Recursão

Algoritmos e Estrutura de Dados II

Prof. Kennedy Lopes

June 26, 2023

Referência

A atual aula tem como principal referência o conteúdo apresentado no livro **Data Structures and Algorithms in C++** do Adam Drozdek:

- Um procedimento ou função é dito **recursivo** se este invoca (realiza uma chamada) a si mesmo.
- Próprias definições podem ser consideradas recursivas:
  - Sistemas de arquivos;

- Conceito está conectado com a relação de recorrência e indução, exemplo:

$$a_n = 2 * a_n - 1; a_0 = 1 \rightarrow a_n = 2^n$$

- Um procedimento ou função é dito **recursivo** se este invoca (realiza uma chamada) a si mesmo.
- Próprias definições podem ser consideradas recursivas:
  - Sistemas de arquivos;
  - Números naturais;
- Conceito está conectado com a relação de recorrência e indução, exemplo:

$$a_n = 2 * a_n - 1; a_0 = 1 \rightarrow a_n = 2^n$$

- Um procedimento ou função é dito **recursivo** se este invoca (realiza uma chamada) a si mesmo.
- Próprias definições podem ser consideradas recursivas:
  - Sistemas de arquivos;
  - Números naturais;
  - Dilema ovo-galinha;

- Conceito está conectado com a relação de recorrência e indução, exemplo:

$$a_n = 2 * a_n - 1; a_0 = 1 \rightarrow a_n = 2^n$$

- Um procedimento ou função é dito **recursivo** se este invoca (realiza uma chamada) a si mesmo.
- Próprias definições podem ser consideradas recursivas:
  - Sistemas de arquivos;
  - Números naturais;
  - Dilema ovo-galinha;
  - Expressões matemáticas;
- Conceito está conectado com a relação de recorrência e indução, exemplo:

$$a_n = 2 * a_n - 1; a_0 = 1 \rightarrow a_n = 2^n$$

- Um procedimento ou função é dito **recursivo** se este invoca (realiza uma chamada) a si mesmo.
- Próprias definições podem ser consideradas recursivas:
  - Sistemas de arquivos;
  - Números naturais;
  - Dilema ovo-galinha;
  - Expressões matemáticas;
  - Lógica de programação.
- Conceito está conectado com a relação de recorrência e indução, exemplo:

$$a_n = 2 * a_n - 1; a_0 = 1 \rightarrow a_n = 2^n$$

Definições

Caso Base

Parte não recursiva, também chamado de âncora, ocorre quando a resposta para o problema é trivial.

Passo indutivo

Parteda definição que especifica como cada elemento (solução) é gerado a partir do precedente.

Figure: Efeito Droste

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \; fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \quad (1)$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

$$f(4) = 4 * 3 * 2 * (1 * 1)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \; fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

$$f(4) = 4 * 3 * 2 * (1 * 1)$$
$$f(4) = 4 * 3 * (2 * 1)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \; fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

$$f(4) = 4 * 3 * 2 * (1 * 1)$$
$$f(4) = 4 * 3 * (2 * 1)$$
$$f(4) = 4 * (3 * 2)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

$$f(4) = 4 * 3 * 2 * (1 * 1)$$
$$f(4) = 4 * 3 * (2 * 1)$$
$$f(4) = 4 * (3 * 2)$$
$$f(4) = (4 * 6)$$

Clássico exemplo

Cálculo fatorial:

$$fat(x) = \begin{cases} 1, & \text{if } x = 0. \\ x \ fat(x-1), & \text{caso contrário.} \end{cases} \tag{1}$$

considerando $x \in \mathbb{N}$

Exemplo fat(4):

$$f(4) = 4 * fat(3)$$
$$f(4) = 4 * 3 * fat(2)$$
$$f(4) = 4 * 3 * 2 * fat(1)$$
$$f(4) = 4 * 3 * 2 * 1 * fat(0)$$

$$f(4) = 4 * 3 * 2 * (1 * 1)$$
$$f(4) = 4 * 3 * (2 * 1)$$
$$f(4) = 4 * (3 * 2)$$
$$f(4) = (4 * 6)$$
$$f(4) = 24$$

Cálculo de uma potência

Calcula a potência com um expoente inteiro de um número em ponto flutuante.

$$pot(x,n) = x^n = \begin{cases} 1, & \text{se } n = 0. \\ \frac{1}{x^{-n}}, & \text{se } n < 0. \\ x(x^{n-1}), & \text{caso contrário.} \end{cases}$$

```
pot(2,-3)
 |__ 1/pot(2,3)
 |    |__ 2*pot(2,2)
 |    |    |__ 2*pot(2,1)
 |    |    |    |__ 2*pot(2,0)
 |    |    |    |    |__ 1
 |    |    |    |__ 2
 |    |    |__ 4
 |    |__ 8
 |__ 0.125
```

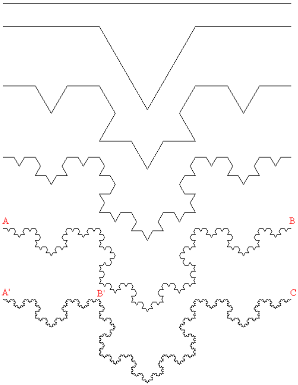
A recursão pode ser observada em um formato de árvore em que cada chamada recursiva abre um novo espaço de processamento.

Exemplo: Números naturais

- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \rightarrow n + 1 \in \mathbb{N}$

Exemplo $2 \in \mathbb{N}$?

- Verificar se $2 \in \mathbb{N}$

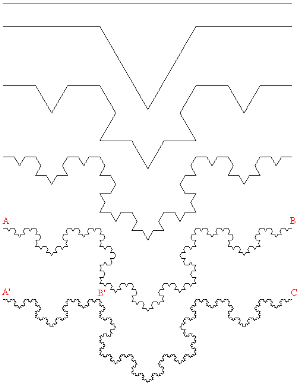


Exemplo: Números naturais

- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \rightarrow n + 1 \in \mathbb{N}$

Exemplo $2 \in \mathbb{N}$?

- Verificar se $2 \in \mathbb{N}$
  - Verificar se $1 \in \mathbb{N}$

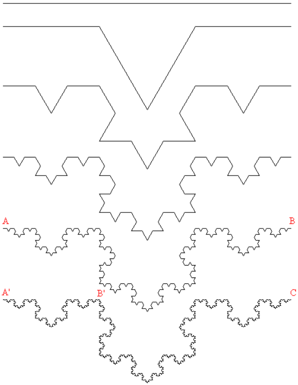


Exemplo: Números naturais

- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \rightarrow n + 1 \in \mathbb{N}$

Exemplo $2 \in \mathbb{N}$?

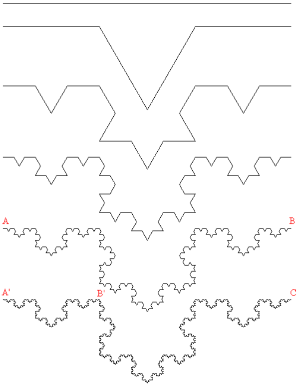
- Verificar se $2 \in \mathbb{N}$
  - Verificar se $1 \in \mathbb{N}$
    - Verificar se $0 \in \mathbb{N}$



- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \rightarrow n + 1 \in \mathbb{N}$

Exemplo $2 \in \mathbb{N}$?

- Verificar se $2 \in \mathbb{N}$
  - Verificar se $1 \in \mathbb{N}$
    - Verificar se $0 \in \mathbb{N}$
    - De fato $0 \in \mathbb{N}$

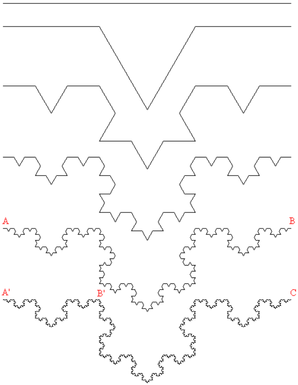


Exemplo: Números naturais

- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \to n + 1 \in \mathbb{N}$

Exemplo $2 \in \mathbb{N}$?

- Verificar se $2 \in \mathbb{N}$
  - Verificar se $1 \in \mathbb{N}$
    - Verificar se $0 \in \mathbb{N}$
    - De fato $0 \in \mathbb{N}$
  - De fato $1 \in \mathbb{N}$

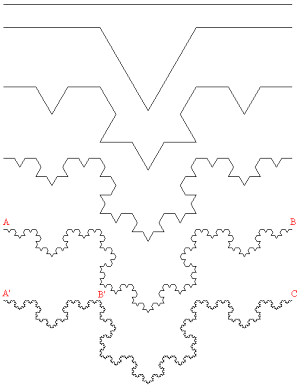


Exemplo: Números naturais

- 0 pertence aos números naturais.
- se $n \in \mathbb{N} \rightarrow n + 1 \in \mathbb{N}$

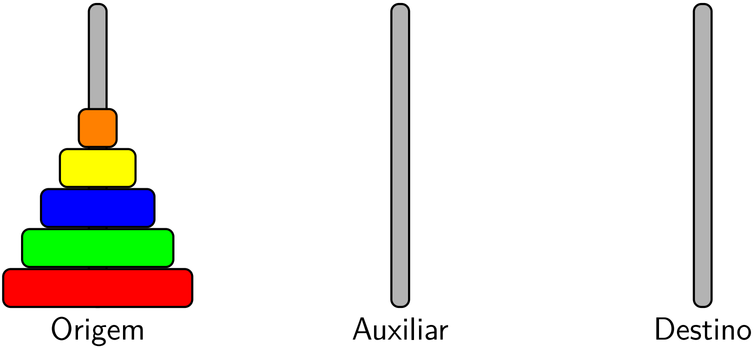
Exemplo $2 \in \mathbb{N}$?

- Verificar se $2 \in \mathbb{N}$
  - Verificar se $1 \in \mathbb{N}$
    - Verificar se $0 \in \mathbb{N}$
    - De fato $0 \in \mathbb{N}$
  - De fato $1 \in \mathbb{N}$
- Portanto $2 \in \mathbb{N}$



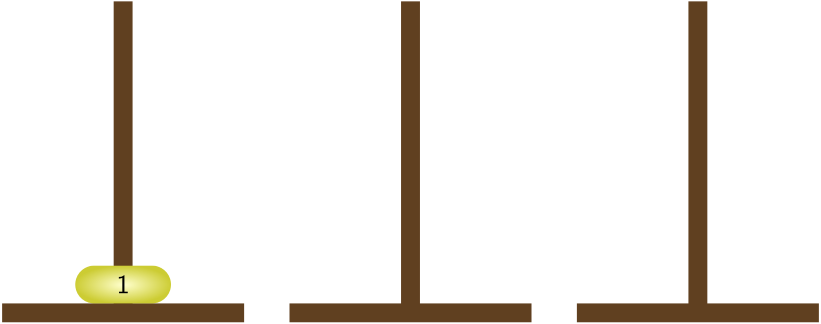
Torre de Hanoi

Objetivo: Mover os blocos da base 1 para a base 3, utilizando, caso necessário, a torre central.[1]
Única regra: Um bloco maior não pode ficar em cima de um bloco menor.

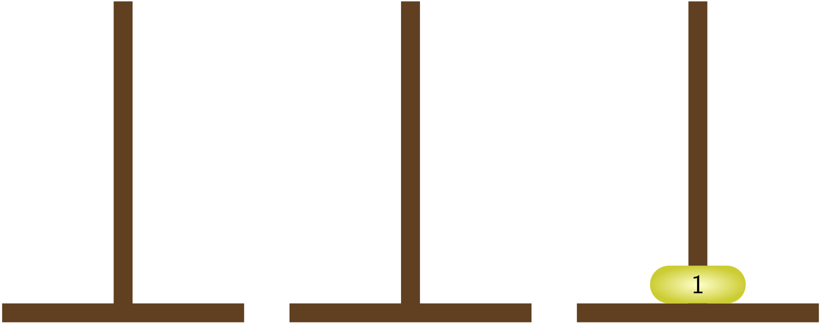


[1] Fonte da animação: https://texample.net/tikz/examples/towers-of-hanoi/

Torre de Hanoi – 1 Disco



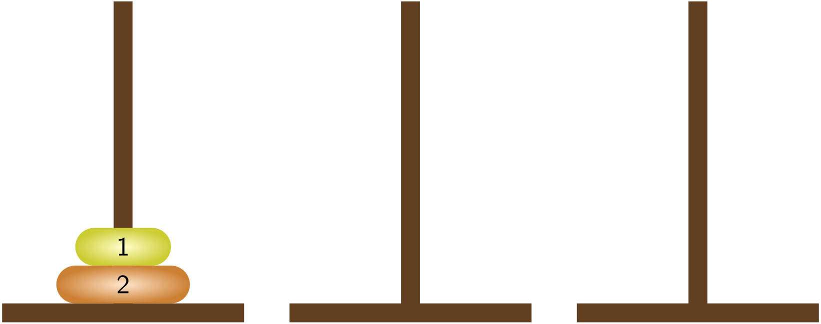
Torre de Hanoi – 1 Disco



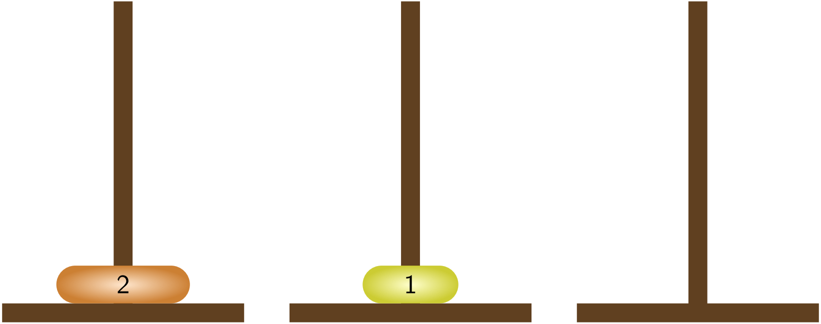
Disco movido de 1 para 3.

Torre de Hanoi – 1 Disco

Torre de Hanoi – 2 Discos

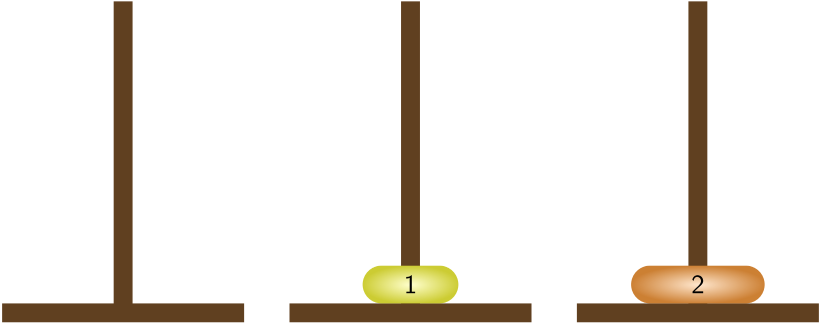


Torre de Hanoi – 2 Discos



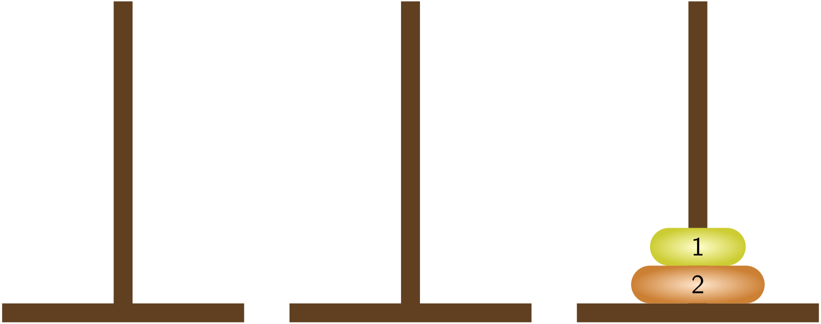
Disco movido de 1 para 2.

Torre de Hanoi – 2 Discos



Disco movido de 1 para 3.

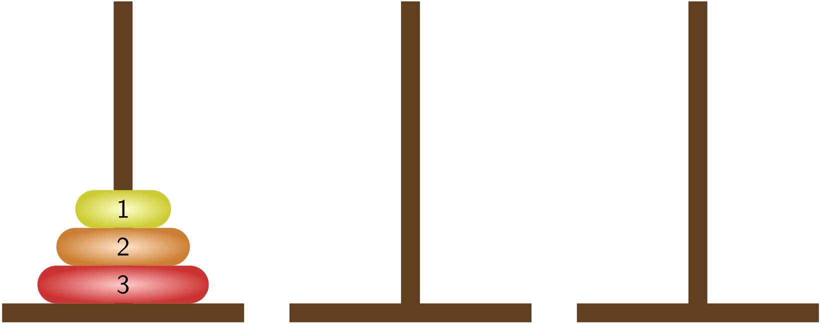
Torre de Hanoi – 2 Discos



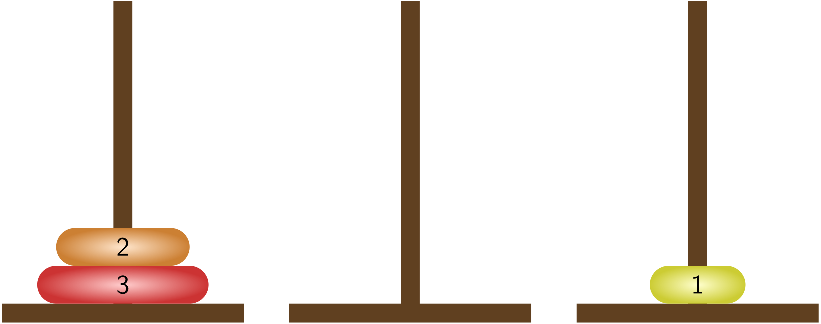
Disco movido de 2 para 3.

Torre de Hanoi – 2 Discos

Torre de Hanoi – 3 Discos

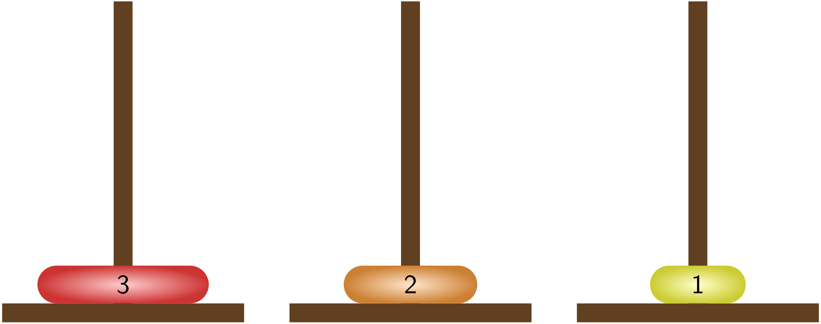


Torre de Hanoi – 3 Discos

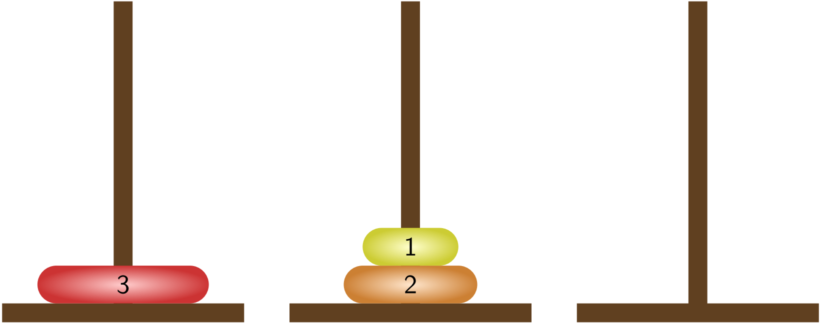


Disco movido de 1 para 3.

Torre de Hanoi – 3 Discos

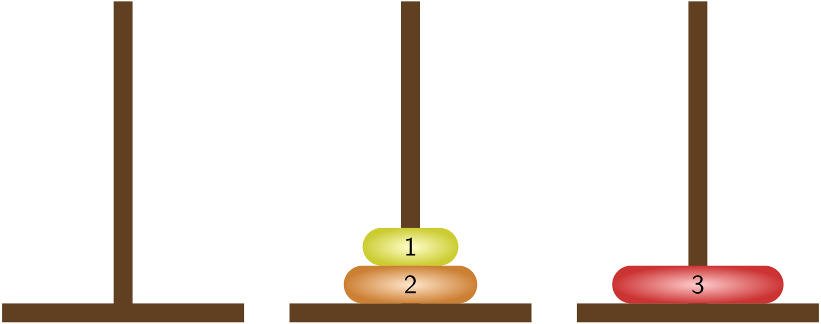


Disco movido de 1 para 2.



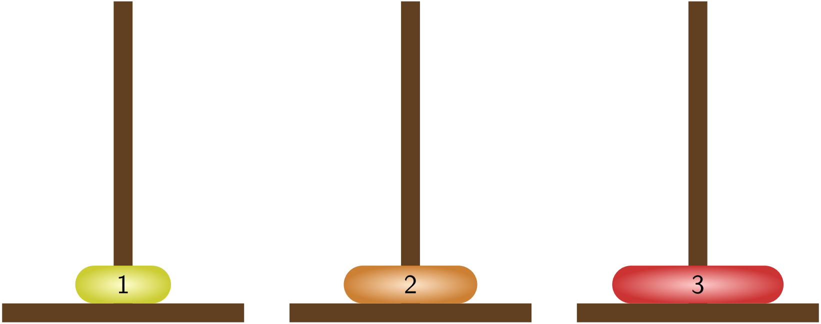
Disco movido de 3 para 2.

Torre de Hanoi – 3 Discos



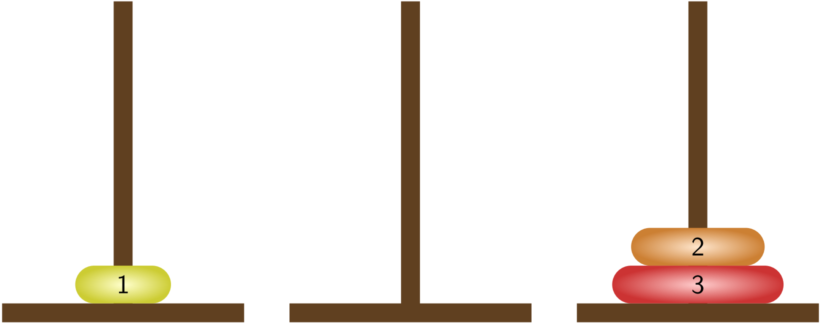
Disco movido de 1 para 3.

Torre de Hanoi – 3 Discos



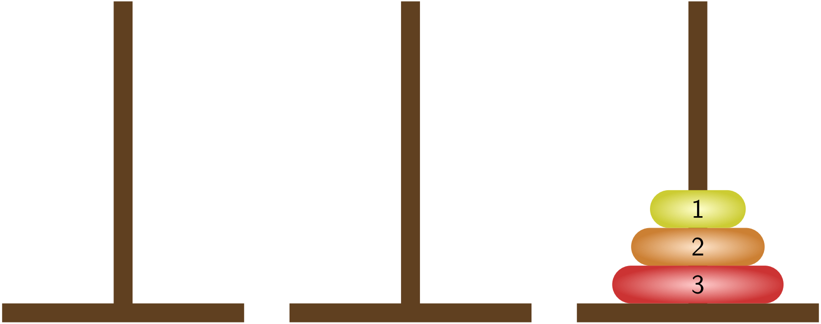
Disco movido de 2 para 1.

Torre de Hanoi – 3 Discos



Disco movido de 2 para 3.

Torre de Hanoi – 3 Discos

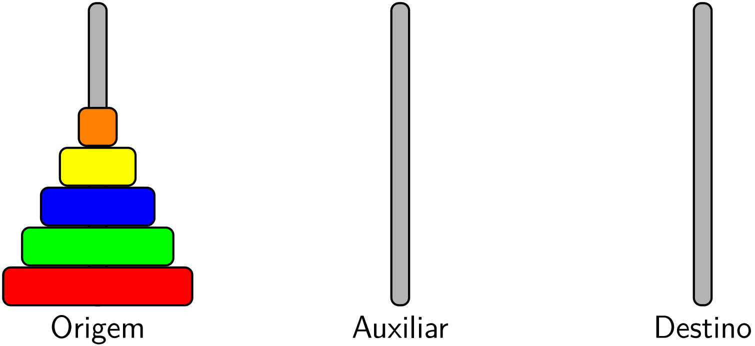


Disco movido de 1 para 3.

Torre de Hanoi – 3 Discos

Torre de Hanoi (Algoritmo) - $N > 1$

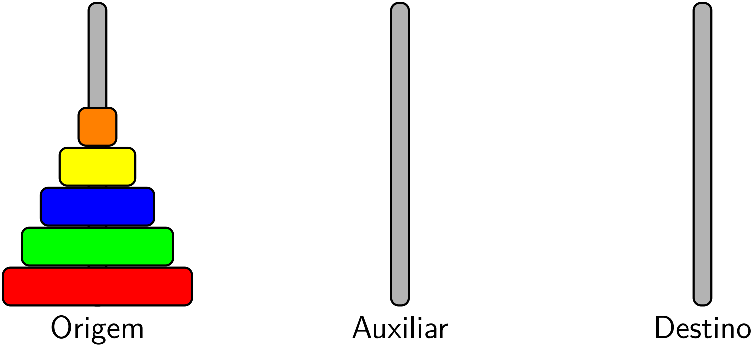
MOVER(N, ORIGEM, DESTINO, AUXILIAR)



Torre de Hanoi (Algoritmo) - $N > 1$

MOVER(N, ORIGEM, DESTINO, AUXILIAR)

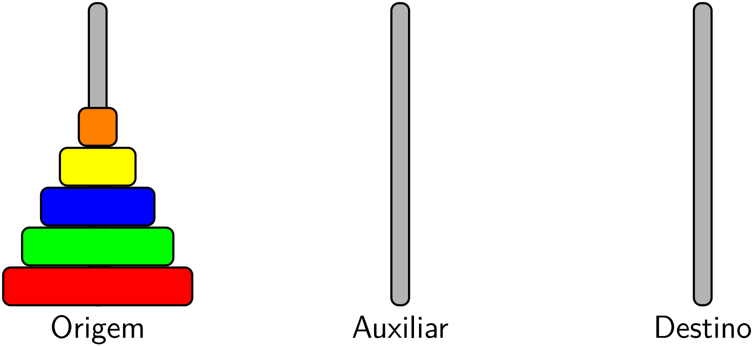
- Passo 1: MOVER(N-1, ORIGEM, AUXILIAR, DESTINO)



Torre de Hanoi (Algoritmo) - $N > 1$

MOVER(N, ORIGEM, DESTINO, AUXILIAR)

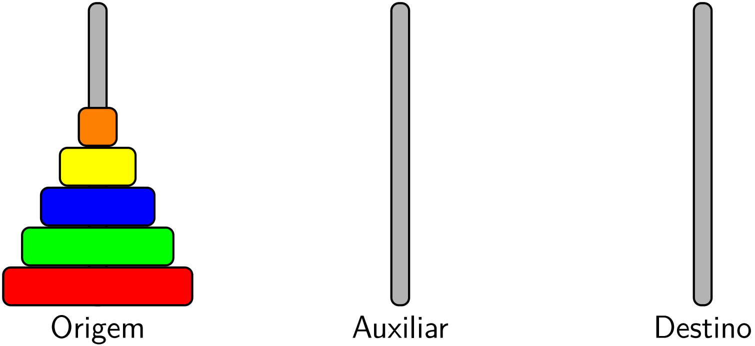
- Passo 1: MOVER(N-1, ORIGEM, AUXILIAR, DESTINO)
- Passo 2: MOVER O DISCO DA **ORIGEM** PARA O **DESTINO**



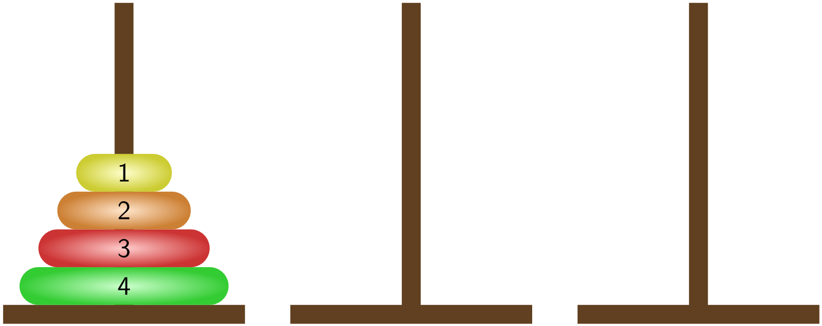
Torre de Hanoi (Algoritmo) - $N > 1$

MOVER(N, ORIGEM, DESTINO, AUXILIAR)

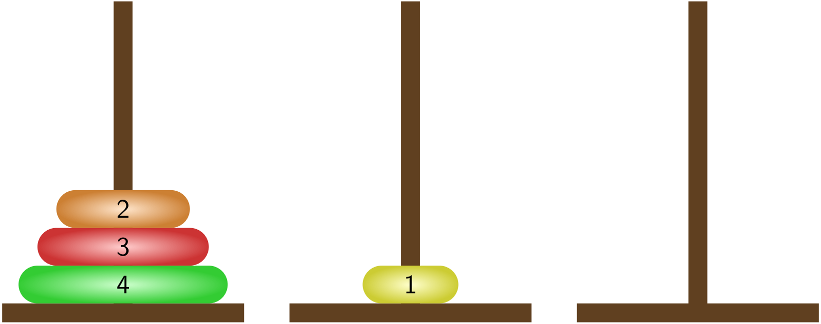
- Passo 1: MOVER(N-1, ORIGEM, AUXILIAR, DESTINO)
- Passo 2: MOVER O DISCO DA **ORIGEM** PARA O **DESTINO**
- Passo 3: MOVER(N-1, AUXILIAR, DESTINO, ORIGEM)



Torre de Hanoi – 4 Discos

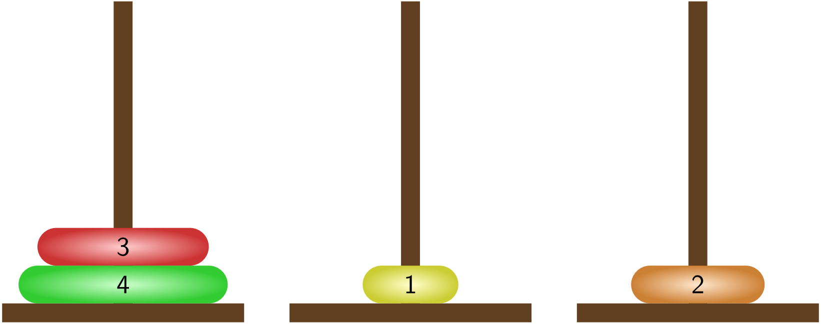


Torre de Hanoi – 4 Discos



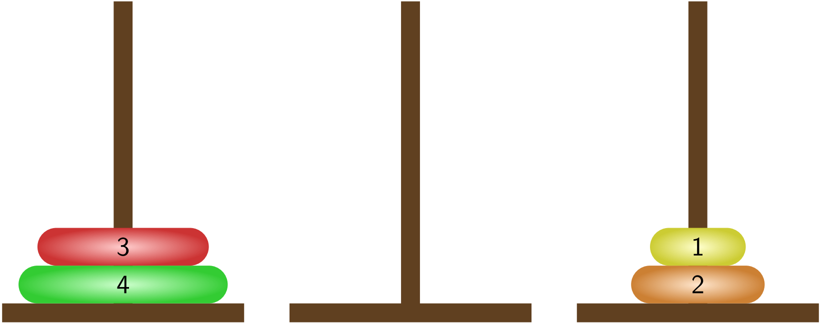
Disco movido de 1 para 2.

Torre de Hanoi – 4 Discos

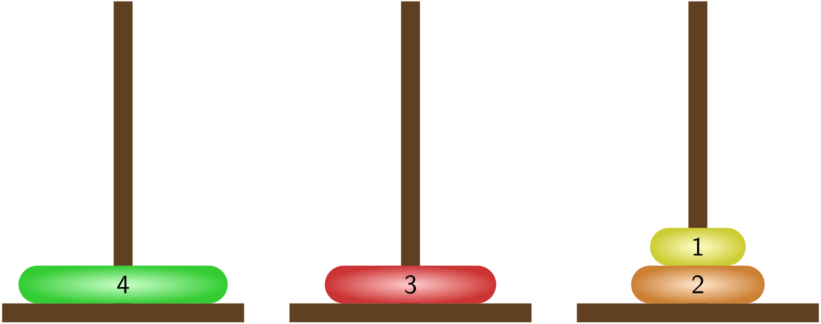


Disco movido de 1 para 3.

Torre de Hanoi – 4 Discos

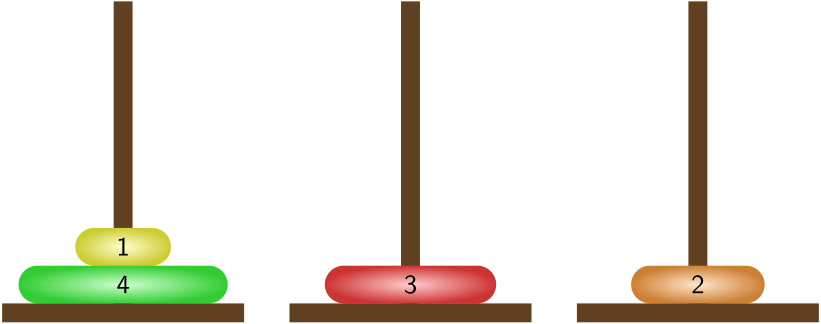


Disco movido de 2 para 3.



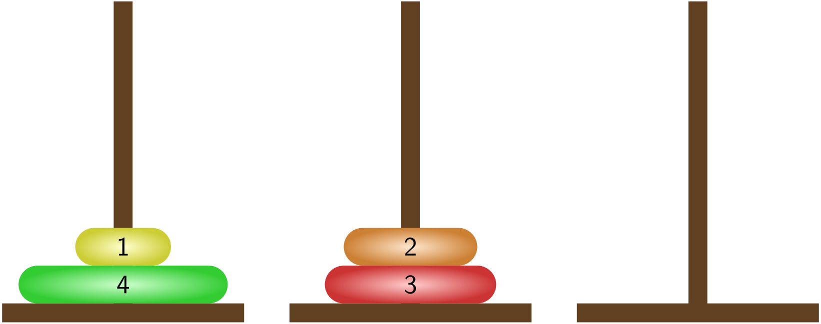
Disco movido de 1 para 2.

Torre de Hanoi – 4 Discos



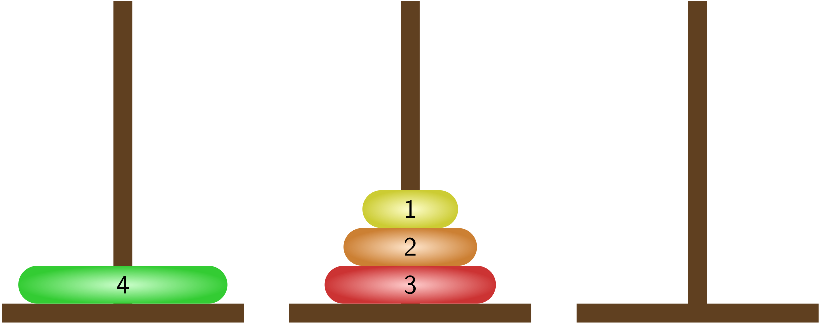
Disco movido de 3 para 1.

Torre de Hanoi – 4 Discos



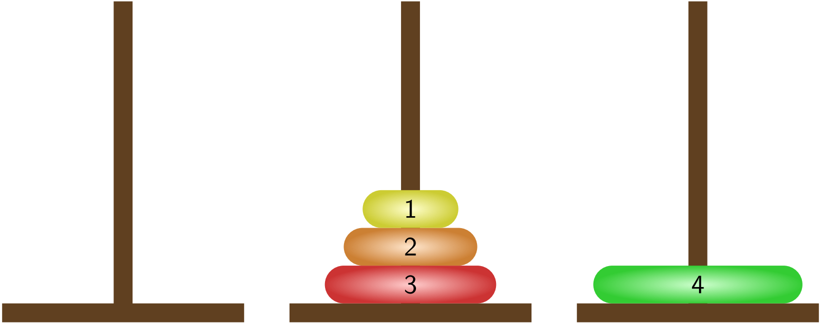
Disco movido de 3 para 2.

Torre de Hanoi – 4 Discos



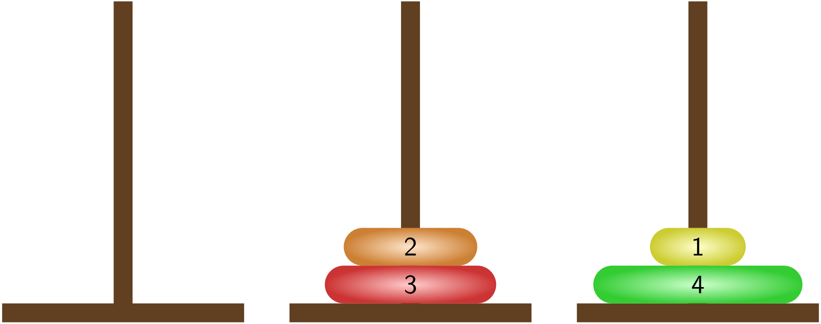
Disco movido de 1 para 2.

Torre de Hanoi – 4 Discos



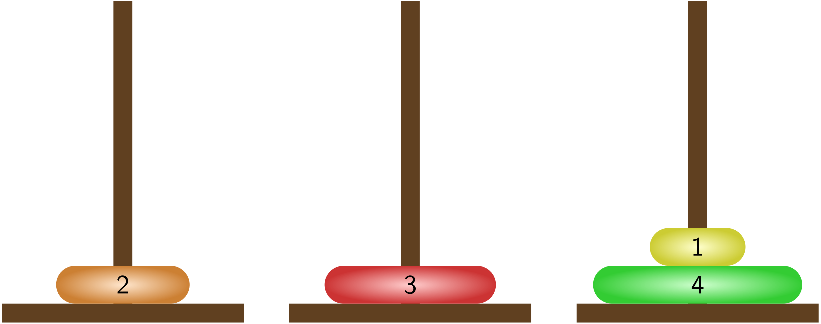
Disco movido de 1 para 3.

Torre de Hanoi – 4 Discos



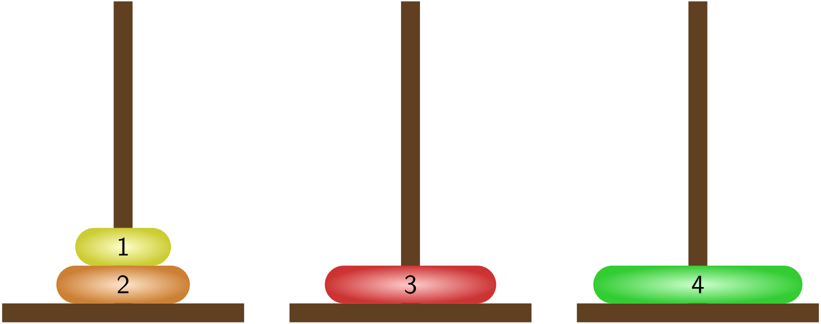
Disco movido de 2 para 3.

Torre de Hanoi – 4 Discos



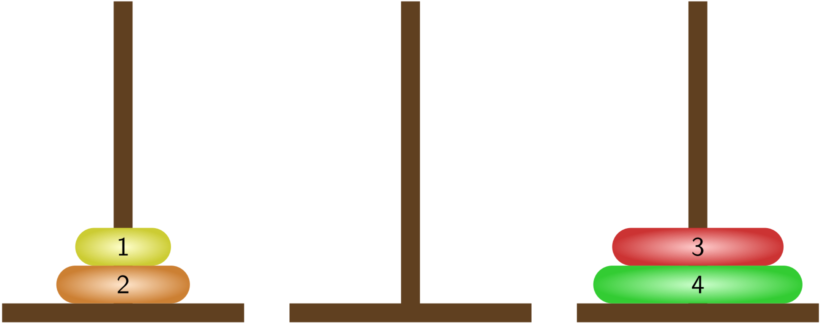
Disco movido de 2 para 1.

Torre de Hanoi – 4 Discos



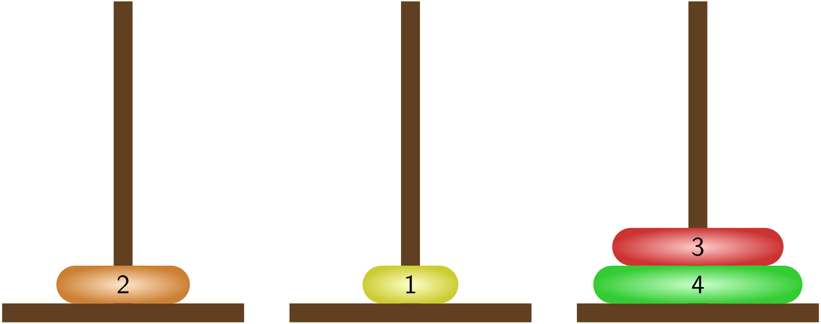
Disco movido de 3 para 1.

Torre de Hanoi – 4 Discos



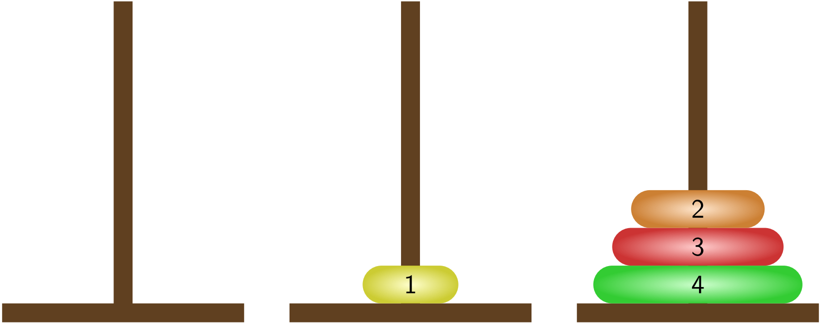
Disco movido de 2 para 3.

Torre de Hanoi – 4 Discos



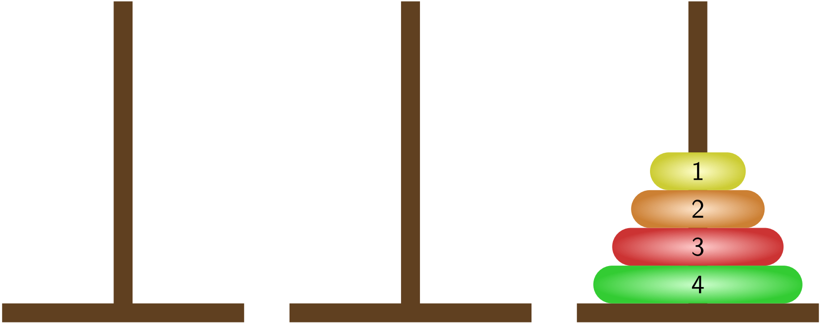
Disco movido de 1 para 2.

Torre de Hanoi – 4 Discos



Disco movido de 1 para 3.

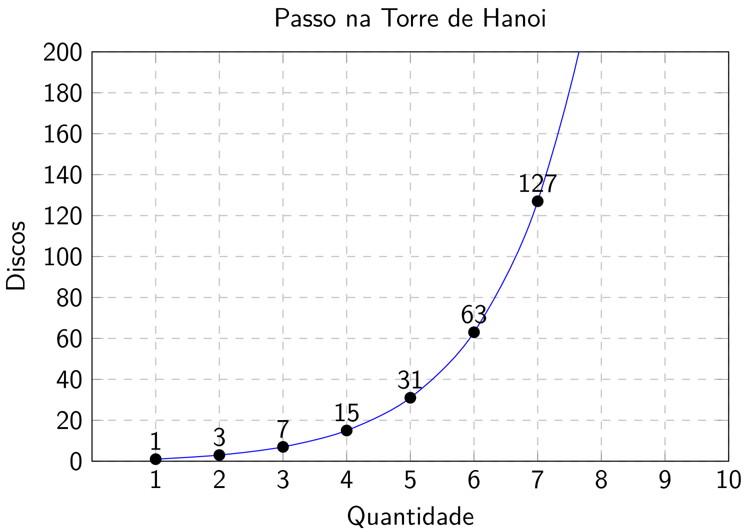
Torre de Hanoi – 4 Discos



Disco movido de 2 para 3.

Torre de Hanoi – 4 Discos

Passos da torre de Hanoi



Exemplo: Formação de sequências

Funções recursivas são comumente utilizadas para geração de sequências.
Exemplo:

$$f(n) = \begin{cases} 1, & \text{if } n = 0. \\ f(n-1) + \frac{1}{f(n-1)}, & \text{caso } n > 0. \end{cases} \tag{2}$$

Exemplo: Formação de sequências

Funções recursivas são comumente utilizadas para geração de sequências.
Exemplo:

$$f(n) = \begin{cases} 1, & \text{if } n = 0. \\ f(n-1) + \frac{1}{f(n-1)}, & \text{caso } n > 0. \end{cases} \tag{2}$$

Sequência gerada:

$$1, 2, \frac{5}{2}, \frac{29}{10}, \frac{941}{240}, \frac{969581}{272890}, \ldots$$

Consideração

Definicões recursivas de sequências tem uma desvantagem: Para determinar o valor de um elemento, e necessário conhecer todos os outros elementos anteriores. Como solução, deve-se encontrar uma expressão iterativa para a resolução sem que seja conhencido os termos anteriores.

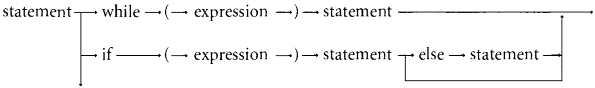
Exemplo:

$$g(n) = \begin{cases} 1, & \text{if } n = 0. \\ 2 * g(n-1), & \text{caso } n > 0. \end{cases} \tag{3}$$

$$\begin{aligned} g(0) &= 1 \\ g(1) &= 2 \\ g(2) &= 4 \end{aligned} \qquad \begin{aligned} g(3) &= 8 \\ &\dots \\ g(n) &= 2^n \end{aligned}$$

Recursão

Recursão para definições sintáticas de algoritmos:



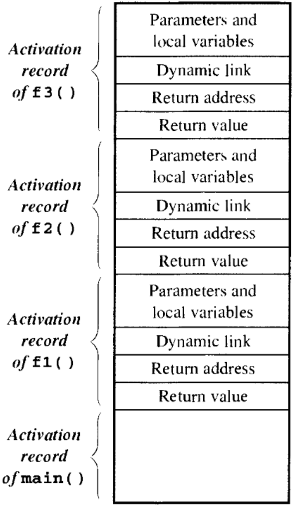
Ou em BNF: *Backus-Naur Form*

<statement> ::= while (<expression>) <statement> |
if (<expression>) <statement> |
if (<expression>) <statement>else<statement> |
...

Chamada de funções e implementação da Recursão

Considere uma função **main()** que chama **f1()** que chama**f2()** que chama **f3()**:

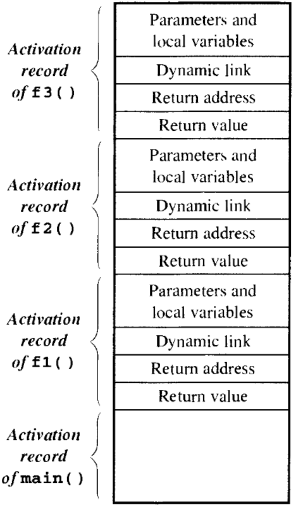
- Para cada função mais do que apenas o endereço de resposta deve ser armazenado.
- Alocação dinâmica é utilizado durante a execução do algoritmo.Cada função com suas próprias variáveis locais.
- Faz-se necessário que o contexto de cada função seja **preservado** antes de cada nova chamada.



Chamada de funções e implementação da Recursão

Considere uma função **main()** que chama **f1()** que chama**f2()** que chama **f3()**:

- Variáveis armazenadas:
    - Valores de todos os parâmetros da função;
    - Variáveis locais que podem ser armazenadas em qualquer outro local;
    - O endereço de retorno para retomar o controle pelo ativador e o endereço da instrução do ativador;
    - Um vínculo dinâmico (ponteiro para o registro de ativação do ativador);
    - O valor retornado par uma função não declarada como *void*.



Anatomia de uma chamada recursiva

Função potência:

$$pot(x,n) = \begin{cases} 1, & \text{if } n = 0. \\ x * pot(x, n-1), & \text{caso } n > 0. \end{cases} \tag{4}$$

Anatomia de uma chamada recursiva

Função potência:

$$pot(x,n) = \begin{cases} 1, & \text{if } n = 0. \\ x * pot(x, n-1), & \text{caso } n > 0. \end{cases} \tag{4}$$

Algoritmo Recursivo

```c
int pot(int x, int n) {
    if (n == 0) {
        return 1;
    } else {
        return x * pot(x, n - 1);
    }
}
```

Recursão de Cauda

Definição

A recursão de cauda é definida pelo uso somente de uma chamada recursiva no final de uma implementação de função. Quando a chamada é recursiva é a **última operação** a ser realizada dentro da função.

Recursão de Cauda

Definição

A recursão de cauda é definida pelo uso somente de uma chamada recursiva no final de uma implementação de função. Quando a chamada é recursiva é a **última operação** a ser realizada dentro da função.

Recursão de Cauda

```c
void tail(int i) {
    if (i > 0) {
        printf("%d", i);
        tail(i - 1);
    }
}
```

Recursão de Cauda

Definição

A recursão de cauda é definida pelo uso somente de uma chamada recursiva no final de uma implementação de função. Quando a chamada é recursiva é a **última operação** a ser realizada dentro da função.

Recursão de Cauda

```c
void tail(int i) {
    if (i > 0) {
        printf("%d", i);
        tail(i - 1);
    }
}
```

Recursão que não é Cauda

```c
void nonTail(int i) {
    if (i > 0) {
        nonTail(i - 1);
        printf("%d", i);
    }
}
```

Recursão de Cauda

A recursão de cauda pode ser **facilmente** substituído por um laço. Convertendo-o a uma função não recursiva.

Recursão que não é Cauda

```c
void iterativoTail_1(int i) {
    for (; i > 0; i--) {
        printf("%d", i);
    }
}

void iterativoTail_2(int i) {
    while (i > 0) {
        printf("%d", i++);
    }
}
```

Conversão

Conversão de algoritmos recursivos para iterativos

Fatorial

```c
int fatorial(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n *
         fat(n - 1);
    }
}
```

Somatório

```c
int soma(int n) {
    if (n < 2) {
        return n;
    } else {
        return n +
        soma(n - 1);
    }
}
```

Fibonacci

```c
int fib(int n){
    if(n < 2){
        return n;
    }else{
        return fib(n-1)
             + fib(n-2);
    }
}
```

Máximo Divisor Comum

```c
int mdc(int a, int b) {
    int r = a;
    if (a<b){
        r = mdc(a, b-a);
    } else if (a>b) {
        r = mdc(a-b, a);
    }return r;
}
```

Fatorial e Fibonacci Iterativo

Fatorial Iterativo

```
int fat2(int n) {
    int i = n, res = 1;
    while (i > 1) {
        res = res * i--;
    }
    return res;
}
```

Fibonacci Iterativo

```
int fib2(int n) {
    if (n == 0 || n == 1) {
        return n;
    }
    int x2 = 1, x1 = 0;
    int i = n, x = 1;
    while (i > 1) {
        x = x1 + x2;
        x1 = x2;
        x2 = x;
        i = i - 1;
    }
    return x;
}
```

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

- Qualquer algoritmo recursivo que não tem como última instrução o passo recursivo;

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

- Qualquer algoritmo recursivo que não tem como última instrução o passo recursivo;
- Ocupam mais espaço na execução;

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

- Qualquer algoritmo recursivo que não tem como última instrução o passo recursivo;
- Ocupam mais espaço na execução;
- Demandam mais tempo;

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

- Qualquer algoritmo recursivo que não tem como última instrução o passo recursivo;
- Ocupam mais espaço na execução;
- Demandam mais tempo;
- O algoritmo tem um tempo de vida menor:
  - Por ocupar mais espaço na memória, o programa é finalizado antecipadamente.
  - Possível erro 1: Memória totalmente ocupada;
  - Possível erro 2: Programa executado compromente a execução dos sistema.

Recursão que não é de Cauda

Algoritmos recursivos que não são de Cauda:

- Qualquer algoritmo recursivo que não tem como última instrução o passo recursivo;
- Ocupam mais espaço na execução;
- Demandam mais tempo;
- O algoritmo tem um tempo de vida menor:
  - Por ocupar mais espaço na memória, o programa é finalizado antecipadamente.
  - Possível erro 1: Memória totalmente ocupada;
  - Possível erro 2: Programa executado compromente a execução dos sistema.

- O algoritmo pode ser re-codificado para tornar de cauda. Exemplo:

Somatório com cauda

```
int soma(int n, int acc){
    if(n == 0){
        return acc;
    }else{
        return
        soma(n-1,acc+n);
    }
}

int soma(int n){
    return soma(n, 0);
}
```

Recursão indireta

A chamada recursiva não é executada diretamente:

Recursão indireta

A chamada recursiva não é executada diretamente:

$$\mathbf{f()} \rightarrow f1() \rightarrow f2() \rightarrow f3() \rightarrow \mathbf{f()}$$

Recursão indireta

A chamada recursiva não é executada diretamente:

$$\mathbf{f()} \rightarrow f1() \rightarrow f2() \rightarrow f3() \rightarrow \mathbf{f()}$$

As funções intermediárias podem ser re-codificadas como apenas uma única função:

$$\mathbf{g()} \equiv f1() \rightarrow f2() \rightarrow f3()$$

Resumindo dessa forma:

Recursão indireta

A chamada recursiva não é executada diretamente:

$$\mathbf{f()} \rightarrow f1() \rightarrow f2() \rightarrow f3() \rightarrow \mathbf{f()}$$

As funções intermediárias podem ser re-codificadas como apenas uma única função:

$$\mathbf{g()} \equiv f1() \rightarrow f2() \rightarrow f3()$$

Resumindo dessa forma:

$$\mathbf{f()} \rightarrow g() \rightarrow \mathbf{f()}$$

Recursão indireta

$$\begin{cases} sen(x) & = sen\left(\frac{x}{3}\right)\frac{\left(3-tan^2\left(\frac{x}{3}\right)\right)}{\left(1+tan^2\left(\frac{x}{3}\right)\right)} \\ tan(x) & = \frac{sen(x)}{cos(x)} \\ cos(x) & = 1 - sen\left(\frac{x}{2}\right) \end{cases}$$

Considerando que:

$$sen(x) \cong x - \frac{x^3}{6}$$

Recursão Infinita

- Recursão Infinita provocada por inexistência de caso base.
- Pode ser que o caso base seja inalcançável;
- O programa irá alcançar o limite da pilha, havendo um *estouro* da memória.

Recursão Infinita

- Recursão Infinita provocada por inexistência de caso base.
- Pode ser que o caso base seja inalcançável;
- O programa irá alcançar o limite da pilha, havendo um *estouro* da memória.

Exemplo:

Algoritmo com recursão infinita

```c
int fat(int n){
    return n*fat(n-1);
}
```

Recursão excessiva

Muito algoritmos recursivos podem ter processamento desnecessário.
Exemplo: Fibonacci

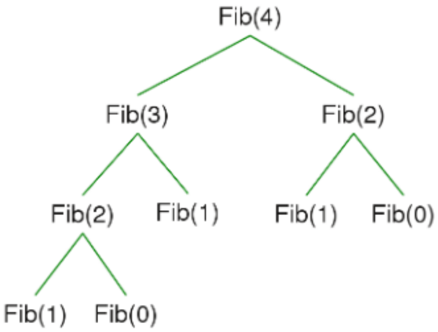


Figure: Árvore de execução da função fibonacci

Recursão aninhada

Exemplo 1:

$$h(n) = \begin{cases} 0, & \text{se } n = 0. \\ \text{n} & \text{se } n > 4 \\ \text{h}(2 + \text{h}(\text{n})), & \text{se } n \leq 4. \end{cases}$$

Recursão aninhada

Exemplo 1:

$$h(n) = \begin{cases} \text{0,} & \text{se } n = 0. \\ \text{n} & \text{se } n > 4 \\ \text{h(2 + h(n)),} & \text{se } n \leq 4. \end{cases}$$

Exemplo 2: (Função de Ackerman)

$$A(m, n) = \begin{cases} \text{m+1,} & \text{se } n = 0. \\ \text{A(n-1, 1)} & \text{se } n > 4 \ e \ m = 0 \\ \text{A(n-1, A(n, m-1)),} & \text{Caso contrário.} \end{cases}$$

Exercícios

1. Escreva uma função recursiva para adicionar os primeiros n termos da série:

$$1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} + \ldots$$

2. Apresente uma versão recursiva da função:

Função cubo

```
void cubos(int n){
    for(int i=1; i<=n; i++){
        printf("%d", i*i*i);
    }
}
```

Exercícios

3. Descreva um algoritmo recursivo para descobrir se uma palavra/expressão é um palíndromo.
4. Execute à mão o algoritmo fibonacci(5).
5. Converta o algoritmo Fatorial recursivo para Fatorial recursivo **com cauda**.
6. Execute a função de Ackerman à mão (A(4,3)).
7. Calcule o valor de $sen(80)$ considerando a aproximação apresentada nessa apresentação para valores menores do que 1^o e suas definições recursivas.