Aula 05:
- Recursão (parte 1)

Prof. João Henrique Kleinschmidt
Material elaborado pelo prof. Jesús P. Mena-Chalco

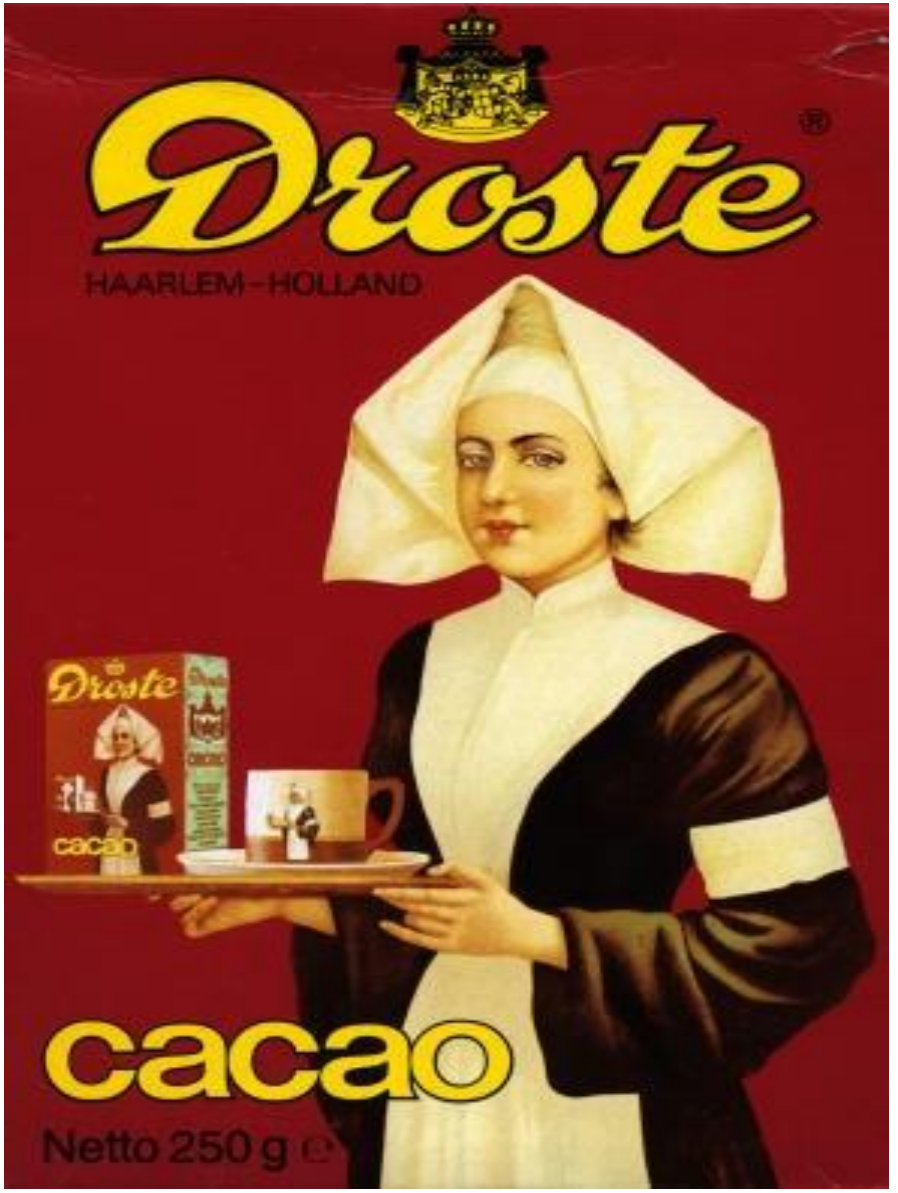
3Q-20108

Recursão:

Se você ainda não entendeu;

Ver: "*Recursão*".

Efeito Droste



Anuncio de cacão com uma imagem recursiva.


Ciclo da Água
Armazenamento de água no gelo
Precipitação
Armazenamento de água na atmosfera
Condensação
Evapotranspiração
Evaporação
Escoamento superficial proveniente do degelo
Escoamento superficial
Caudal no rio
Infiltração
Nascente
Armazenamento de água doce
Descarga do aquífero
Armazenamento da água nos oceanos
Armazenamento de água subterrânea
USGS
U.S. Department of the Interior
U.S. Geological Survey
http://ga.water.usgs.gov/edu/watercycle.html

Recursão

- O conceito de recursão é de fundamental importância em computação!

- Muitos problemas computacionais têm a seguinte
- propriedade:

Cada instância do problema contém uma instância menor do mesmo problema.

→ Dizemos que esses problemas têm

estrutura recursiva.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Recursão

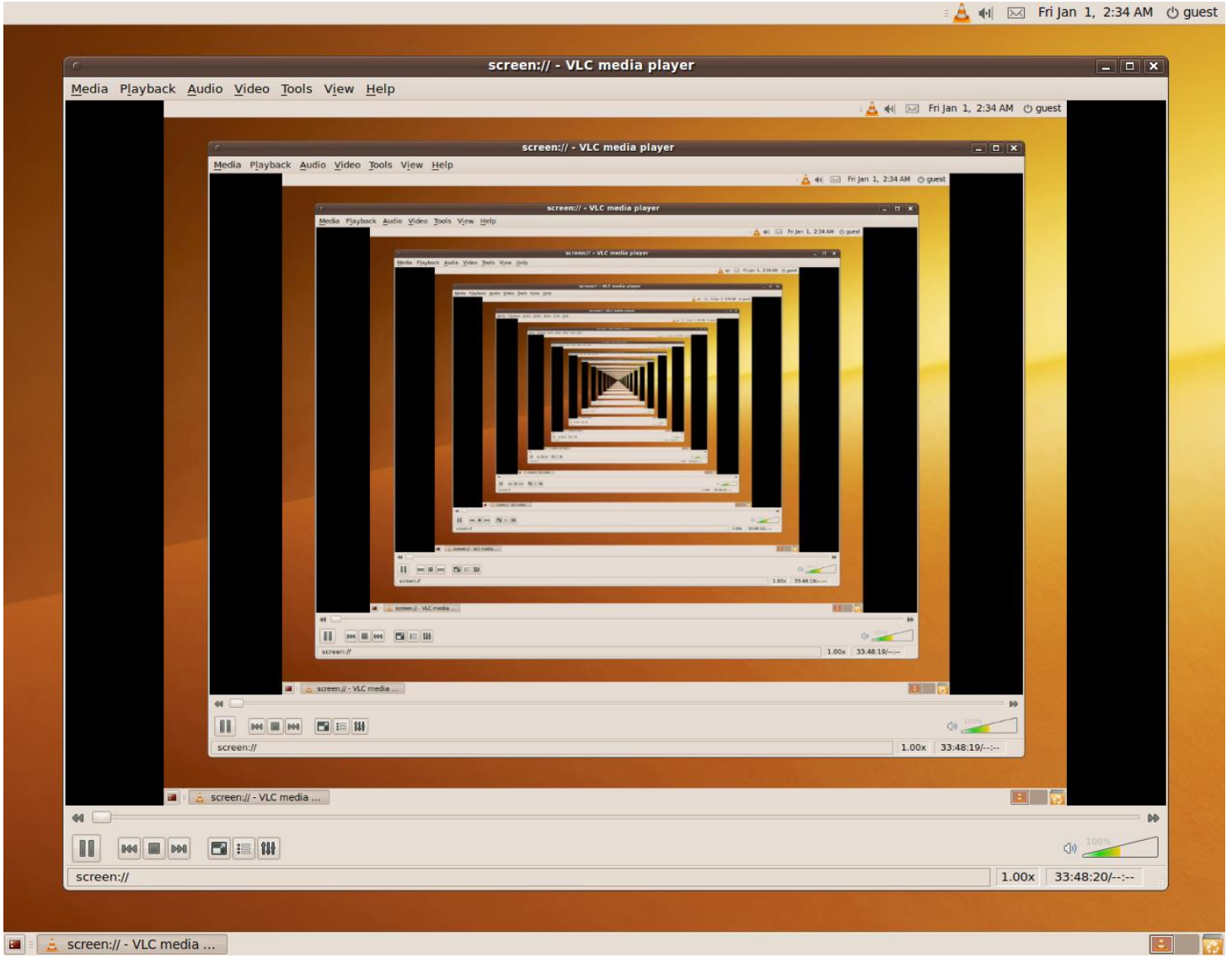
Para resolver um tal problema é natural aplicar o seguinte método:

- **Se a instância em questão é pequena:**
- → Resolva-a diretamente
  (use força bruta se necessário)
- **Senão**
- → Reduza-a a uma instância menor do mesmo problema
- → Aplique o método à instância menor
  e volte à instância original.

A aplicação do método produz um algoritmo recursivo.

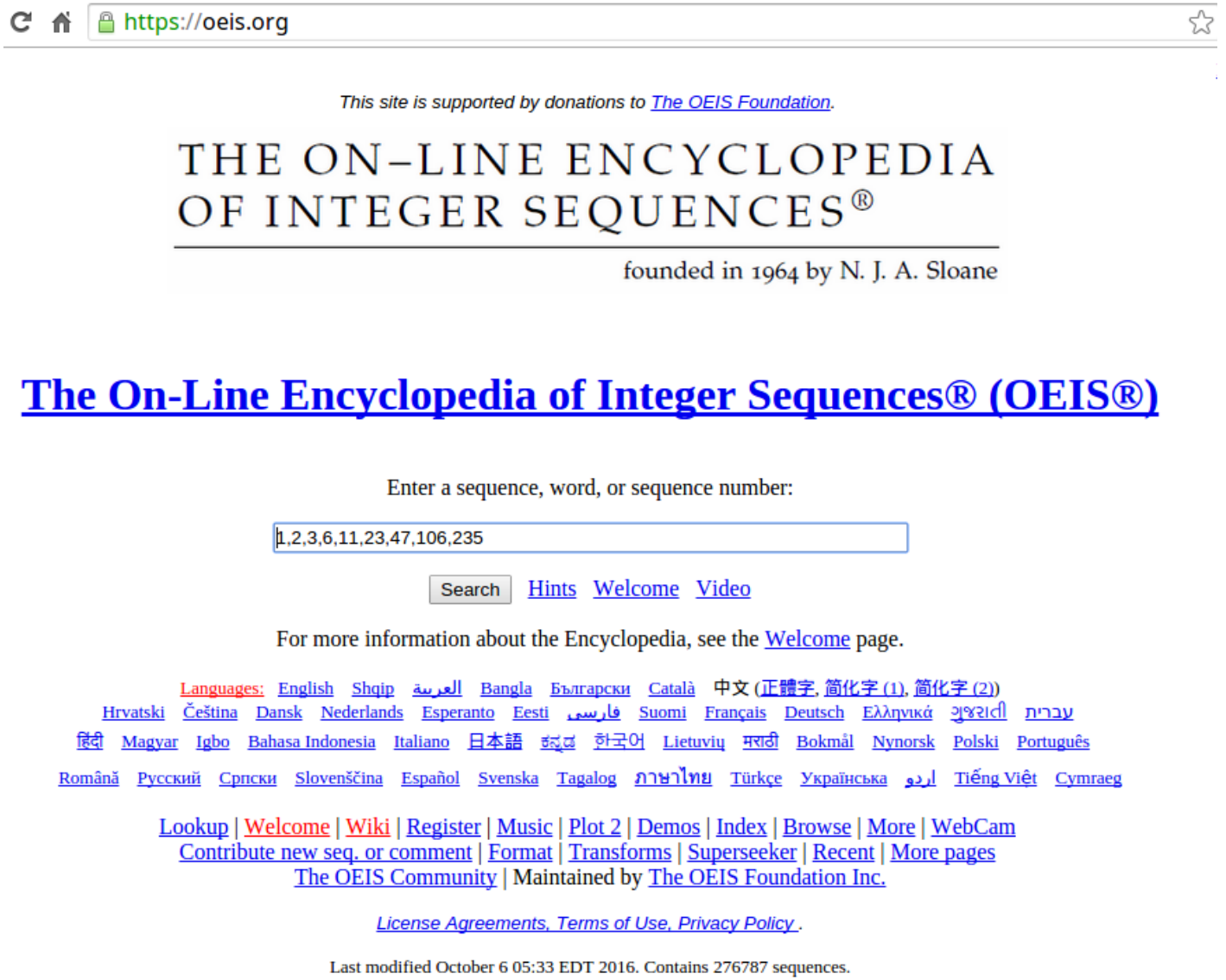
(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

A recursão pode ser infinita.
Não esqueça de definir **o caso base** (condição de parada)



Fonte: http://en.wikipedia.org/wiki/Recursion

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000



https://oeis.org/

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES®

founded in 1964 by N. J. A. Sloane

1,2,6,24,120 Search Hints

(Greetings from The On-Line Encyclopedia of Integer Sequences!)

Search: **seq:1,2,6,24,120**

Displaying 1-10 of 348 results found. page 1 2 3 4 5 6 7 8 9 10 ... 35

Sort: relevance | references | number | modified | created Format: long | short | data

A000142 Factorial numbers: n! = 1*2*3*4*...*n (order of symmetric group S_n, number of permutations of n letters). +20 1721

(Formerly M1675 N0659)

1, **1**, **2**, **6**, **24**, **120**, 720, 5040, 40320, 362880, 3628800, 39916800, 479001600, 6227020800, 87178291200, 1307674368000, 20922789888000, 355687428096000, 6402373705728000, 121645100408832000, 2432902008176640000, 51090942171709440000, 1124000727777607680000 (list; graph; refs; listen; history; text; internal format)

OFFSET 0,3

COMMENTS The earliest publication that discusses this sequence appears to be the Sepher Yezirah [Book of Creation], circa AD 300. (See Knuth, also the Zeilberger link) - N. J. A. Sloane, Apr 07 2014

For n >= 1, a(n) is the number of n X n (0,1) matrices with each row and column containing exactly one entry equal to 1.

This sequence is the BinomialMean transform of A000354. (See A075271 for definition.) - John W. Layman, Sep 12 2002. This is easily verified from the Paul Barry formula for A000354, by interchanging summations and using the formula: Sum_k (-1)^k C(n-i, k) = KroneckerDelta(i,n). - David Callan, Aug 31 2003

Number of distinct subsets of T(n-1) elements with 1 element A, 2 elements B,..., n - 1 elements X (e.g., at n = 5, we consider the distinct subsets of ABBCCCDDDD and there are 5! = 120). - Jon Perry, Jun 12 2003

n! is the smallest number with that prime signature. E.g., 720 = 2^4 * 3^2 * 5. - Amarnath Murthy, Jul 01 2003

a(n) is the permanent of the n X n matrix M with M(i, j) = 1. - Philippe Deléham, Dec 15 2003

Given n objects of distinct sizes (e.g., areas, volumes) such that each object is sufficiently large to simultaneously contain all previous objects, then n! is the total number of essentially different arrangements using all n objects. Arbitrary levels of nesting of objects are permitted within arrangements. (This application of the sequence was inspired by considering leftover moving boxes.) If the restriction exists that each object is able or permitted to contain at most one smaller (but possibly nested) object at a time, the resulting sequence begins 1,2,5,15,52 (Bell Numbers?). Sets of nested wooden boxes or traditional nested Russian dolls come to mind here. - Rick L. Shepherd, Jan 14 2004

From Michael Somos, Mar 04 2004; edited by M. F. Hasler, Jan 02 2015: (Start)

Stirling transform of [2, 2, 6, 24, 120, ...] is A052856 = [2, 2, 4, 14, 76,

Fatorial de um número

Fatorial de um número

Considere a função fatorial (representado por **n!**) para um número inteiro, **n,** não-negativo arbitrário

$$\text{fatorial}(n) = \begin{cases} 1 & \text{, se n=0} \\ n \times \text{fatorial}(n-1) & \text{, caso contrario} \end{cases}$$

$$\text{fatorial}(n) = \begin{cases} 1 & \text{, se n=0} \\ 1 & \text{, se n=1} \\ n \times \text{fatorial}(n-1) & \text{, caso contrario} \end{cases}$$

Fatorial de um número

```c
#include <stdio.h>

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

int main()
{
    int num;

    scanf("%d", &num);
    printf("%d\n", fatorial(num) );

    return 0;
}
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Número de vezes em que a função **Fatorial** é chamada?

Fatorial de um número

```c
#include <stdio.h>

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

int main()
{
    int num;

    scanf("%d", &num);
    printf("%d\n", fatorial(num) );

    return 0;
}
```

	N	Factorial
1	1	1
2	2	2
3	3	6
4	4	24
5	5	120
6	6	720
7	7	5040
8	8	40320
9	9	362880
10	10	3628800
11	11	39916800
12	12	479001600
13	13	6227020800
14	14	87178291200
15	15	1307674368000
16	16	20922789888000
17	17	355687428096000
18	18	6402373705728000
19	19	121645100408832000
20	20	2432902008176640000

Número de vezes em que a função **Fatorial** é chamada? **n+1**

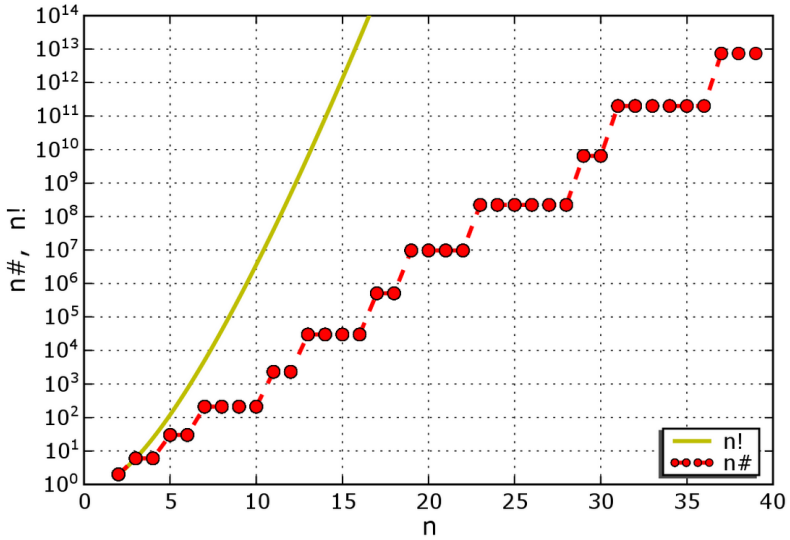
Fatorial → Primorial

O primorial de um número inteiro positivo **n** é o produto de todos os primos menores ou iguais a **n**.

- É denotado por **n#**

$$1\# = 1$$
$$2\# = 2$$
$$3\# = 2 \cdot 3 = 6$$
$$4\# = 2 \cdot 3 = 6$$
$$5\# = 2 \cdot 3 \cdot 5 = 30$$
$$6\# = 2 \cdot 3 \cdot 5 = 30$$
$$7\# = 2 \cdot 3 \cdot 5 \cdot 7 = 210$$

https://en.wikipedia.org/wiki/Primorial



- **[LAB]** Crie uma função recursiva que, dado um número inteiro positivo, devolva:
- (a) o seu Primorial.
- (b) o número de multiplicações necessárias para calcular seu primorial.

Somatório de números

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função iterativa para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \cdots + n^k$$

```
long int somatoriaIte(int n, int k) {
    long int i, soma=0;

    for (i=1; i<=n; i++)
        soma = soma + pow(i, k);

    return soma;
}
```

Um exemplo de somatória

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

long int somatoriaIte(int n, int k) {
    long int i, soma=0;

    for (i=1; i<=n; i++)
        soma = soma + pow(i, k);

    return soma;
}

int main()
{
    int num, k;

    scanf("%d %d", &num, &k);
    printf("%ld\n", somatoriaIte(num, k) );
}
```

```
$ gcc somatoria.c -lm -o somatoria.exe
$ ./somatoria.exe
4
20
1102999460754
```

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função recursiva para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \cdots + n^k$$

$$S(n,k) = \begin{cases} 0 & , \text{ se } n = 0 \\ n^k + S(n-1,k) & , \text{ se } n > 0 \end{cases}$$

Um exemplo de somatória

Dados dois número inteiros, **n** e **k**, crie uma função recursiva para calcular a seguinte somatória:

$$1^k + 2^k + 3^k + \cdots + n^k$$

$$S(n,k) = \begin{cases} 0 & , \text{ se } n = 0 \\ n^k + S(n-1,k) & , \text{ se } n > 0 \end{cases}$$

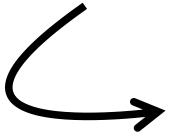
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

n=4, k=99

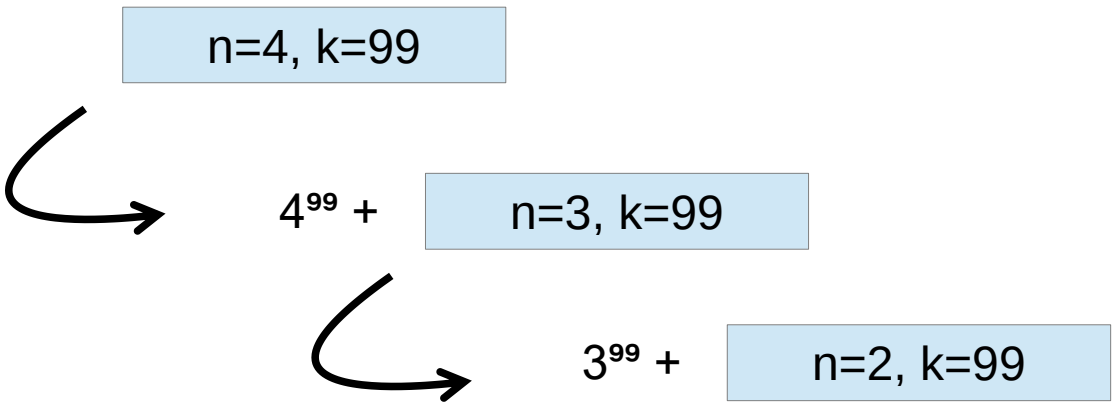
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

n=4, k=99

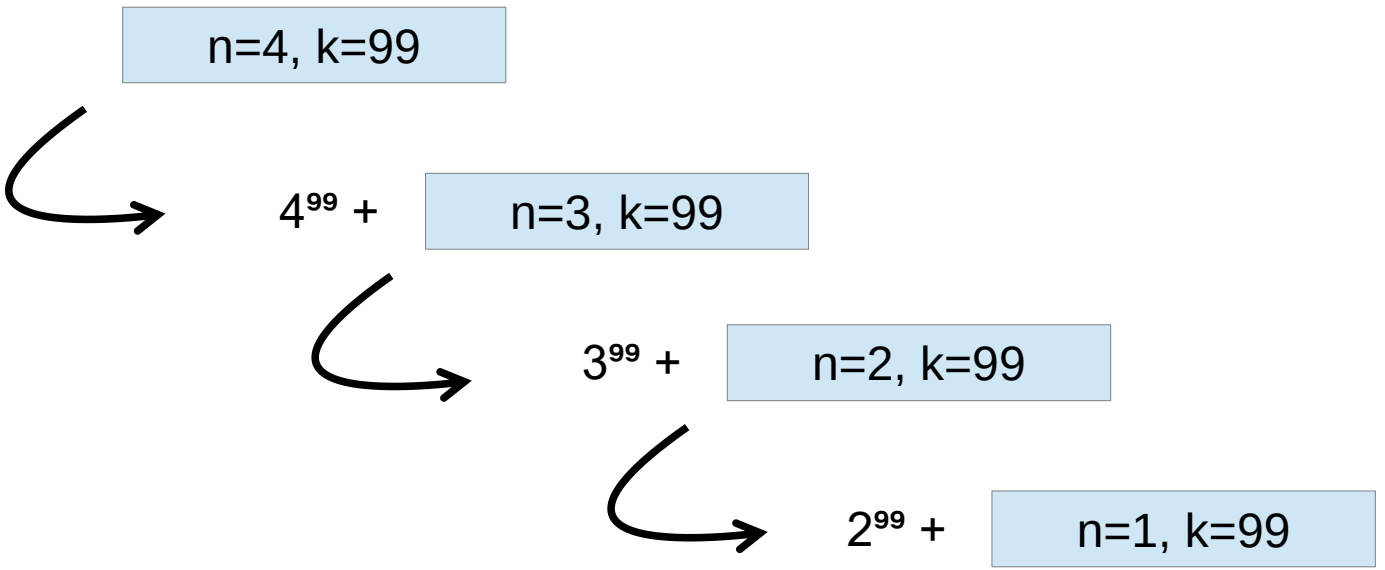


4^{99} + n=3, k=99

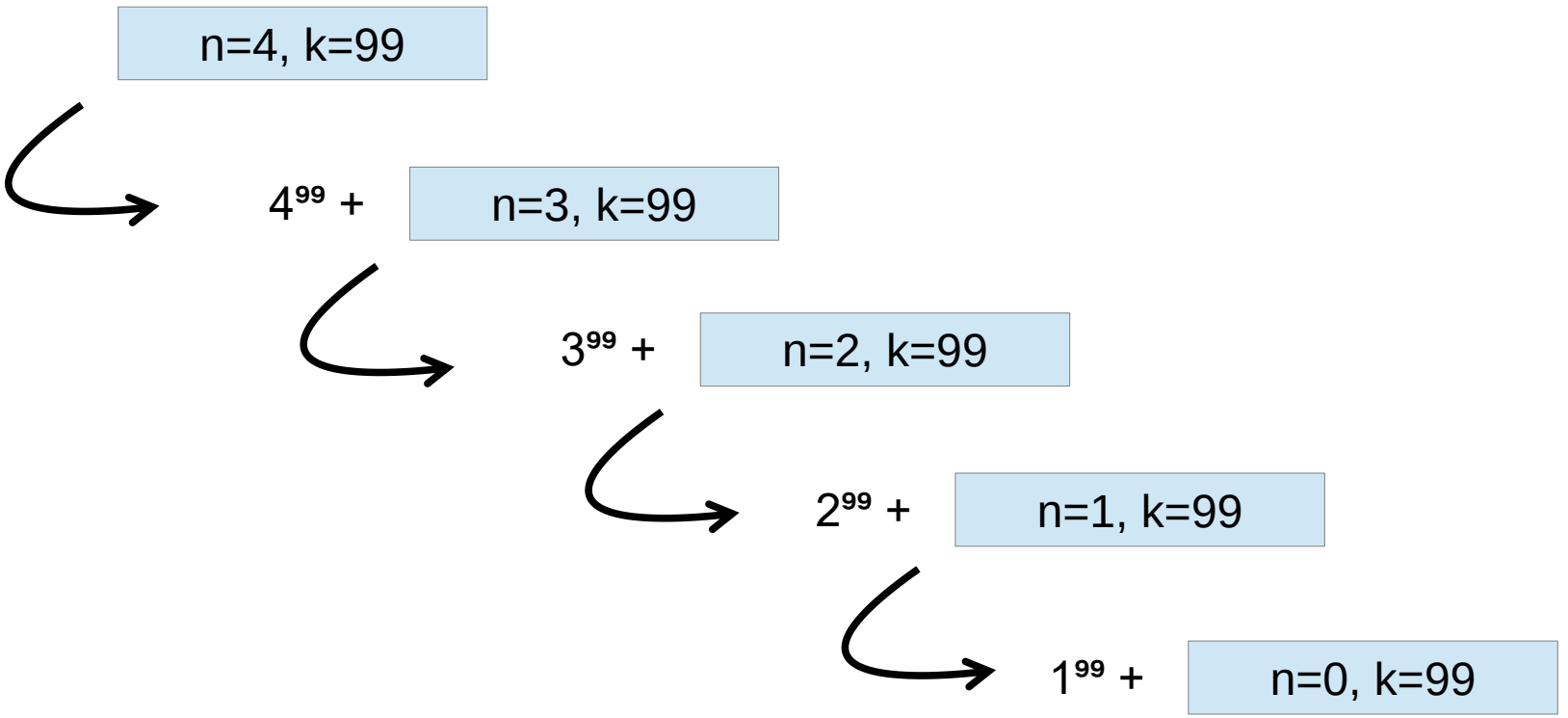
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



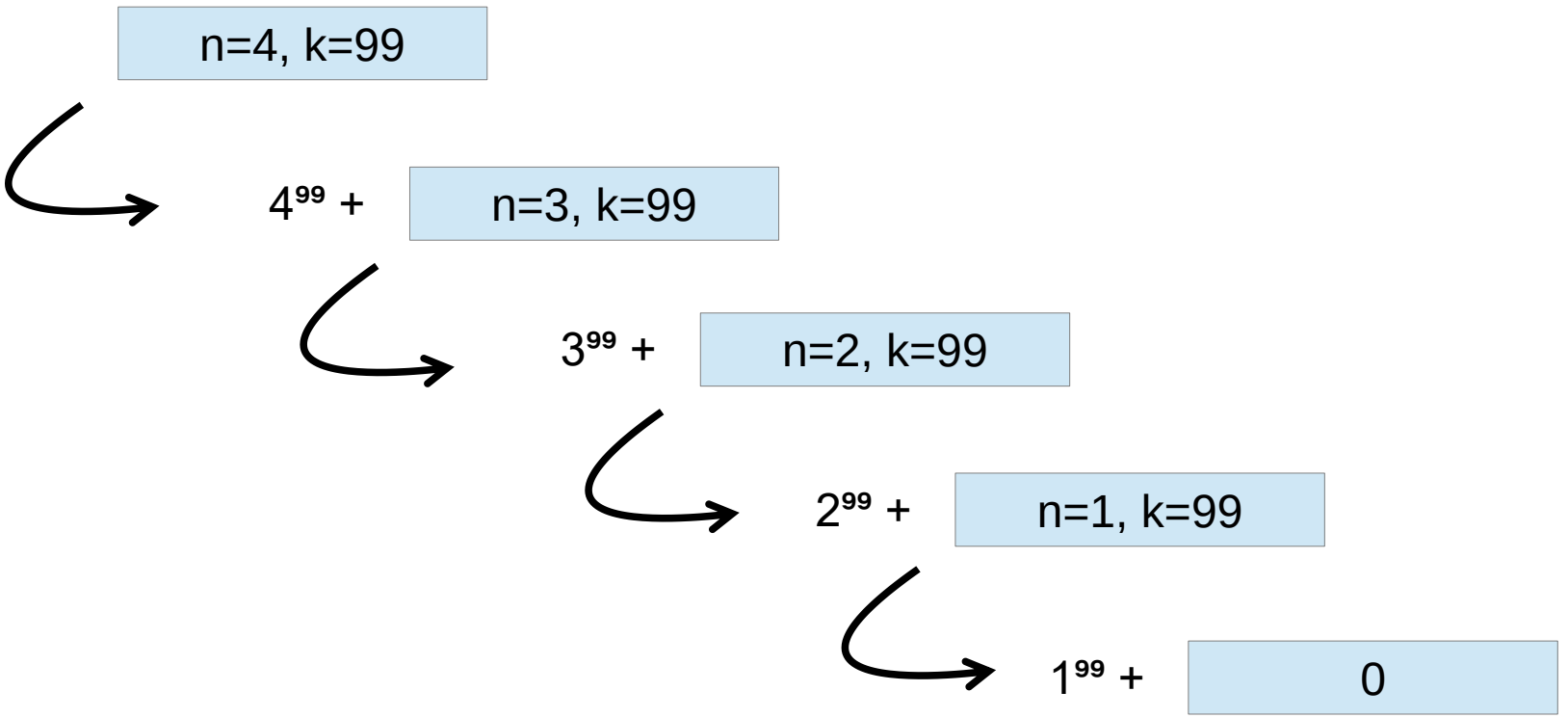
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



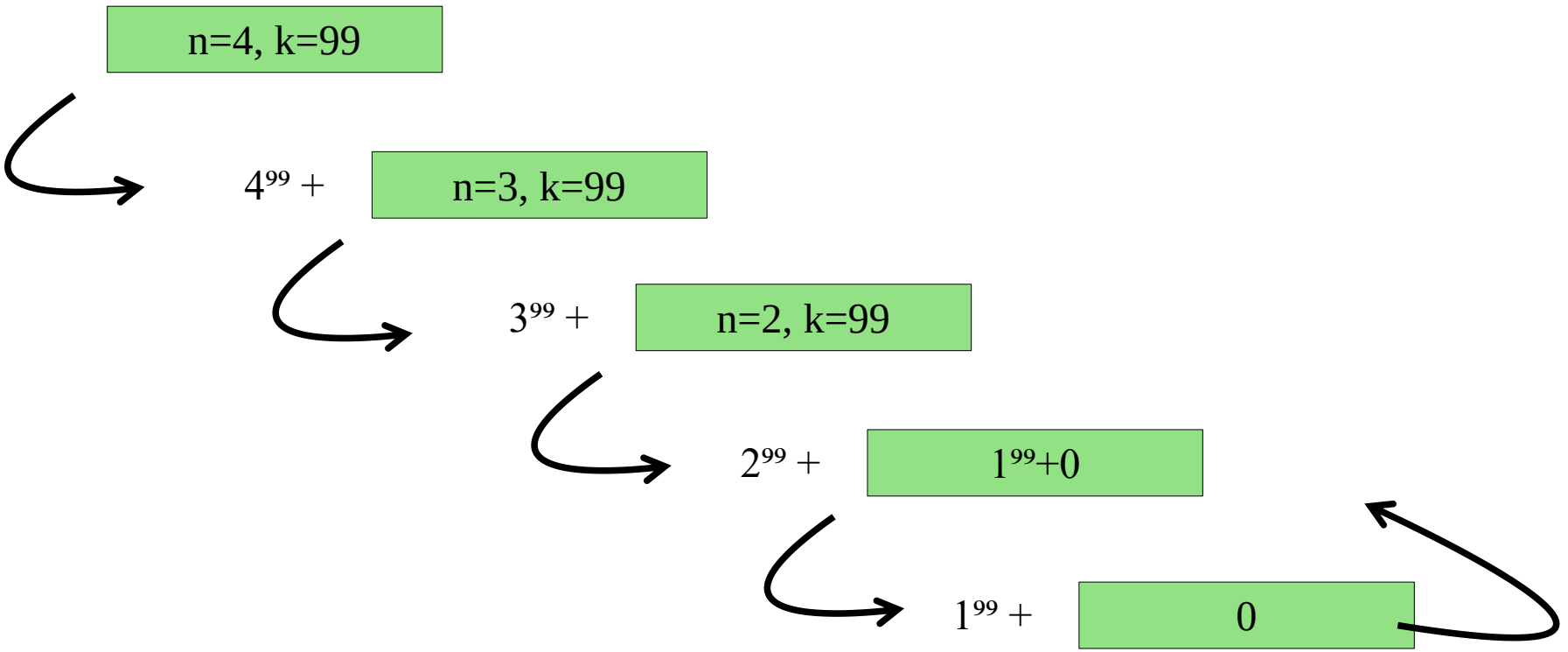
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



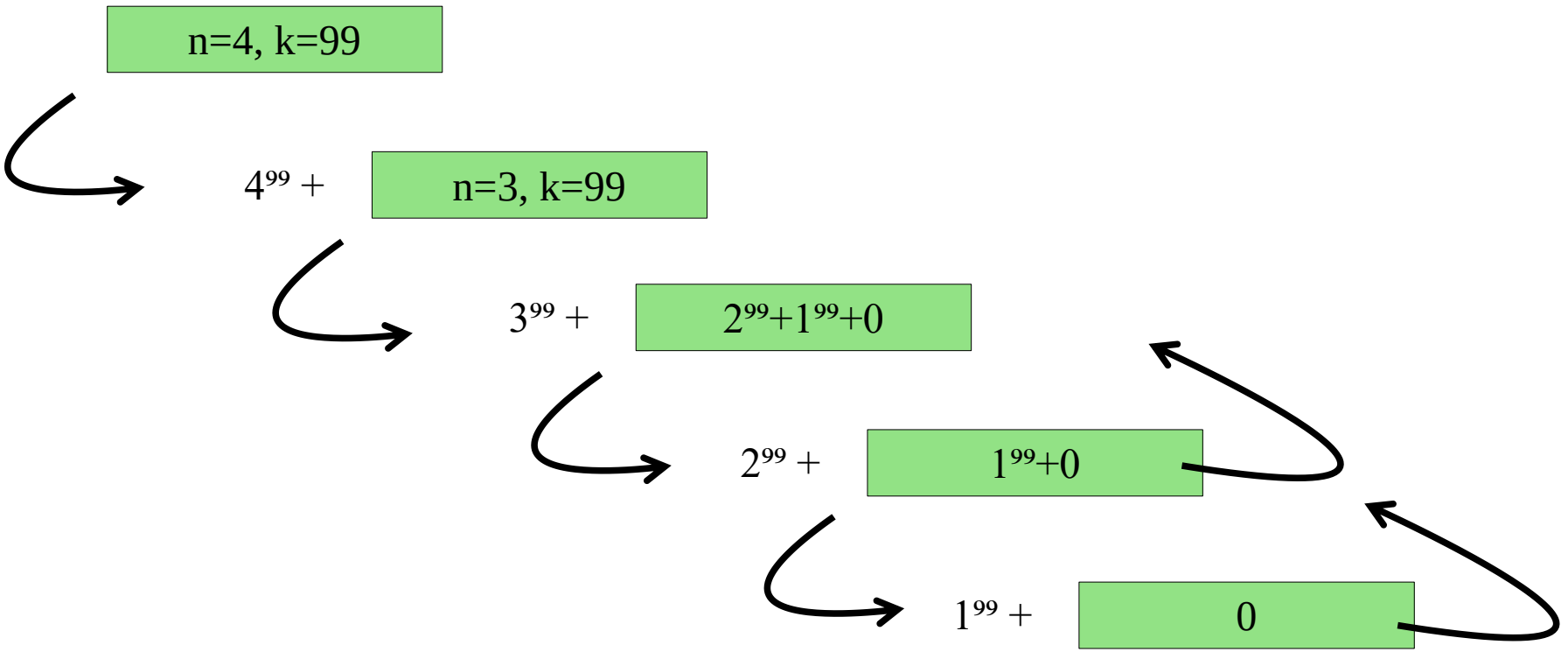
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



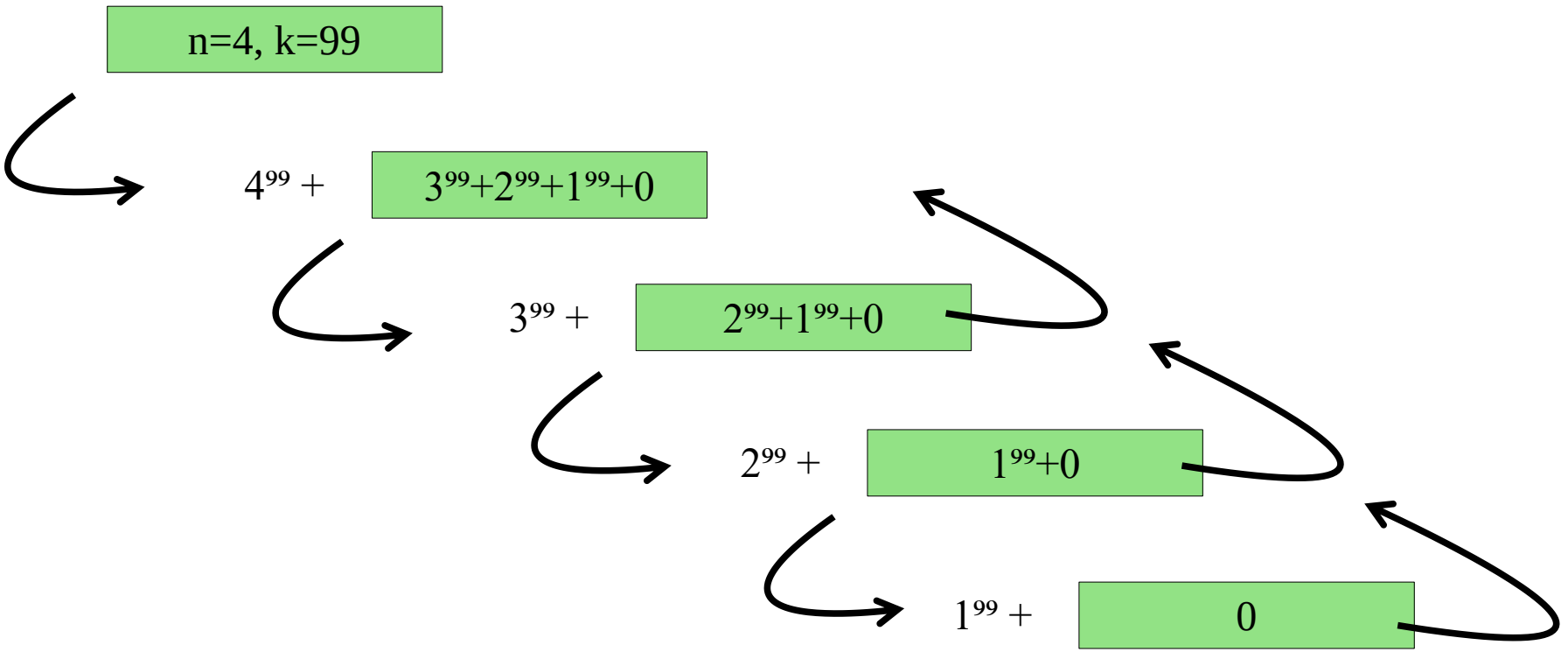
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



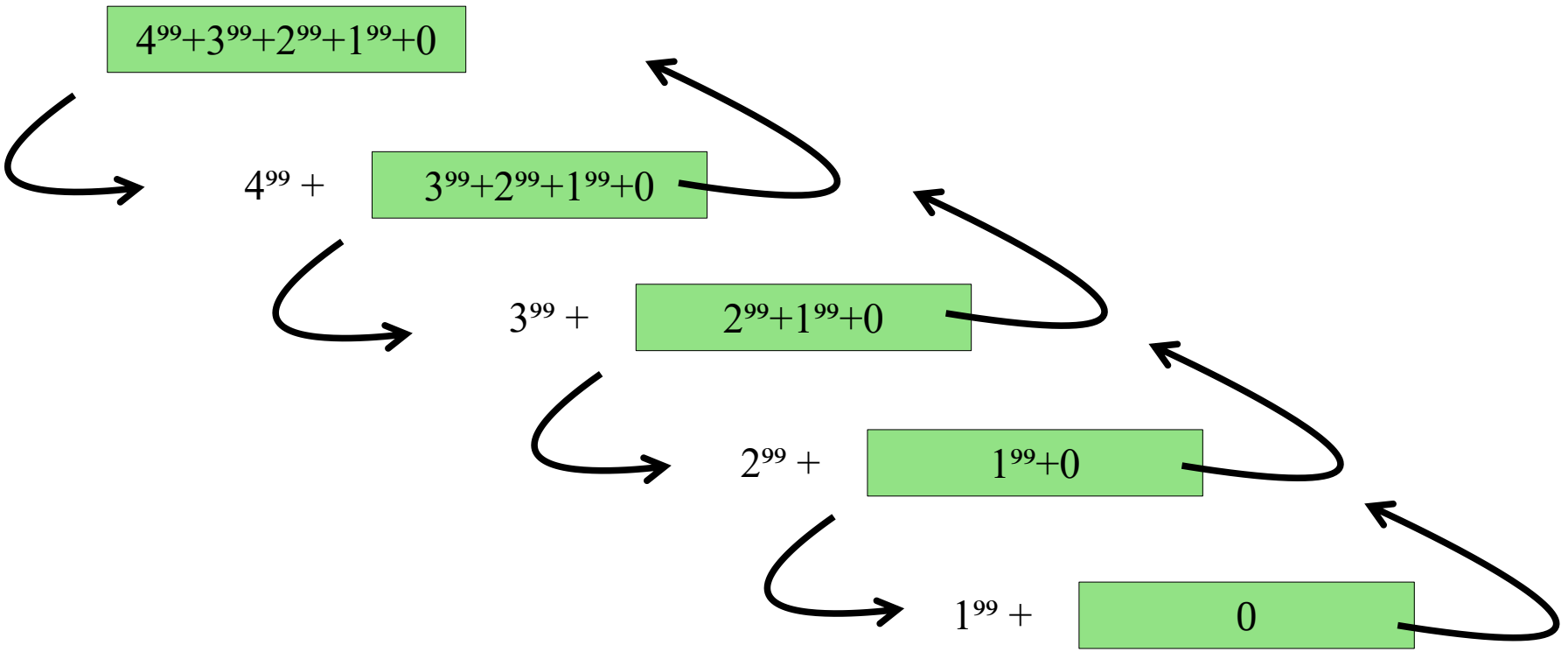
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

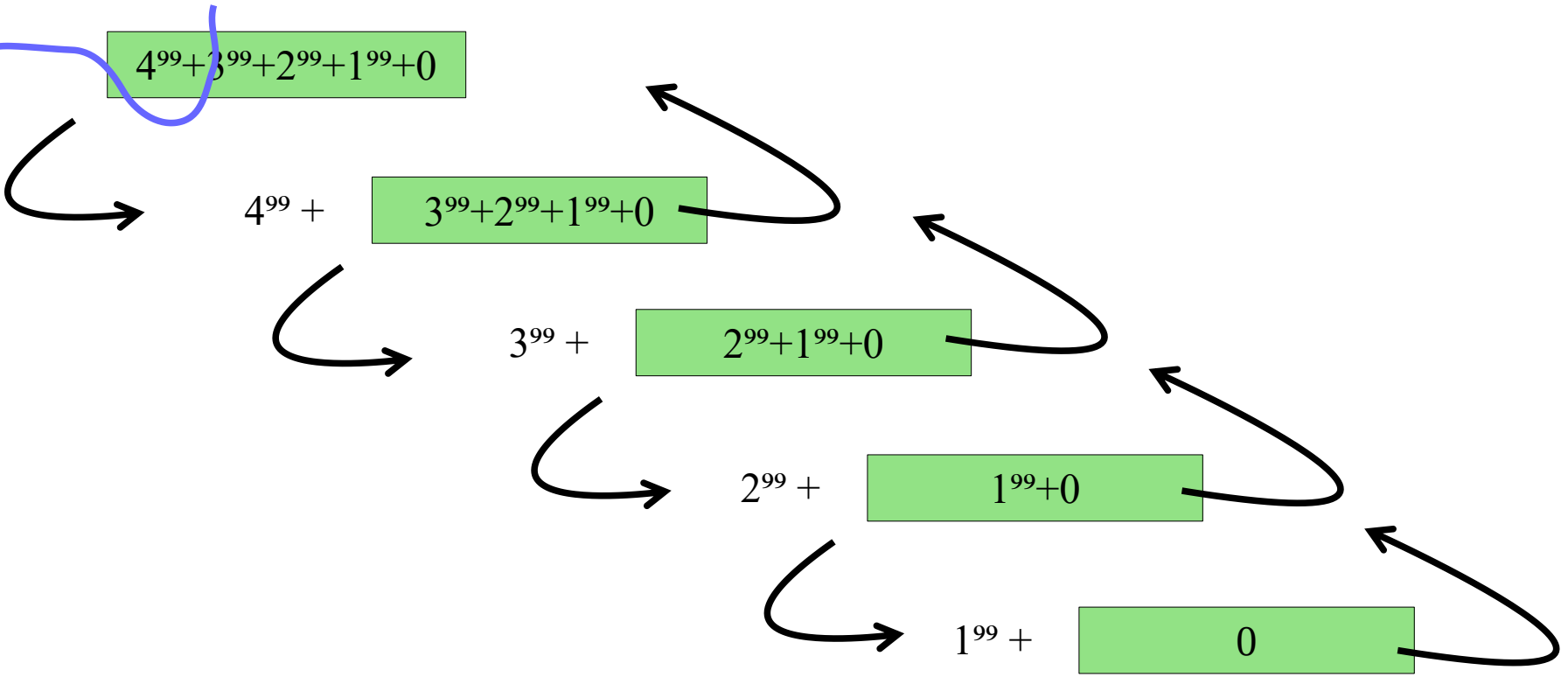


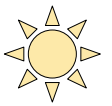
```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```



```
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

$$1^{99} + 2^{99} + 3^{99} + 4^{99}$$





```c
long int somatoriaIte(int n, int k) {
    long int i, soma=0;

    for (i=1; i<=n; i++)
        soma = soma + pow(i, k);

    return soma;
}
```

Um laço e um acumulador

```c
long int somatoriaRec(int n, int k) {
    if (n==0)
        return 0;
    else
        return pow(n, k) + somatoriaRec(n-1, k);
}
```

Sem laço e sem acumulador?

Matrioska

Matrioska

Função recursiva

- Uma função recursiva é definida em seus próprios termos.

- Toda função pode ser escrita como função recursiva sem o uso de interação (laços).

- Reciprocamente, qualquer função recursiva pode ser descrita através de interações sucessivas.

- **Ingredientes**:
- Definição de casos bases (que não envolvem recursão)
- Passos recursivos, com decremento na entrada, no sentido do caso base.

Recursão

Para resolver um tal problema é natural aplicar o seguinte método:

- **Se a instância em questão é pequena:**
- → Resolva-a diretamente
  (use força bruta se necessário)
- **Senão**
- → Reduza-a a uma instância menor do mesmo problema
- → Aplique o método à instância menor
  e volte à instância original.

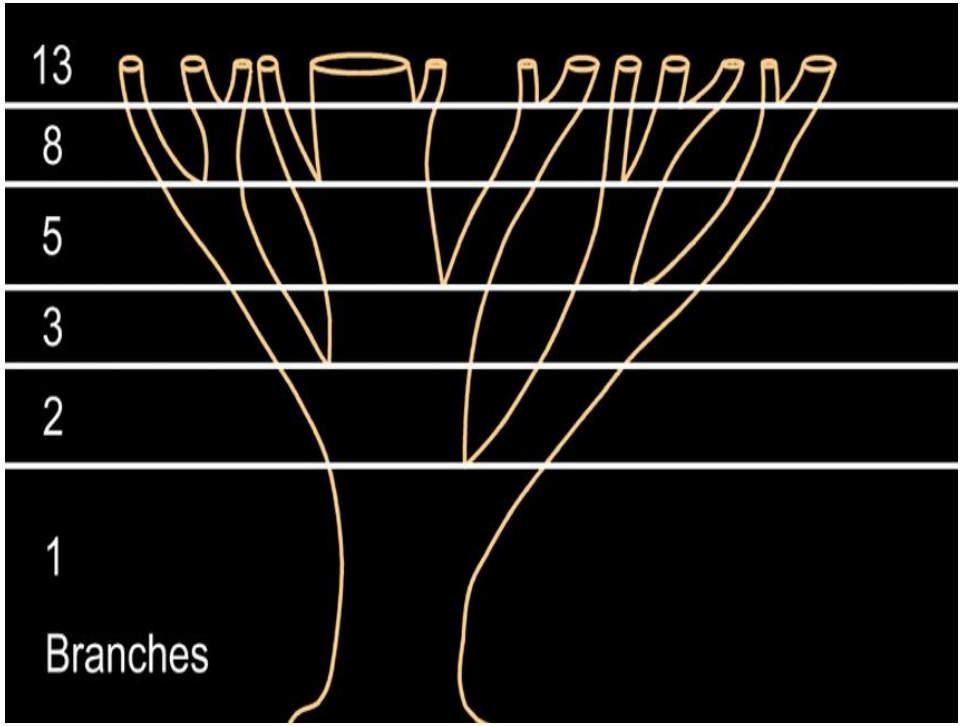
A aplicação do método produz um algoritmo recursivo.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Números de Fibonacci

Números de Fibonacci

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765



(*) fonte http://britton.disted.camosun.bc.ca/fibslide/jbfibslide.htm

Números de Fibonacci

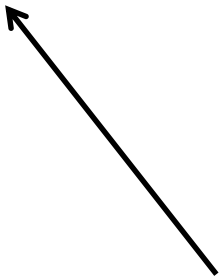
F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n-1) + Fib(n-2) & , \text{ se } n > 1 \end{cases}$$

Números de Fibonacci

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}	F_{16}	F_{17}	F_{18}	F_{19}	F_{20}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610	987	1597	2584	4181	6765

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n-1) + Fib(n-2) & , \text{ se } n > 1 \end{cases}$$



Função é duplamente recursiva
(efetua mais de uma chamada a Fib

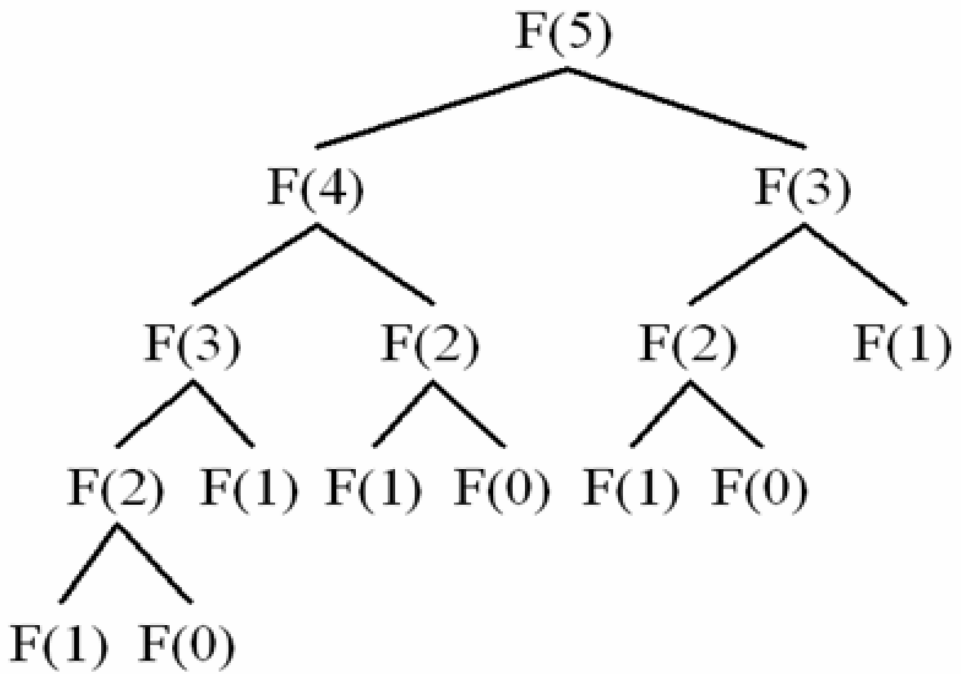
Números de Fibonacci

$$Fib(n) = \begin{cases} 0 & , \text{ se } n = 0 \\ 1 & , \text{ se } n = 1 \\ Fib(n-1) + Fib(n-2) & , \text{ se } n > 1 \end{cases}$$

```c
#include <stdio.h>

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

int main() {
    int num;
    scanf("%d", &num);
    printf("%ld\n", Fib(num));
}
```

Números de Fibonacci

Fib (5)
- Fib (4)
  - Fib (3)
    - Fib (2)
      - **Fib (1)**
      - **Fib (0)**
    - **Fib (1)**
  - Fib (2)
    - **Fib (1)**
    - **Fib (0)**
- Fib (3)
  - Fib (2)
    - **Fib (1)**
    - **Fib (0)**
  - **Fib (1)**



Pensou na eficiência da abordagem recursiva?

Números de Fibonacci

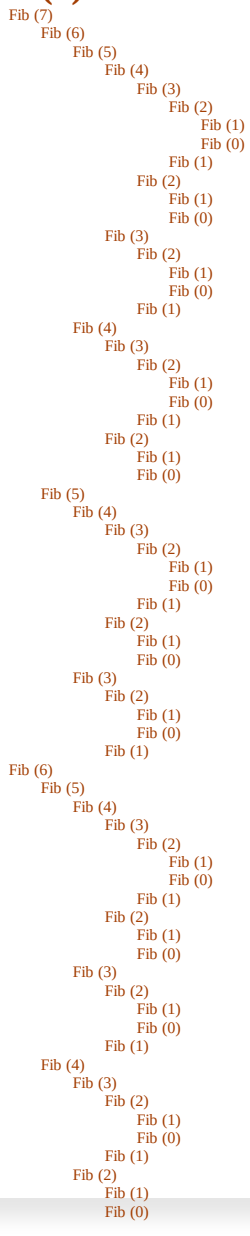
Fib (6)

- Fib (5)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)
- Fib (4)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)
  - Fib (2)
    - Fib (1)
    - Fib (0)

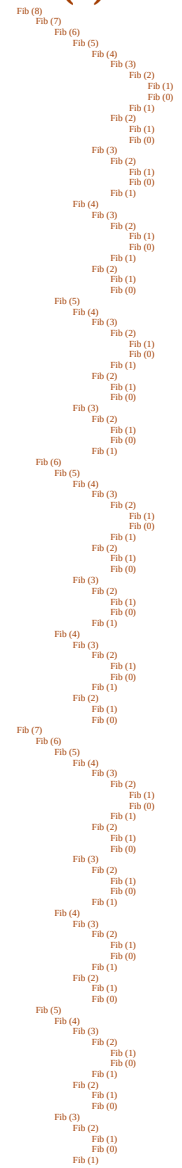
Fib (7)

- Fib (6)
  - Fib (5)
    - Fib (4)
      - Fib (3)
        - Fib (2)
          - Fib (1)
          - Fib (0)
        - Fib (1)
      - Fib (2)
        - Fib (1)
        - Fib (0)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
- Fib (5)
  - Fib (4)
    - Fib (3)
      - Fib (2)
        - Fib (1)
        - Fib (0)
      - Fib (1)
    - Fib (2)
      - Fib (1)
      - Fib (0)
  - Fib (3)
    - Fib (2)
      - Fib (1)
      - Fib (0)
    - Fib (1)

Fib (8)



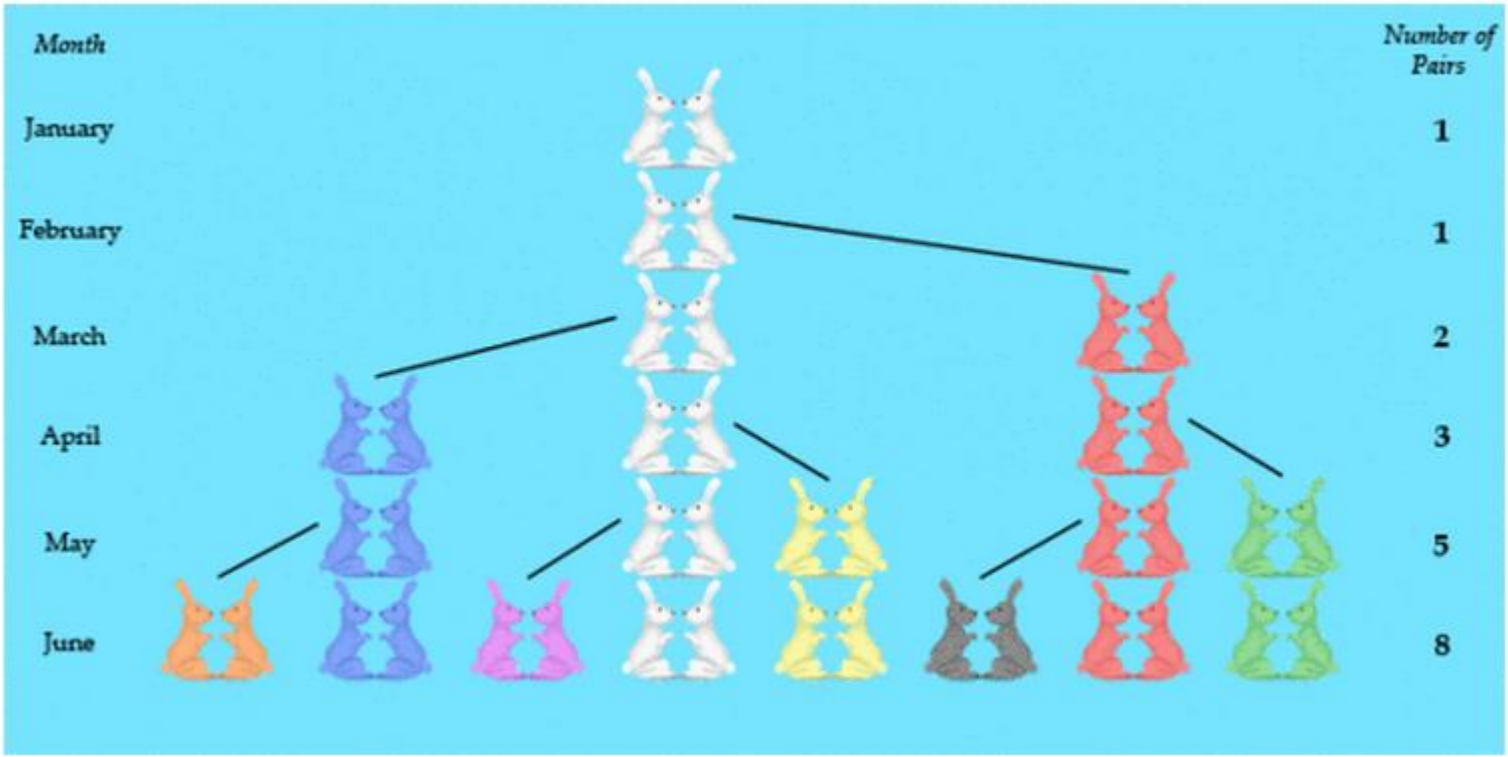
Fib (9)



Qual o número de vezes que a função Fib é chamada?

Números de Fibonacci

Os números de Fibonacci foram propostos por Leonardo di Pisa (Fibonacci), em 1202, como uma solução para o problema de determinar o tamanho da população de coelhos



(*) fonte http://www.oxfordmathcenter.com/drupal7/node/487

Número de dígitos binários

Número de dígitos binários

Crie uma função que calcula o **número** mínimo de dígitos binários para representar um número inteiro decimal positivo **n.**

Equivalent:	128	0	0	0	8	0	2	0
Power:	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value:	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
Position:	7	6	5	4	3	2	1	0

128 é representado com, no mínimo, 8 dígitos binários

Número de dígitos binários

$$Num(n) = \begin{cases} 1 & , \quad \text{se } n = 0 \text{ ou } n = 1 \\ 1 + Num(\frac{n}{2}) & , \quad \text{caso contrario} \end{cases}$$

Número de dígitos binários

$$Num(n) = \begin{cases} 1 & , \quad \text{se } n = 0 \text{ ou } n = 1 \\ 1 + Num(\frac{n}{2}) & , \quad \text{caso contrario} \end{cases}$$

```c
#include <stdio.h>

int bin(int n) {
    if (n==1 || n==0)
        return 1;
    else
        return bin(n/2) + 1;
}

int main() {
    int num;
    scanf("%d", &num);
    printf("%d\n", bin(num));
}
```

Número de vezes em que a função **bin** é chamada?

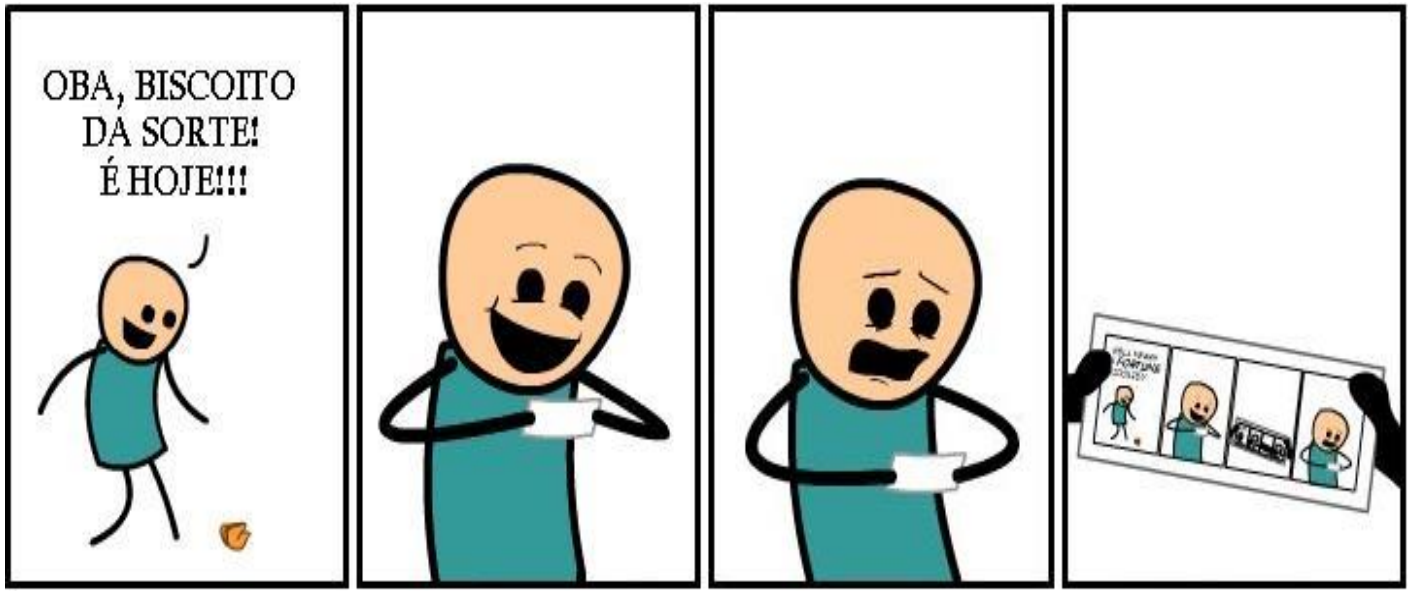
Número de dígitos binários

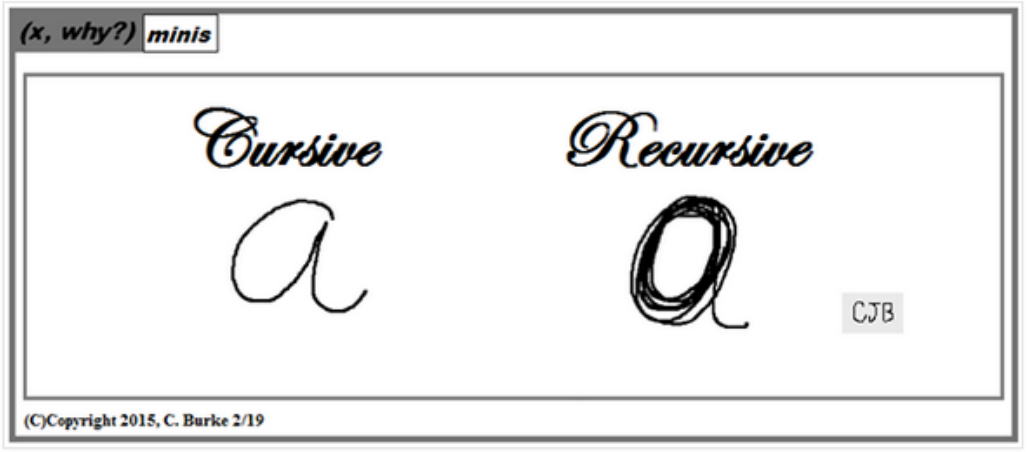
$$Num(n) = \begin{cases} 1 & , \quad \text{se } n = 0 \text{ ou } n = 1 \\ 1 + Num(\frac{n}{2}) & , \quad \text{caso contrario} \end{cases}$$

```c
#include <stdio.h>

int bin(int n) {
    if (n==1 || n==0)
        return 1;
    else
        return bin(n/2) + 1;
}

int main() {
    int num;
    scanf("%d", &num);
    printf("%d\n", bin(num));
}
```

Número de vezes em que a função **bin** é chamada?

Recursividade é uma das coisas mágicas e interessantes em Lógica de Programação





Atividade em aula

Questão 1

- Qual é o resultado da execução das seguintes funções para n=5?

```
void conta1(int n) {
    if (n>0) {
        printf("\n%d", n);
        conta1(n-1);
    }
}
```

```
void conta2(int n) {
    if (n>0) {
        conta2(n-1);
        printf("\n%d", n);
    }
}
```

Resposta de conta1 para n=5

5
4
3
2
1

Resposta de conta2 para n=5

1
2
3
4
5

Questão 2

- Indique o que devolvem as funções F1 e F2 para valores de: a=2, k=5.
- Construa sua representação matemática recursiva.
- Descreva em português o que calcula cada função.

```
int f1(int a, int k) {
    if (k==1)
        return a;
    else
        return a*f1(a, k-1);
}
```

```
int f2(int a, int k) {
    int x;

    if (k==1)
        return a;
    else {
        x = f2(a, k/2);
        if (k%2==0)
            return x*x;
        else
            return x*x*a;
    }
}
```

$$F1(a,k) = \begin{cases} a & , \text{ se } k = 1 \\ a \times F1(a, k-1) & , \text{ caso contrario} \end{cases}$$

$$F2(a,k) = \begin{cases} a & , \text{ se } k = 1 \\ F2(a, \frac{k}{2})^2 & , \text{ se } k \text{ é par} \\ a \times F2(a, \frac{k}{2})^2 & , \text{ se } k \text{ é impar} \end{cases}$$

Ambas as funções calculam a^k .
Para a=2, e b=5 a função devolve 32.

F2 é mais eficiente!

Questão 3

- Descreva em português o que calcula a função M.
- Qual o número total de comparações?

```
1  #include <stdio.h>
2
3  int M(int v[], int n) {
4      if (n==1)
5          return v[0];
6      else {
7          int x;
8          x = M(v, n-1);
9          if (x <= v[n-1])
10             return x;
11         else
12             return v[n-1];
13     }
14 }
15
16 int main() {
17     int v[] = {1,2,3,4,5,6,7,-5,100};
18     printf("%d\n", M(v,9));
19 }
```

A função M, dada um vetor v[0,…,n-1] de n elementos devolve o menor valor contido em v.

Número de comparações = n

Questão 4

- Escreva uma versão iterativa da função M.

```
int MIter(int v[], int n) {
    int i;
    int min = v[0];
    for (i=1; i<n; i++){
        if (min>v[i])
            min = v[i];
    }
    return min;
}
```

Versão iterativa

Recursão

- Uma função recursiva é aquela que se chama a si mesma (obrigatoriamente)?

```c
#include <stdio.h>

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

int main() {
    int num;
    scanf("%d", &num);
    printf("%ld\n", Fib(num));
}
```

Recursão

Uma função recursiva não necessariamente é aquela que se chama a si mesma

```
int F1(parametros)
{
   .
   .
   .
   F2(parametros)
   .
   .
}
```

```
int F2(parametros)
{
   .
   .
   .
   F1(parametros)
   .
   .
}
```

F1(...) → F2(...) → F3(...) → …. → F1(….)

Análise de algoritmos recursivos

- Para fazer a análise é necessário obter a relação de recorrência.
- A fórmula fechada (resolução) da recorrência pode ser obtida por diferentes abordagens, por exemplo:
- Árvore de recorrência
- Método mestre
- Funções geradoras