



Recursão

PrepTech Google

Recursão

Conceito

É quando uma função chama a si mesma durante sua execução.

Funcionamento:

1 **Caso base:** é a condição de parada que interrompe a recursão, retornando um valor.

2. **Caso Recursivo:** é a parte da função que chama a si mesma.

Precisa ser usada com cuidado, vide problema de stack overflow pelo excesso de chamadas recursivas em uma mesma função

Exemplo - Fatorial

Fatorial

Conceito

$$0! = 1$$

$$1! = 1$$

$$2! = 2 \times 1 = 2$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

...

Fatorial

Conceito

$$0! = 1$$

$$1! = 1$$

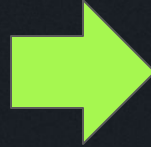
$$2! = 2 \times 1 = 1$$

$$3! = 3 \times 2 \times 1 = 6$$

$$4! = 4 \times 3 \times 2 \times 1 = 24$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

...



$$0! = 1$$

$$1! = 1 \times 0!$$

$$2! = 2 \times 1!$$

$$3! = 3 \times 2!$$

$$4! = 4 \times 3!$$

$$5! = 5 \times 4!$$

...

Fatorial

Conceito

Exemplo clássico de recursão é a função fatorial

O fatorial de um número natural n , denotado por $n!$ é definido da seguinte forma:

Caso base: $0! = 1$

Caso indutivo: $n! = n \cdot (n-1)!$, para $n \geq 1$

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Conceito

Exemplo clássico de recursão é a função fatorial

O fatorial de um número natural n , denotado por $n!$ é definido da seguinte forma:

Caso base: $0! = 1$

Caso indutivo: $n! = n \cdot (n-1)!$, para $n \geq 1$

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```


Fatorial

Funcionamento

O que ocorre quando chama o fatorial para 5?

fatorial(5)

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```


Fatorial

Funcionamento

`fatorial(5)`

`5 * fatorial(4)`

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

2 * fatorial(1)

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```


Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

2 * fatorial(1)

1 * fatorial(0)

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

2 * fatorial(1)

1 * fatorial(0)

1

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

2 * fatorial(1)

1 * 1

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * fatorial(2)

2 * 1

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```


Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * fatorial(3)

3 * 2

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * fatorial(4)

4 * 6

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

5 * 24

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```

Fatorial

Funcionamento

fatorial(5)

120

```
def fatorial(n):  
    if n == 0: # Caso base  
        return 1  
    else: # Caso recursivo  
        return n * fatorial(n-1)
```




Exemplo - Série de Fibonacci

Série de Fibonacci

Conceito

Primeiros elementos da série: 0, 1, 1, 2, 3, 5, 8, 13, ...

Pode ser definida por indução.

Funcionamento:

1 Caso base: termo 0 é 0, já termo 1 é 1.

2. Caso Recursivo: $\text{fib}(n) = \text{fib}(n-2) + \text{fib}(n-1)$, $n > 1$

Em outras palavras, o próximo termo é obtido pela soma dos dois anteriores

Vamos revisar um problema já resolvido iterativamente?

Fibonacci Number

C

509. Fibonacci Number

Solved ✓

Easy Topics Companies

The **Fibonacci numbers**, commonly denoted $F(n)$ form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

$$\begin{aligned} F(0) &= 0, F(1) = 1 \\ F(n) &= F(n-1) + F(n-2), \text{ for } n > 1. \end{aligned}$$

Given n , calculate $F(n)$.

Example 1:

Input: $n = 2$
Output: 1
Explanation: $F(2) = F(1) + F(0) = 1 + 0 = 1$.

Example 2:

Input: $n = 3$
Output: 2
Explanation: $F(3) = F(2) + F(1) = 1 + 1 = 2$.

Example 3:

Input: $n = 4$
Output: 3
Explanation: $F(4) = F(3) + F(2) = 2 + 1 = 3$.

Constraints:

- $0 \leq n \leq 30$



Fibonacci Number

Prática no LeetCode - <https://leetcode.com/problems/fibonacci-number/description/>

Fibonacci Number

```
int fib(int n){  
    if( n == 0)  
        return 0;  
    else if (n == 1)  
        return 1;  
  
    return fib(n-2) + fib(n-1);  
}
```


Fibonacci Number

Prática no LeetCode - <https://leetcode.com/problems/fibonacci-number/description/>

Fibonacci Number (iterativa)

```
int fib(int n){  
  
    if( n == 0 || n == 1)  
        return n;  
  
    int fib1 = 0;  
    int fib2 = 1;  
    for(int i = 2; i <= n; i++) {  
        int aux = fib2;  
        fib2 = fib2 + fib1;  
        fib1 = aux;  
    }  
    return fib2;  
}
```

Fibonacci Number

Prática no LeetCode - <https://leetcode.com/problems/fibonacci-number/description/>

Compare o tempo de processamento dessa versão recursiva com a sua iterativa.

Linguagem: C

Recursiva: 8ms

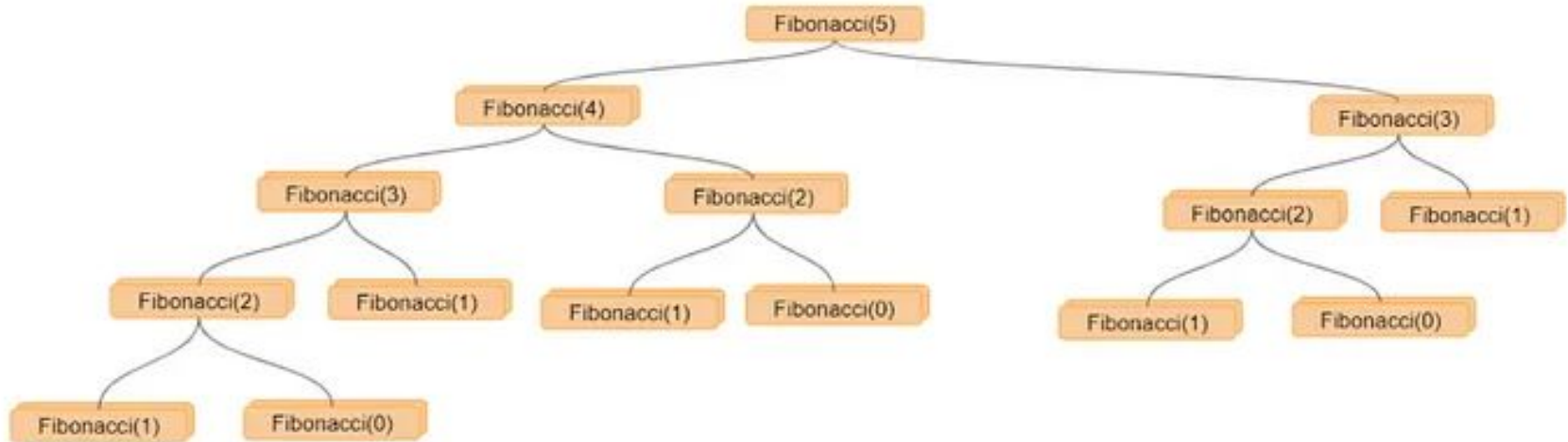
Iterativa: 0ms

Por que a recursiva tem um desempenho pior que a iterativa?

A resposta tem a ver com a pilha de execução. O mesmo termo pode ser recalculado diversas vezes.

Série de Fibonacci

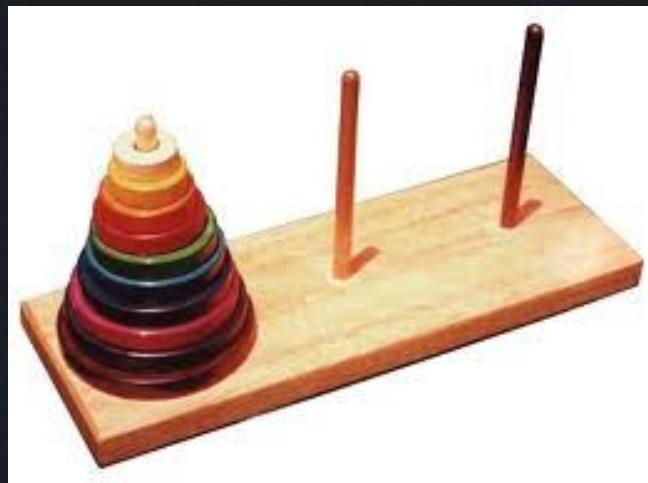
Pilha de Execução



Exemplo - Torres de Hanoi

Torres de Hanoi

Apresentação



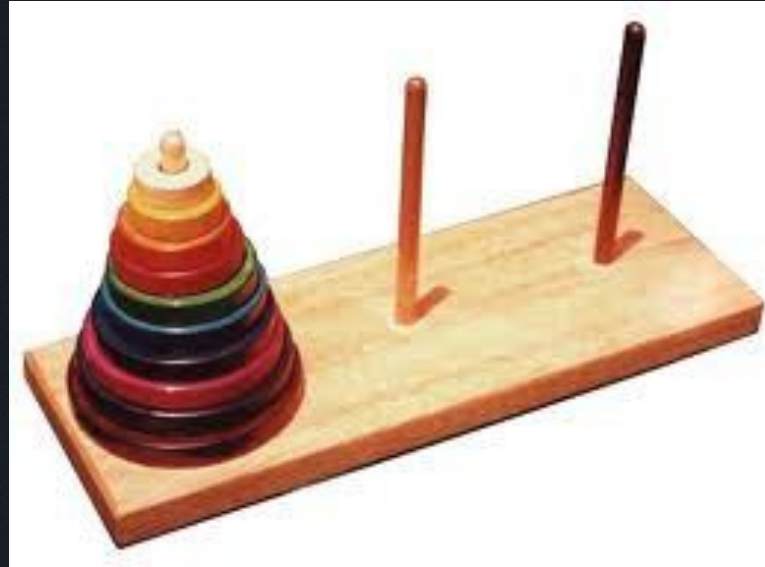
Torres de Hanoi

Definições

Há três hastes e um número natural N de discos

Os N discos são de tamanhos diferentes

Inicialmente, todos os N discos estão na primeira haste



Torres de Hanoi

Regras

- Move-se um único disco por vez
- Um disco deve estar sempre em uma das três hastes ou em movimento de uma haste para outra
- Um disco maior não pode ser colocado sobre um disco menor

Objetivo:

- Mover os N discos da primeira haste para a terceira haste

Torres de Hanoi

Lenda

- Foi proposto inicialmente pelo matemático francês Edouard Lucas, em 1883
- Ele criou a lenda que um grupo de monges deveria mover 64 discos concêntricos da primeira para a terceira hastes, respeitando as regras descritas
- Proposta inicial: criar uma regra de movimentação ótima

Torres de Hanoi

Exemplo de estado



Aqui não parecem estar resolvendo o problema da melhor forma :)

O estado da figura é válido, pois não há um disco maior sobre um menor

Torres de Hanoi

Algoritmo Recursivo

```
hanoi( N, sourceStem, auxStem, targetStem )
```

- Caso base:
 - Não precisa fazer nada se o N for 0
- Caso recursivo:
 - Usando a função hanoi, move os N-1 discos de sourceStem para auxStem, usando o targetStem como auxiliar
 - Move o disco N (o maior) de sourceStem diretamente para o targetStem
 - Usando a função hanoi, move os N-1 discos de auxStem para targetStem, usando o sourceStem como auxiliar

Torres de Hanoi

Algoritmo Recursivo

```
def hanoi(N, sourceStem, auxStem, targetStem):  
    if N>=1:  
        hanoi(N-1, sourceStem, targetStem, auxStem)  
        print('move', N, 'from', sourceStem, 'to', targetStem)  
        hanoi(N-1, auxStem, sourceStem, targetStem)
```

- Onde está o caso base? Por que o algoritmo pára quando $N < 1$?

Torres de Hanoi

Funcionamento do código

- Olhem o que imprime quando chama `hanoi(3, 'A', 'B', 'C')`

```
move 1 from A to C  
move 2 from A to B  
move 1 from C to B  
move 3 from A to C
```

`hanoi(2, 'A', 'C', 'B')`

```
move 1 from B to A  
move 2 from B to C  
move 1 from A to C
```

`hanoi(2, 'B', 'A', 'C')`

Torres de Hanoi

Funcionamento do código

- Olhem o que imprime quando chama `hanoi(5, 'A', 'B', 'C')`

```
move 1 from A to C
move 2 from A to B
move 1 from C to B
move 3 from A to C
move 1 from B to A
move 2 from B to C
move 1 from A to C
move 4 from A to B
move 1 from C to B
move 2 from C to A
move 1 from B to A
move 3 from C to B
move 1 from A to C
move 2 from A to B
move 1 from C to B
move 5 from A to C
```

(... continua ao lado ...)

`hanoi(4, 'A', 'C', 'B')`

(... continuando ...)

```
move 1 from B to A
move 2 from B to C
move 1 from A to C
move 3 from B to A
move 1 from C to B
move 2 from C to A
move 1 from B to A
move 4 from B to C
move 1 from A to C
move 2 from A to B
move 1 from C to B
move 3 from A to C
move 1 from B to A
move 2 from B to C
move 1 from A to C
```

`hanoi(4, 'B', 'A', 'C')`

Torres de Hanoi

Complexidade do Algoritmo

Quantas operações são necessárias?


$$T(N) = \begin{cases} 1, & \text{se } N=1 \\ 2*T(N-1)+1, & \text{se } N>1 \end{cases}$$

- Isso significa que a quantidade de operações, dado N com entrada é $2^N - 1$
- Torres de hanoi é um problema computacionalmente intensivo

Torres de Hanoi

Complexidade do Algoritmo

N	Operações
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023



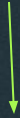
Problemas do LeetCode

- <https://leetcode.com/problems/merge-two-sorted-lists/description/>
- <https://leetcode.com/problems/palindrome-linked-list/description/>
- <https://leetcode.com/problems/decode-string/description/>

decode("3[a2[c]]")



3 * decode("a2[c]")



"a" + 2 * decode("c]")



"c"

decode("3[a2[c]]")



3 * decode("a2[c]")



"a" + 2 * decode("c]")



"c"

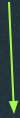
"a" + 2 * "c"



decode("3[a2[c]]")



3 * decode("a2[c]")

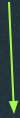


"a" + 2 * "c"  "a" + "cc"

decode("3[a2[c]]")



3 * decode("a2[c]")

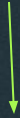


"a" + 2 * "c" \longrightarrow "a" + "cc" \longrightarrow "acc"

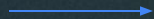
decode("3[a2[c]]")



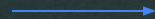
3 * decode("a2[c]")



"a" + 2 * "c"



"a" + "cc"



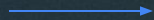
"acc"



decode("3[a2[c]]")



3 * "acc"



"accaccacc"

decode("3[a2[c]]") → "accaccacc"



Obrigado