Recursão

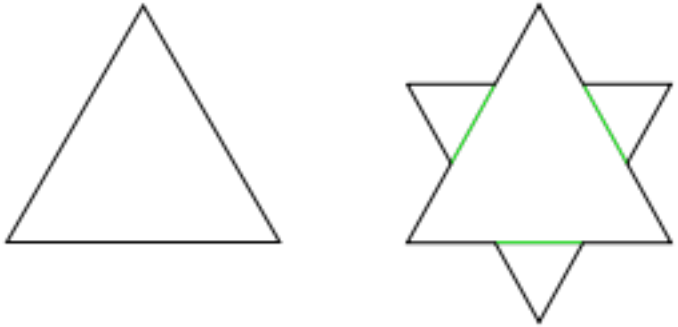
Recursividade - definição

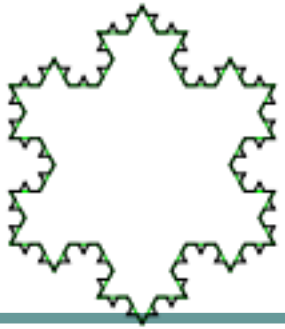
Recursividade - definição

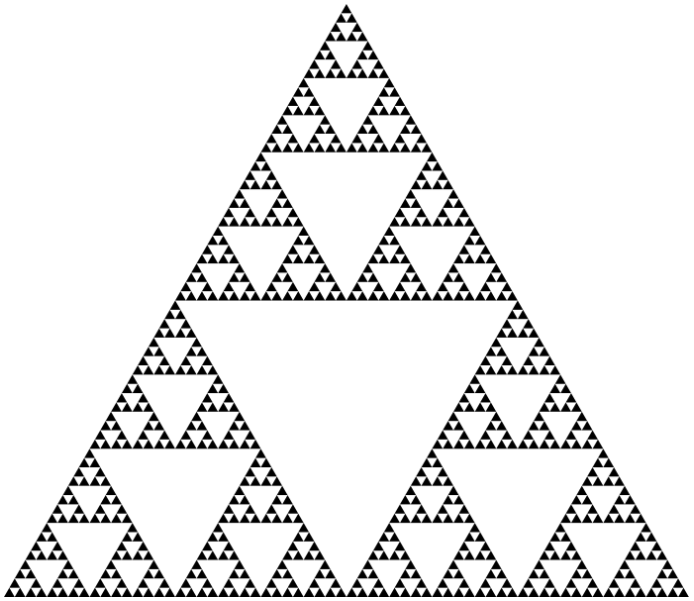
- A recursividade é muito usada na matemática:
  - Funções e séries recursivas
  - Fractais

Recursividade - definição

- Fractais e figuras recursivas







Recursividade - definição

- Séries recursivas
  - Série de Fibonacci

$$f_n = \begin{cases} 1, n = 0 \\ 1, n = 1 \\ f_{n-1} + f_{n-2}, \forall\ n \geq 2 \end{cases}$$

  - Fatorial

$$n! = \begin{cases} 1, n = 0 \\ n \times (n-1)!, \forall\ n > 0 \end{cases}$$

Recursividade - Elementos

- **Caso(s) base**: casos bem conhecidos onde definimos a solução *a priori.* Também conhecido como caso trivial.

- **Regra de recursão**: regra que identifica como derivar qualquer outro valor, baseada no caso base. Método geral que reduz o problema a um ou mais problemas menores (subproblemas) da mesma natureza.

Recursão

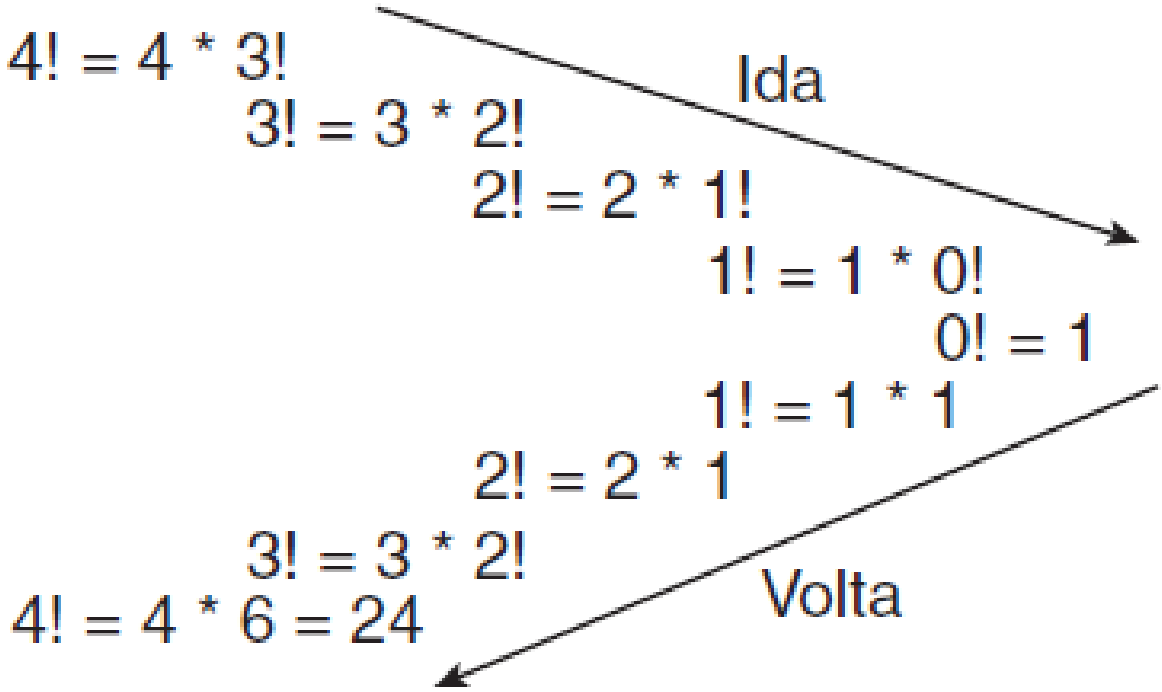
0! = 1

1! = 1 * 0!

2! = 2 * 1!

3! = 3 * 2!

4! = 4 * 3!



n! = n * (n - 1)! : fórmula geral

0! = 1 : caso-base

Recursividade na programação

- Também podemos usar a ideia de recursividade em programas
- Como calcular o fatorial de forma recursiva?
- Como calcular a série de fibonacci de forma recursiva?
- Podemos usar funções recursivas

Recursão - Fatorial

Sem Recursão

```
int fatorial (int n){
if (n == 0)
    return 1;
else {
    int i;
    int f=1;
    for (i=2; i <= n;i++)
          f = f * i;
    return f;
    }
}
```

Recursão - Fatorial

Com Recursão

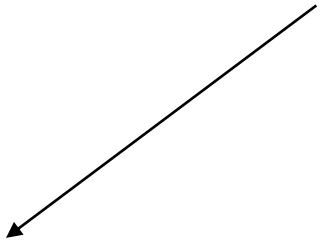
```
int fatorial (int n){
if (n == 0)
    return 1;
else
    return n*fatorial(n-1);
}
```

Recursão - Fatorial

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

Pilha de Execução

Empilha `fatorial(4)`

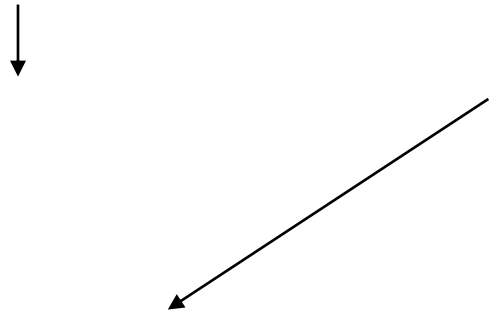


`fatorial(4)` -> `return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução



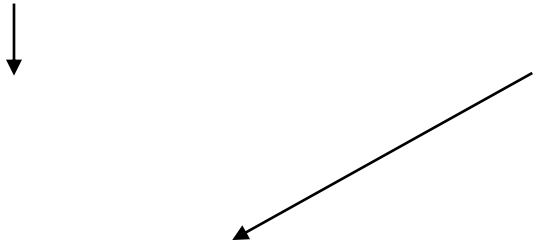
Empilha `fatorial(3)`

`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução



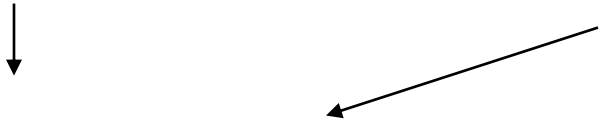
Empilha `fatorial(2)`

`fatorial(2)`	`-> return 2*fatorial(1)`
`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução



Empilha `fatorial(1)`

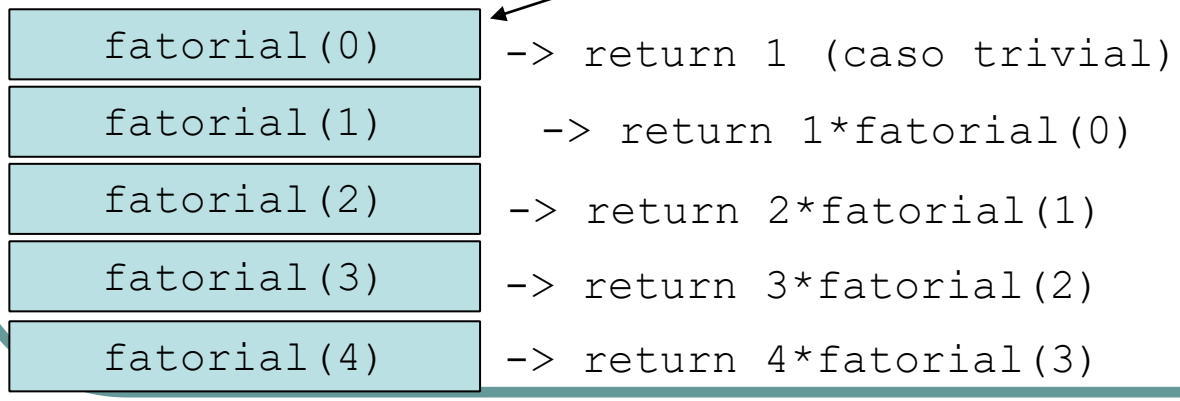
`fatorial(1)`	`-> return 1*fatorial(0)`
`fatorial(2)`	`-> return 2*fatorial(1)`
`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução

Empilha `fatorial(0)`



Recursão - Fatorial

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

Pilha de Execução

Desempilha `fatorial(0)`

`fatorial(0)`	`-> return 1 (caso trivial)`
`fatorial(1)`	`-> return 1*`**`fatorial(0)`**
`fatorial(2)`	`-> return 2*fatorial(1)`
`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução

Desempilha `fatorial(1)`

`fatorial(1)`	`-> return 1*1`
`fatorial(2)`	`-> return 2*fatorial(1)`
`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução

Desempilha `fatorial(2)`

`fatorial(2)`	`-> return 2*(1*1)`
`fatorial(3)`	`-> return 3*fatorial(2)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução

Desempilha `fatorial(3)`

`fatorial(3)`	`-> return 3*(2*1*1)`
`fatorial(4)`	`-> return 4*fatorial(3)`

Recursão - Fatorial

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

Pilha de Execução

Desempilha `fatorial(4)`

`fatorial(4)` -> return 4***(3*2*1*1)**

Recursão - Fatorial

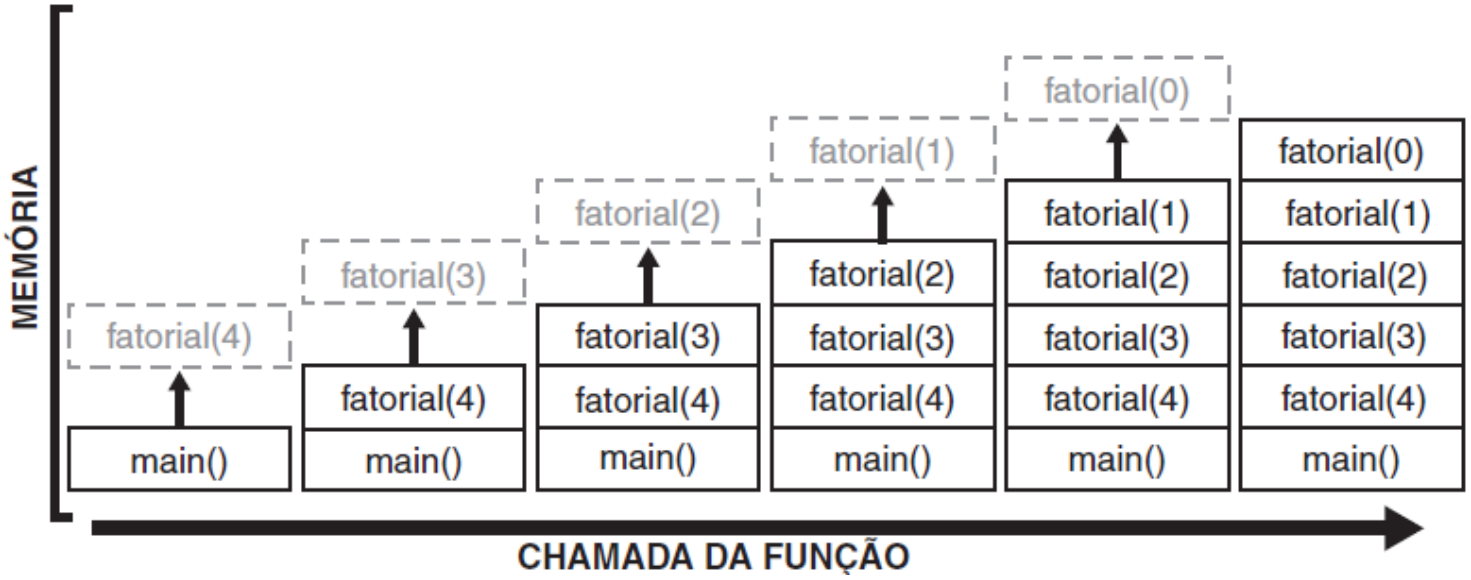
```
int fatorial(int n)
{
  if (n == 0)
    return 1;
  else
    return n * fatorial(n-1);
}
```

Resultado = 24

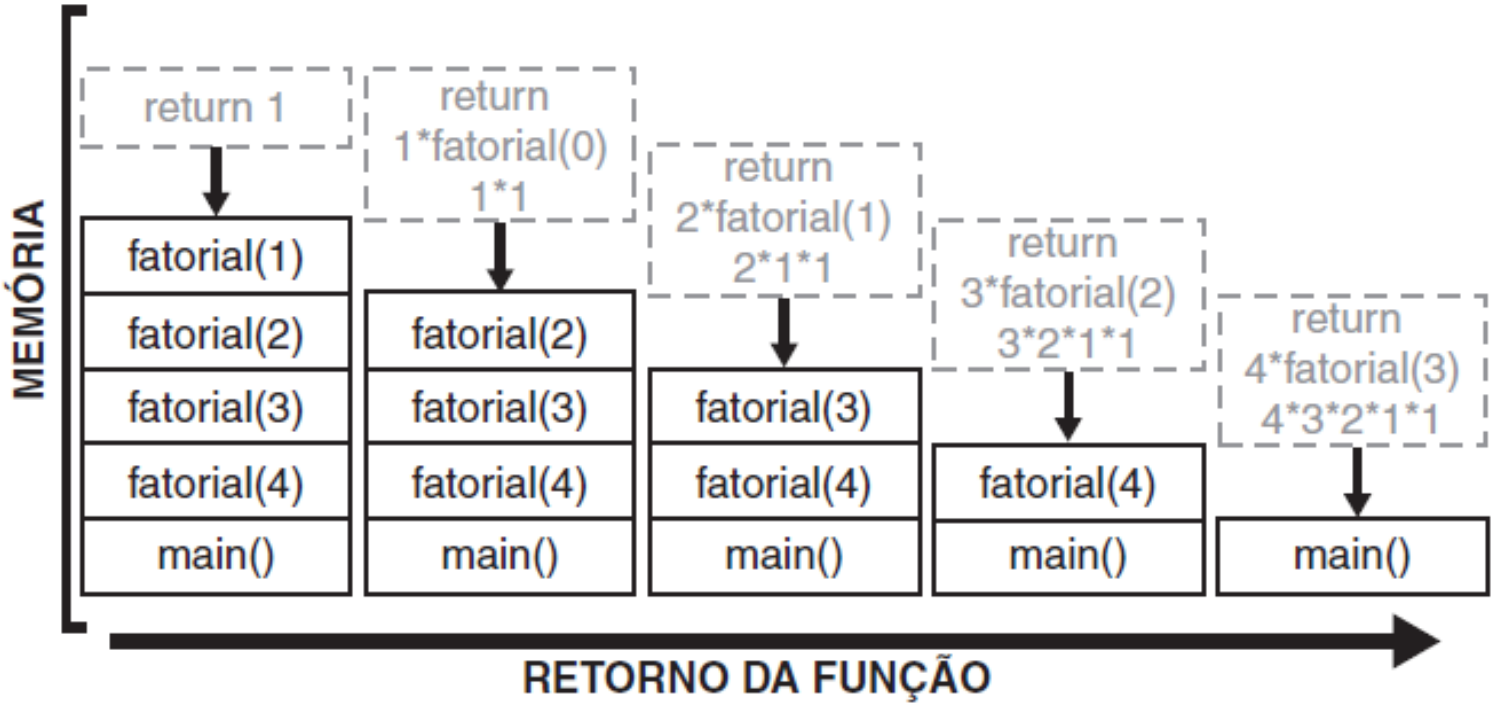
Recursão - Fatorial

- O que acontece na chamada da função fatorial com um valor como n = 4?

```
int x = fatorial (4);
```



Recursão - Fatorial



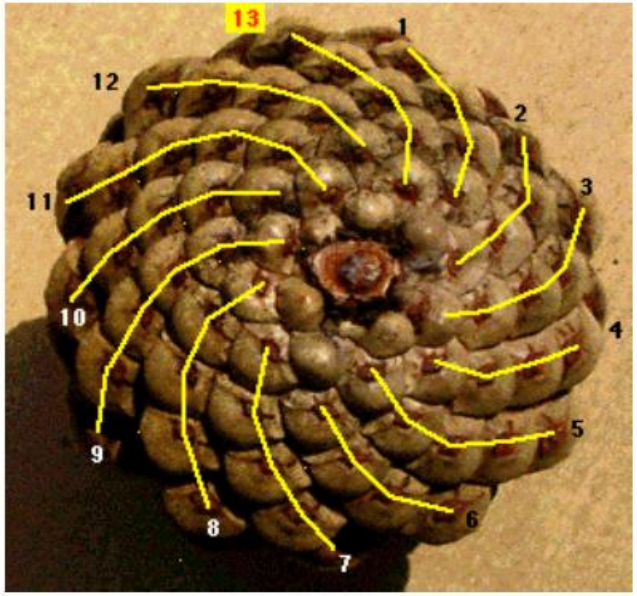
Recursão - Fatorial

- Erros comuns
  - Critério de parada: determina quando a função deverá parar de chamar a si mesma.
  - O parâmetro da chamada recursiva deve ser sempre modificado, de forma que a recursão chegue a um término.

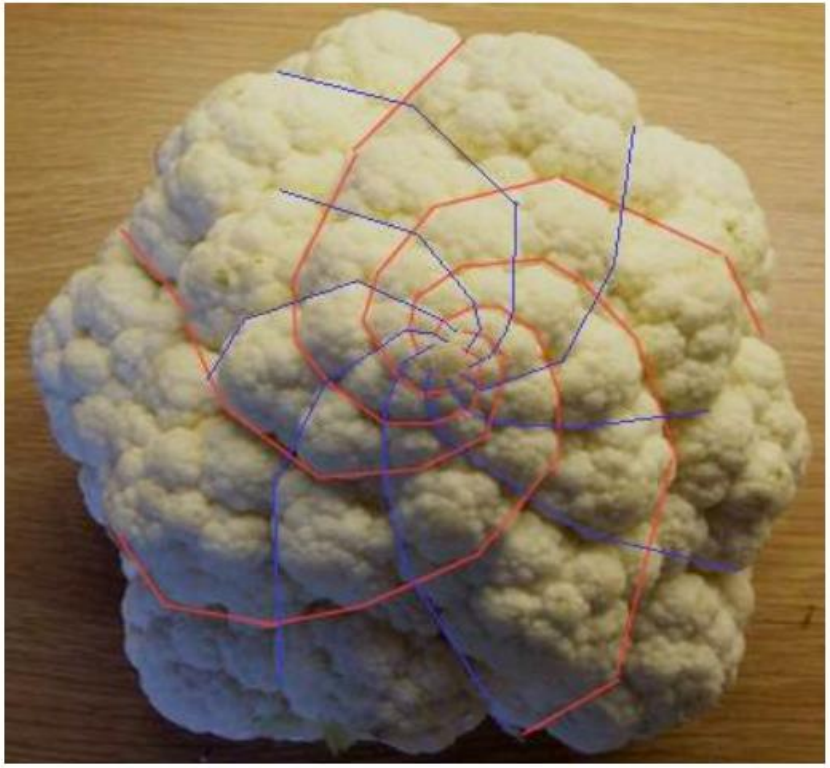
Recursão - Fatorial

```
int fatorial (int n){
if (n == 0) //critério de parada
    return 1;
else
    return n*fatorial(n-1); //parâmetro muda
}
```

Sequência de Fibonacci e a Natureza

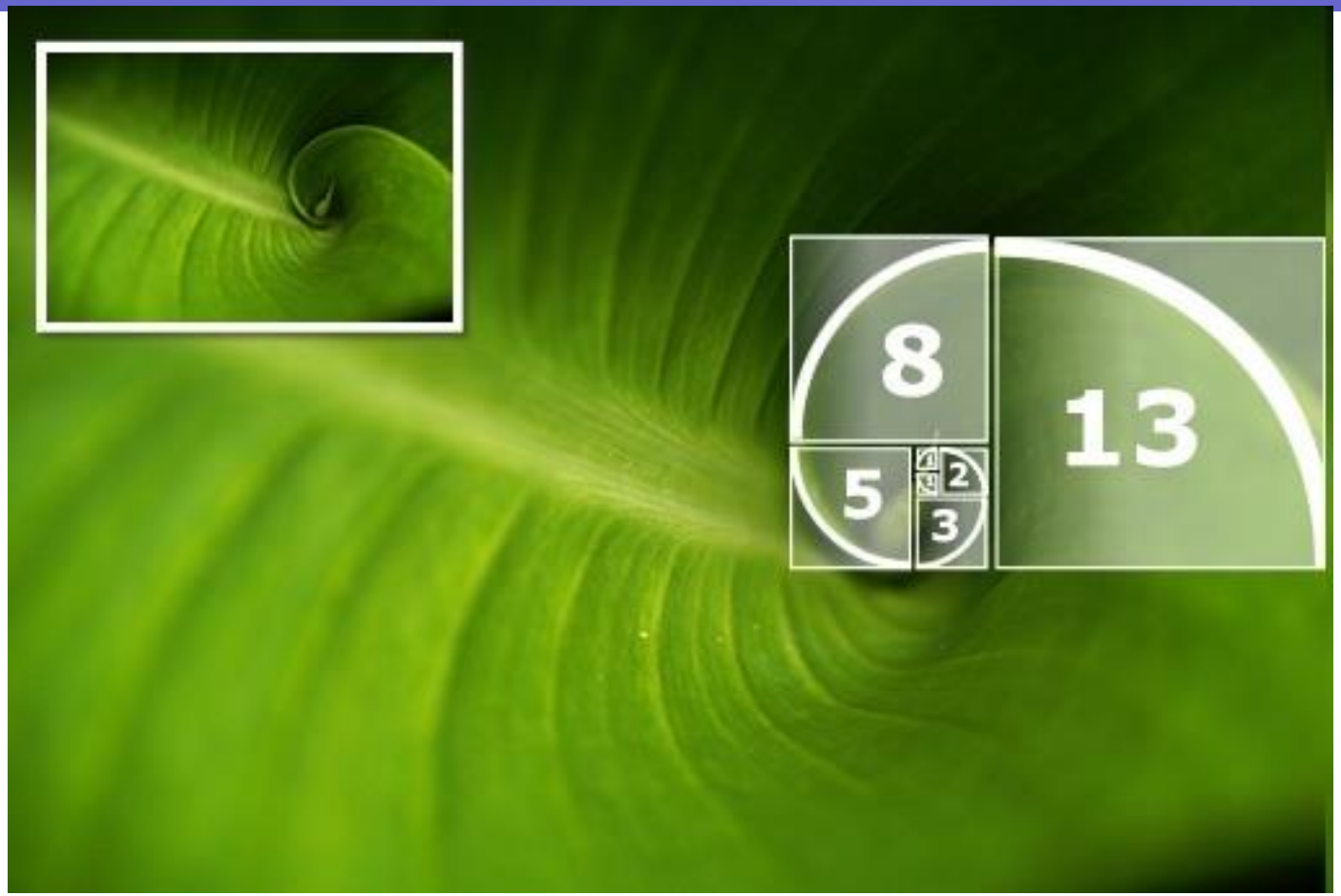


pinecone



cauliflower

Sequência de Fibonacci e a Natureza



Fibonacci

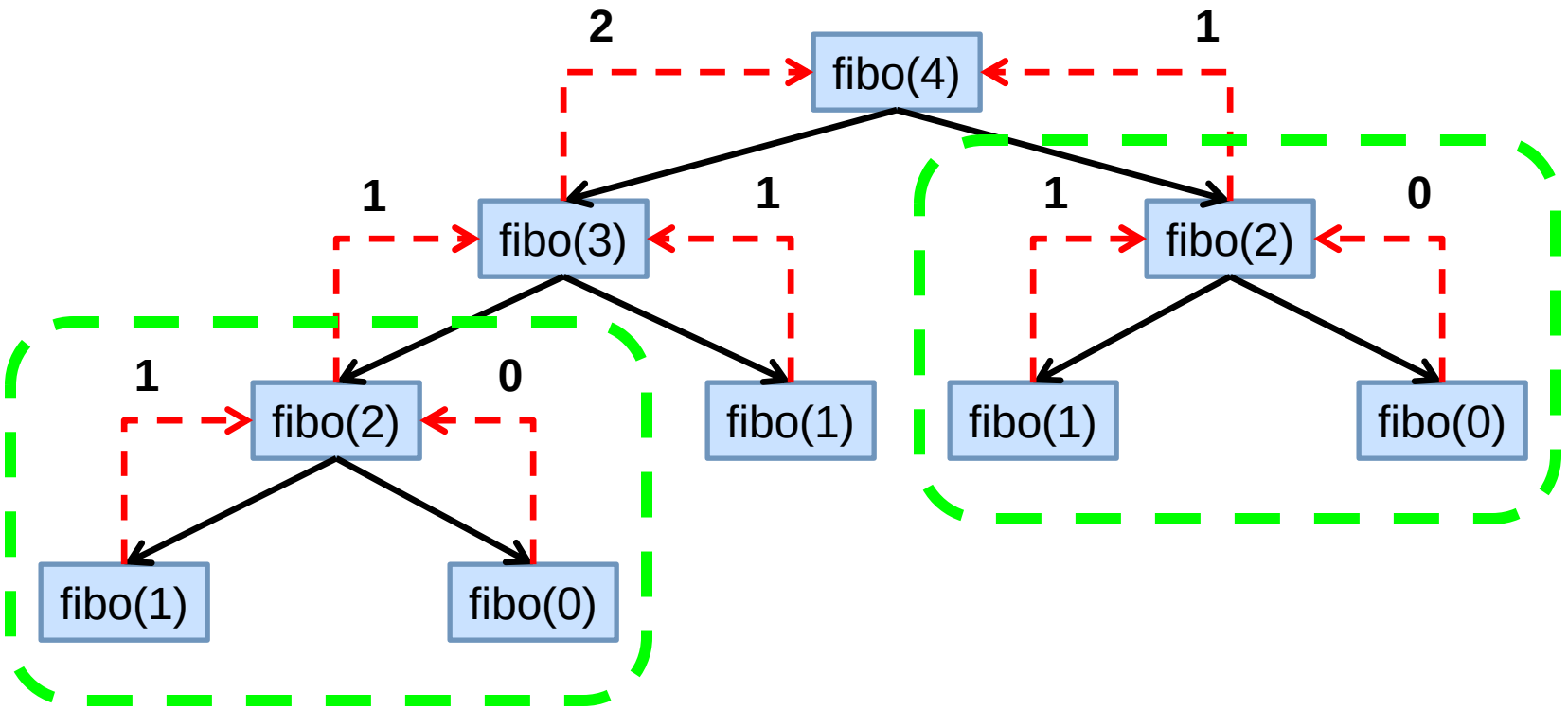
Essa sequência é um clássico da recursão

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

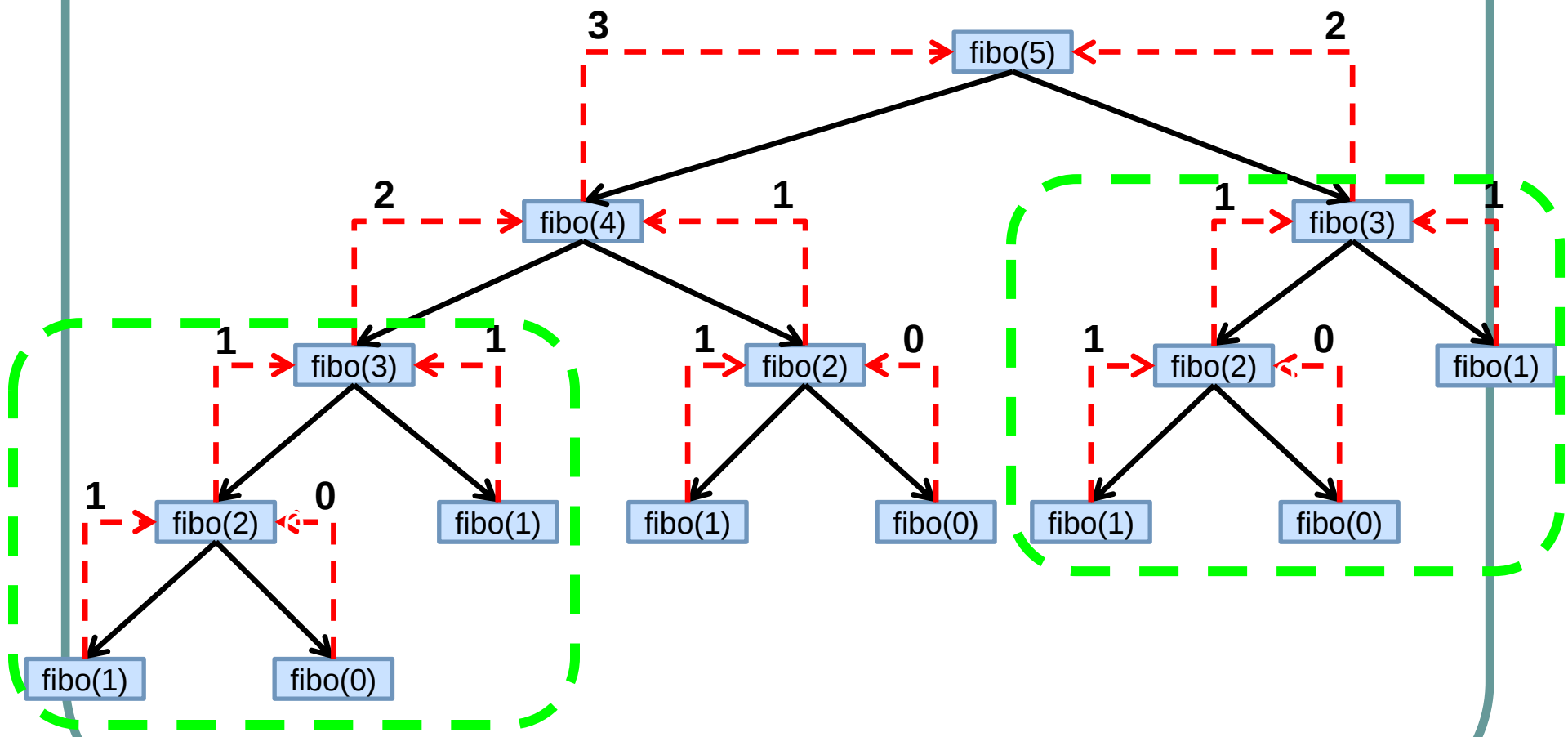
- Sua solução recursiva

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

Fibonacci


2
1
fibo(4)
1
1
fibo(3)
1
0
fibo(2)
1
0
fibo(2)
fibo(1)
fibo(1)
fibo(0)
fibo(1)
fibo(0)

Fibonacci


3
fibo(5)
2
2
fibo(4)
1
1
fibo(3)
1
1
fibo(3)
1
1
fibo(2)
0
1
fibo(2)
0
1
fibo(2)
0
1
fibo(1)
1
fibo(1)
fibo(0)
1
fibo(1)
fibo(0)
1
fibo(1)
fibo(0)

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms				
iterativo	0.17 ms				

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s			
iterativo	0.17 ms	0.33 ms			

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min		
iterativo	0.17 ms	0.33 ms	0.50 ms		

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	•1 s	•2 min	•21 dias	
iterativo	0.17 ms	•0.33 ms	•0.50 ms	•0.75 ms	

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min	21 dias	??????
iterativo	0.17 ms	0.33 ms	0.50 ms	0.75 ms	1,50 ms

Fibonacci

O livro dos autores Brassard e Bradley (*Fundamentals of Algorithmics*, 1996, pág. 73) apresenta um quadro comparativo de tempos de execução das versões iterativa e recursiva:

n	10	20	30	50	100
recursivo	8 ms	1 s	2 min	21 dias	10^9 anos
iterativo	0.17 ms	0.33 ms	0.50 ms	0.75 ms	1,50 ms

- Portanto: um algoritmo recursivo nem sempre é o melhor.
- No entanto, a recursividade muitas vezes torna o algoritmo mais simples.