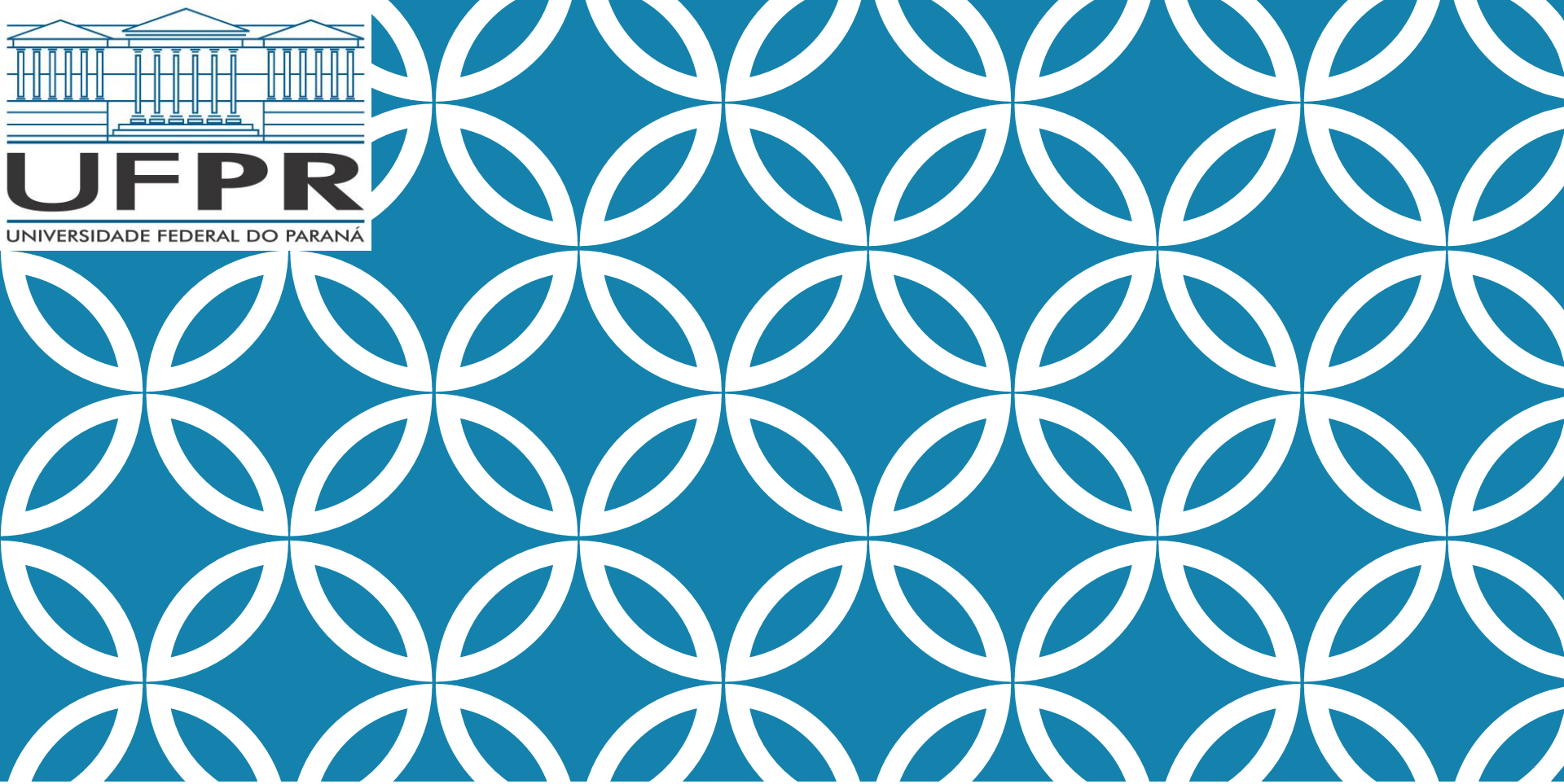


INTRODUÇÃO À RECURSÃO

Algoritmos e Estrutura de Dados II

Prof. André Vignatti

FATORIAL

Já sei (magicamente) o valor de $26!$. Como calcular $27!$?

$$27! = 27 \times 26!$$

Já sei (magicamente) o valor de $52!$. Como calcular $53!$?

$$53! = 53 \times 52!$$

Já sei (magicamente) o valor de $(n-1)!$. Como calcular $n!$?

$$n! = n \times (n-1)!$$

FATORIAL

chamando $f(n) = n!$, então

$$f(n) = n \times f(n-1)$$

o que há de "estranho" na função acima?

"é a primeira vez que vejo uma função definida em termos dela mesma!"

- de fato, isso é estranho... geralmente vemos algo como $f(n) = n^2$ ou $f(n) = e^n$

basta "plugar" o valor de n desejado, e podemos calcular a função

FATORIAL

$$f(n) = n \times f(n-1)$$

"como raciocinar com uma função definida em termos dela mesma?"

$$\begin{aligned} f(n) &= n \times f(n-1) \\ &= n \times (\overbrace{(n-1) \times f(n-2)}) \\ &= n \times (n-1) \times f(n-2) \\ &= n \times (n-1) \times (\overbrace{(n-2) \times f(n-3)}) \\ &= n \times (n-1) \times (n-2) \times f(n-3) \end{aligned}$$

o fatorial está se formando!!!!!

FATORIAL

$$f(n) = n \times f(n-1)$$

"tem um problema: isso nunca para!!"

vamos corrigir:

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

testar para $f(4)$

FATORIAL

uma função matemática recebe uma **entrada,** e devolve uma **saída**

uma função em linguagem de programação faz a mesma coisa...

seria possível usar essa ideia em linguagens de programação?

SIM!!!!

em linguagem C:

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

em linguagem Pascal:

```pascal
function f(n: integer): integer;
begin
  if n = 0 then
     f := 1
  else
     f := n * f(n-1);
end;
```

RECURSÃO

definir uma função (matemática ou computacional) a partir dela mesma é chamado de **recursão**

ALG-2: (quase sempre) somente algoritmos recursivos!

excepcionalmente, nesta aula: veremos *códigos reais*, e não pseudo-códigos

senão, é difícil acreditar concretamente que algoritmos recursivos funcionam...

RECURSÃO

Definição. *Uma função definida recursivamente é chamada de* função recursiva *ou relação de recorrência. Tal função sempre tem dois elementos:*

1. ***base**: o(s) caso(s) computados sem necessidade de recursão.*
2. ***recursão**: computação recursiva para os casos que não são base.*

uma vantagem: pouco esforço para traduzir a função recursiva matemática na função recursiva em código de computador

EXEMPLO

Exemplo. *Defina x^n como uma função (matemática) recursiva e em seguida traduza para código de computador:*

$$x^n = \begin{cases} 1 & se\ n = 0 \\ x \cdot x^{n-1} & se\ n > 0. \end{cases}$$

```
float potencia (float x, int n) {
  if (n==0) return 1.0;
  return x * potencia (x,n-1);
}
```

compare com a versão não recursiva:

```
float potencia_nao_recursiva (float x, int n) {
  if (n == 0)
      return 1;

  float result = x;

  while (n>1) {
     result = result * x;
     n = n - 1;
  }
  return result;
}
```

VANTAGENS DE RECURSÃO

código recursivos tem algumas vantagens:

- é mais intuitivo: parece com a definição matemática
- são menores
- aumenta a legibilidade, simplifica a codificação

UM PASSO A PASSO PARA RECURSÃO

O passo-a-passo para escrever um código recursivo é:

1. Escreva o protótipo da sua função: definir as entradas e saídas.
2. Escreva o código para o caso base.
3. Tome um exemplo concreto e responda: como usar o valor já calculado de um exemplo mais próxima da base para calcular o valor do exemplo mais distante da base?
4. Generalize o passo anterior para escrever o código da parte recursiva.

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

1. *Protótipo:* `void count_down (int n)`

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

1. *Protótipo:* `void count_down (int n)`

2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

1. *Protótipo:* `void count_down (int n)`

2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down(8)`. *Ideia: imprime* 8 *e chama* `count_down(7)`.

Exemplo (Contagem regressiva)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, n-2, \ldots, 3, 2, 1$.

1. *Protótipo:* `void count_down (int n)`

2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down(8)`. *Ideia: imprime* 8 *e chama* `count_down(7)`.

4. *Código da parte recursiva:*

```
   else {
      printf("%d, ", n);
      count_down (n - 1);
   }
}
```

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$.

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$.

1. *Protótipo:* `void count_down_up (int n)`

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$*.*

1. *Protótipo:* `void count_down_up (int n)`
2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$*.*

1. *Protótipo:* `void count_down_up (int n)`
2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down_up(4)`*. Ideia: imprime* 4*, chama* `count_down_up(3)`*, imprime* 4*.*

Exemplo (Contagem pra baixo e pra cima)**.** *Faça um programa recursivo que recebe um inteiro* $n \geq 1$ *e imprime* $n, n-1, \ldots, 1, \ldots, n-1, n$*.*

1. *Protótipo:* `void count_down_up (int n)`

2. *Código do caso base:*

```
{
   if (n == 1)
   printf("%d", n);
```

3. *Exemplo concreto:* `count_down_up(4)`*. Ideia: imprime* 4*, chama* `count_down_up(3)`*, imprime* 4*.*

4. *Código da parte recursiva:*

```
   else {
      printf("%d, ", n);
      count_down_up (n - 1);
      printf("%d, ", n);
   }
}
```

FIBONACCI

a sequência de Fibonacci são os números na seguinte sequência:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

por definição:

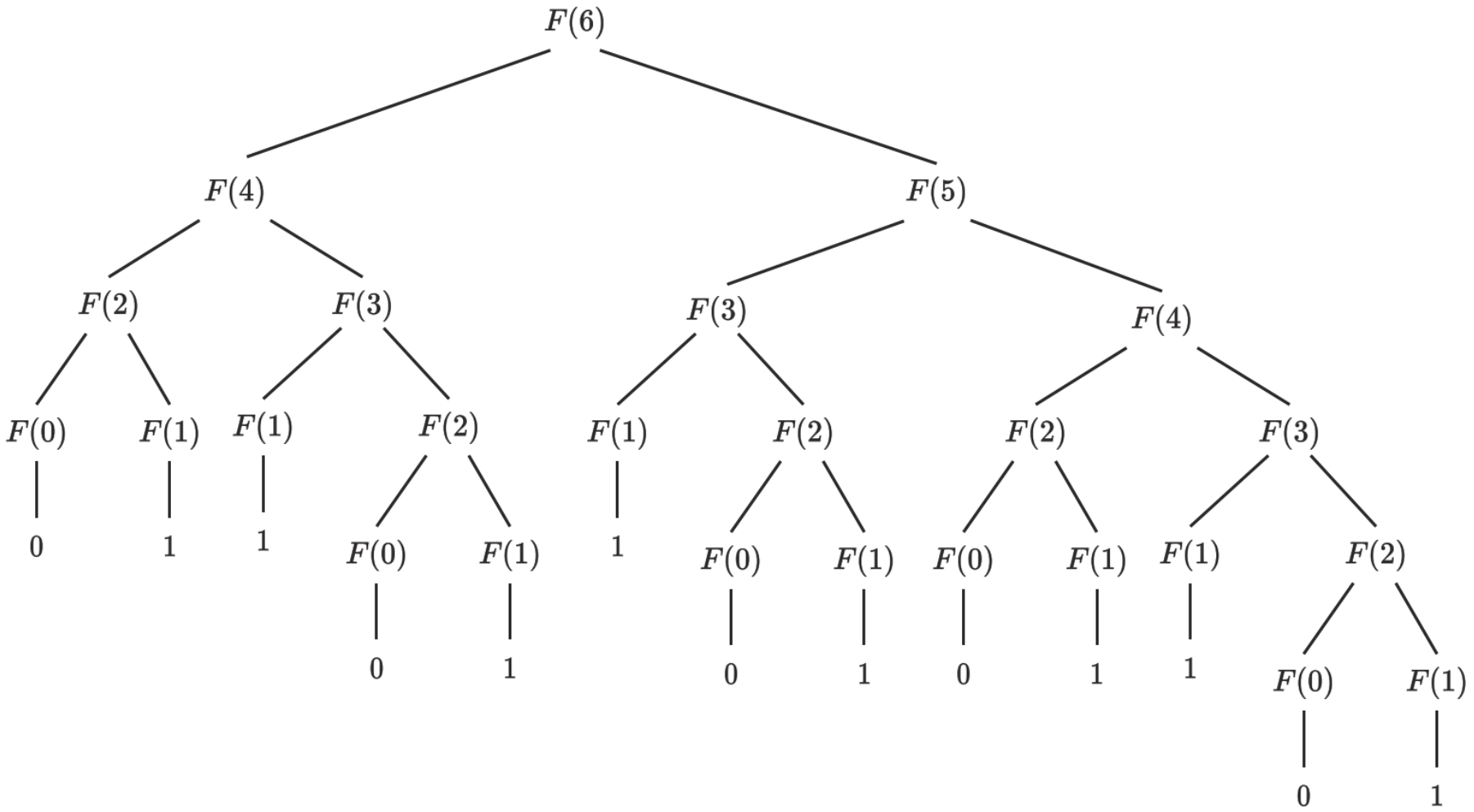
- os primeiros dois números são 0 e 1
- cada número subsequente é a soma dos dois anteriores

como calcular o n-ésimo número da sequência de Fibonacci?

$$F(n) = \begin{cases} n & \text{se } n \leq 1 \\ F(n-1) + F(n-2) & \text{se } n > 1 \end{cases}$$

escreva o código recursivo e iterativo (não-recursivo) que calcula $F(n)$

árvore de chamadas recursivas de $F(n)$



n	$F(n+1)$	num. de adições	num. chamadas de função
6	13	12	25
10	89	88	177
15	987	986	1973
20	10946	10945	21891
25	121393	121392	242785
30	1346269	1346268	2692537

Teorema. *O número de adições que a versão recursiva de $F(n)$ executa é igual a $F(n+1)-1$.*

- uma **desvantagem** da recursão: chamada de função é custosa