递归函数

车万翔

哈尔滨工业大学

两个和尚的故事

“从前有座山,山里有座庙,庙里有个老和尚给小和尚讲故事,讲什么呢?” “从前有座山,山里有座庙,庙里有个老和尚给小和尚讲故事,讲什么呢?”

“从前有座山,山里有座庙,庙里有个老和尚给小和尚讲故事,讲什么呢?” ……

递归的定义

递归：程序调用自身

形式：在函数定义有直接或间接调用自身

阶乘

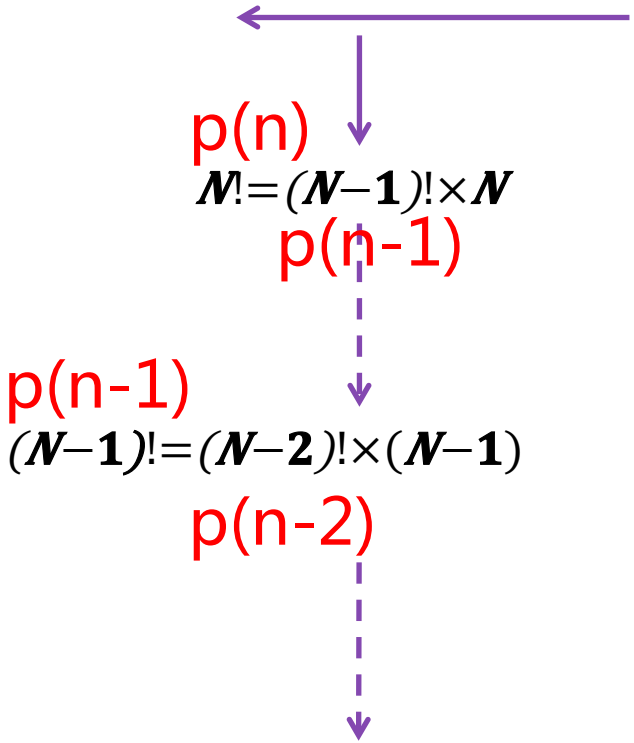
- 阶乘：

```
def p(n):
    x = 1
    i = 1
    while i <= n:
        x = x * i
        i= i + 1
    return x

n = int(raw_input("请输入一个整数:"))
print n, "!的值为", p(n)
```

阶乘

- 阶乘：



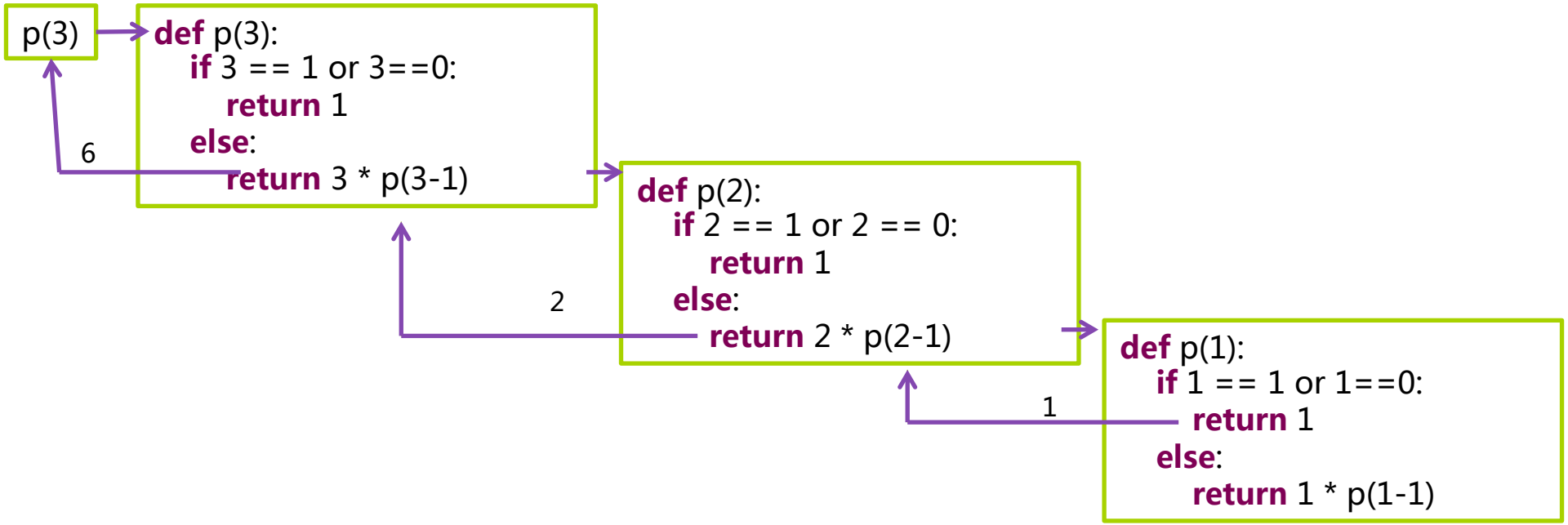
阶乘

- 阶乘：

```
def p(n):
    if n == 1 or n == 0:
        return 1
    else:
        return n * p(n-1)

n = int(raw_input("请输入一个整数:"))
print n, " !的值为：", p(n)
```

阶乘


p(3)
def p(3):
if 3 == 1 or 3==0:
return 1
else:
return 3 * p(3-1)
6
def p(2):
if 2 == 1 or 2 == 0:
return 1
else:
return 2 * p(2-1)
2
def p(1):
if 1 == 1 or 1==0:
return 1
else:
return 1 * p(1-1)
1

阶乘

```
def p(n):
    if n == 1 or n == 0:
        return 1
    else:
        return n * p(n-1)
```

初始条件

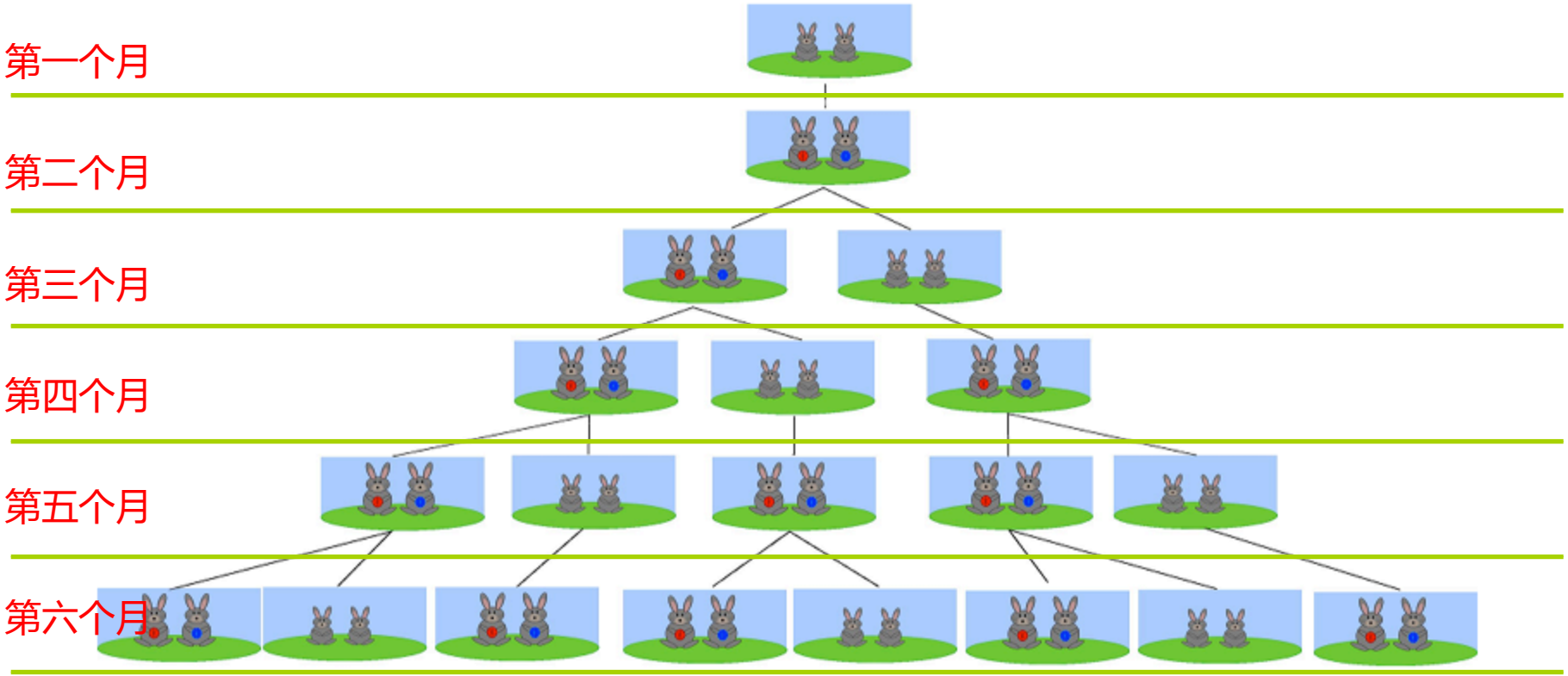
递归

掐头去尾留中间

递归解决问题的思想

- if 问题足够简单：
  - 直接解决问题
  - 返回解
- else:
  - 将问题分解为与原问题同构的一个或多个更小的问题
  - 逐个解决这些更小的问题
  - 将结果组合为，获得最终的解
  - 返回解

兔子数列

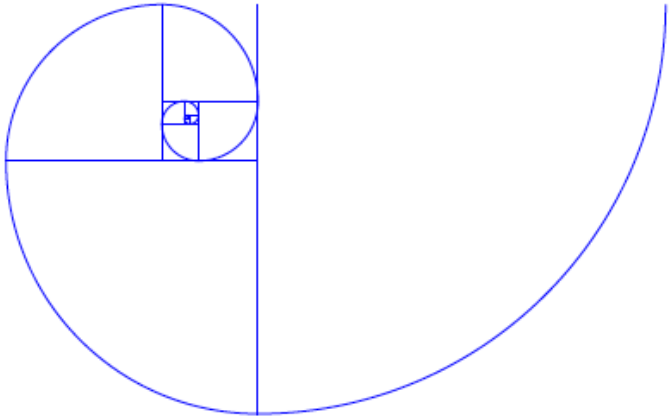


兔子数列

斐波那契数列

- 是这样一个数列：1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89……

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



斐波那契数列

- **斐波那契数列：1, 1, 2, 3, 5, 8, 13, 21……**

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

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

初始条件

递归

斐波那契数列

fib(4)

```
def fib(4):
    if 4 == 1 or 4 == 2:
        return 1
    else:
        return fib(4-1) + fib(4-2)
```

```
def fib(2):
    if 2 == 1 or  2 == 2:
        return 1
    else:
        return fib(2-1) + fib(2-2)
```

```
def fib(3):
    if 3 == 1 or  3 == 2:
        return 1
    else:
        return fib(3-1) + fib(3-2)
```

```
def fib(1):
    if 1 == 1 or  1 == 2:
        return 1
    else:
        return fib(1-1) + fib(1-2)
```

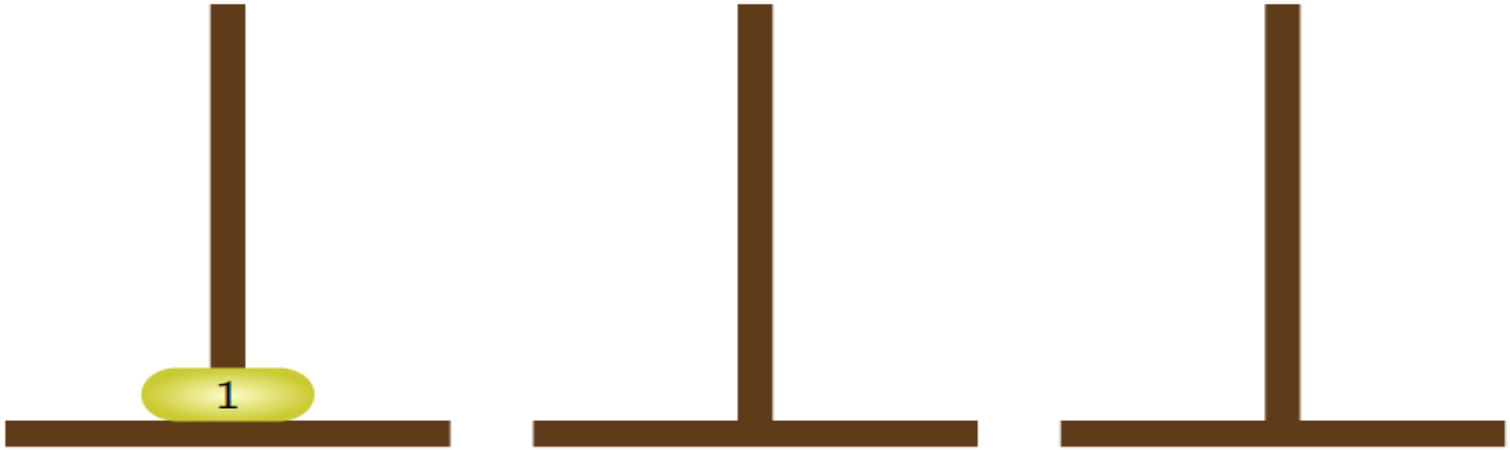
递归 - 汉诺塔

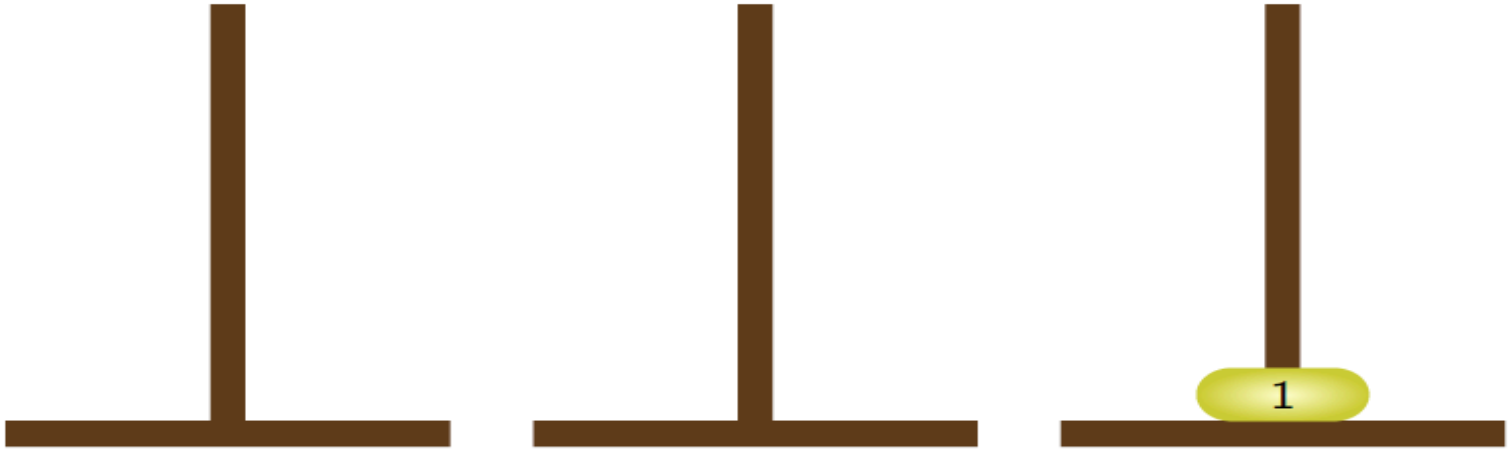
- **在印度，有这么一个古老的传说:**

开天辟地的神勃拉玛（和中国的盘古差不多的神）在一个庙里留下了三根金刚石的棒，第一根上面套着64个圆的金片，最大的一个在底下，其余一个比一个小，依次叠上去，庙里的众僧不倦地把它们一个个地从这根棒搬到另一根棒上，规定可利用中间的一根棒作为帮助，但每次只能搬一个，而且大的不能放在小的上面

移动圆片的次数:

18446744073709551615

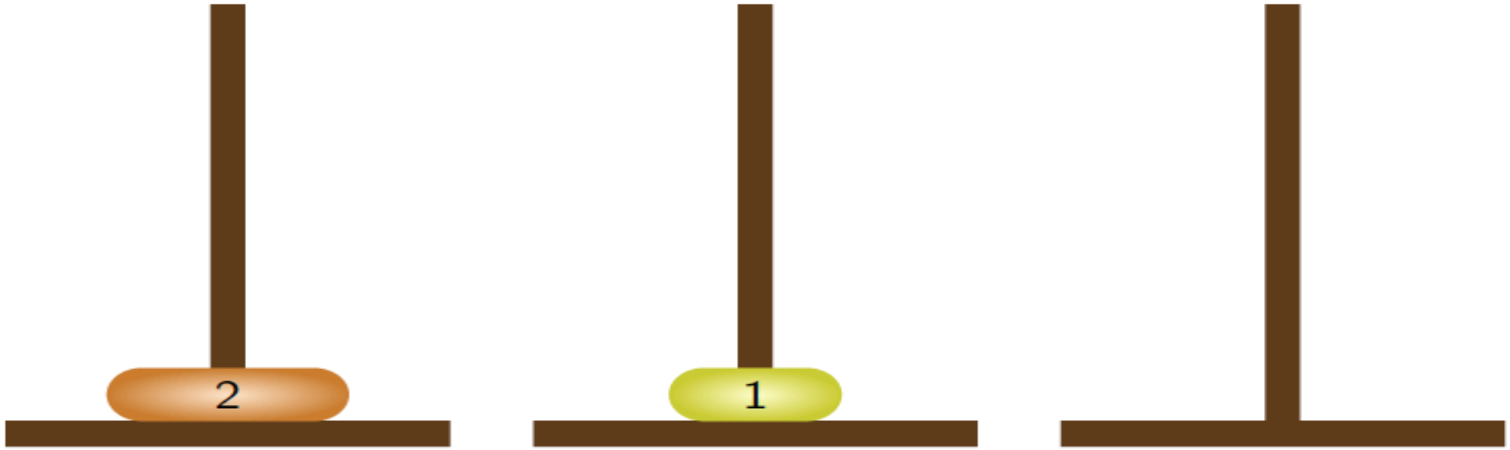

1



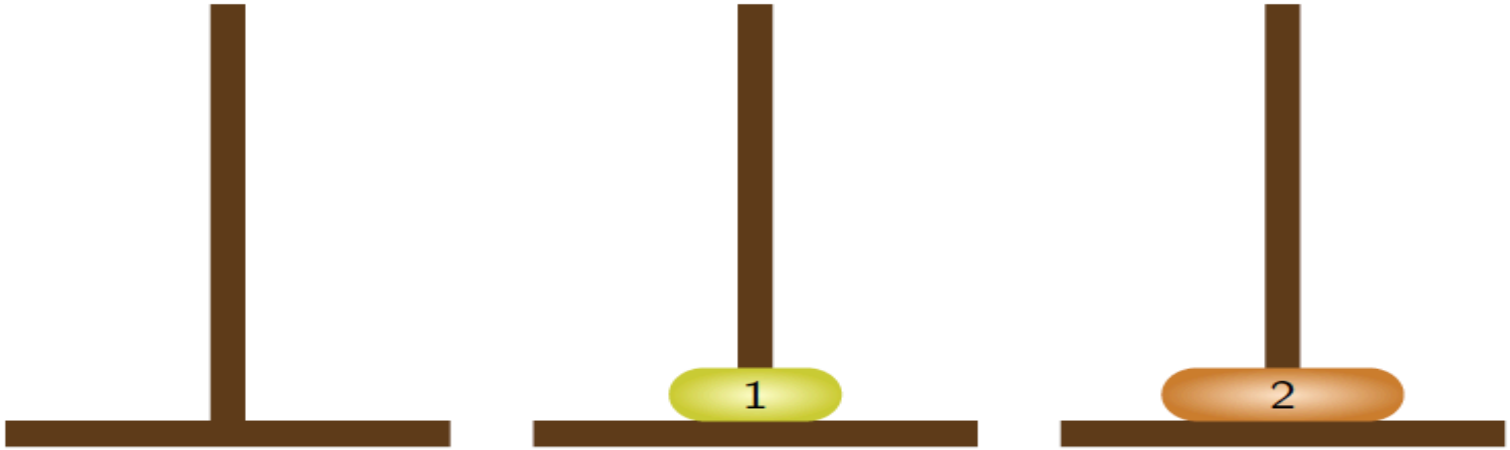
Moved disc from pole 1 to pole 3.


ok
1

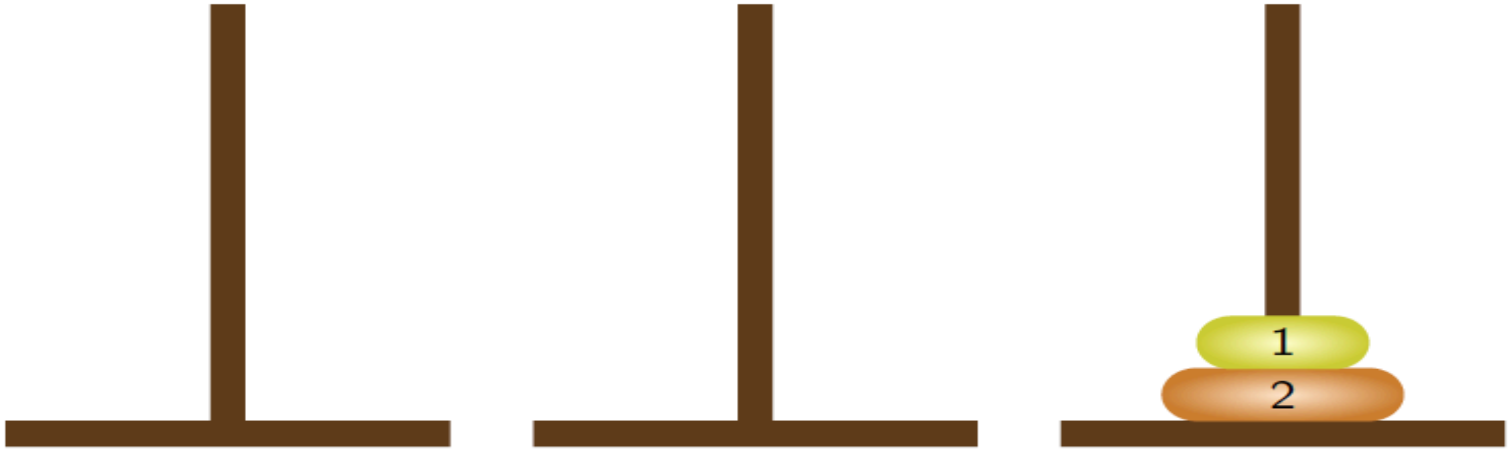

1
2



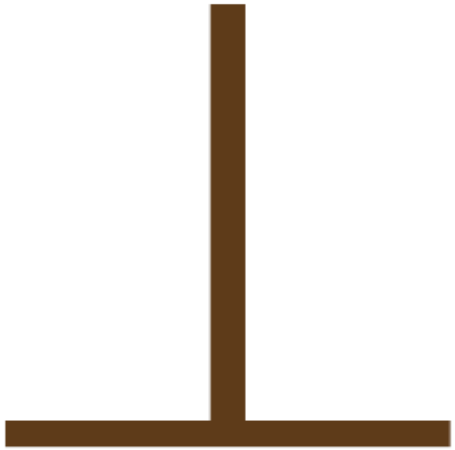
Moved disc from pole 1 to pole 2.



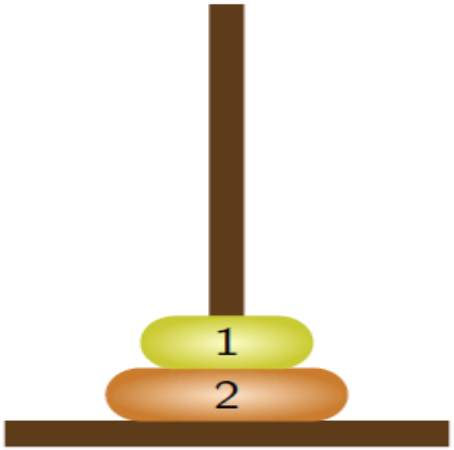
Moved disc from pole 1 to pole 3.

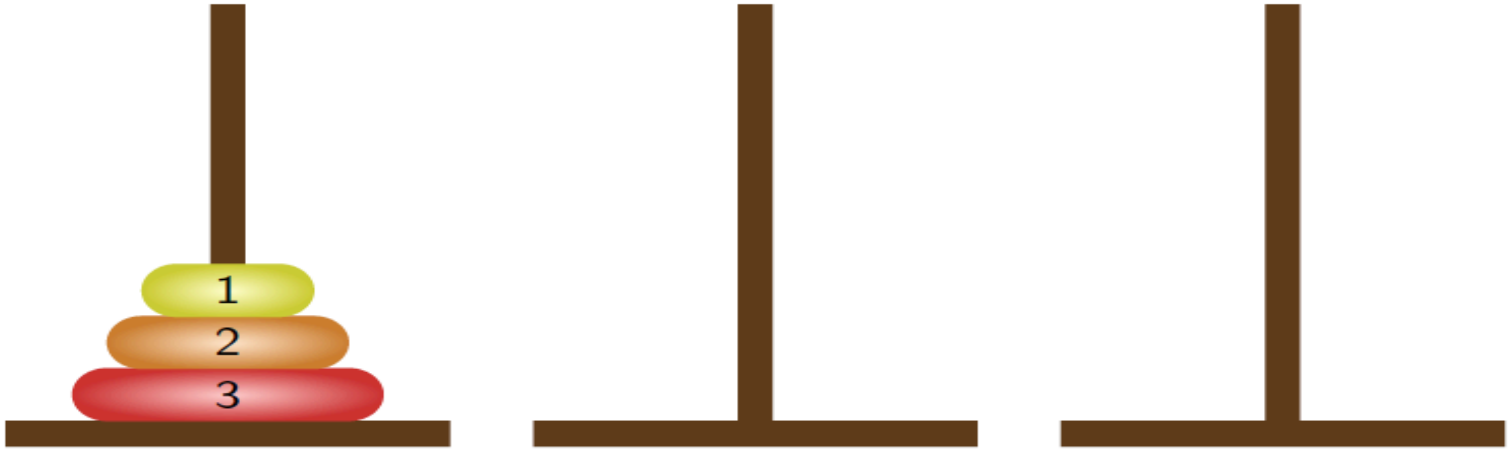


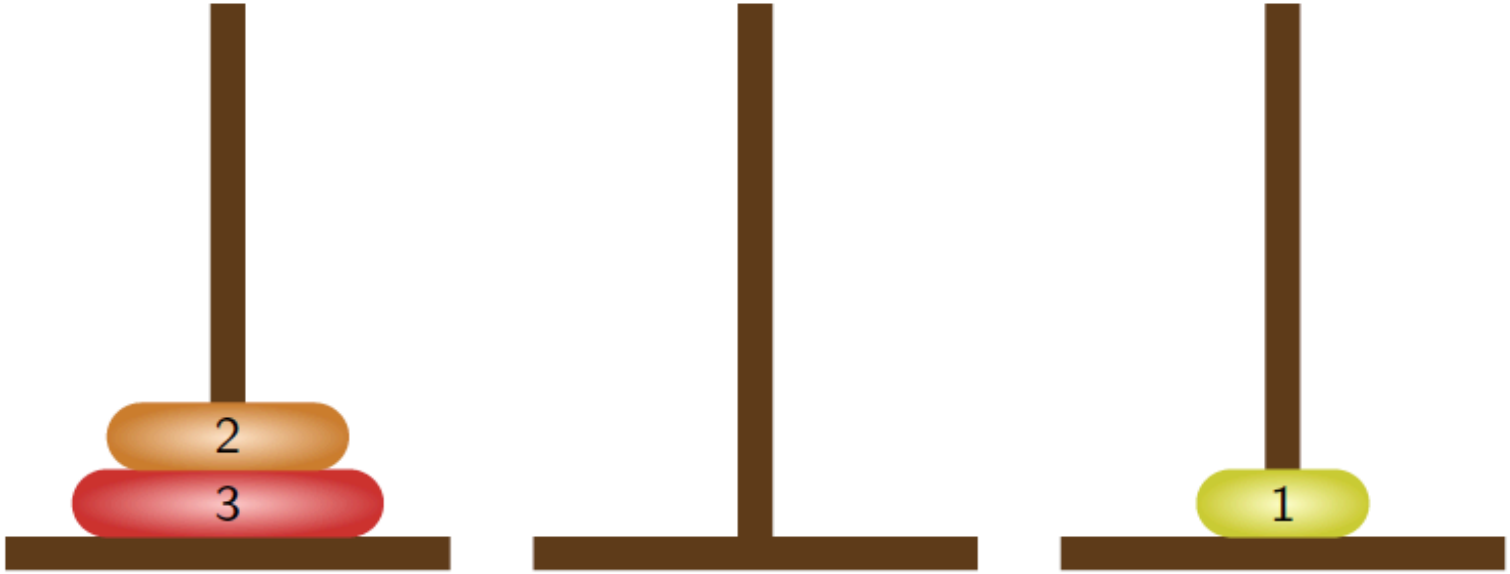
Moved disc from pole 2 to pole 3.



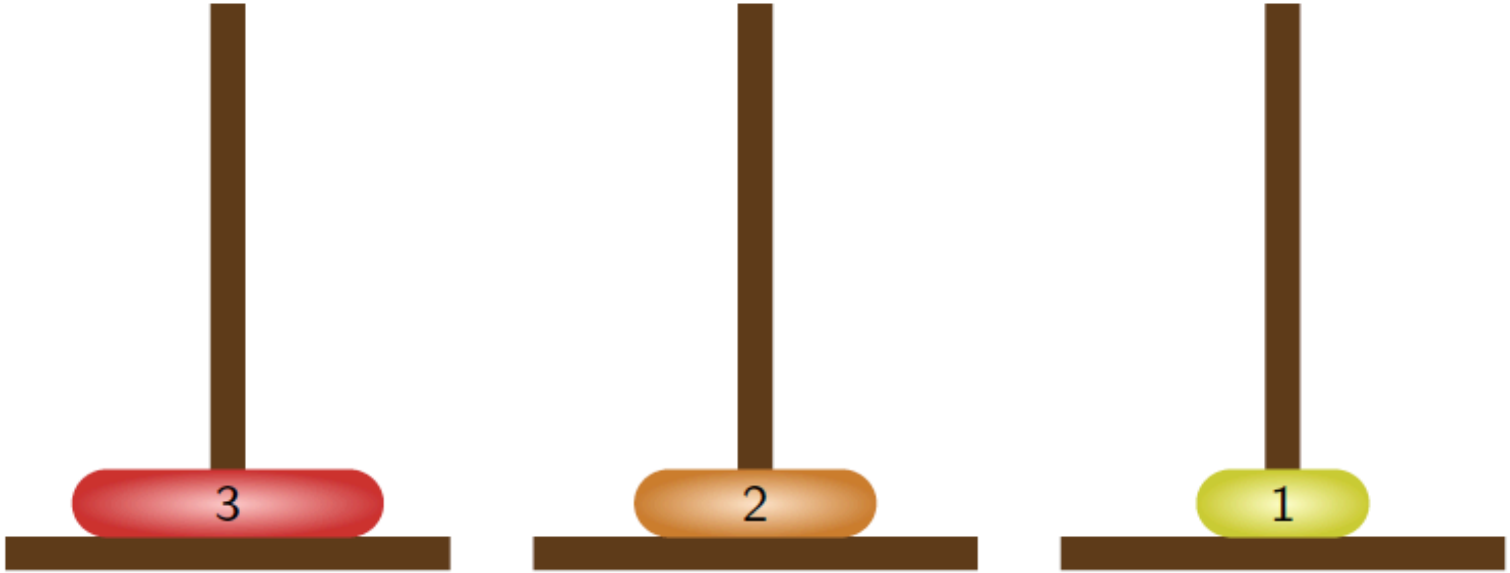

ok


1
2

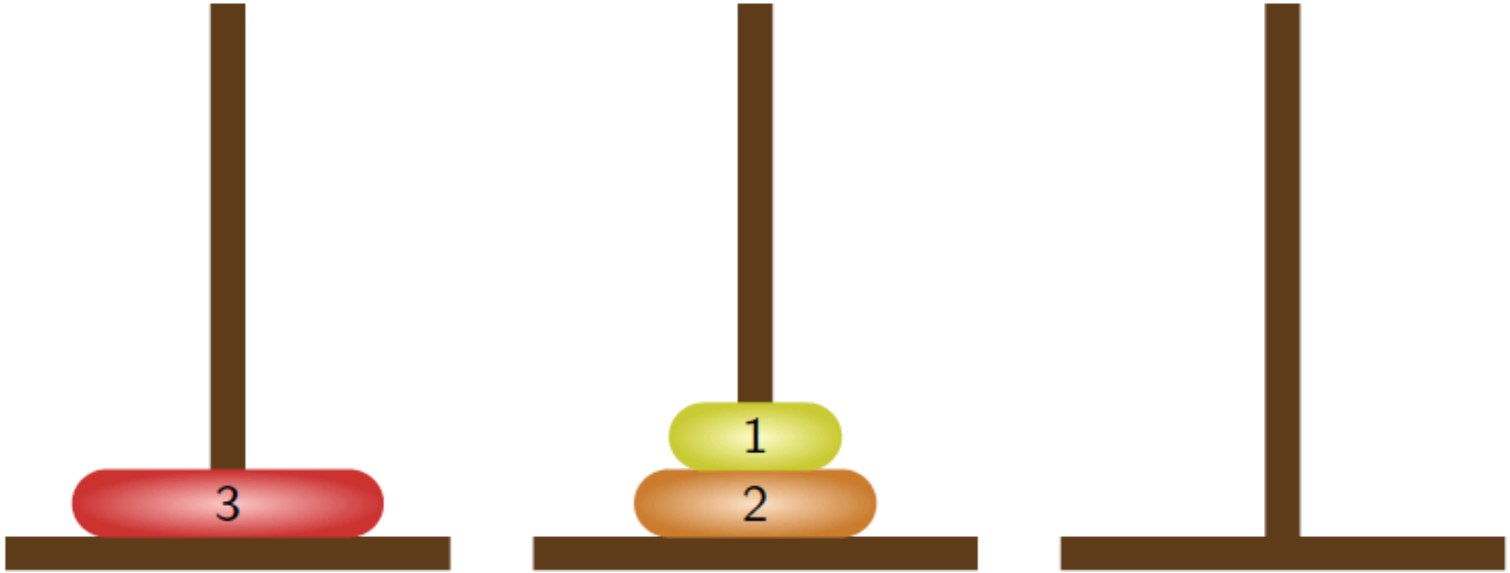

1
2
3



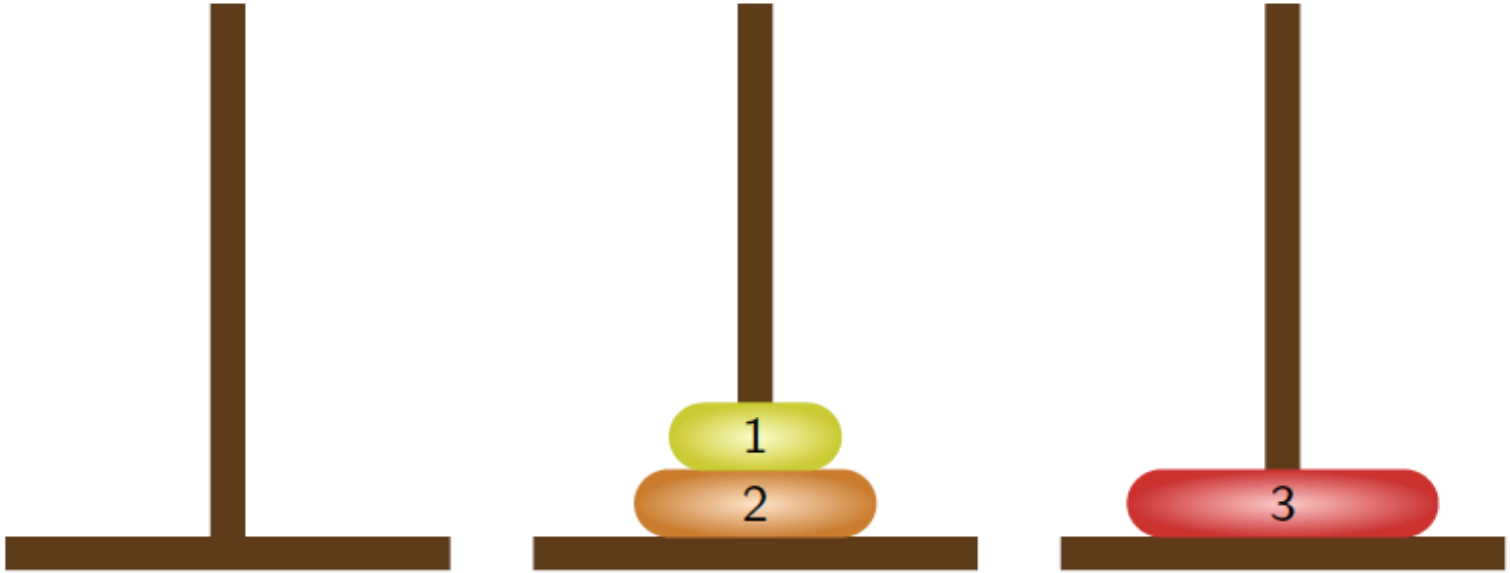
Moved disc from pole 1 to pole 3.



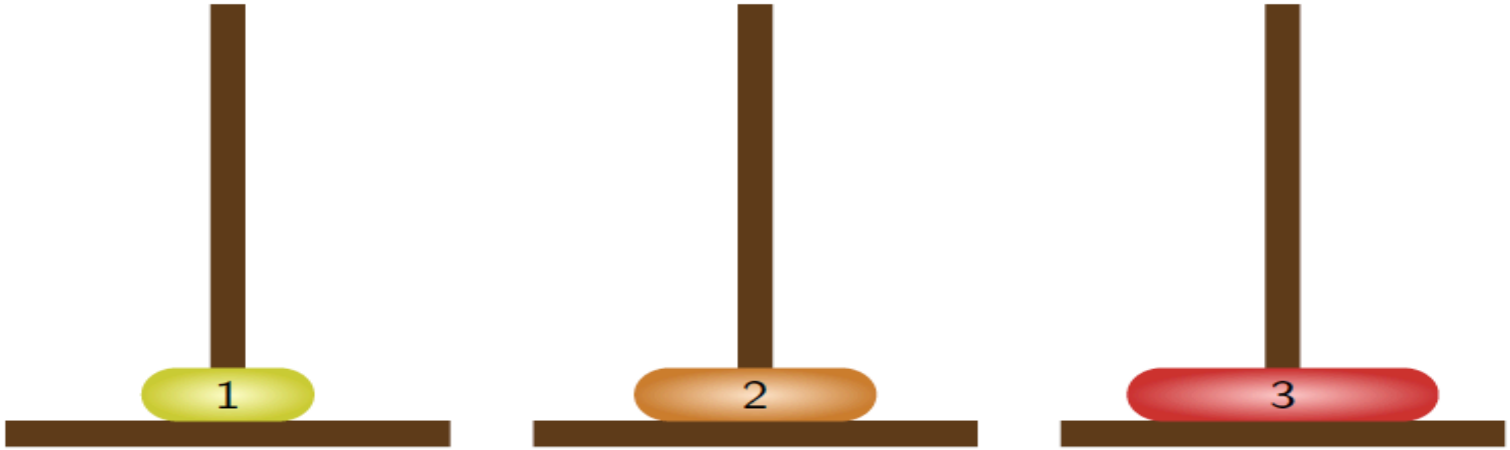
Moved disc from pole 1 to pole 2.



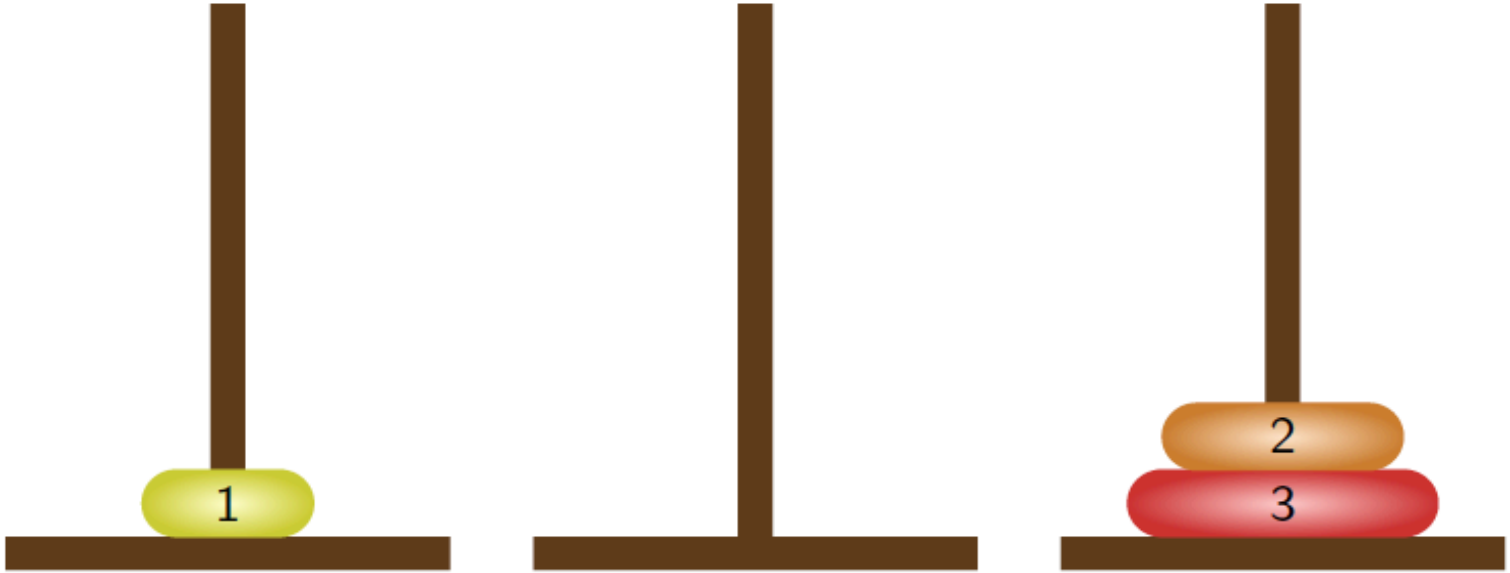
Moved disc from pole 3 to pole 2.



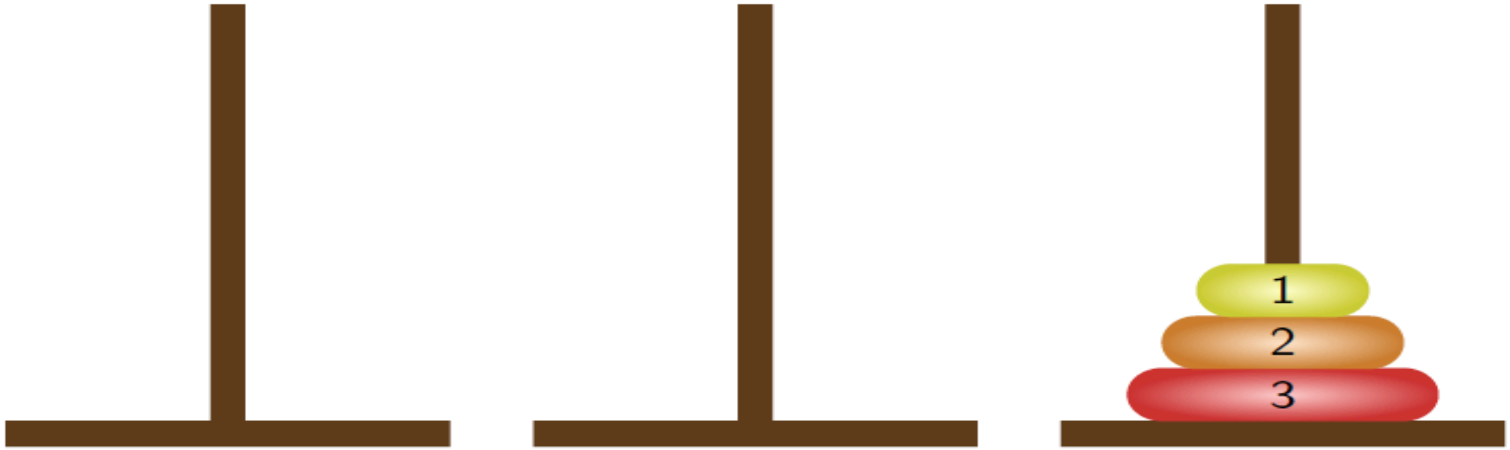
Moved disc from pole 1 to pole 3.



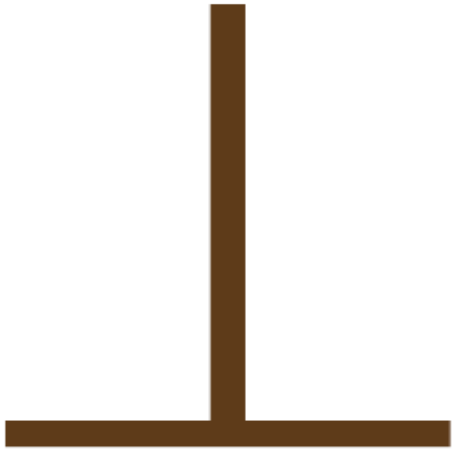
Moved disc from pole 2 to pole 1.



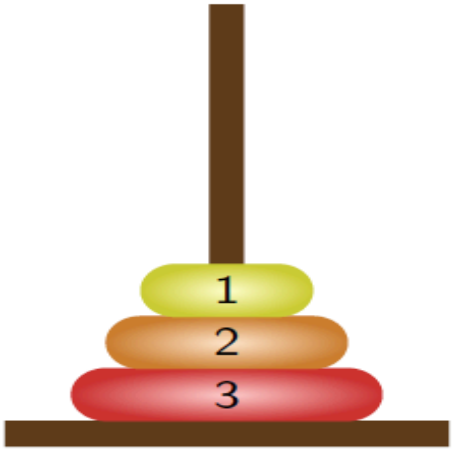
Moved disc from pole 2 to pole 3.

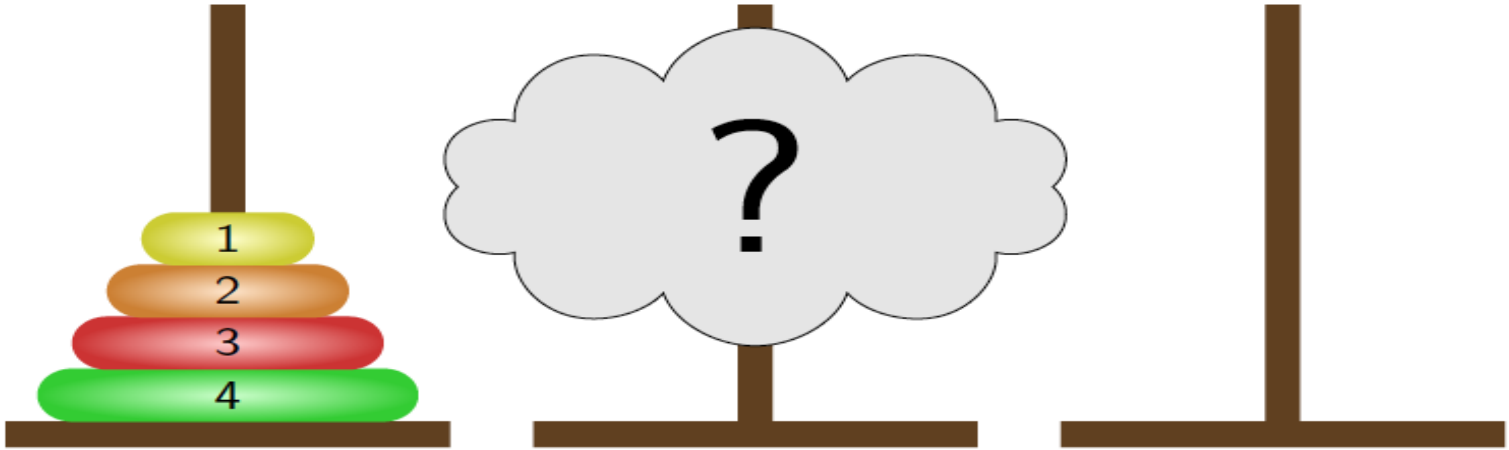


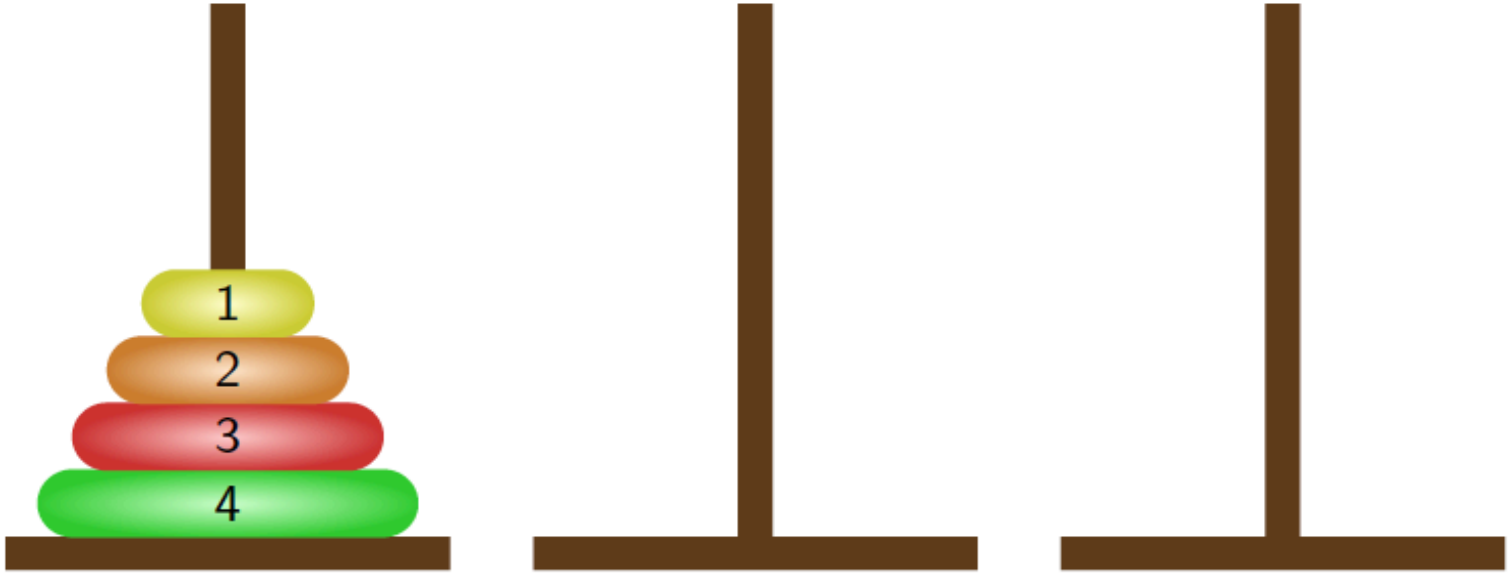
Moved disc from pole 1 to pole 3.

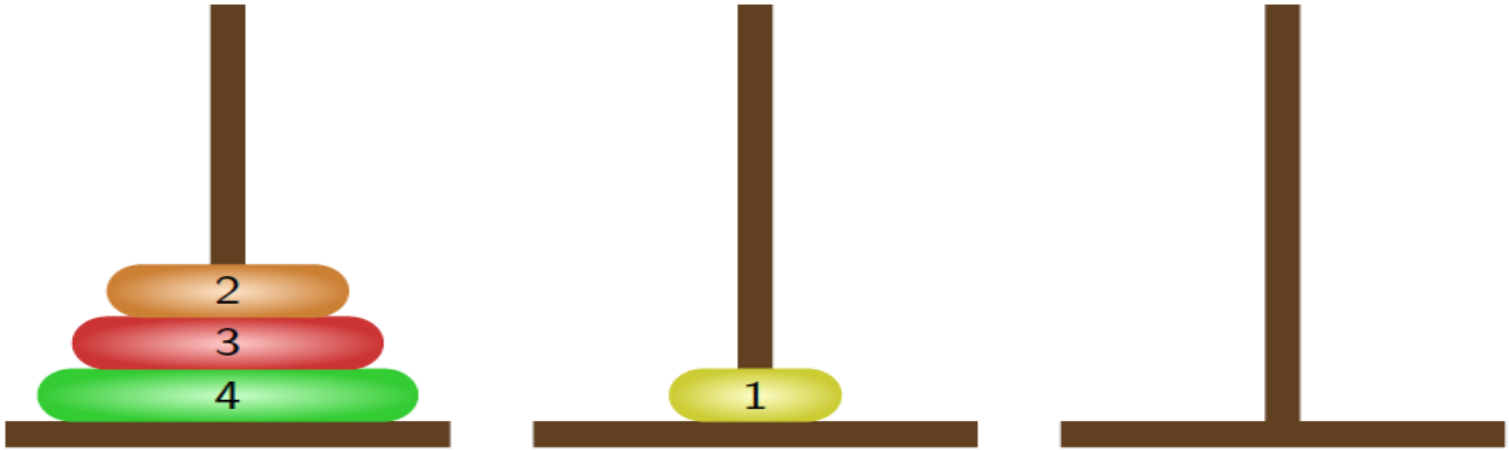



ok

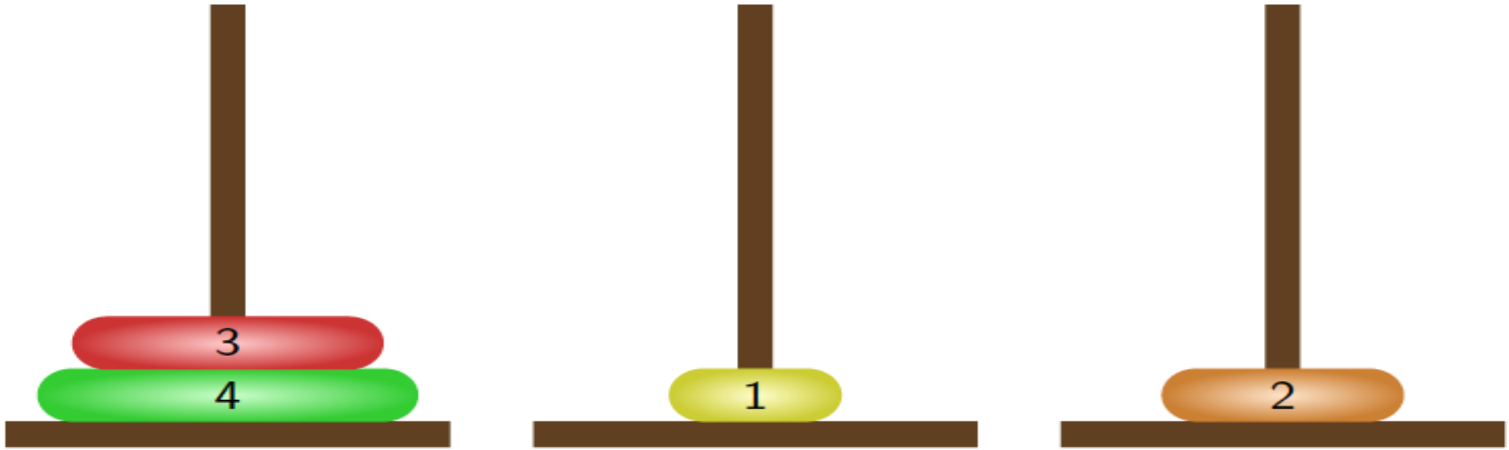

1
2
3


1
2
3
4
?


1
2
3
4

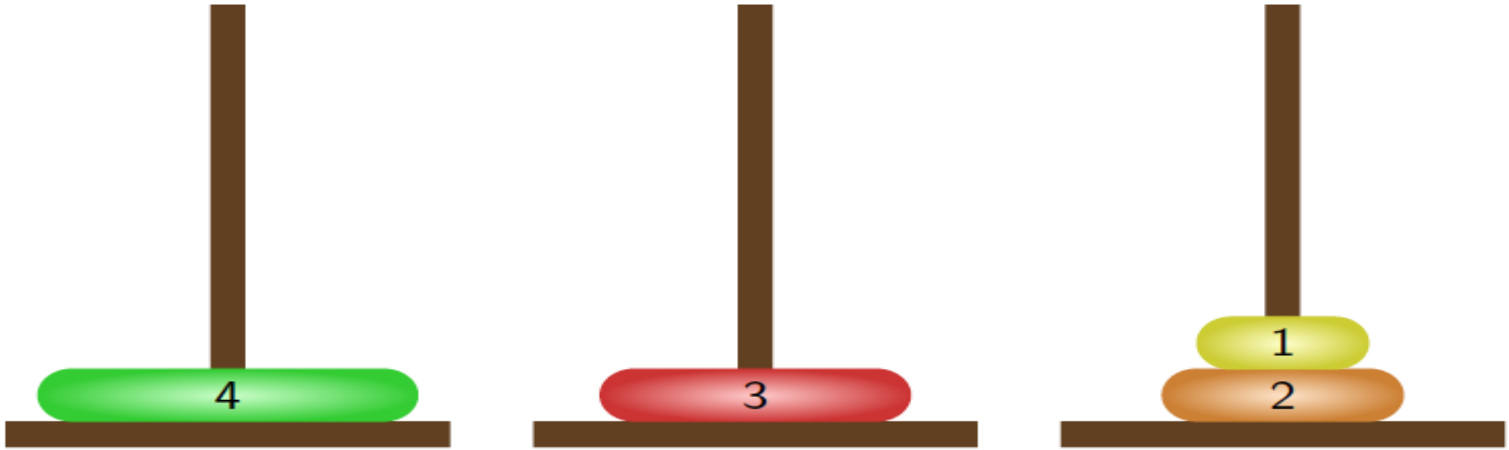


Moved disc from pole 1 to pole 2.

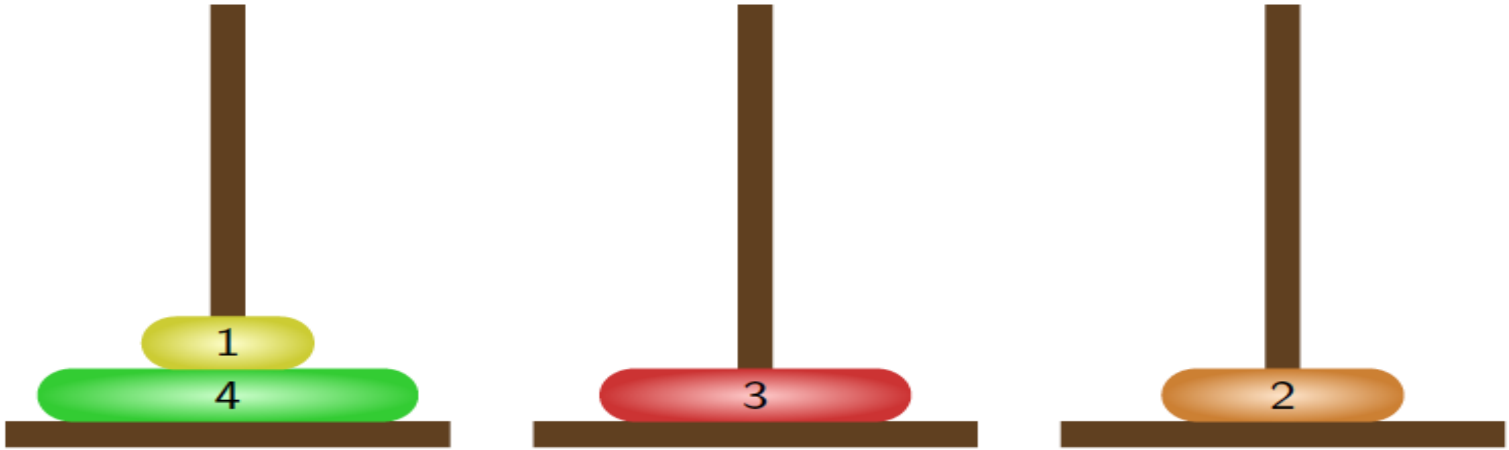


Moved disc from pole 1 to pole 3.

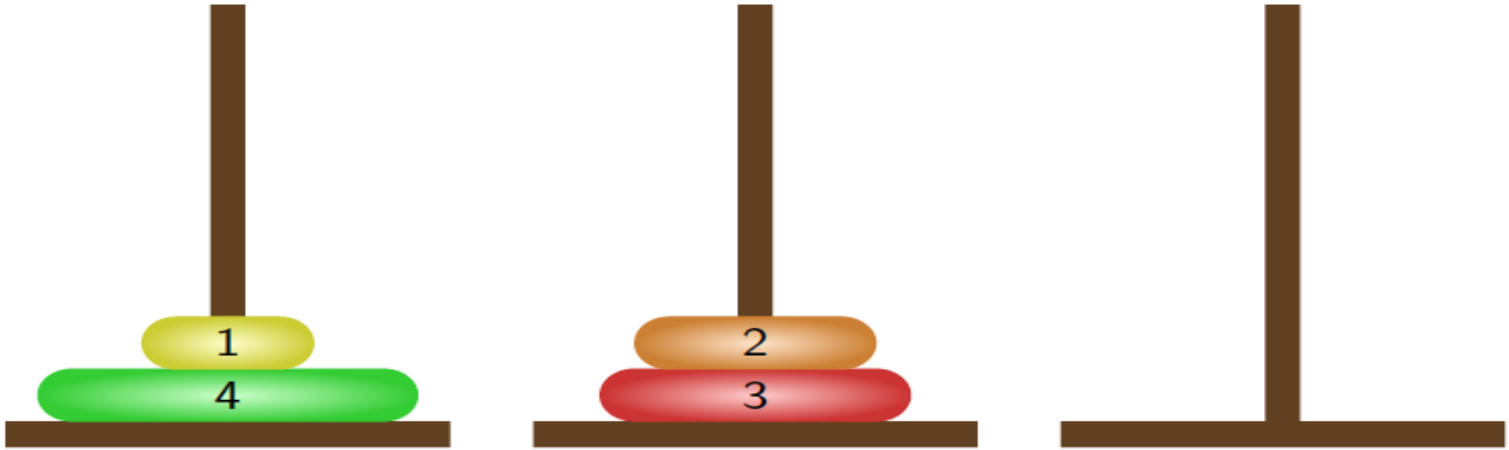
Moved disc from pole 2 to pole 3.



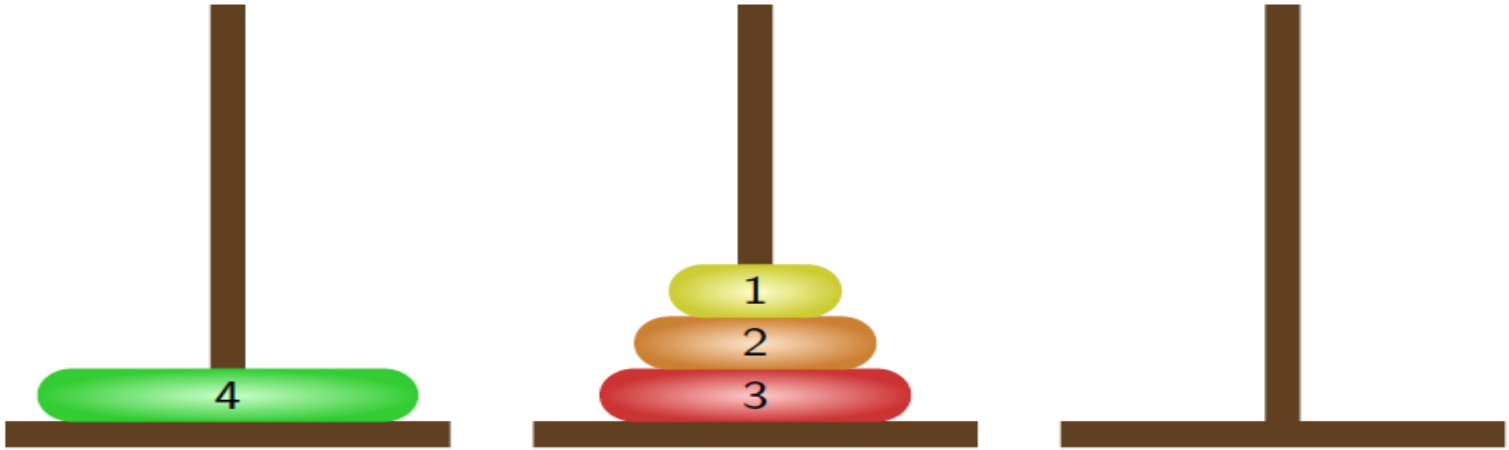
Moved disc from pole 1 to pole 2.



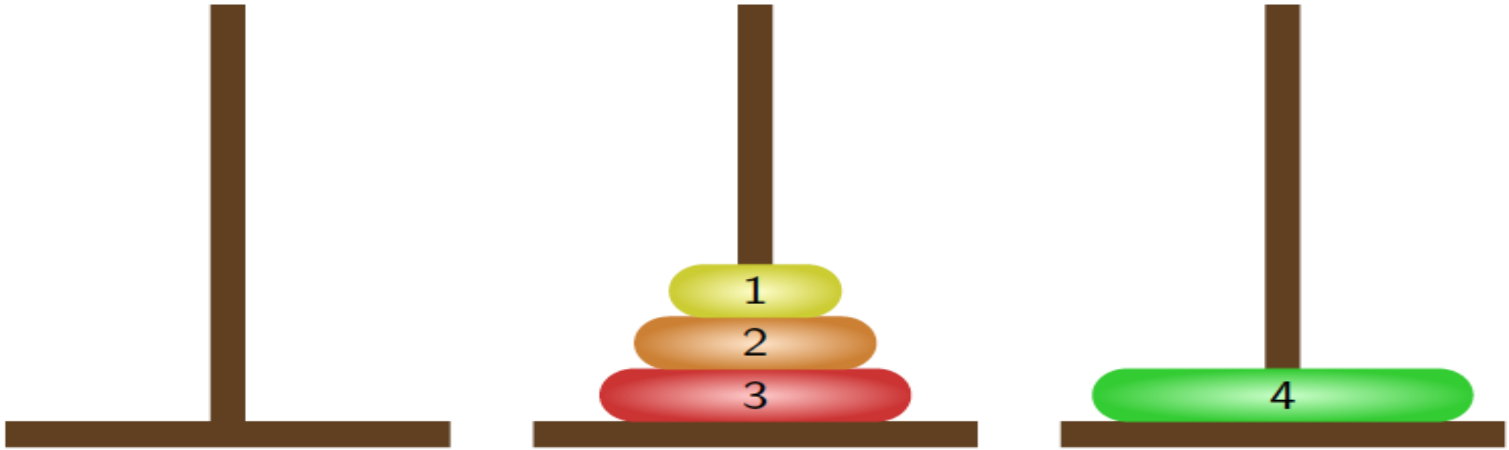
Moved disc from pole 3 to pole 1.



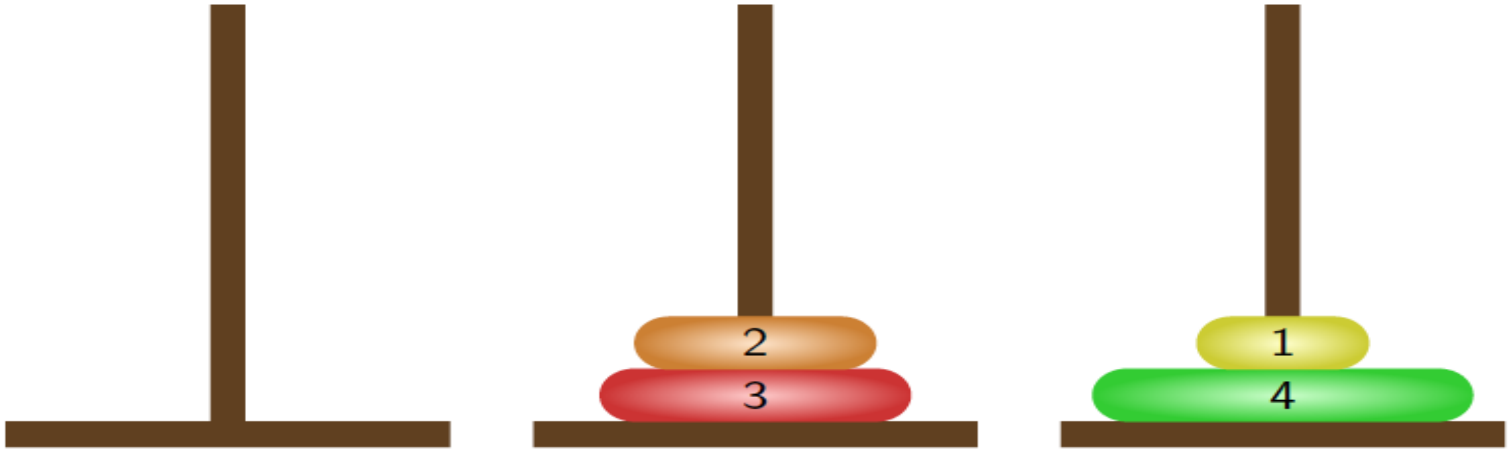
Moved disc from pole 3 to pole 2.



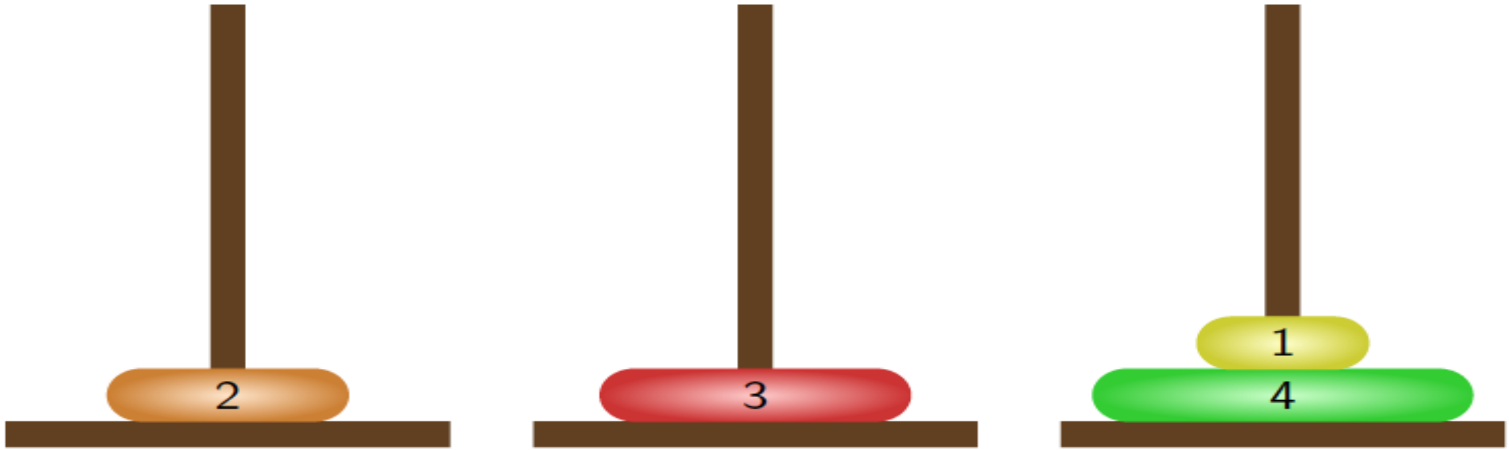
Moved disc from pole 1 to pole 2.



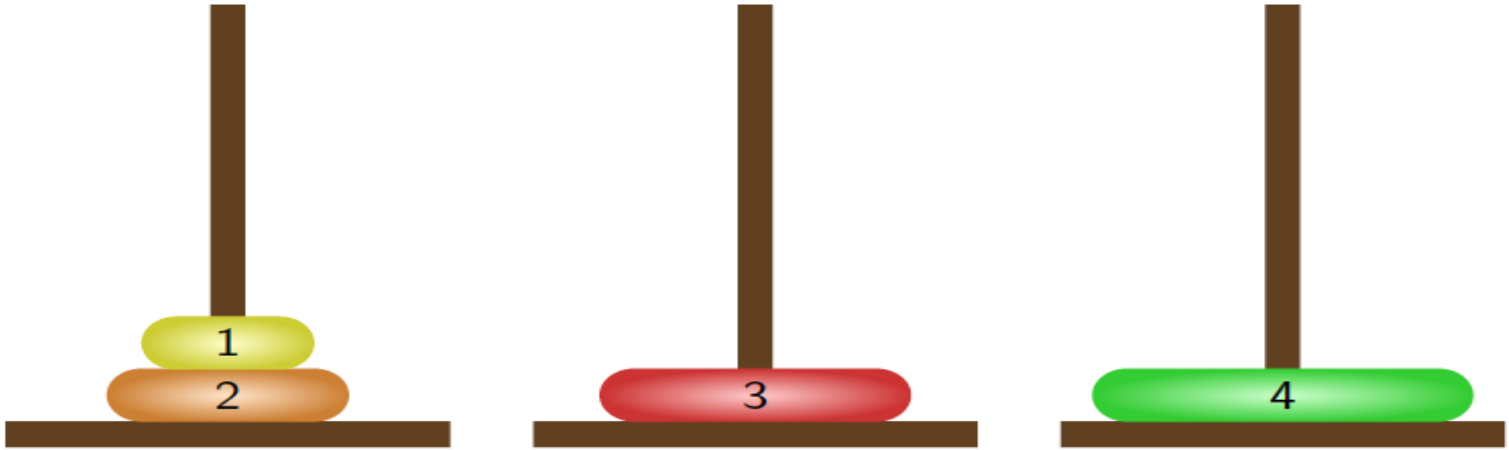
Moved disc from pole 1 to pole 3.



Moved disc from pole 2 to pole 3.

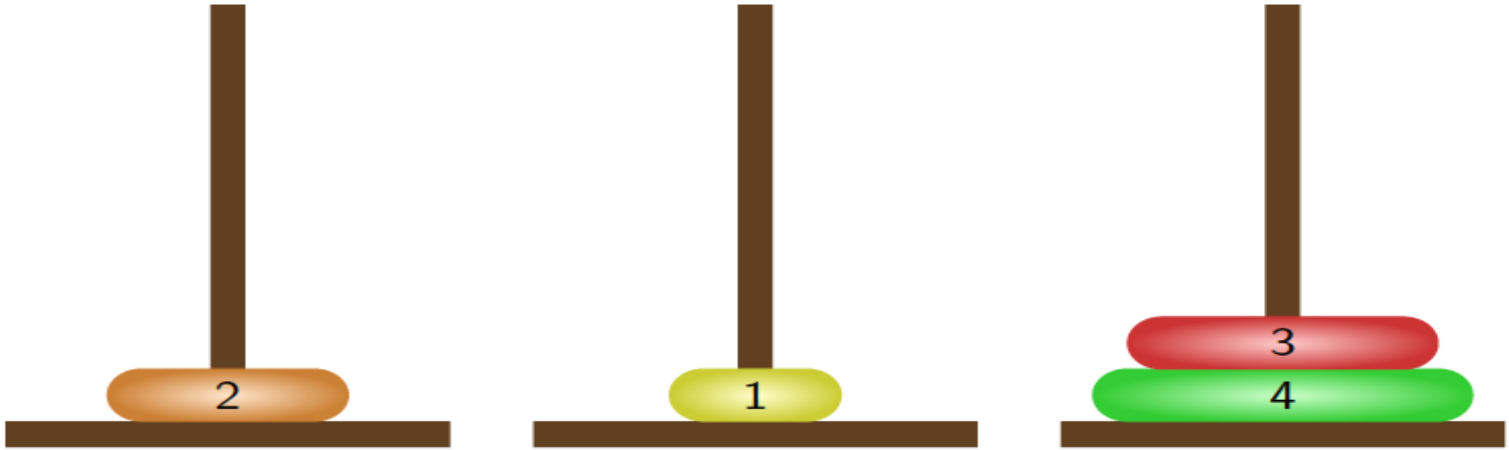


Moved disc from pole 2 to pole 1.

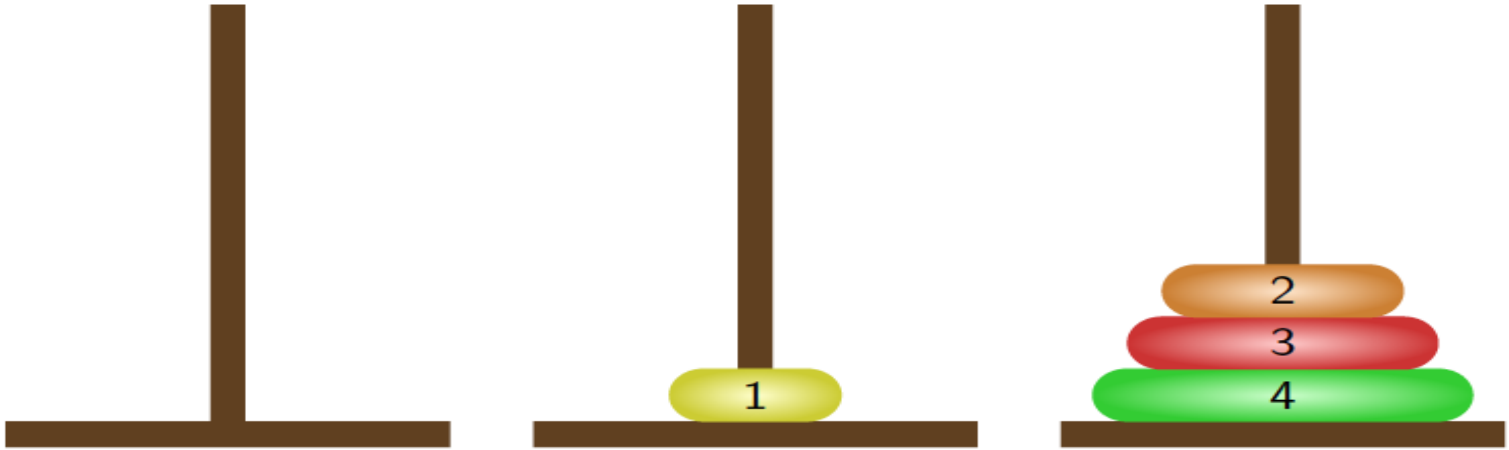


Moved disc from pole 3 to pole 1.

Moved disc from pole 2 to pole 3.



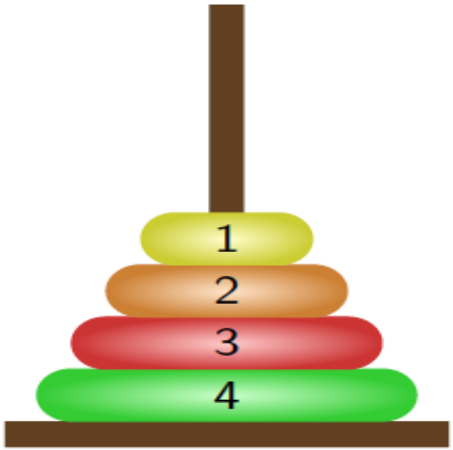
Moved disc from pole 1 to pole 2.

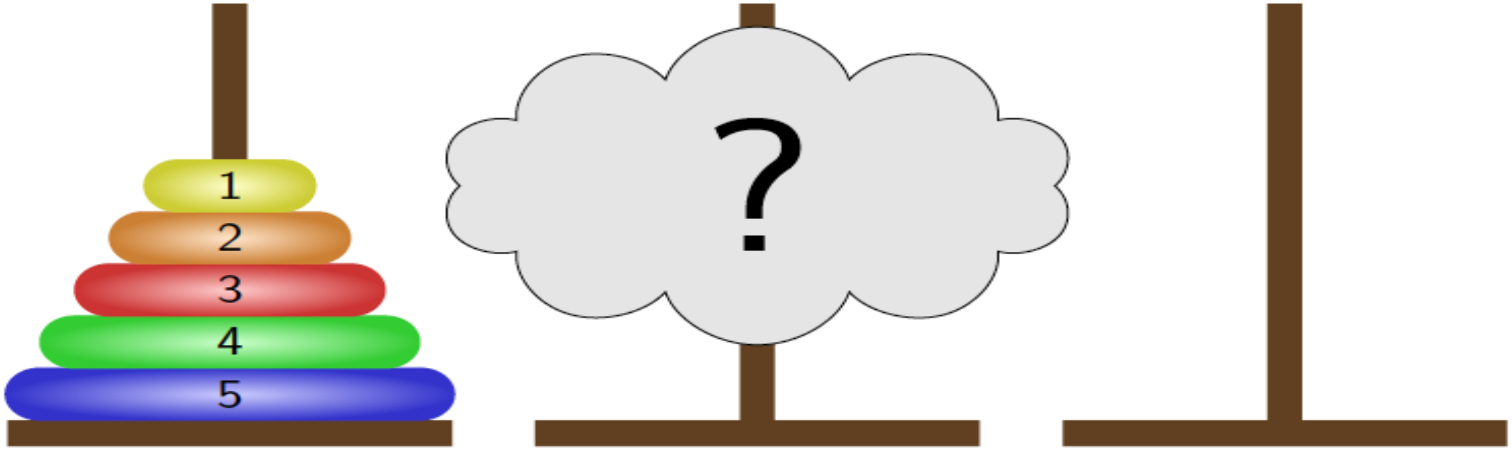


Moved disc from pole 1 to pole 3.

Moved disc from pole 2 to pole 3.


ok


1
2
3
4

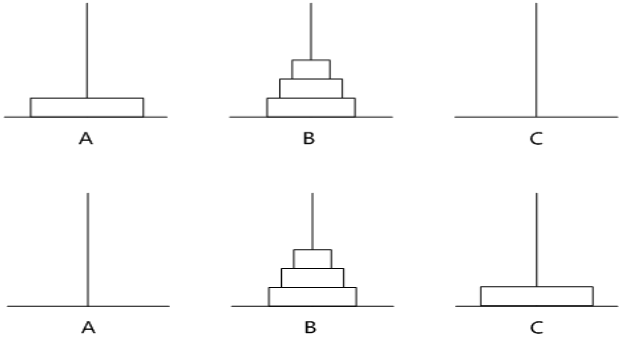

1
2
3
4
5
?

递归解决方案

- 将前 n-1 个盘子，通过 C，从 A 移动到 B
- 从 A 到 C 移动第 n 个盘子
- 将前 n-1 个盘子，通过 A，从 B 移动到 C

递归 - 汉诺塔

- 定义函数hanoi(n, A, B, C)表示把A上的n个盘子移动到C上，其中可以用到B



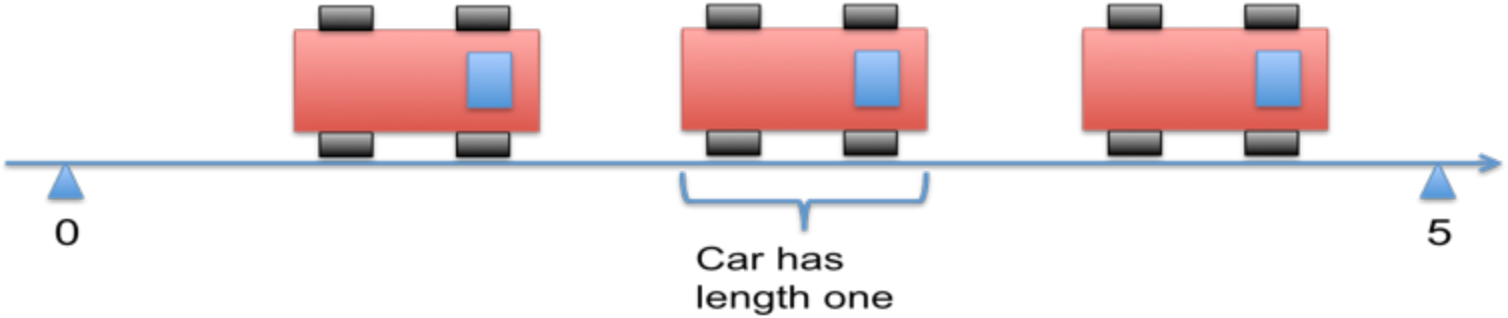

```
def hanoi(n, A, B, C):
    if n == 1:
        print "Move disk ", n, " from ", A, " to ", C
    else:
        hanoi (n-1, A, C, B)
        print "Move disk ", n, " from ", A, " to ", C
        hanoi (n-1, B, A, C)

n = int(raw_input("请输入一个整数: "))
hanoi(n, '左', '中','右')
```

路边停车

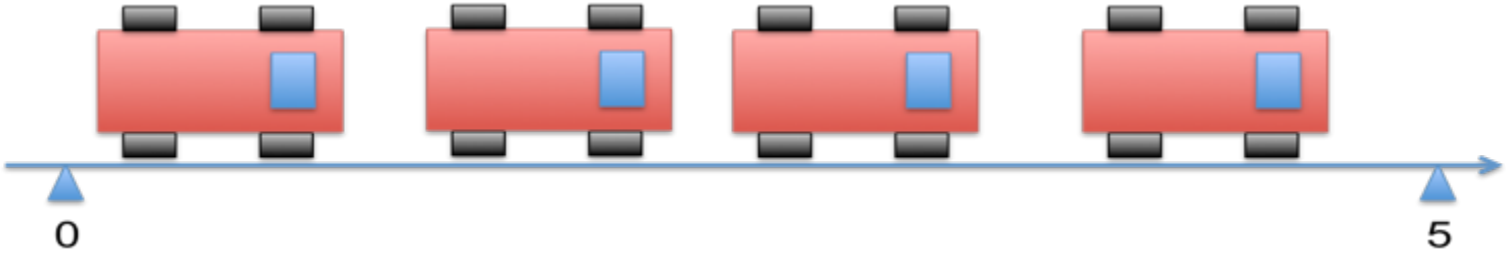
随机停车

- 长度为5的马路，平均能停多少量长度为1的汽车？

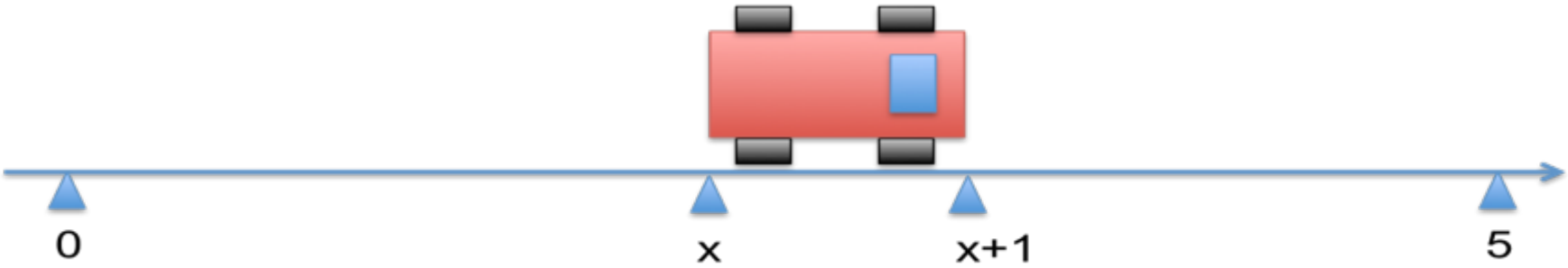


随机停车

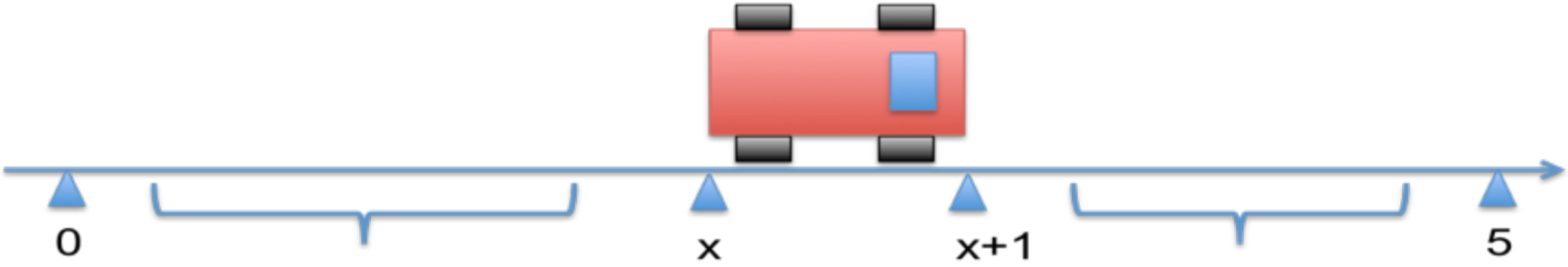
- 长度为5的马路，平均能停多少量长度为1的汽车？



随机停车



随机停车



Place cars randomly in these ranges

随机停车


```
def park_randomly(low, high):
    if high - low < 1:
        return 0;
    else:
        x = random.uniform(low, high - 1)
        return 1 + park_randomly(low, x) \
                 + park_randomly(x + 1, high)
```

随机停车

- 当宽度w足够大时，平均停车约 0.7475972w 辆
- 常数 0.7475972 被称作 Renyi 停车常数
- 该算法巧妙的运用了递归思想，将大问题分解为更小的、独立的相似问题，然后分别加以解决
- 许多问题能够以此方式解决

递归的时间开销


```
def fib_loop(n):
    if n == 1 or  n == 2:
        return 1
    else:
        i = 2
        f1 = 1
        f2 = 1
        while(i < n):
            f3 = f1 + f2
            f1 = f2
            f2 = f3
            i = i + 1
        return f3
```

```
def fib_recursive(n):
    if n == 1 or n == 2:
        return 1
    else:
        return fib_recursive(n-1) + fib_recursive(n-2)
```

fib_loop(100)	不到0.01秒
fib_recursive(100)	超过1小时

时间都去哪了？

递归的时间开销

fib(4)

```
def fib(4):
    if 4 == 1 or 4 == 2:
        return 1
    else:
        return fib(4-1) + fib(4-2)
```

```
def fib(2):
    if 2 == 1 or  2 == 2:
        return 1
    else:
        return fib(2-1) + fib(2-2)
```

```
def fib(3):
    if 3 == 1 or  3 == 2:
        return 1
    else:
        return fib(3-1) + fib(3-2)
```

```
def fib(1):
    if 1 == 1 or  1 == 2:
        return 1
    else:
        return fib(1-1) + fib(1-2)
```

递归的优劣分析

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

- **优势（strength）**
  - 它能使一个蕴含递归关系且结构复杂的程序简洁精炼，增加可读性
  - 特别是在难于找到从边界到解的全过程的情况下，如果把问题推进一步,其结果仍维持原问题的关系
- **劣势（weakness）**
  - 嵌套层次深，函数调用开销大
  - 重复计算