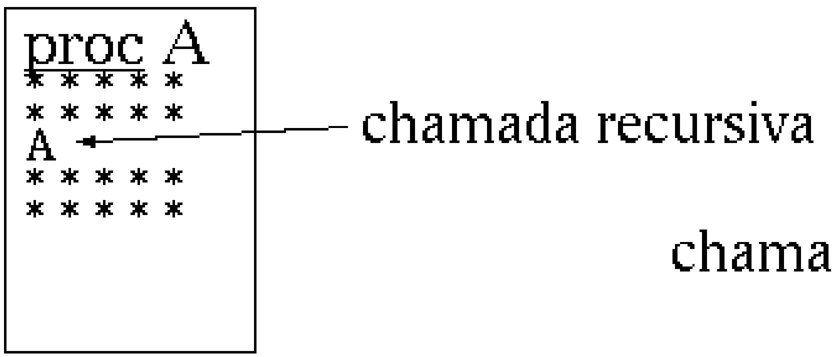
Aula 2: Recursividade

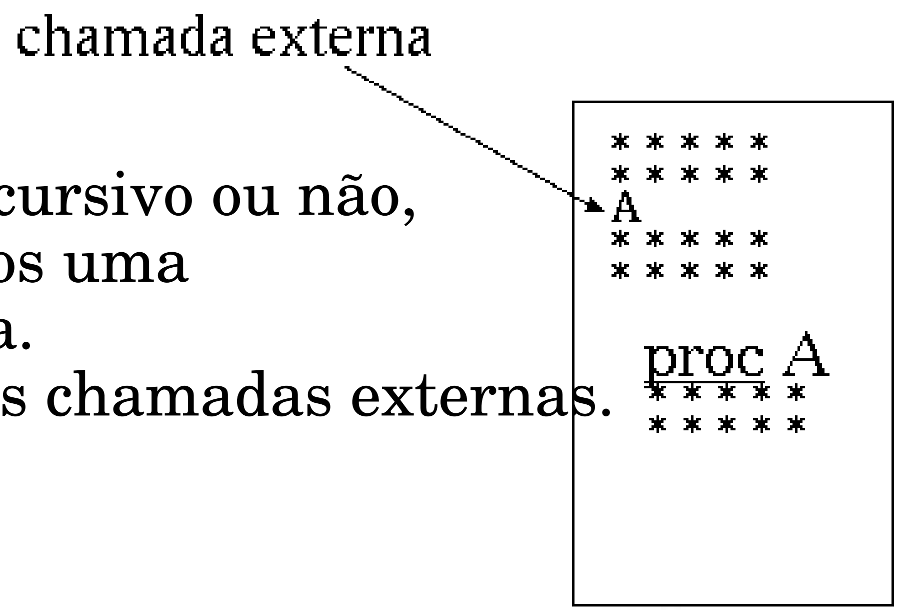
- O conceito de recursividade
- Algoritmo do fatorial recursivo
- Problema da Torre de Hanói

Recursividade

- Procedimentos ou funções recursivas correspondem a aqueles que contêm chamadas a si mesmo.

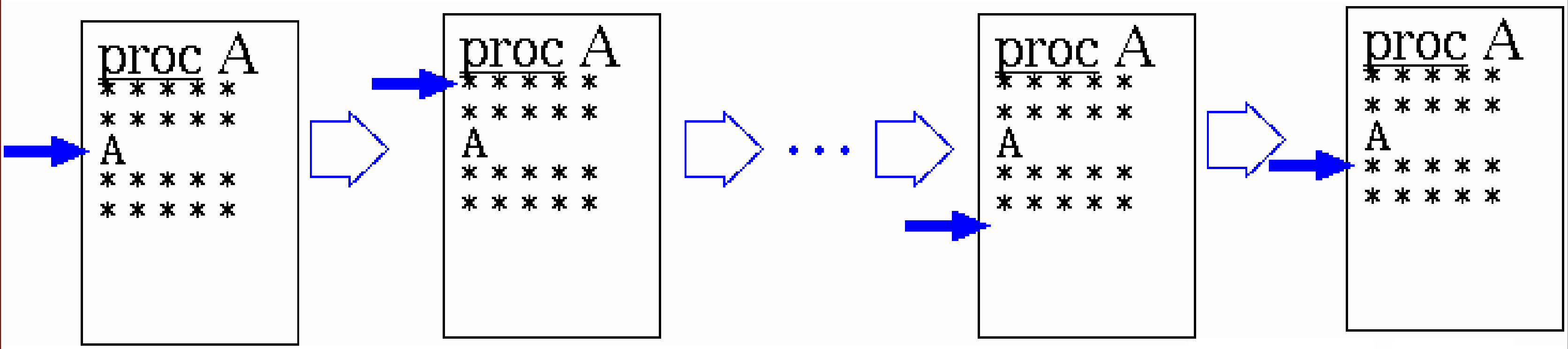


- Todo procedimento, recursivo ou não, deve possuir pelo menos uma chamada não recursiva. Essas são denominadas chamadas externas.



Recursividade

- Se ocorrer uma chamada recursiva:
  - o algoritmo é interrompido e reiniciado no princípio do procedimento recursivo;
  - após o término da computação do procedimento correspondente à chamada recursiva, o algoritmo é interrompido e reiniciado no ponto seguinte ao da ocorrência da chamada recursiva.



Recursão x Iteração

- Recursão versus Iteração
  - De um modo geral, a cada processo recursivo corresponde um outro iterativo
- Vantagens da recursão

  Geralmente,
  - algoritmos mais concisos;
  - prova de correção mais simples.
- Vantagens da iteração
  - Geralmente, mais eficiente.

Cálculo de Fatorial

- Primeiro exemplo: cálculo de fatorial

  $n! = n.(n-1)\ldots 1$
  $5! = 5 \;.\; 4 \;.\; 3 \;.\; 2 \;.\; 1 = 120$

  Convenção: $0! = 1$

- Escrever um algoritmo para calcular n!

- Idéia: $n! = n.(n-1)!$

- O algoritmo abaixo usa a função recursiva fat

  Algoritmo: fatorial(recursivo)

```
função fat(i)
   se i ≤ 1 então 1 senão i.fat(i-1)
```

  chamada externa: fat(n)

Recursividade Passo a Passo

- Como seguir os passos de um algoritmo recursivo?
- Exemplo: fatorial recursivo, com n = 5

fat(5)

Resposta: 120

Calcular **Voltar**

Cálculo Iterativo do Fatorial

- Cálculo iterativo de fatorial
  - a variável `fat` representa um vetor, e não mais uma função
  - o elemento `fat[i]` contém o valor `i`!, 0 <= `i` <= `n`

Algoritmo 1.3: fatorial (iterativo)

```
fat[0]:=1
para j=1,..., n faça
    fat[j] := j . fat[j-1]
```

Iteração Passo a Passo

- Como seguir os passos de um algoritmo iterativo?

fat[0]	fat[1]	fat[2]	fat[3]	fat[4]	fat[5]
1	1	2	6	24	120

Resposta: 120

Calcular

Voltar

Exercício

- Calcular o valor n!, através de um algoritmo iterativo, de modo que prescinda do armazenamento de qualquer vetor.

Tempo: 12 minutos.

Exercício (solução)

- Solução:

```
algoritmo fatorial

ant := fat := 1

se n > 1 então
    para i = 2, ..., n faça
            fat := ant . i
            ant := fat
```

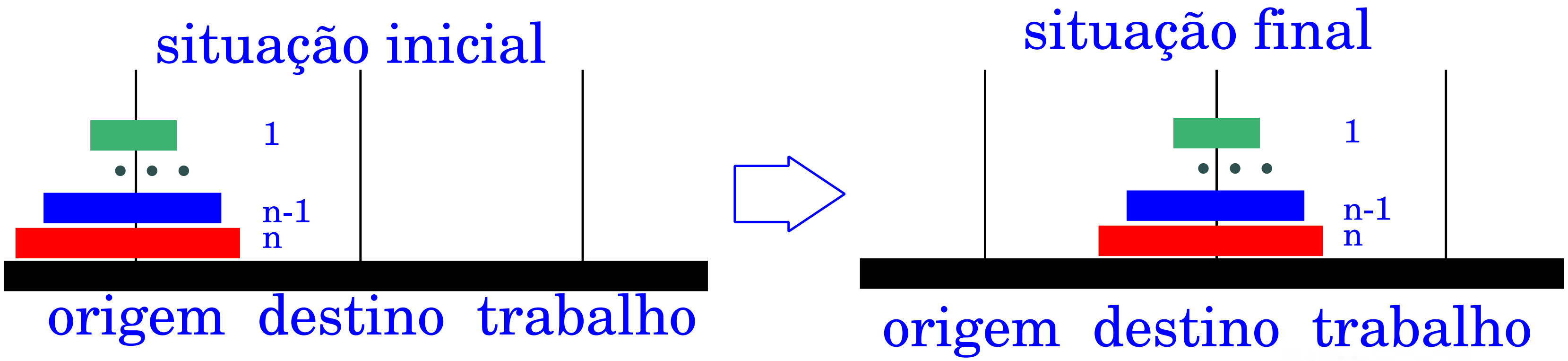
- Ao final do algoritmo, a variável fat contém o valor de n!

Problema da Torre de Hanói

O problema consiste de 3 pinos, A, B e C, denominados origem, destino e trabalho, respectivamente. Além dos pinos, há n discos de diâmetros diferentes. Inicialmente, todos os discos se encontram empilhados no pino origem, em ordem crescente de tamanho, de cima para baixo. O objetivo é empilhar todos os discos no pino destino, atendendo às seguintes restrições:

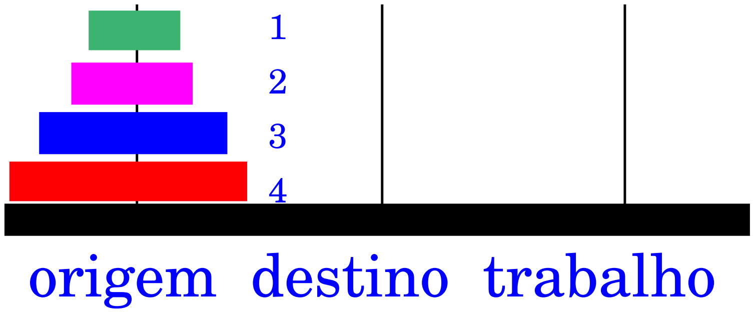
(i) apenas um disco pode ser movido de cada vez;

(ii) qualquer disco não pode jamais ser colocado sobre outro de menor tamanho.

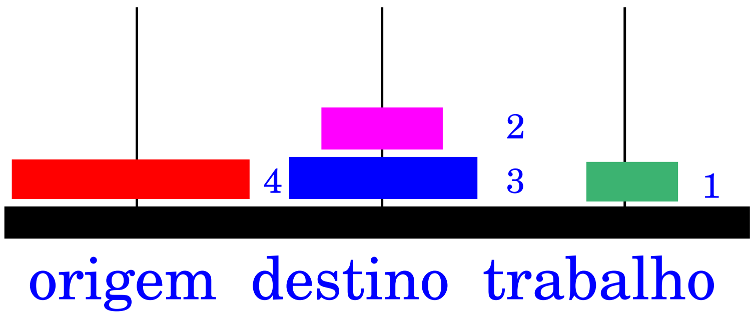


Torre de Hanói

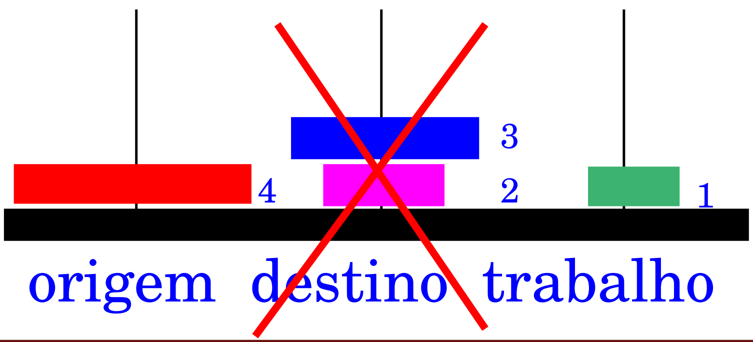
Problema com n = 4 discos



situação inicial



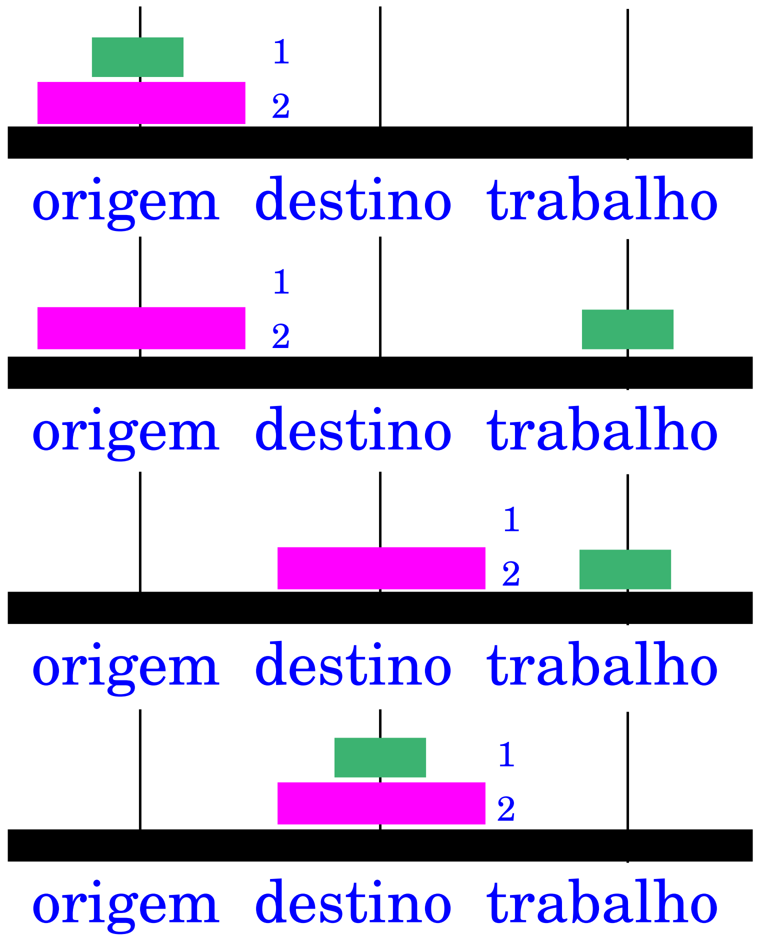
possível situação intermediária



situação proibida

Torre de Hanói

- Problema com n = 2 discos



situação inicial

movimento 1

movimento 2

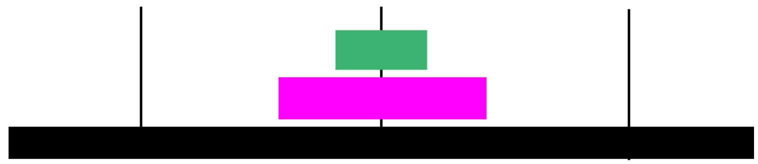
movimento 3

total: 3 movimentos

Torre de Hanói

- Problema com n = 2 discos

Mover Voltar



Total: 3 movimentos

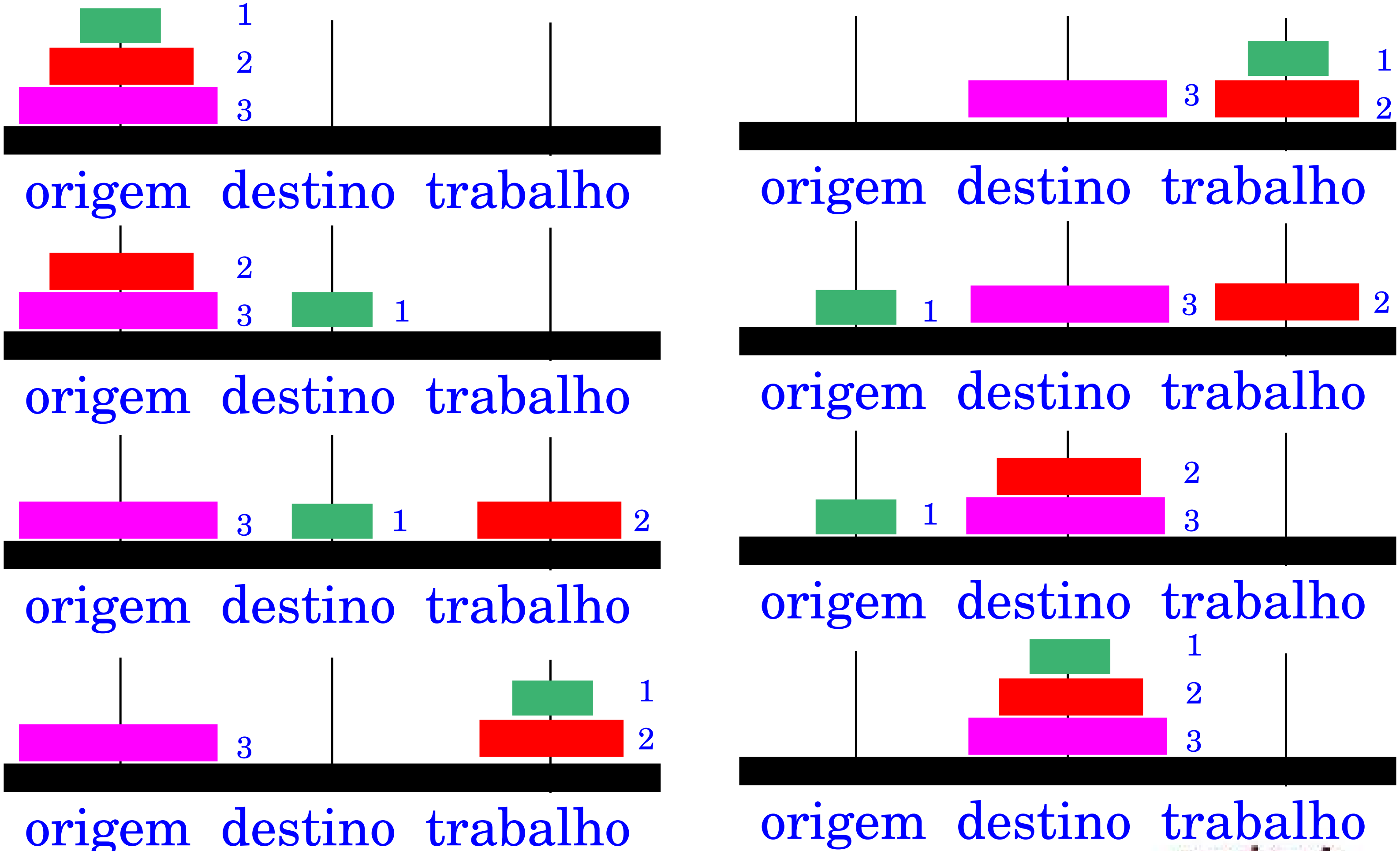
- Exercício: resolver o problema da Torre de Hanói

com n = 3 discos

Tempo: 12 minutos

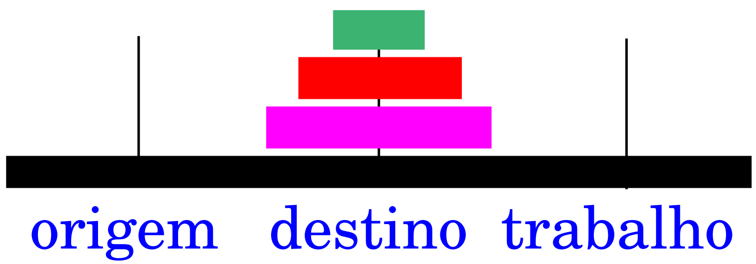
Torre de Hanói

Solução do problema da Torre de Hanói: 3 discos, 7 movimentos



Torre de Hanói

Solução do problema da Torre de Hanói: 3 discos



Total: 7 movimentos

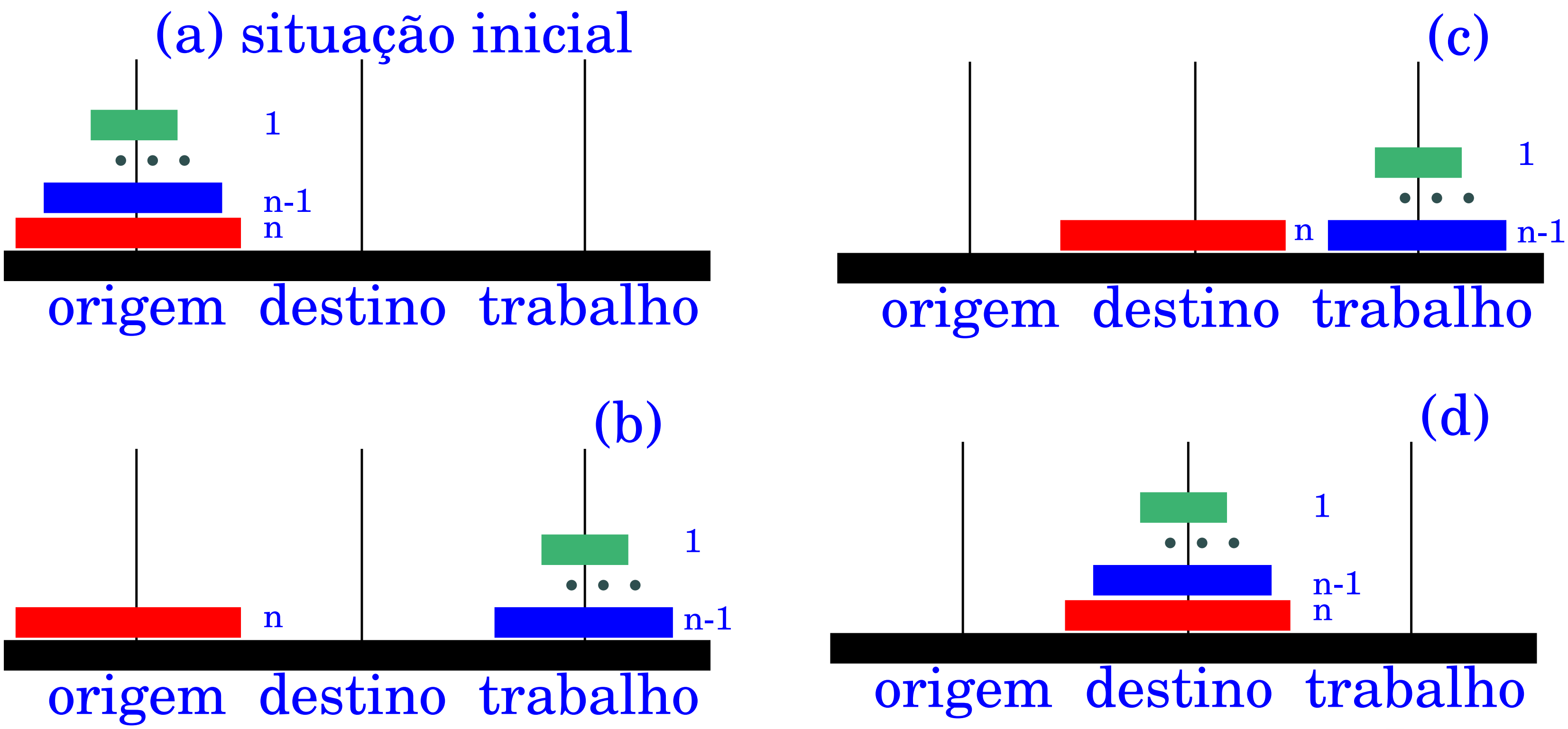
Mover Voltar

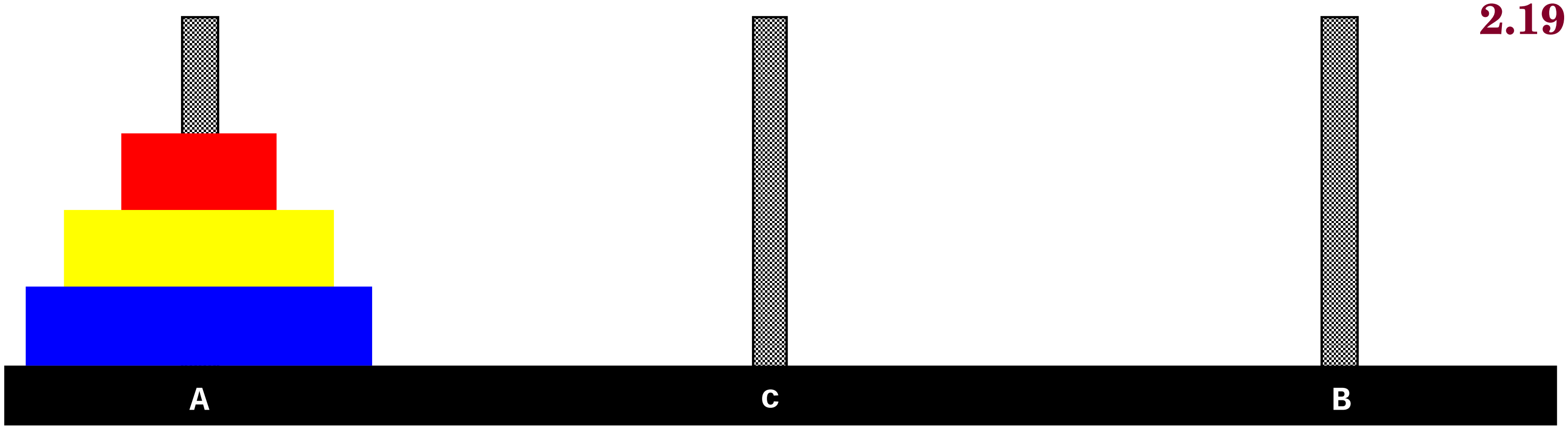
Torre de Hanói

- Solução geral do problema da Torre de Hanói
- Suponha que se saiba como resolver o problema até `n-1` discos, `n` > 1, de forma recursiva.
- A extensão para n discos pode ser obtida pela realização dos seguintes passos:
  - Resolver o problema da Torre de Hanói com os `n-1` discos do topo do pino A, usando A como origem, B como trabalho e C como destino;
  - Mover o n-ésimo pino (maior de todos) de A para B;
  - Resolver o problema da Torre de Hanói com os `n-1` discos do topo do pino C, usando A como trabalho, B como destino e C como origem.

Torre de Hanói

Esquema da solução do problema da Torre de Hanói





Objetivo: mover 3 peças de A para B

n = 3; origem: A; destino: B; trabalho: C

```
mover n-1 peças de ORIGEM para TRABALHO
mover 1 peça de ORIGEM para DESTINO
mover n-1 peças de TRABALHO para DESTINO
```

Reiniciar

Animar

Torre de Hanói

Algoritmo: Torre de Hanói

```
proc hanoi( n, A, B, C )
        se n > 0 então
             hanoi( n-1, A, C, B )
             mover o disco do topo de A para B
             hanoi( n-1, C, B, A )
```

Parâmetros: 1°) número de discos
2°) pino de origem
3°) pino de destino
4°) pino de trabalho

Torre de Hanói

- Exercícios finais:
  - Mostrar que o algoritmo apresentado para o Problema da Torre de Hanói requer, exatamente, $2^n - 1$ movimentos de disco.
  - Reescrever o algoritmo do Problema da Torre de Hanói de forma que a recursividade pare no nível correspondente a $n = 1$, e não $n = 0$. Há alguma vantagem em realizar esta modificação? Qual?