19. Recursividade

DIM0321

2015.1

Outline

Matemática

Definir um problem **recursivamente** = resolver o problema a partir da solução de uma instância menor do problema.

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ n * (n-1)! & \text{senão} \end{cases}$$

Em programação

Uma forma de especificar uma função (sub-rotina) que é definida através de sua própria definição.

$$fib(n) = \begin{cases} x & x \leq 1 \\ fib(n-1) + fib(n-2) & n \geq 1 \end{cases}$$

A recursão pode ser usada também como procedimento ou mecanismo de alteração de dados.

Fatorial

```c
#include <stdio.h>
#include <assert.h>
unsigned fact(unsigned n)
{
     if (n == 0) return 1;
     return n * fact(n - 1);
}


int main(void)
{
     unsigned n;
     scanf("%u", &n);
     printf("fact(%u) = %u\n", n, fact(n));
     return 0;
}
```


Sequência de Fibonacci

```c
#include <stdio.h>
#include <assert.h>
unsigned fib(unsigned n)
{
     if (n == 0) return 0;
     else if (n == 1) return 1;
     else return fib(n - 1) + fib(n - 2);
}


int main(void)
{
     unsigned n;
     scanf("%u", &n);
     printf("fib(%u) = %u\n", n, fib(n));
     return 0;
}
```

Ordenação

Problema

Dado um conjunto $A = \{a_1, a_2, a_3, \cdots, a_n\}$, ordenar A tal que para $1 \leq i < n, a_i < a_{i+1}$

Ideia

- Se $\{a_2, \cdots, a_n\}$ estiver ordenado basta inserir a_1 na sua posição ordenada em $\{a_2, \cdots, a_n\}$
- Se $\{a_2, \cdots, a_n\}$ não estiver ordenado, aplica o mesmo princípio (mas o problema está menor)
- Um conjunto com apenas 1 elemento está obrigatoriamente ordenado

Parada

- A execução de uma função recursiva provoca a chamada a própria função
- Quando parar?

A base!

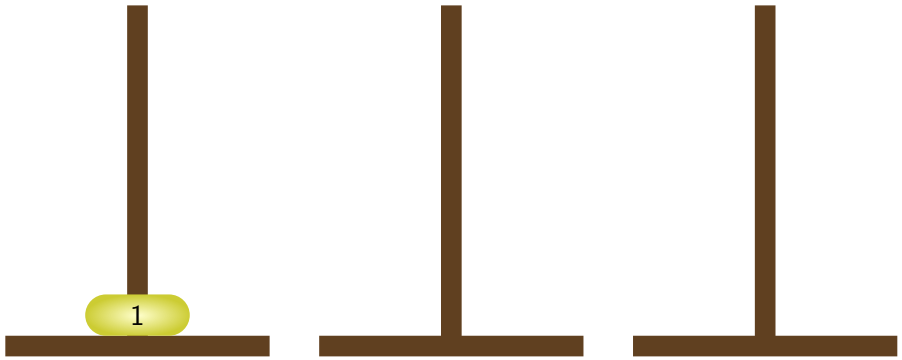
Definir e não esquecer o **caso base**

1. O corpo da função deve incluir ao menos um caso onde o valor de retorno não depende de uma chamada recursiva. Este é o **caso base**.
2. Se o cálculo de f(a) chama recursivamente f(b), então b tem que estar mais próximo do caso-base do que a

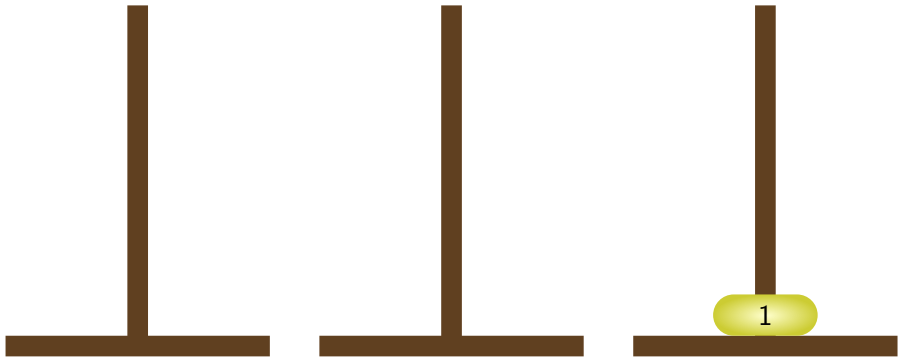
Recursão mútua

```c
#include <stdbool.h>
#include <assert.h>
#include <stdio.h>

bool even(unsigned n);
bool odd(unsigned n);

bool even(unsigned n)
{
     printf("even %d\n", n);
     if (n == 0) return true;
     else return odd(n - 1);
}


bool odd(unsigned n)
{
     printf("odd %d\n", n);
     if (n == 0) return false;
     else return even(n - 1);
}

int main(void)
{
     int n;
     scanf("%d", &n);
     if (n < 0) {
          n = -n;
     }

```

Torre de Hanoi – 1 disco



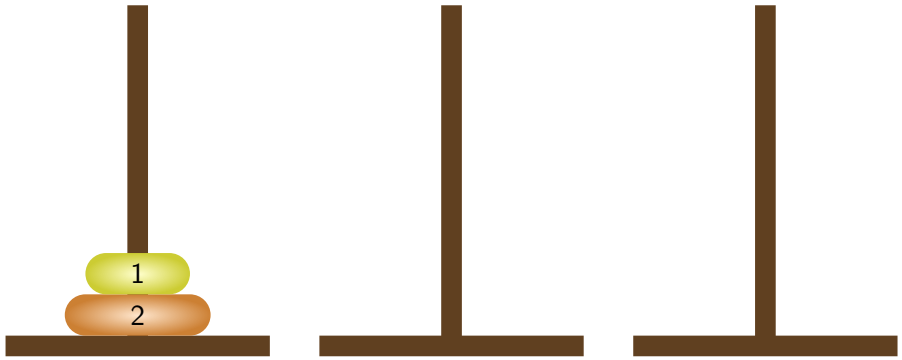
Torre de Hanoi – 1 disco



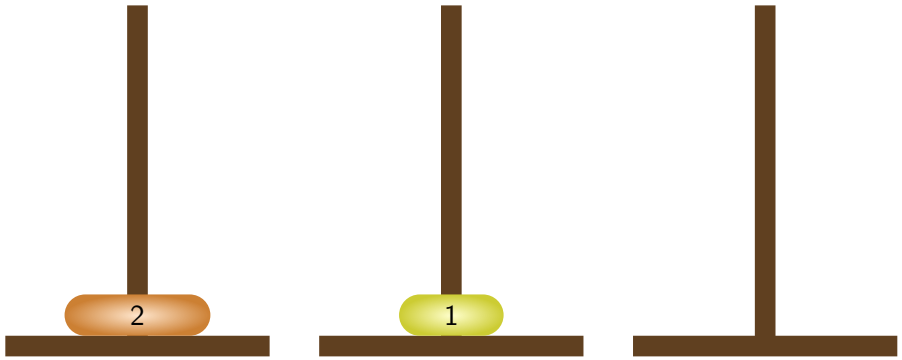
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 1 disco

Torre de Hanoi – 2 discos

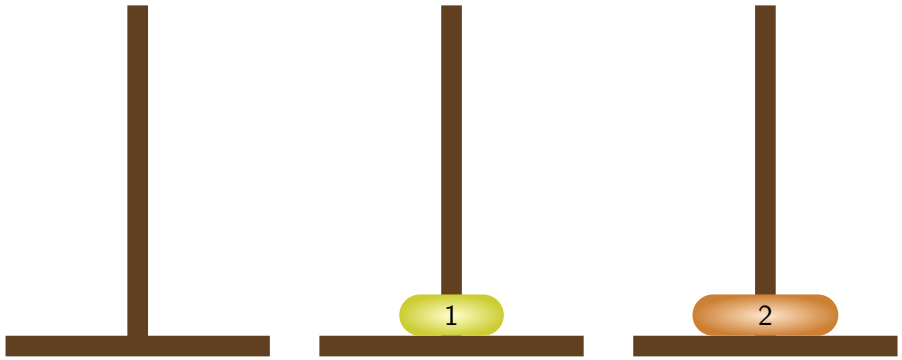


Torre de Hanoi – 2 discos



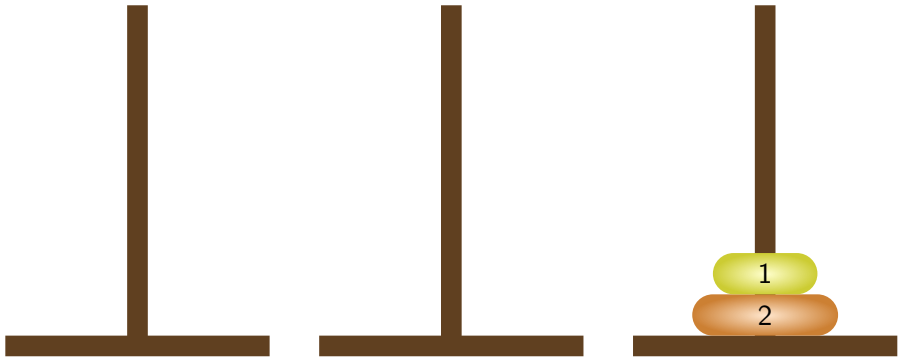
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 2 discos



Mudei disco do pino 1 ao pino 3.

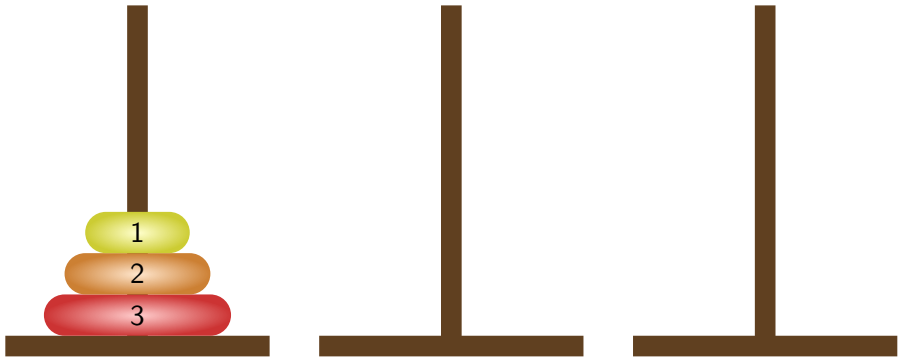
Torre de Hanoi – 2 discos



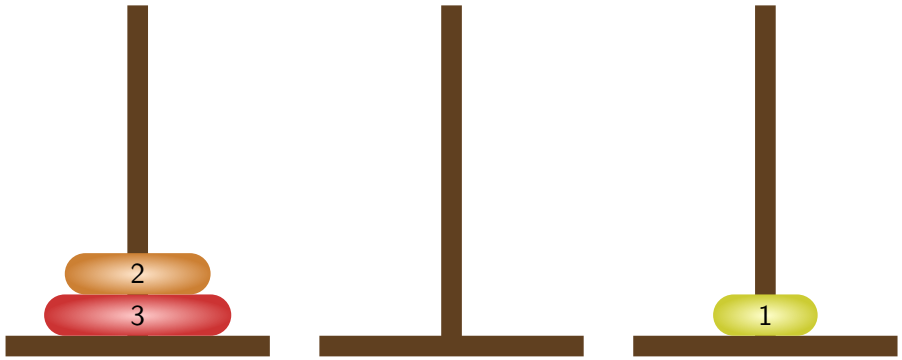
Mudei disco do pino 2 ao pino 3.

Torre de Hanoi – 2 discos

Torre de Hanoi – 3 discos

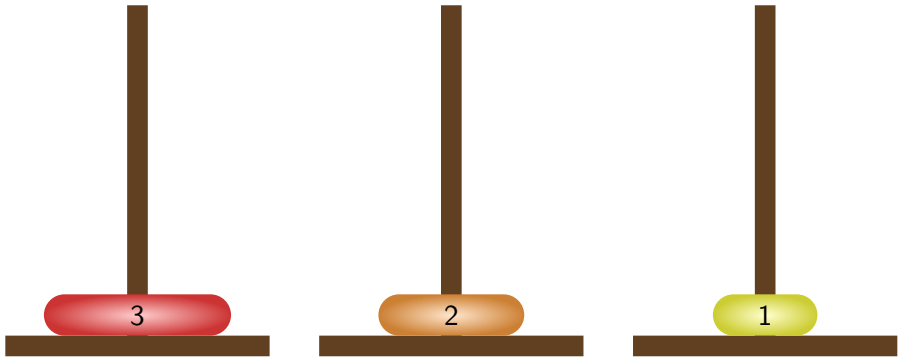


Torre de Hanoi – 3 discos



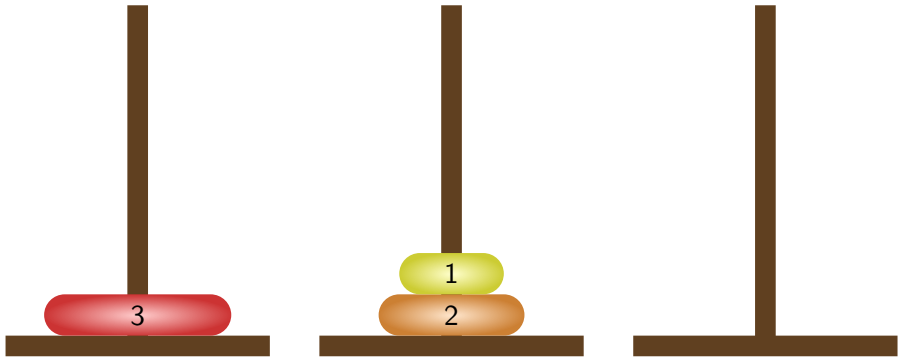
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 3 discos



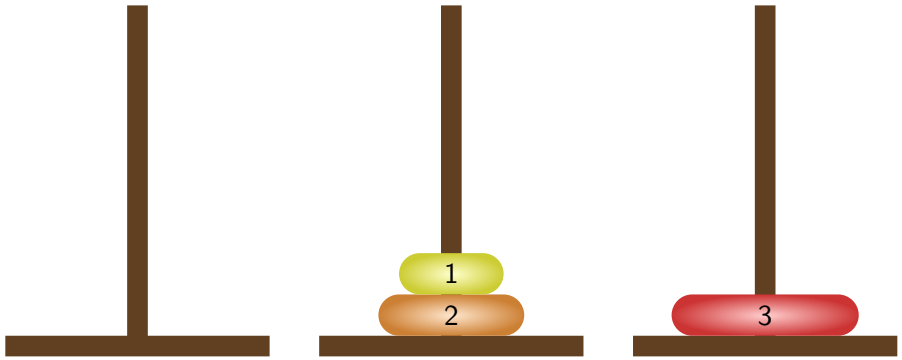
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 3 discos



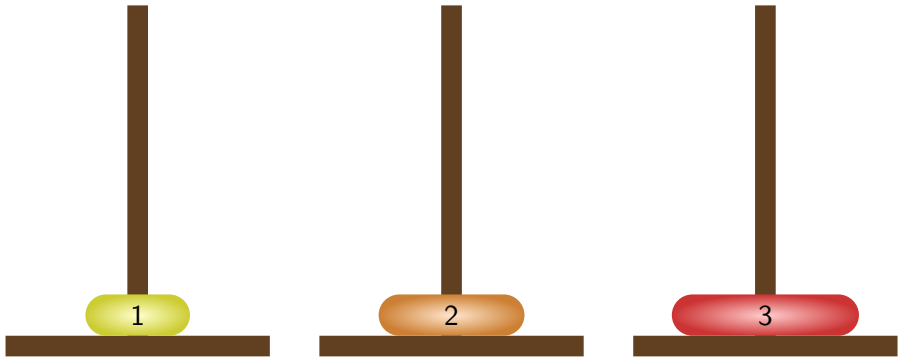
Mudei disco do pino 3 ao pino 2.

Torre de Hanoi – 3 discos



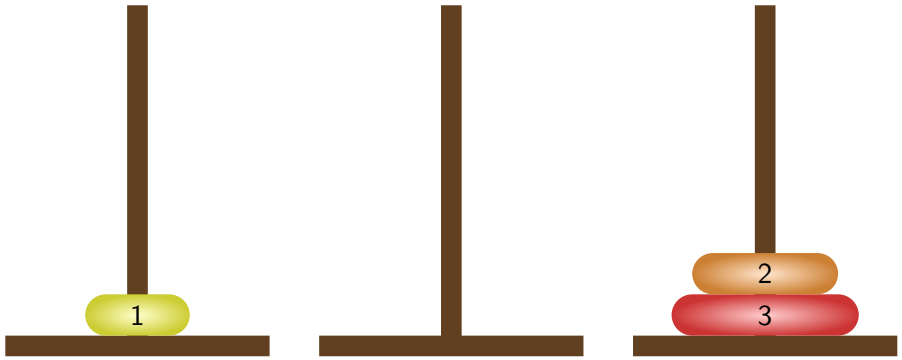
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 3 discos



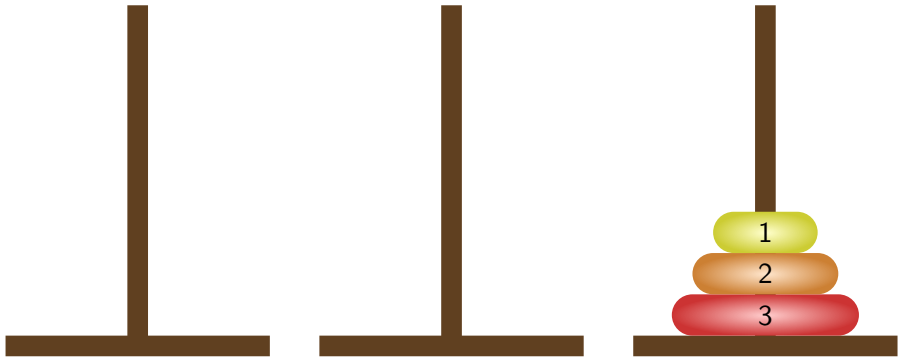
Mudei disco do pino 2 ao pino 1.

Torre de Hanoi – 3 discos



Mudei disco do pino 2 ao pino 3.

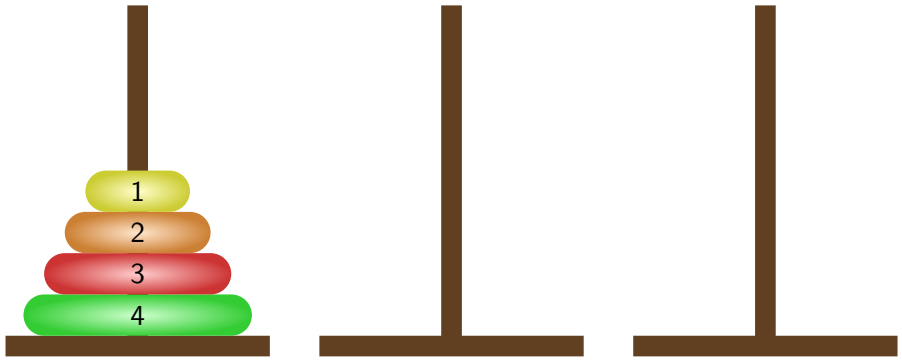
Torre de Hanoi – 3 discos



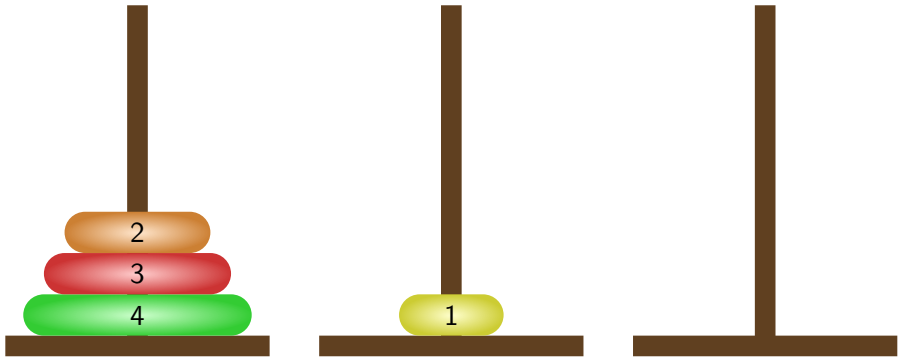
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 3 discos

Torre de Hanoi – 4 discos

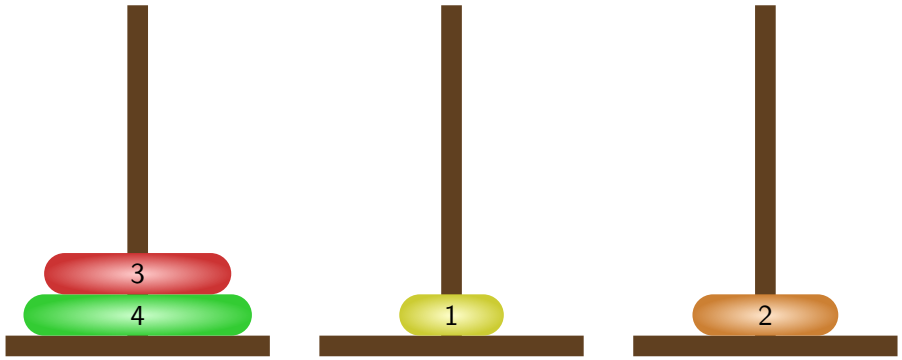


Torre de Hanoi – 4 discos



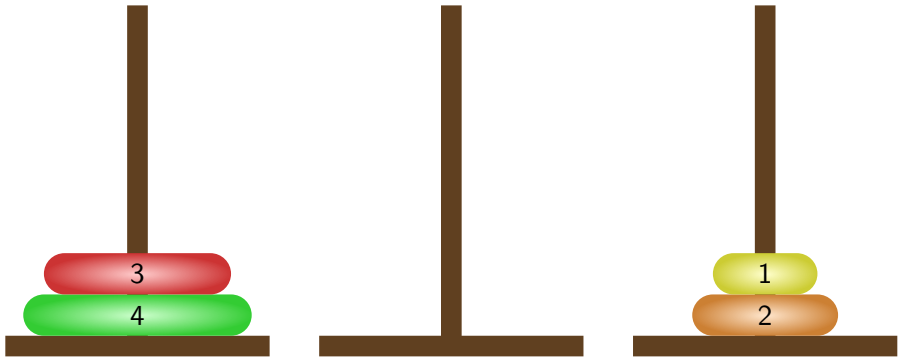
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 4 discos



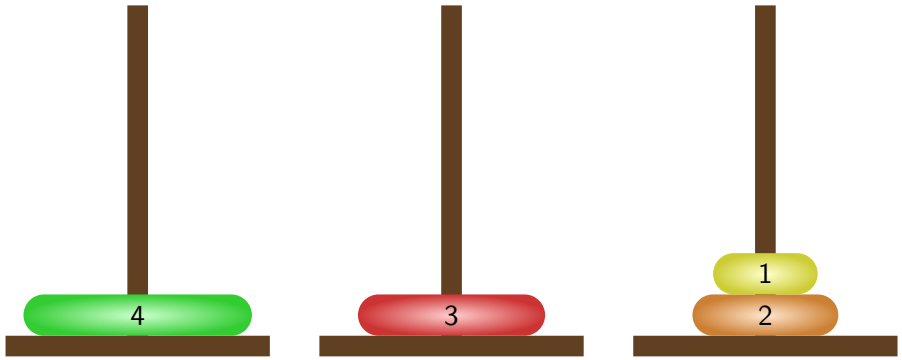
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 4 discos



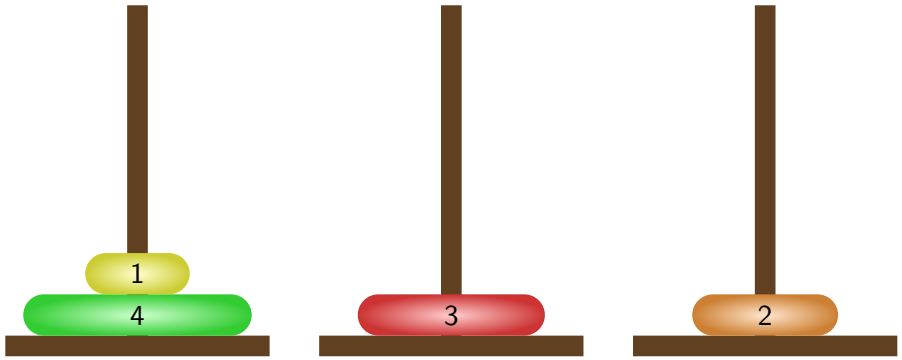
Mudei disco do pino 2 ao pino 3.

Torre de Hanoi – 4 discos



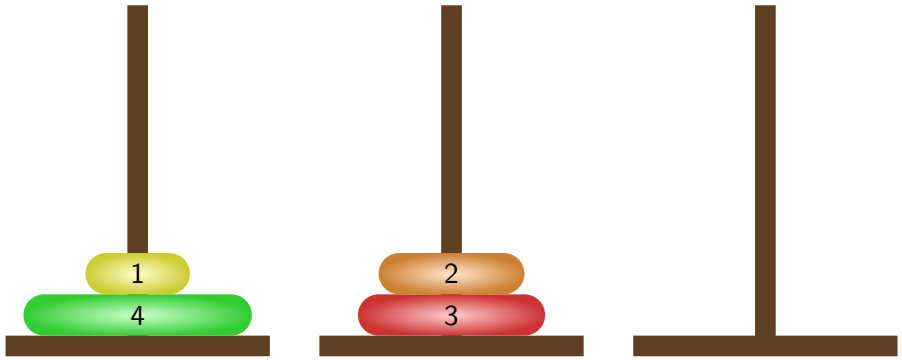
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 4 discos



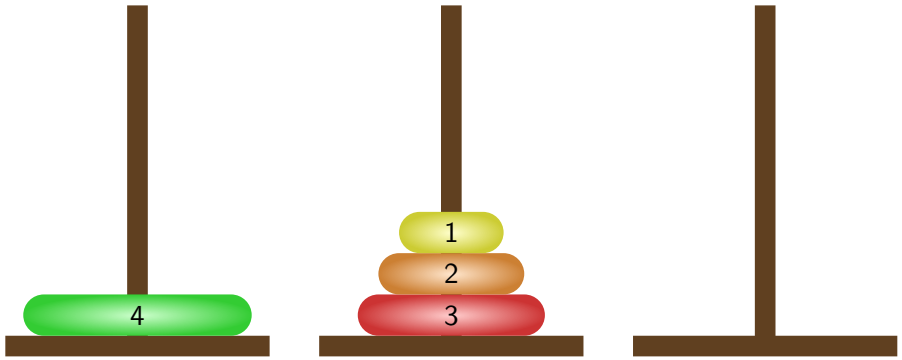
Mudei disco do pino 3 ao pino 1.

Torre de Hanoi – 4 discos



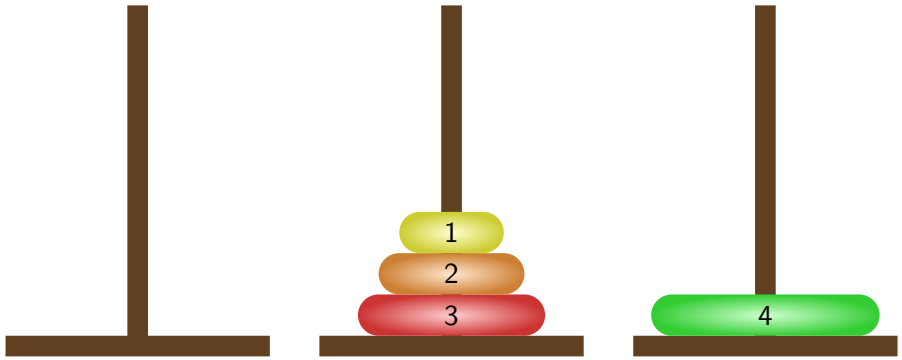
Mudei disco do pino 3 ao pino 2.

Torre de Hanoi – 4 discos



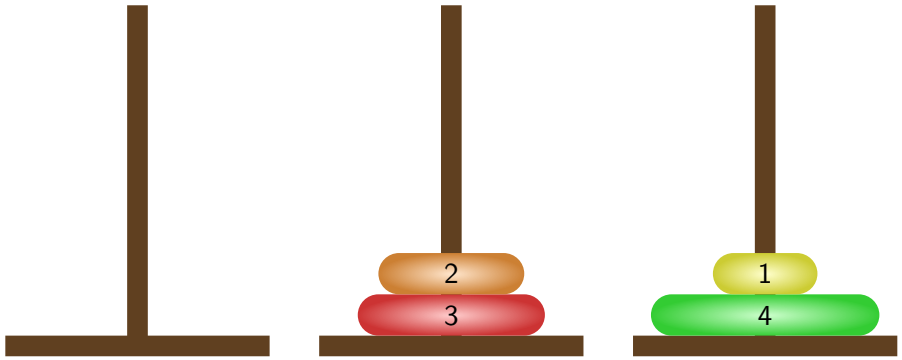
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 4 discos



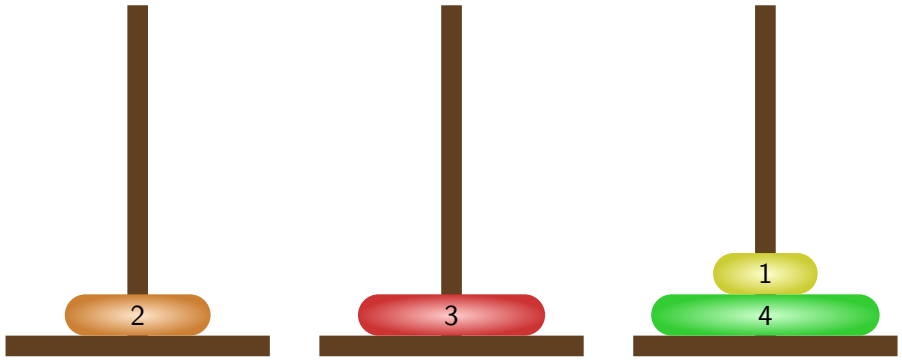
Mudei disco do pino 1 ao pino 3.

Torre de Hanoi – 4 discos



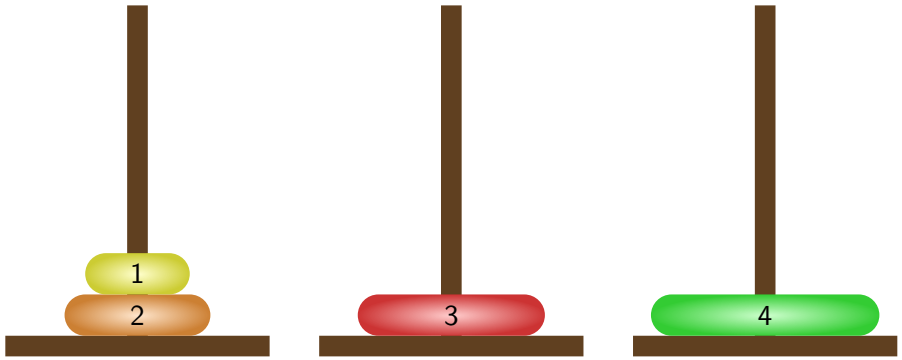
Mudei disco do pino 2 ao pino 3.

Torre de Hanoi – 4 discos



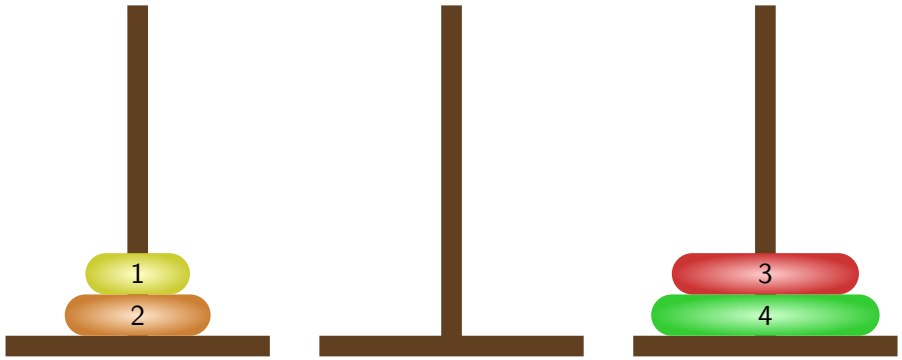
Mudei disco do pino 2 ao pino 1.

Torre de Hanoi – 4 discos



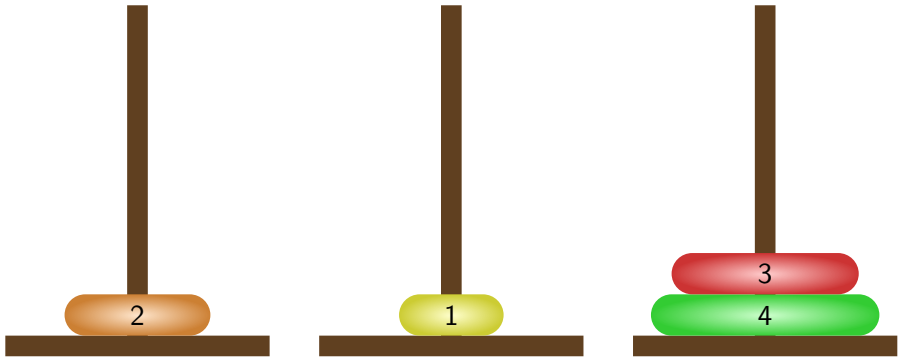
Mudei disco do pino 3 ao pino 1.

Torre de Hanoi – 4 discos



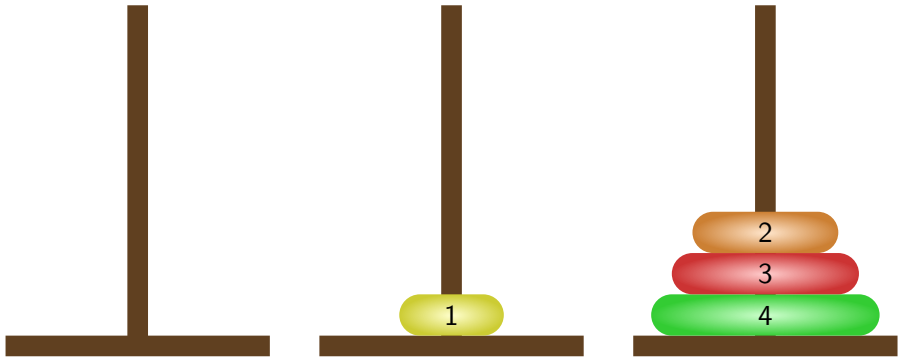
Mudei disco do pino 2 ao pino 3.

Torre de Hanoi – 4 discos



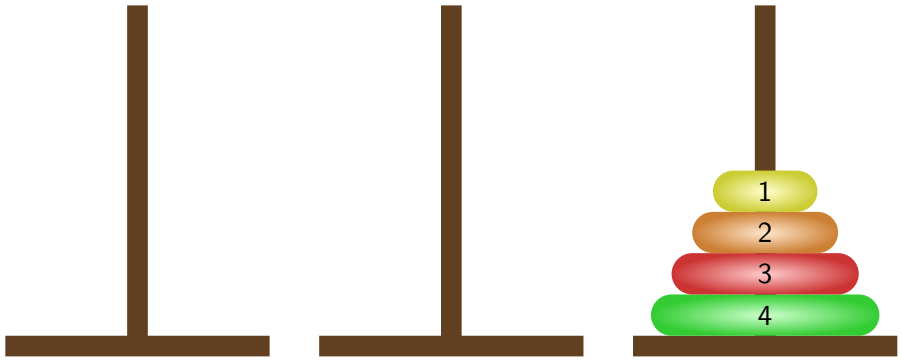
Mudei disco do pino 1 ao pino 2.

Torre de Hanoi – 4 discos



Mudei disco do pino 1 ao pino 3.

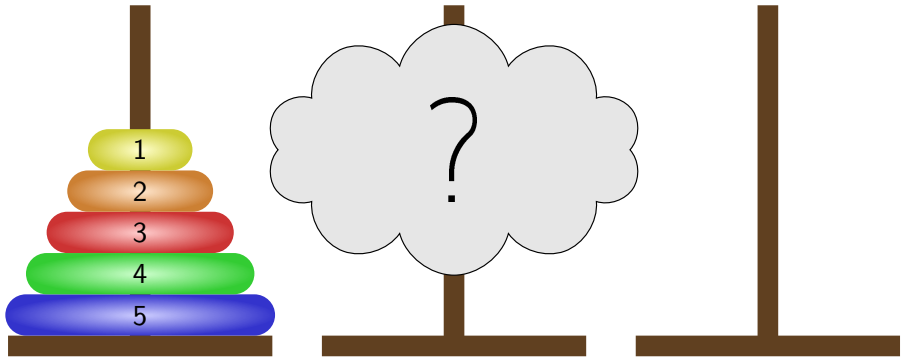
Torre de Hanoi – 4 discos



Mudei disco do pino 2 ao pino 3.

Torre de Hanoi – 4 discos

Tower of Hanoi – 5 Disc



História

- Proposto pelo matemático Édouard Lucas em 1883.

Assunto

- Considerando uma torre de n discos de circunferência crescente, inicialmente empilhados em ordem crescente de circunferência sobre uma entre três varas.
- O objetivo é transferir a torre inteira para outra vara, movendo apenas um disco de cada vez, e nunca colocando um disco maior sobre um menor.

Resolução programática

Definir uma sub-rotina que imprime as instruções para resolver o quebra-cabeça:

- Ter como parâmetro o número de discos
- Imprimir uma lista de movimentos: "mover X para Y" onde $1 \leq X, Y \leq 3$.

Exemplo, para 2 discos:

- mover 1 para 2
- mover 1 para 3
- mover 2 para 3

Solução recursiva

```c
#include <stdio.h>
#include <assert.h>

int nmoves; // Init = 0;

void move_disc(char orig, char dest)
{
     nmoves++;
     printf("%d: moving %c -> %c\n", nmoves, orig, dest);
}

void move(int n, char orig, char dest, char aux)
{
     if (n == 1) move_disc(orig, dest);
     else {
          move(n - 1, orig, aux, dest);
          move_disc(orig, dest);
          move(n - 1, aux, dest, orig);
     }

}

int main(void)
{
     int n;
     printf("Tower of Hanoi\n");
     printf("How many discs ( > 0)? ");
     scanf("%d", &n);
     assert(n > 0);
     nmoves = 0;
```


Soluçao imperativa

- Para mais tarde !

Referências

Precisões

[?] 5.1 – 5.4

[?] 10

Perguntas ?

http://dimap.ufrn.br/~richard/dim0321