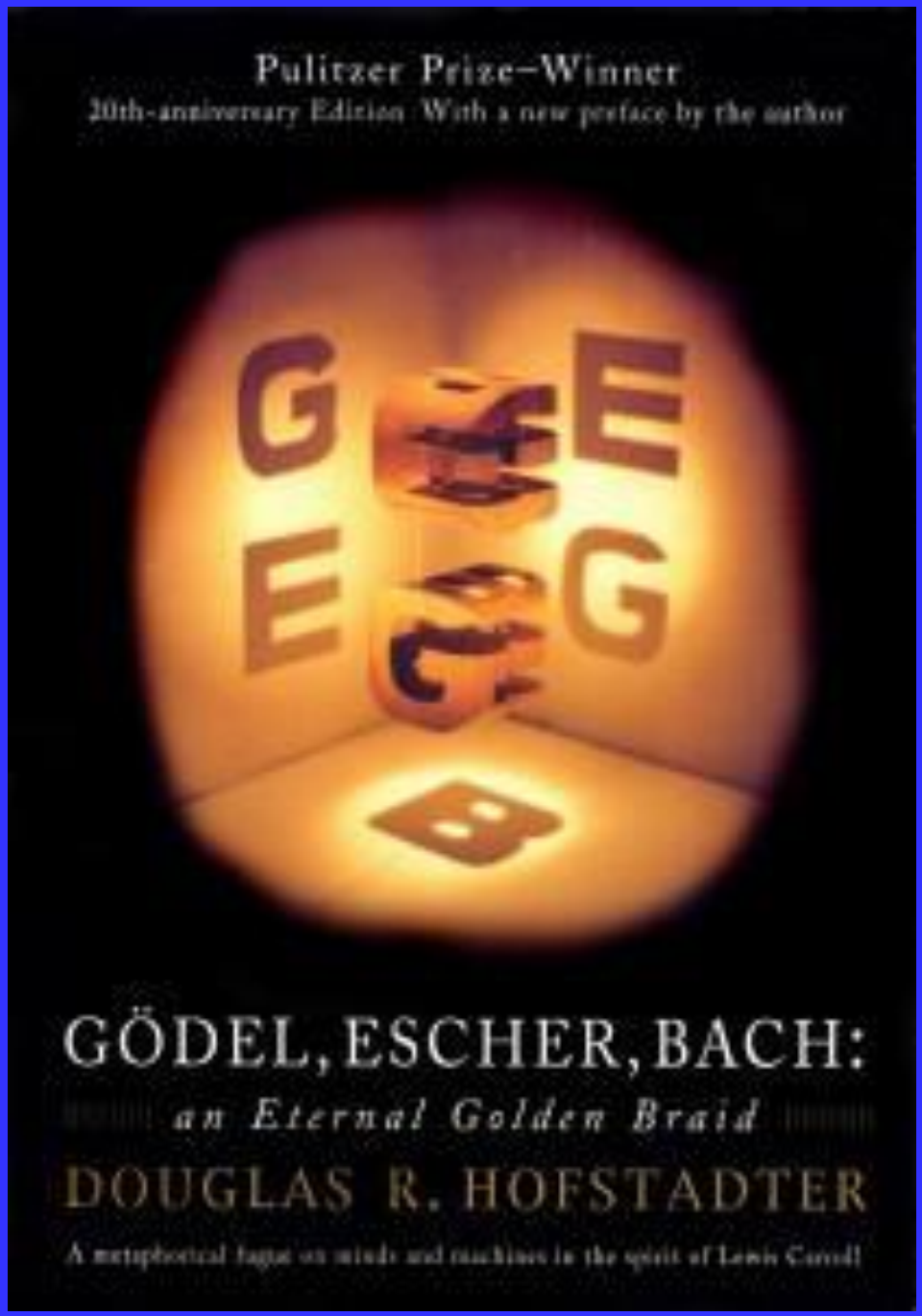
Iteration

```
def pol(length, angle, n):
    for i in range(n):
        turtle.forward(length)
        turtle.left(angle)
```

Example of range:

range(5)=[0,1,2,3,4]

Whatever you can do with recursion you can do with iteration and the other way around.
Usually iteration is used instead of single recursion but not instead of multiple recursion.
Note: to compute the Fibonacci numbers with recursion can be very inefficient.
Some Artificial Intelligence programming languages, like Lisp and Prolog, use recursion heavily.
They belong to different programming paradigms than the most commonly used languages.
The language that we use influences the way we think.



Self-reference,
recursion,
self-reproduction
and much more.

Douglas Hofstadter,
Gödel, Escher, Bach:
an Eternal Golden Braid
(GEB)

From *Gödel, Escher, Bach*

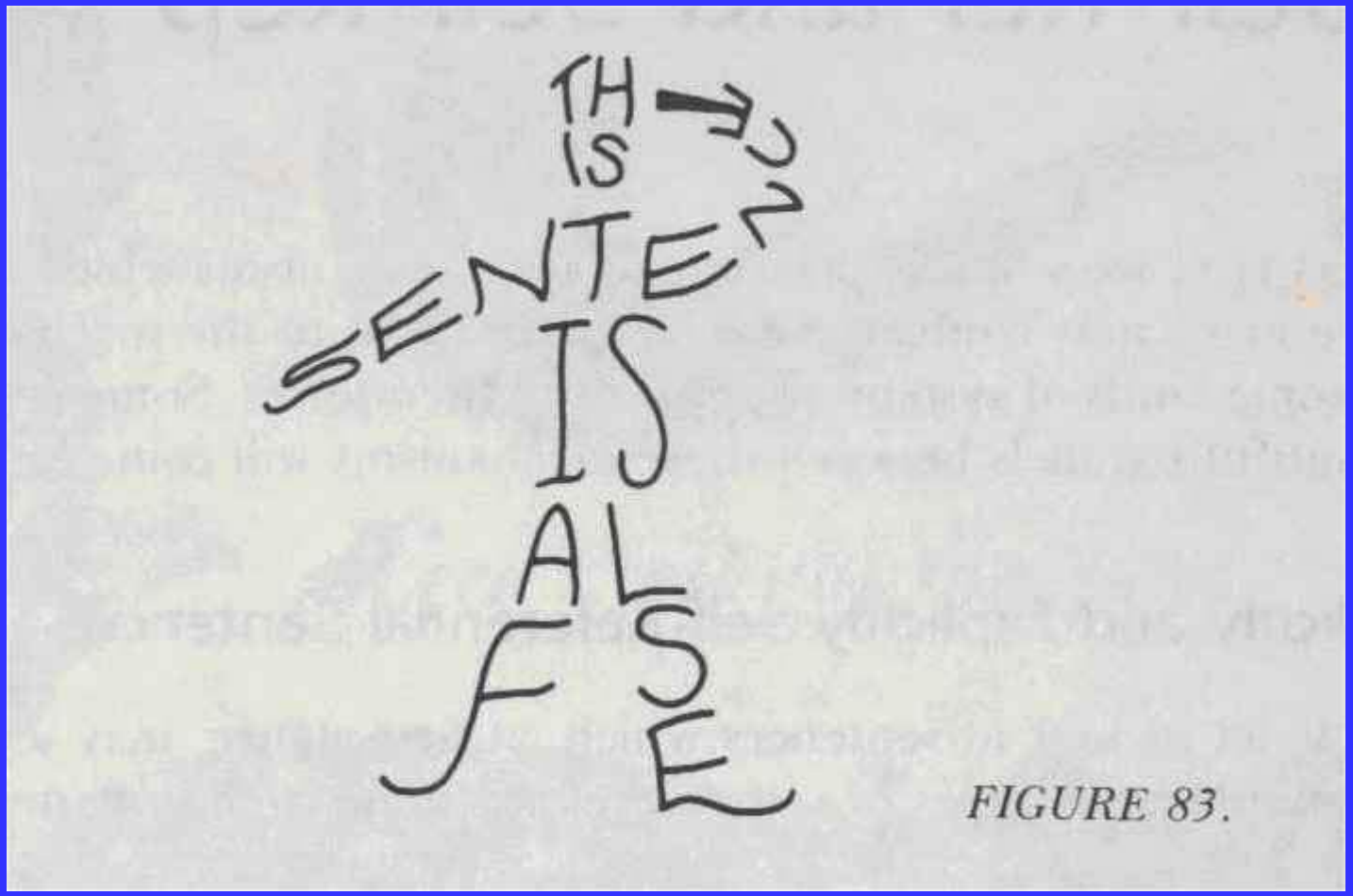


FIGURE 83.

From *Gödel, Escher, Bach*

The sentence

"The sentence

"The sentence

"The sentence

…

…

etc,.,etc.

…

….

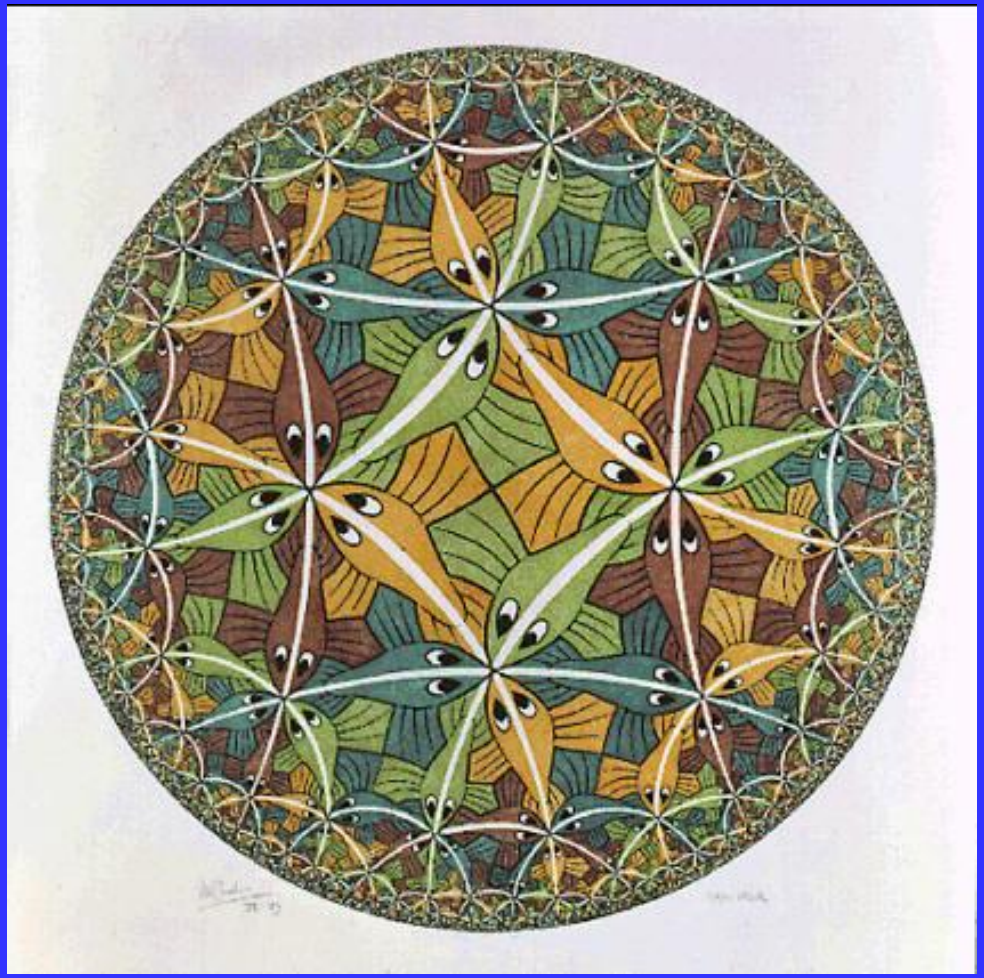
is infinitely long"

is infinitely long"

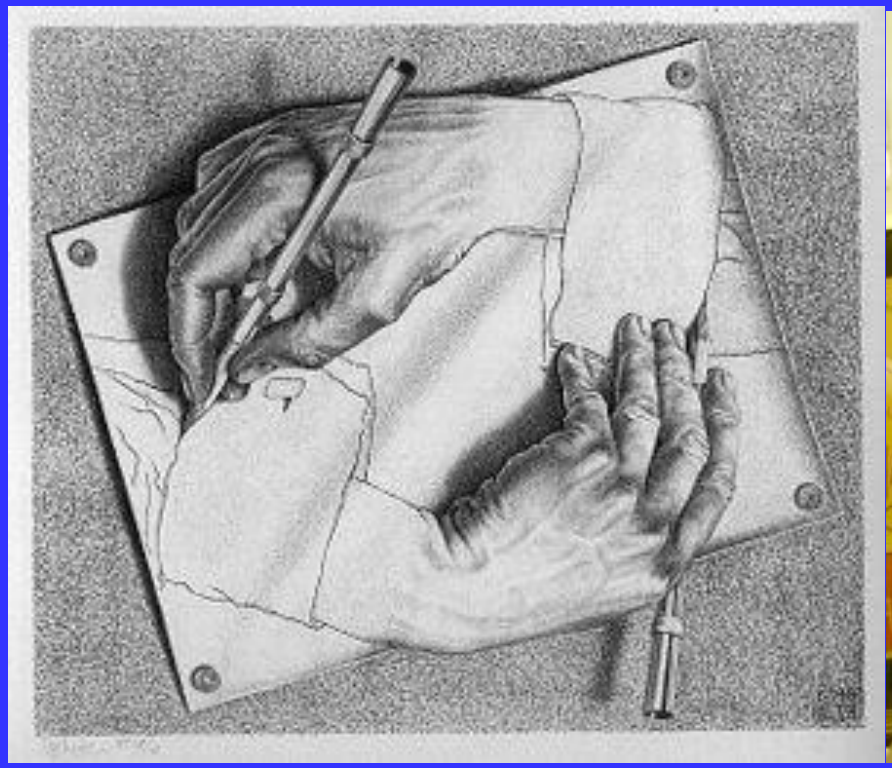
is infinitely long"

is infinitely long.

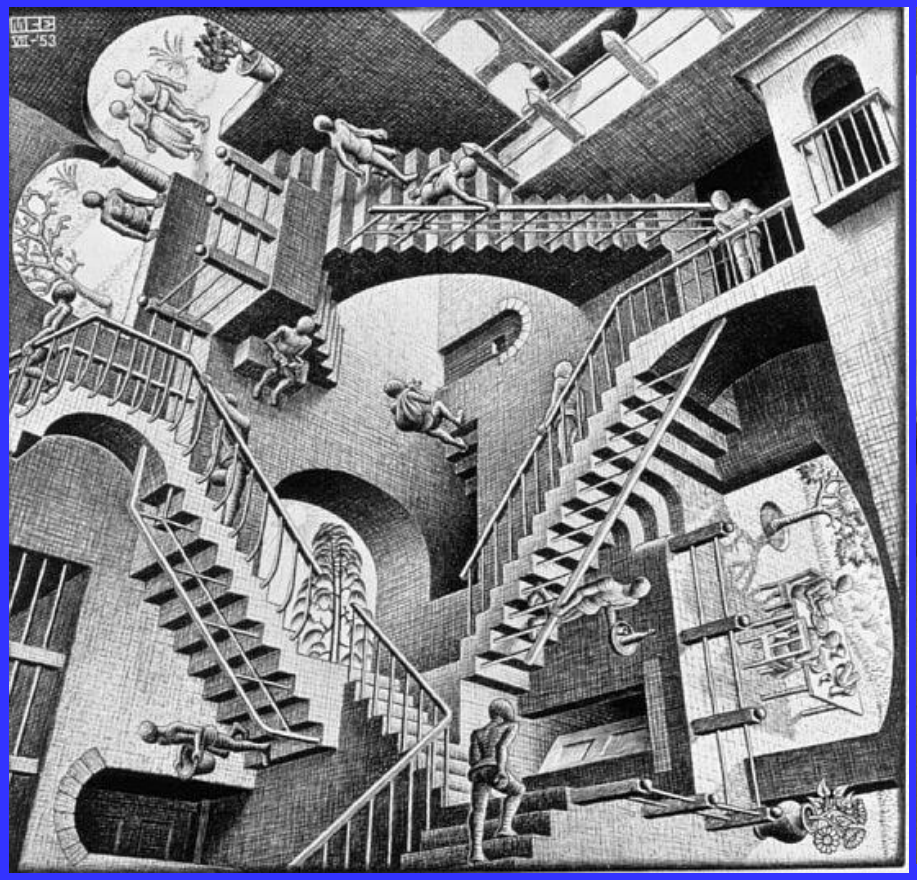
M. C. Escher, *Circle Limit III*

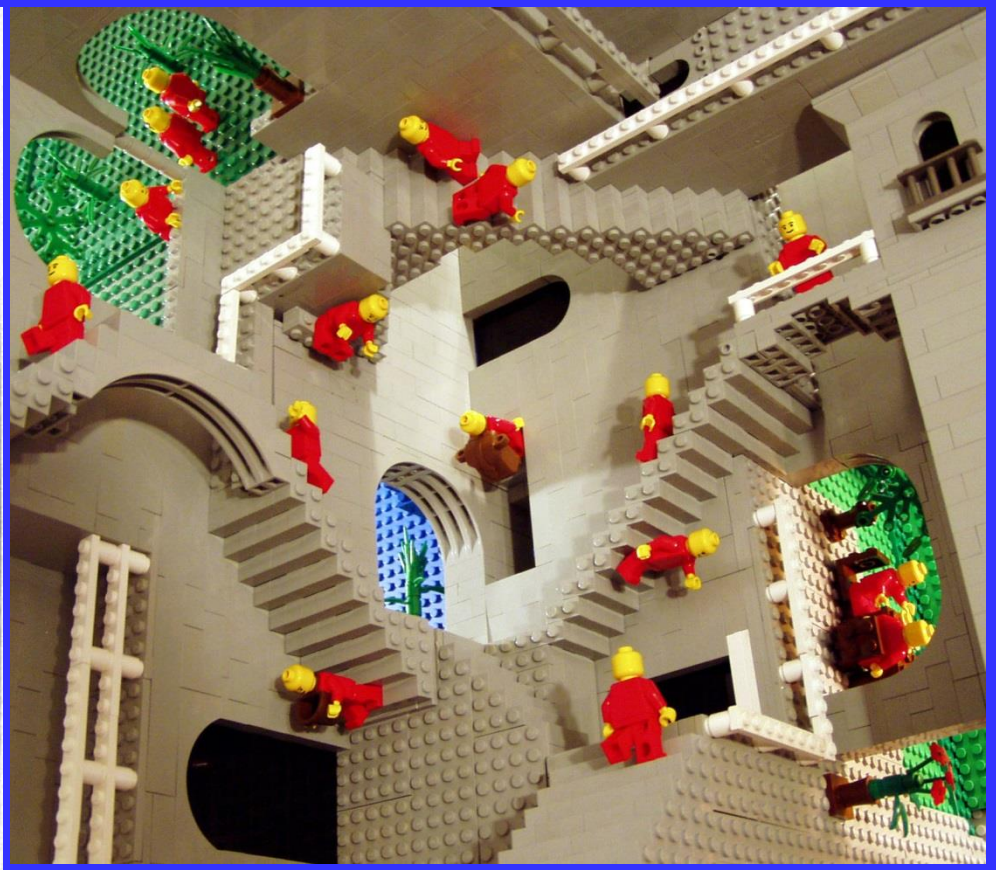


M. C. Escher, *Drawing Hands*

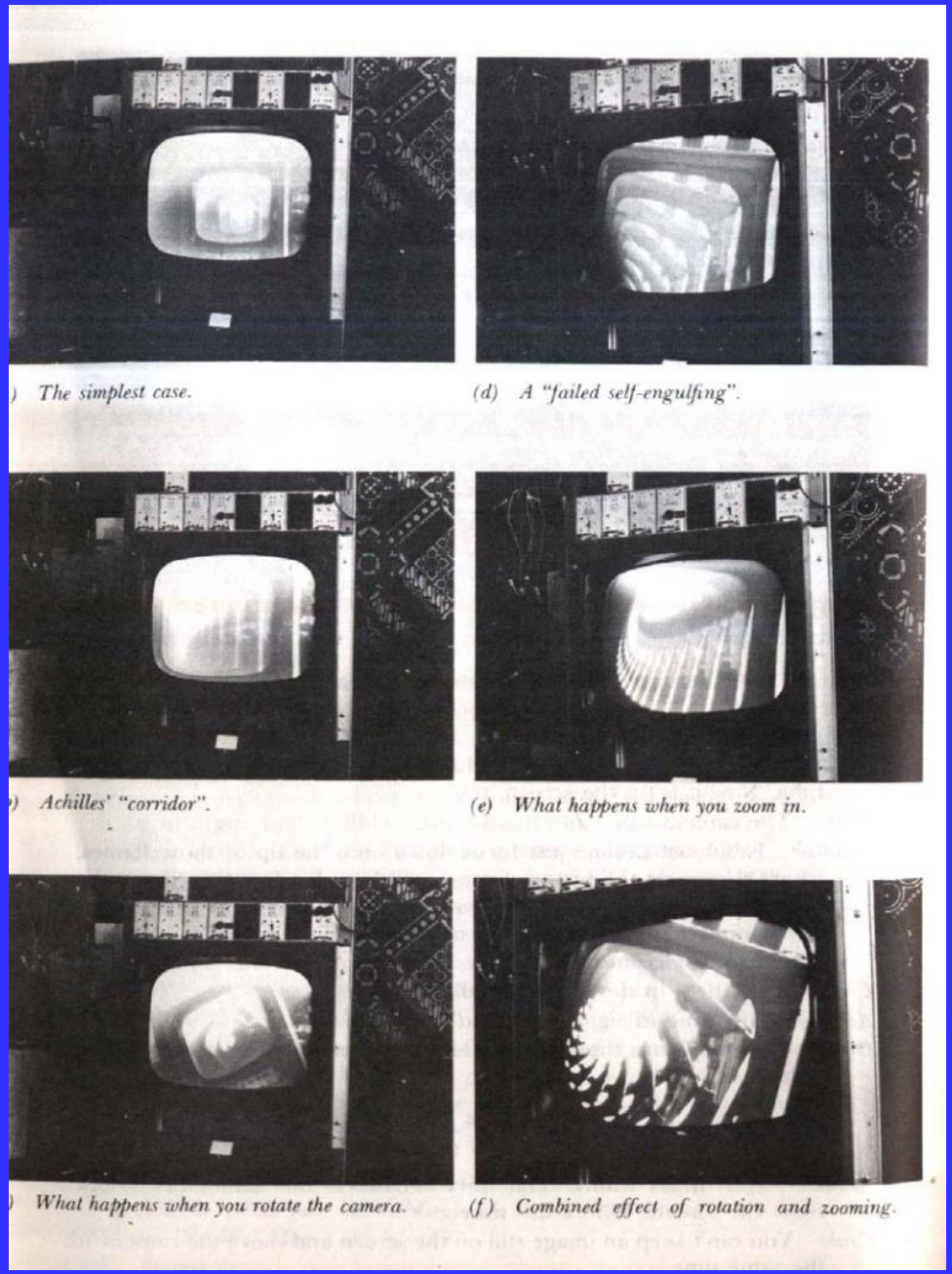


M. C. Escher, *Relativity*





Relativity in Lego from www.andrewlipson.com



) *The simplest case.*

(d) *A "failed self-engulfing".*

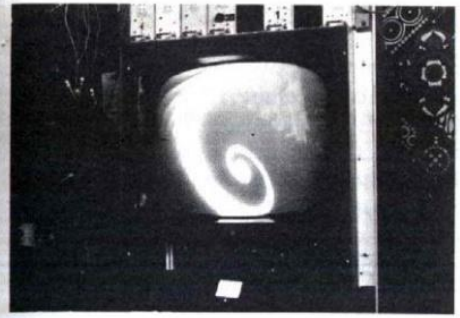
) *Achilles' "corridor".*

(e) *What happens when you zoom in.*

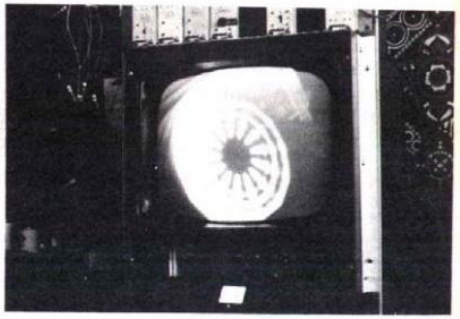
) *What happens when you rotate the camera.*

(f) *Combined effect of rotation and zooming.*

Optical feedback from *Gödel, Escher, Bach*

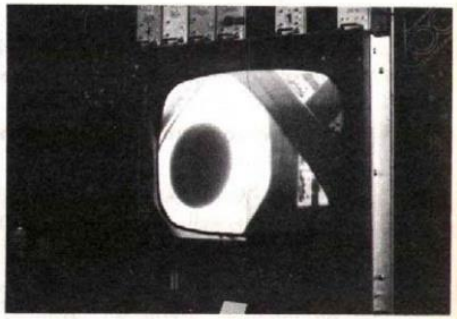


(g) *Starting to get weird . . .*

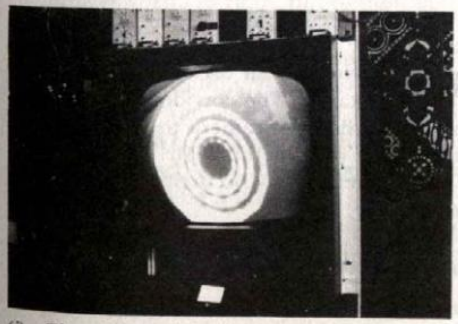


(j) *The late stages of a galaxy. Count the number of spokes!*

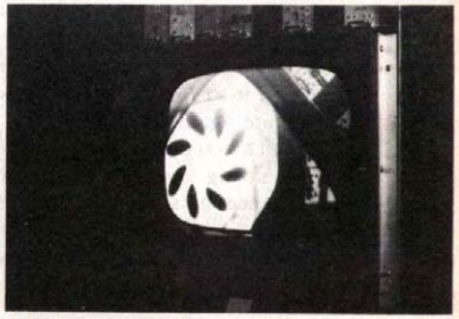
(h) *A "galaxy" is born.*



(k) *The galaxy has burned itself out, and become—a black hole!*



(i) *The galaxy evolves . . .*



(l) *A "pulsating petal pattern", caught in the middle of one of its pulsations.*

Optical feedback from *Gödel, Escher, Bach*

Self-reference - Optical feedback

I need two volunteers…

That’s all!

Please take a copy of the answers.

Thank you.