Estrutura de dados I

Recursão

Prof. Rodrigo Minetto

Universidade Tecnológica Federal do Paraná

Sumário

Introdução

Muitos problemas têm a seguinte propriedade: cada instância do problema contém uma instância menor do mesmo problema (estrutura recursiva). Em uma definição recursiva um item é definido em termos de si mesmo (aparece como parte da definição).

Introdução

Motivos comuns para usar recursão:

- O problema é naturalmente recursivo
- Os dados tem estrutura recursiva

No entanto, **toda** solução recursiva tem uma solução iterativa. Embora, versões iterativas para certos problemas sejam menos elegantes, mais difíceis de implementar, compreender e provar que algoritmos recursivos.

Estrutura

Algoritmos recursivos são divididos em:

- **caso base**: menor instância do problema (caso mais simples), que não pode ser mais decomposto. O caso base geralmente corresponde ao vazio (lista, conjunto, árvore vazia).

RECURSÃO (**argumentos**)

1. **se** **caso base** **então**
2. Determine a solução (sem recursão);
3. **senão**
4. **Divida** o problema em subproblemas
5. **Invoque** a função **recursivamente**
6. **Reorganize** as soluções

Estrutura

Algoritmos recursivos são divididos em:

- **passo recursivo**: que decompõe instâncias maiores do problema em menores (mais simples), caso contrário a recursão pode nunca terminar.

RECURSÃO (**argumentos**)

1. **se** **caso base** **então**
2. Determine a solução (sem recursão);
3. **senão**
4. **Divida** o problema em subproblemas
5. **Invoque** a função **recursivamente**
6. **Reorganize** as soluções

Sumário

Fatorial

O fatorial de um número inteiro n (não negativo), definido como $n!$, é o produto de todos os inteiros positivos menores ou iguais a n

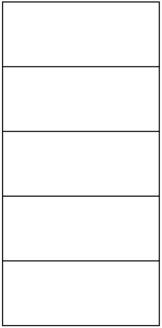
$$n! = n \times (n-1)!$$

$$n! = n \times n - 1 \times n - 2 \times \ldots 3 \times 2 \times 1$$

A sequência de fatoriais para $\mathbf{n = 0,1,2,3,\ldots}$ é dada por $\mathbf{1,1,2,6,\ \ldots}$

Recursão: fatorial

```
FATORIAL (n)
se n = 0 então
    return 1;
return n * FATORIAL (n − 1);
```



pilha
(recursão)
(abstração)

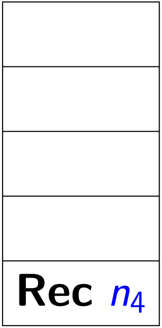
Recursão: fatorial

▷ $\textsc{Fatorial}\ (n = 4)$

```
se n = 0 então
    return 1;
return n * FATORIAL (n − 1);
```

n_4

Fatorial (4)



pilha
(recursão)
(abstração)

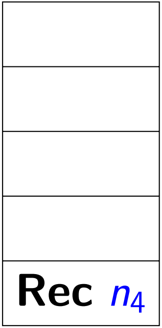
Recursão: fatorial

FATORIAL ($n = 4$)

▷ 1. **se** **n** $= 0$ **então**
2. **return** 1;
3. **return** **n** * FATORIAL (**n** $- 1$);

n_4

Fatorial (4)



pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 4$)

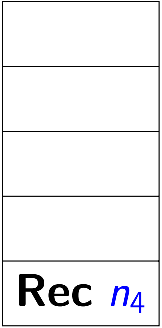
---

1. **se** **n** $= 0$ **então**
2.   **return** 1;

▷ 3. **return** **n** * FATORIAL (**n** $- 1$);

n_4

Fatorial (4)



pilha
(recursão)
(abstração)

Recursão: fatorial

$\triangleright$ FATORIAL ($n = 3$)

```
se n = 0 então
    return 1;
return n * FATORIAL (n − 1);
```

n_4 n_3

4 * Fatorial (3)

Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

Fatorial ($n = 3$)

1. **se n** $= 0$ **então**
2. **return** 1;
3. **return n** * Fatorial (**n** $-$ 1);

n_4 n_3

4 * Fatorial (3)

Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 3$)

1. **se** **n** $= 0$ **então**
2. **return** 1;

▷ 3. **return** **n** * FATORIAL (**n** − 1);

n_4 n_3

4 * Fatorial (3)

Rec n_3
Rec n_4

pilha (recursão) (abstração)

Recursão: fatorial

▷ FATORIAL ($n = 2$)

---

se $\mathbf{n} = 0$ **então**
    return 1;
return $\mathbf{n}$ * FATORIAL ($\mathbf{n} - 1$);

n_4 n_3 n_2

4 * (3 * Fatorial (2))

Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 2$)

```
▷ se n = 0 então
      return 1;
  return n * FATORIAL (n − 1);
```

n_4 n_3 n_2

4 * (3 * Fatorial (2))

Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 2$)

---

se **n** $= 0$ **então**
    return 1;
▷ **return** **n** * FATORIAL (**n** $- 1$);

n_4 n_3 n_2

4 * (3 * Fatorial (2))

Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

▷ FATORIAL ($n = 1$)

1. **se n** $= 0$ **então**
2. **return** 1;
3. **return n** * FATORIAL (**n** $- 1$);

n_4 n_3 n_2 n_1

4 * (3 * (2 * Fatorial (1)))

Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

$\textsc{Fatorial}\ (n = 1)$

$\triangleright$	1.	**se** **n** $= 0$ **então**
	2.	**return** 1;
	3.	**return** **n** * $\textsc{Fatorial}\ (\mathbf{n} - 1)$;

$$\underset{n_4}{\mathbf{4}} * (\underset{n_3}{\mathbf{3}} * (\underset{n_2}{\mathbf{2}} * \textbf{Fatorial}\ (\underset{n_1}{\mathbf{1}})))$$

Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 1$)

```
   se n = 0 então
       return 1;
▷  return n * FATORIAL (n − 1);
```

$$\overset{n_4}{4} * (\overset{n_3}{3} * (\overset{n_2}{2} * \text{Fatorial } (\overset{n_1}{1})))$$

Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

▷ FATORIAL $(n = 0)$

1. **se n** $= 0$ **então**
2. **return** 1;
3. **return n** $*$ FATORIAL $(\mathbf{n} - 1)$;

$n_4 \quad n_3 \quad n_2 \quad n_1 \quad n_0$

4 * (3 * (2 * (1 * Fatorial (0))))

Rec n_0
Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL $(n = 0)$

```
▷ se n = 0 então
      return 1;
  return n * FATORIAL (n − 1);
```

$$\overset{n_4}{4} * (\overset{n_3}{3} * (\overset{n_2}{2} * (\overset{n_1}{1} * \textbf{Fatorial}\ (\overset{n_0}{0}))))$$

Rec n_0
Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha (recursão) (abstração)

Recursão: fatorial

FATORIAL ($n = 0$)

1. **se n** $= 0$ **então**
2. ▷ **return** 1;
3. **return n** * FATORIAL (**n** $- 1$);

n_4 n_3 n_2 n_1 n_0

4 * (3 * (2 * (1 * Fatorial (0))))

Rec n_0
Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL $(n = 0)$

```
    se n = 0 então
▷       return 1;
    return n * FATORIAL (n − 1);
```

$n_4 \quad n_3 \quad n_2 \quad n_1 \quad n_0$

$$\mathbf{4 * (3 * (2 * (1 * 1)))}$$

Rec n_0
Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL $(n = 1)$

1. **se n** $= 0$ **então**
2. **return** 1;

$\triangleright$ 3. **return n** $*$ FATORIAL $(\mathbf{n} - 1)$;

$$\begin{array}{ccccccc} n_4 & & n_3 & & n_2 & & n_1 \\ \mathbf{4} & * & (\mathbf{3} & * & (\mathbf{2} & * & (\mathbf{1} * \mathbf{1}))) \end{array}$$

Rec n_1
Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 2$)

1. **se** **n** $= 0$ **então**
2. **return** 1;

▷ 3. **return** **n** * FATORIAL (**n** $- 1$);

n_4 n_3 n_2

$\mathbf{4 * (3 * (2 * 1))}$

Rec n_2
Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 3$)

---

se $\mathbf{n} = 0$ **então**
    return 1;
▷ **return** $\mathbf{n}$ * FATORIAL ($\mathbf{n} - 1$);

$$\underset{\mathbf{4}}{n_4} * (\underset{\mathbf{3}}{n_3} * \mathbf{2})$$

Rec n_3
Rec n_4

pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 4$)

```
   se n = 0 então
       return 1;
▷  return n * FATORIAL (n − 1);
```

n_4

$4 * 6$

Rec n_4

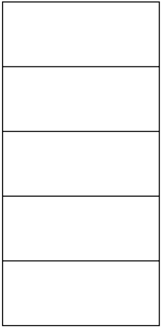
pilha
(recursão)
(abstração)

Recursão: fatorial

FATORIAL ($n = 4$)

```
se n = 0 então
    return 1;
return n * FATORIAL (n − 1);
```

24

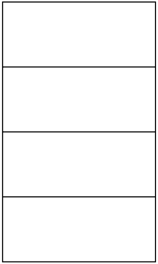


pilha (recursão) (abstração)

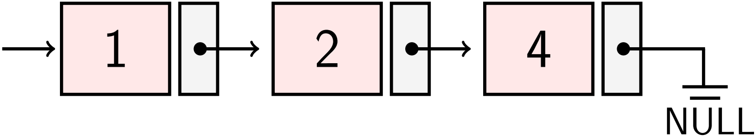
Sumário

Recursão em listas: imprimir

```
PRINT (List *ℓ)
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
    PRINT (ℓ→next);
```



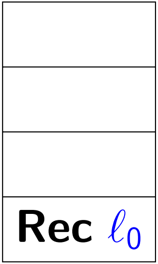
pilha (recursão)



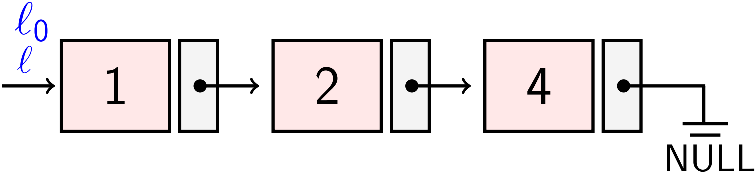
Recursão em listas: imprimir

$\triangleright$ PRINT (List $*\ell$)

```
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
    PRINT (ℓ→next);
```

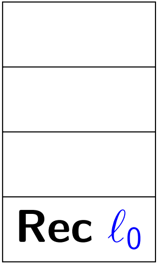


pilha (recursão)

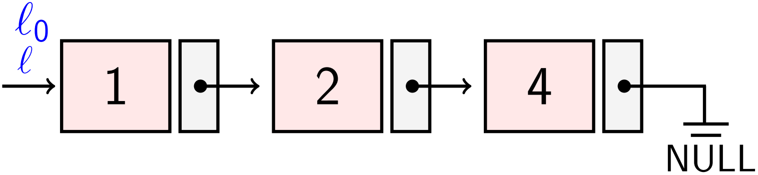


Recursão em listas: imprimir

```
    PRINT (List *ℓ)
▷   se ℓ ≠ NULL então
        printf ("%d ", ℓ→elem);
        PRINT (ℓ→next);
```



pilha (recursão)

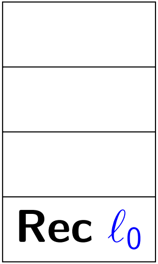


Recursão em listas: imprimir

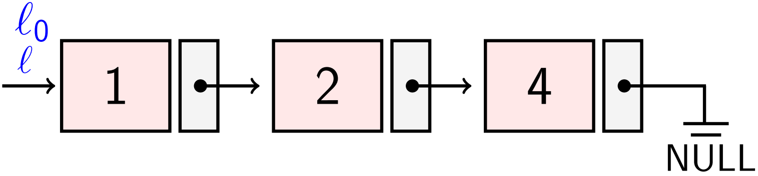
PRINT (List $*\ell$)

---

se $\ell \neq NULL$ **então**
$\triangleright$ **printf** ("%d ", $\ell \rightarrow$elem);
PRINT ($\ell \rightarrow$next);

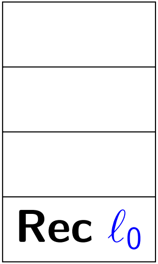


pilha
(recursão)

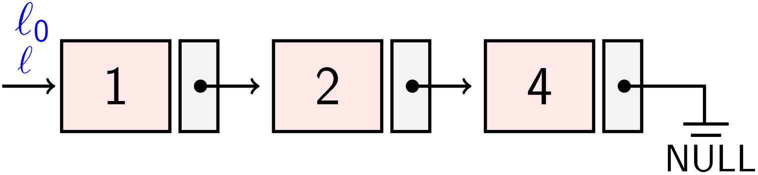


Recursão em listas: imprimir

```
PRINT (List *ℓ)
------------------------------
  se ℓ ≠ NULL então
      printf ("%d ", ℓ→elem);
▷     PRINT (ℓ→next);
```



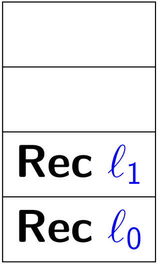
pilha (recursão)



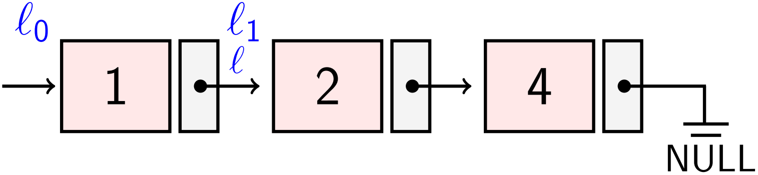
Recursão em listas: imprimir

▷ PRINT (List $*\ell$)

```
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
    PRINT (ℓ→next);
```



pilha (recursão)

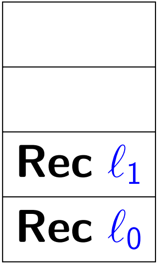


Recursão em listas: imprimir

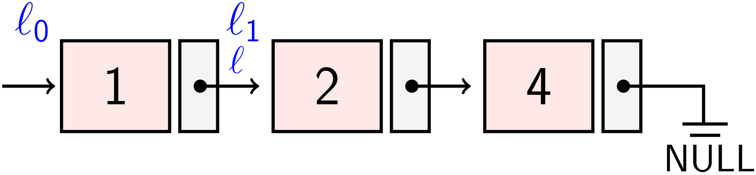
PRINT (List $*\ell$)

---

se $\ell \neq NULL$ **então**
    printf ("%d ", $\ell \rightarrow$elem);
    PRINT ($\ell \rightarrow$next);



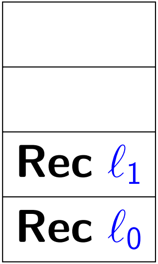
pilha (recursão)



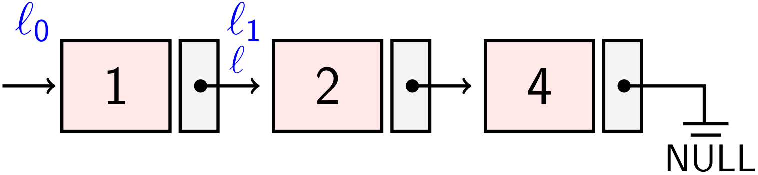
Recursão em listas: imprimir

PRINT (List *ℓ)

```
    se ℓ ≠ NULL então
▷       printf ("%d ", ℓ→elem);
        PRINT (ℓ→next);
```



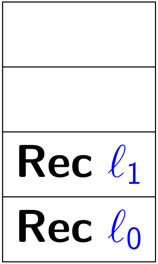
pilha (recursão)



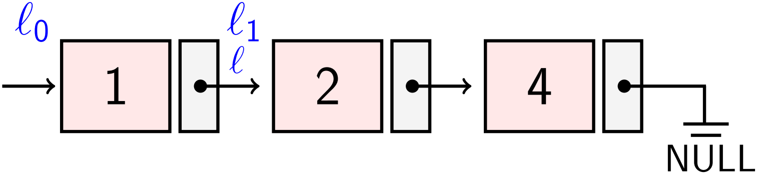
Recursão em listas: imprimir

Print (List *ℓ)

```
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
▷   Print (ℓ→next);
```



pilha (recursão)



Recursão em listas: imprimir

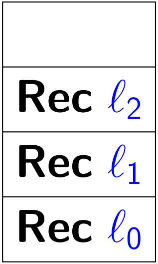
▷ PRINT (List $*\ell$)

---

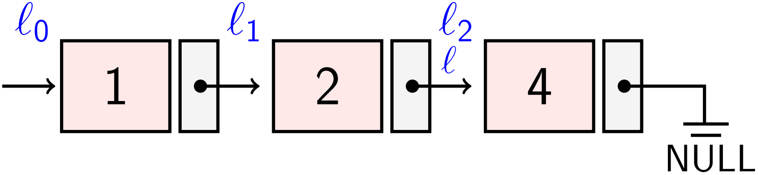
se $\ell \neq NULL$ **então**

 printf ("%d ", $\ell\rightarrow$elem);

 PRINT ($\ell\rightarrow$next);



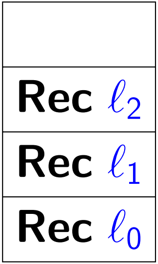
pilha (recursão)



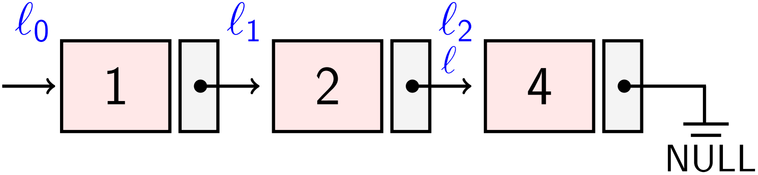
Recursão em listas: imprimir

PRINT (List *ℓ)

```
▷ se ℓ ≠ NULL então
      printf ("%d ", ℓ→elem);
      PRINT (ℓ→next);
```

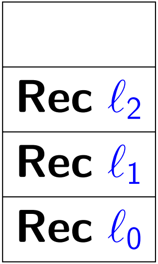


pilha (recursão)

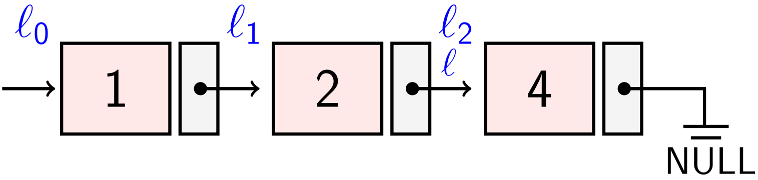


Recursão em listas: imprimir

```
PRINT (List *ℓ)
----------------------------------
  se ℓ ≠ NULL então
▷     printf ("%d ", ℓ→elem);
      PRINT (ℓ→next);
```



pilha (recursão)

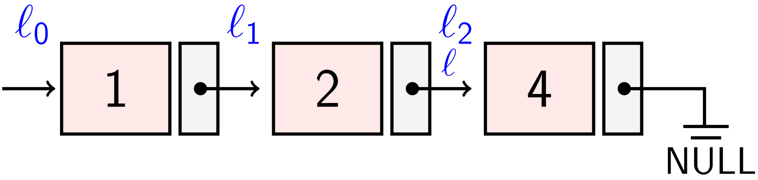


Recursão em listas: imprimir

```
PRINT (List *ℓ)
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
    PRINT (ℓ→next);
```

Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)



Recursão em listas: imprimir

▷ PRINT (List $*\ell$)

---

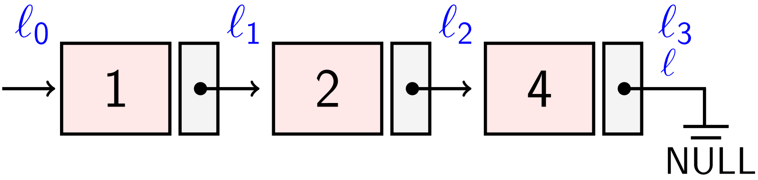
se $\ell \neq NULL$ **então**

  printf (“%d ”, $\ell \rightarrow$elem);

  PRINT ($\ell \rightarrow$next);

Rec ℓ_3
Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)



Recursão em listas: imprimir

PRINT (List *ℓ)

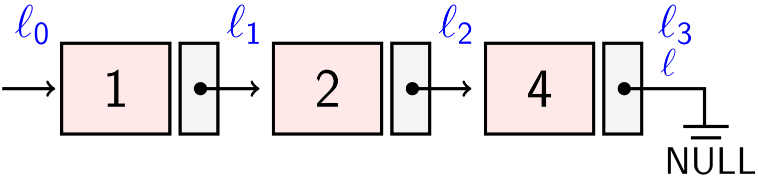
$\triangleright$ 1. **se** $\ell \neq NULL$ **então**

2. **printf** ("%d ", $\ell \rightarrow$elem);

3. PRINT ($\ell \rightarrow$next);

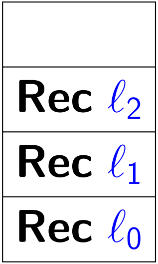
Rec ℓ_3
Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)

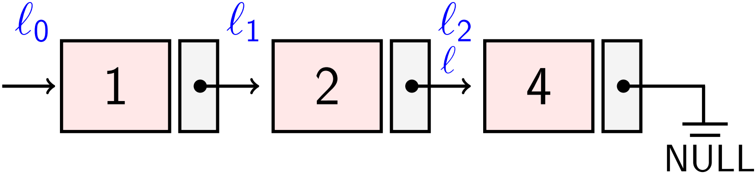


Recursão em listas: imprimir

```
Print (List *ℓ)
--------------------------------
    se ℓ ≠ NULL então
        printf ("%d ", ℓ→elem);
▷       Print (ℓ→next);
```



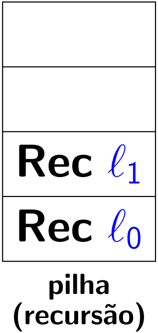
pilha (recursão)

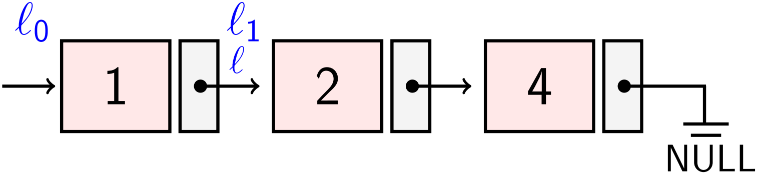


Recursão em listas: imprimir

PRINT (List $*\ell$)

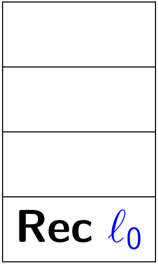
1. **se** $\ell \neq NULL$ **então**
2. **printf** ("%d ", $\ell \rightarrow$elem);
3. PRINT ($\ell \rightarrow$next);



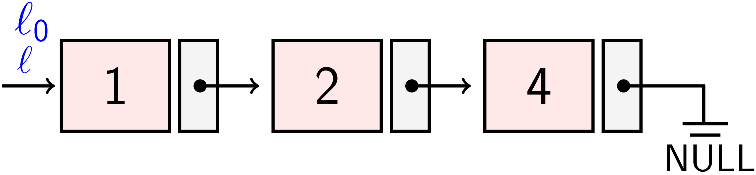


Recursão em listas: imprimir

```
PRINT (List *ℓ)
---------------------------------
   se ℓ ≠ NULL então
       printf ("%d ", ℓ→elem);
 ▷     PRINT (ℓ→next);
```



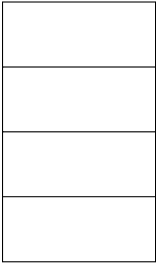
pilha
(recursão)



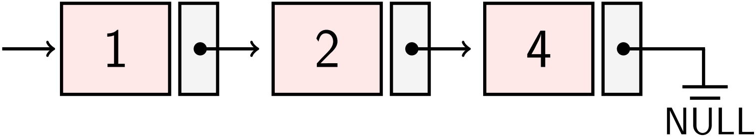
Recursão em listas: imprimir

PRINT (List $*\ell$)

```
se ℓ ≠ NULL então
    printf ("%d ", ℓ→elem);
    PRINT (ℓ→next);
```



pilha (recursão)

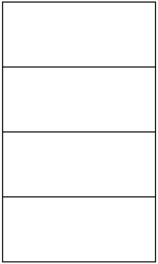


Sumário

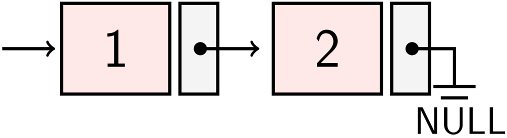
Recursão: inserção no final

INSERT (List *ℓ, int **k**)

```
se ℓ ≠ NULL então
    ℓ→next=INSERT(ℓ→next,k);
else
    ℓ = (List*)malloc(sizeof(List));
    ℓ→elem = k;
    ℓ→next = NULL;
return ℓ
```



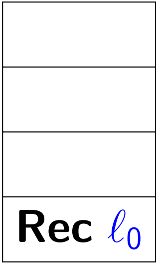
pilha (recursão)



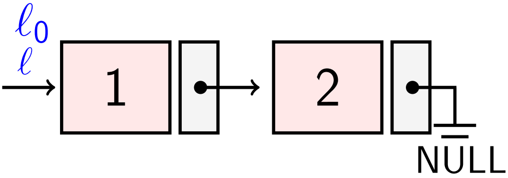
Recursão: inserção no final

▷ Insert (List $*\ell$, int $\mathbf{k} = 4$)

1. **se** $\ell \neq NULL$ **então**
2. $\quad \ell \rightarrow$next$=$Insert($\ell \rightarrow$next,$\mathbf{k}$);
3. **else**
4. $\quad \ell =$ (List*)malloc(sizeof(List));
5. $\quad \ell \rightarrow$elem $= \mathbf{k}$;
6. $\quad \ell \rightarrow$next $=$ NULL;
7. **return** ℓ



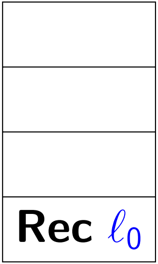
pilha (recursão)



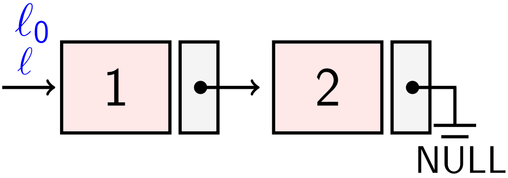
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

▷ 1. **se** $\ell \neq NULL$ **então**
2. $\quad \ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,**k**);
3. **else**
4. $\quad \ell =$ (List*)malloc(sizeof(List));
5. $\quad \ell \rightarrow$elem $=$ **k**;
6. $\quad \ell \rightarrow$next $=$ NULL;
7. **return** ℓ

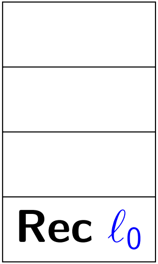


pilha
(recursão)

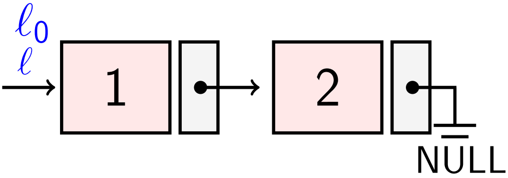


Recursão: inserção no final

```
INSERT (List *ℓ, int k = 4)
-----------------------------------------------
   se ℓ ≠ NULL então
▷      ℓ→next=INSERT(ℓ→next,k);
   else
       ℓ = (List*)malloc(sizeof(List));
       ℓ→elem = k;
       ℓ→next = NULL;
   return ℓ
```



pilha (recursão)

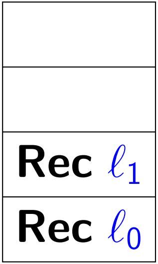


Recursão: inserção no final

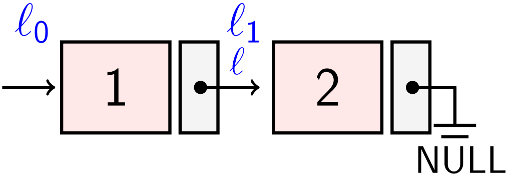
▷ INSERT (List $*\ell$, int $\mathbf{k} = 4$)

---

se $\ell \neq \mathit{NULL}$ **então**
    $\ell \rightarrow$next$=$INSERT($\ell\rightarrow$next,$\mathbf{k}$);
else
    ℓ = (List*)malloc(sizeof(List));
    $\ell \rightarrow$elem $= \mathbf{k}$;
    $\ell \rightarrow$next $=$ NULL;
return ℓ



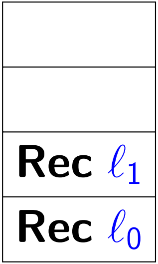
pilha (recursão)



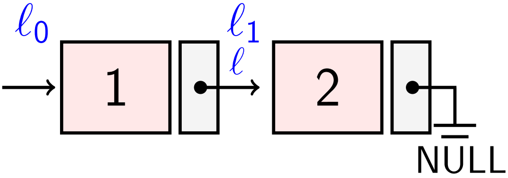
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

```
▷ se ℓ ≠ NULL então
      ℓ→next=INSERT(ℓ→next,k);
  else
      ℓ = (List*)malloc(sizeof(List));
      ℓ→elem = k;
      ℓ→next = NULL;
  return ℓ
```



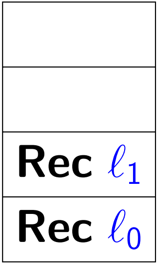
pilha (recursão)



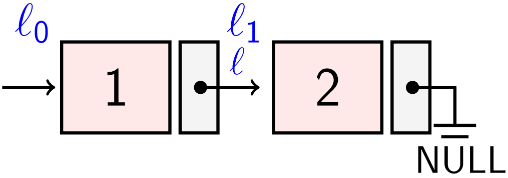
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

1. **se** $\ell \neq NULL$ **então**
2. $\triangleright$ $\ell \rightarrow$next=INSERT($\ell \rightarrow$next,**k**);
3. **else**
4. ℓ = (List*)malloc(sizeof(List));
5. $\ell \rightarrow$elem = **k**;
6. $\ell \rightarrow$next = NULL;
7. **return** ℓ



pilha (recursão)

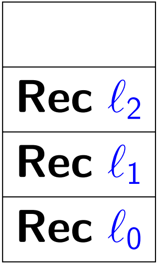


Recursão: inserção no final

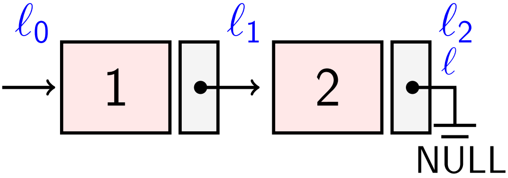
▷ INSERT (List $*\ell$, int $\mathbf{k} = 4$)

---

se $\ell \neq$ *NULL* **então**
    $\ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,$\mathbf{k}$);
else
    ℓ = (List*)malloc(sizeof(List));
    $\ell \rightarrow$elem = $\mathbf{k}$;
    $\ell \rightarrow$next = NULL;
return ℓ



pilha (recursão)



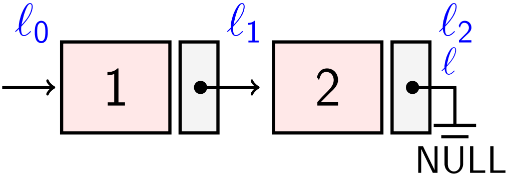
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

```
▷ se ℓ ≠ NULL então
      ℓ→next=INSERT(ℓ→next,k);
  else
      ℓ = (List*)malloc(sizeof(List));
      ℓ→elem = k;
      ℓ→next = NULL;
  return ℓ
```

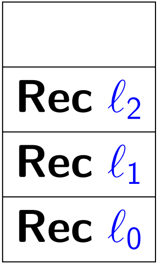
Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)

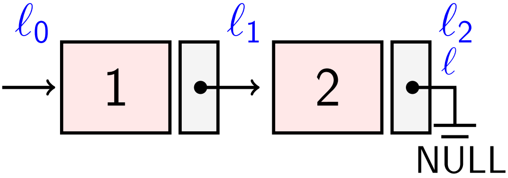


Recursão: inserção no final

```
INSERT (List *ℓ, int k = 4)
--------------------------------------------
   se ℓ ≠ NULL então
       ℓ→next=INSERT(ℓ→next,k);
▷  else
       ℓ = (List*)malloc(sizeof(List));
       ℓ→elem = k;
       ℓ→next = NULL;
   return ℓ
```



pilha (recursão)



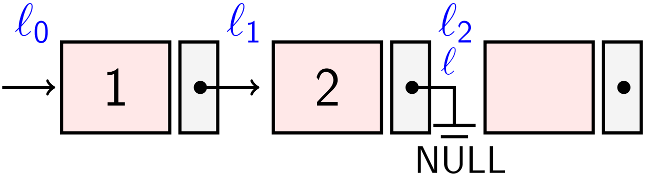
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

```
    se ℓ ≠ NULL então
        ℓ→next=INSERT(ℓ→next,k);
    else
▷       ℓ = (List*)malloc(sizeof(List));
        ℓ→elem = k;
        ℓ→next = NULL;
    return ℓ
```

Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)

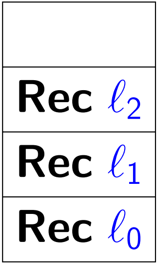


Recursão: inserção no final

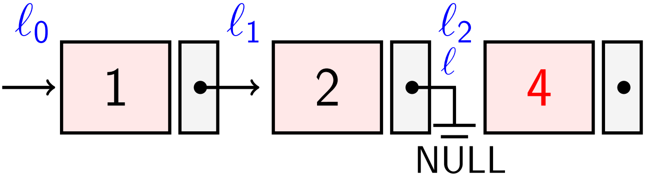
INSERT (List $*\ell$, int $\mathbf{k} = 4$)

---

se $\ell \neq NULL$ **então**
  $\ell \rightarrow$next$=$INSERT($\ell\rightarrow$next,$\mathbf{k}$);
else
  ℓ = (List*)malloc(sizeof(List));
▷ $\ell\rightarrow$elem $= \mathbf{k}$;
  $\ell\rightarrow$next = NULL;
return ℓ



pilha (recursão)



Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

---

se $\ell \neq NULL$ **então**

  $\ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,$\mathbf{k}$);

else

  ℓ = (List*)malloc(sizeof(List));

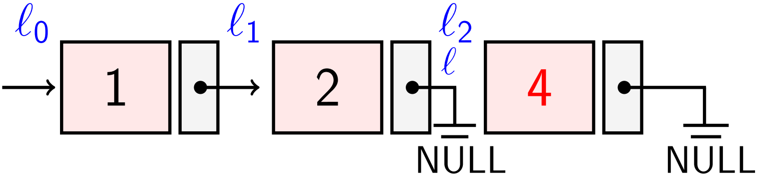
  $\ell \rightarrow$elem $= \mathbf{k}$;

▷ $\ell \rightarrow$next = NULL;

return ℓ

Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

pilha (recursão)



Recursão: inserção no final

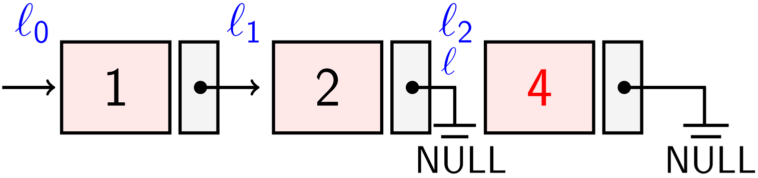
INSERT (List $*\ell$, int $\mathbf{k} = 4$)

---

se $\ell \neq$ *NULL* **então**
    $\ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,**k**);
else
    ℓ = (List*)malloc(sizeof(List));
    $\ell \rightarrow$elem = **k**;
    $\ell \rightarrow$next = NULL;
▷ **return** ℓ

Rec ℓ_2
Rec ℓ_1
Rec ℓ_0

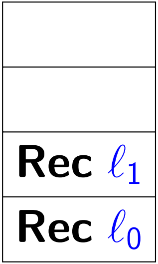
pilha (recursão)



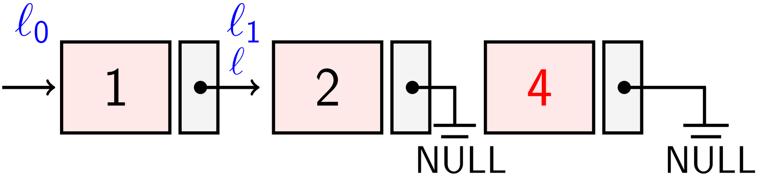
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

1. **se** $\ell \neq NULL$ **então**
2. ▷ $\ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,**k**);
3. **else**
4. ℓ = (List*)malloc(sizeof(List));
5. $\ell \rightarrow$elem = **k**;
6. $\ell \rightarrow$next = NULL;
7. **return** ℓ



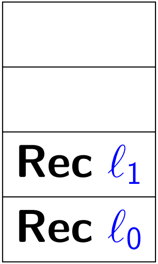
pilha (recursão)



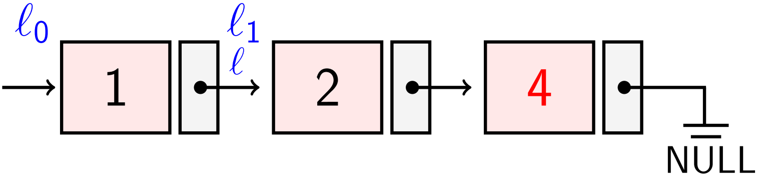
Recursão: inserção no final

Insert (List $*\ell$, int $\mathbf{k} = 4$)

```
   se ℓ ≠ NULL então
▷      ℓ→next = INSERT(ℓ→next, k);
   else
       ℓ = (List*)malloc(sizeof(List));
       ℓ→elem = k;
       ℓ→next = NULL;
   return ℓ
```



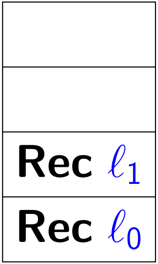
pilha (recursão)



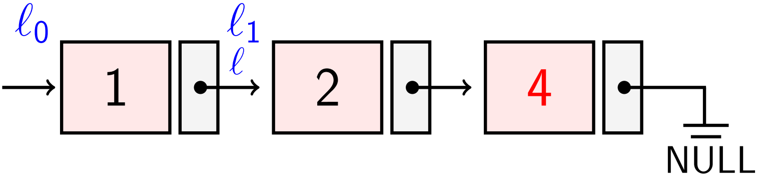
Recursão: inserção no final

INSERT (List $*\ell$, int $\mathbf{k} = 4$)

```
   se ℓ ≠ NULL então
       ℓ→next=INSERT(ℓ→next,k);
   else
       ℓ = (List*)malloc(sizeof(List));
       ℓ→elem = k;
       ℓ→next = NULL;
▷  return ℓ
```



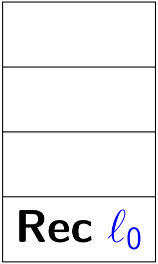
pilha (recursão)



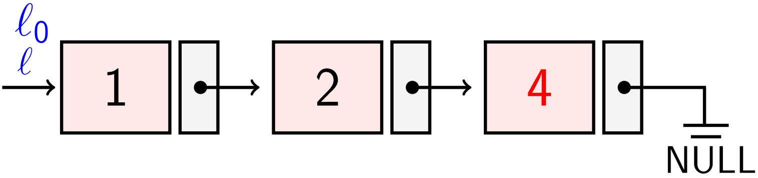
Recursão: inserção no final

Insert (List $*\ell$, int **k** $= 4$)

```
  se ℓ ≠ NULL então
▷     ℓ→next=Insert(ℓ→next,k);
  else
      ℓ = (List*)malloc(sizeof(List));
      ℓ→elem = k;
      ℓ→next = NULL;
  return ℓ
```



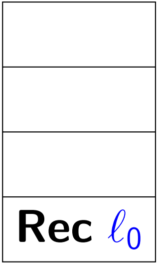
pilha (recursão)



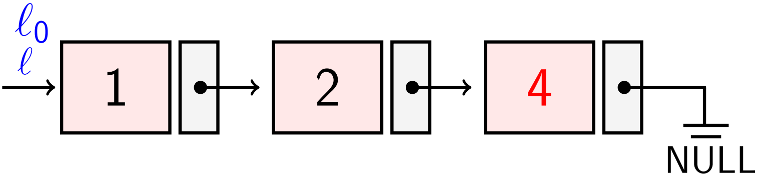
Recursão: inserção no final

$\textsc{Insert}$ (List $*\ell$, int $\mathbf{k} = 4$)

```
se ℓ ≠ NULL então
      ℓ→next=INSERT(ℓ→next,k);
else
      ℓ = (List*)malloc(sizeof(List));
      ℓ→elem = k;
      ℓ→next = NULL;
▷ return ℓ
```



pilha (recursão)

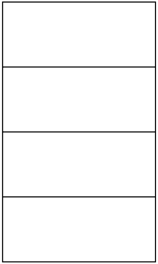


Recursão: inserção no final

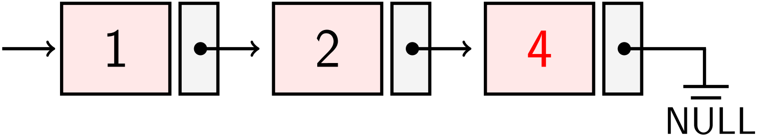
INSERT (List $*\ell$, int **k**)

---

se $\ell \neq NULL$ **então**
    $\ell \rightarrow$next$=$INSERT($\ell \rightarrow$next,**k**);
else
    $\ell =$ (List*)malloc(sizeof(List));
    $\ell \rightarrow$elem $=$ **k**;
    $\ell \rightarrow$next $=$ NULL;
return ℓ



pilha (recursão)



Sumário

Fibonacci

A sequência de Fibonacci foi descrita em 1202 através da observação da evolução de uma população de coelhos. Esta sequência tem aplicações na análise de mercados financeiros, teoria dos jogos, e tem relação com certas configurações biológicas. Em termos matemáticos, a sequência é definida por

$$F_n = F_{n-1} + F_{n-2}$$

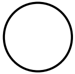
tal que $F_0 = 0$, $F_1 = 1$ (dois casos base), e $n \geq 2$. Os dez primeiros valores da sequência são:

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

Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n**)

```
se n ≤ 1 então
    return n;
return Fib(n − 1) + Fib(n − 2);
```



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=4**)

```
se n ≤ 1 então
    return n;
return Fib(n − 1) + Fib(n − 2);
```

F_4

Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=4**)

▷ 1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;

F_4

Recursão: Fibonacci

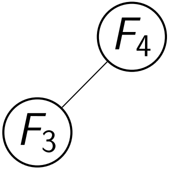
```
    int Fib (int n=4)
----------------------------------------
    se n ≤ 1 então
        return n;
▷   return Fib(n − 1) + Fib(n − 2);
```

F_4

Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int** **n=3**)

1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



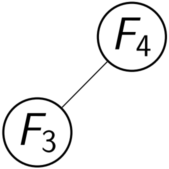
Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=3**)

▷ 1. **se** **n** ≤ 1 **então**

2. **return** **n**;

3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



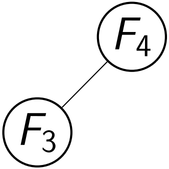
Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=3**)

---

1. **se** **n** ≤ 1 **então**
2. **return** **n**;

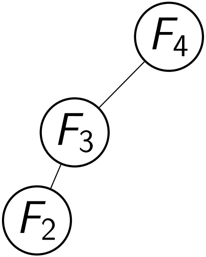
$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

$\triangleright$ **int** $\textsc{Fib}$ (**int n=2**)

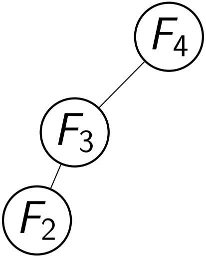
1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

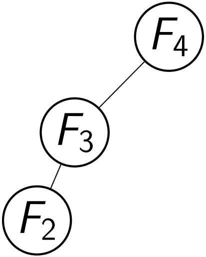
int Fib (**int n=2**)

▷ 1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\text{Fib}(\mathbf{n}-1) + \text{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

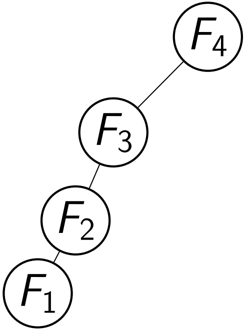
```
int Fib (int n=2)
─────────────────────────────
   se n ≤ 1 então
       return n;
▷  return Fib(n − 1) + Fib(n − 2);
```



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=1**)

1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



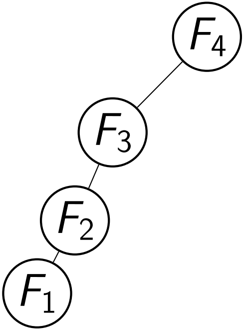
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=1**)

▷ 1. **se** $\mathbf{n} \leq 1$ **então**

2. **return n**;

3. **return** $\textsc{Fib}(\mathbf{n} - 1) + \textsc{Fib}(\mathbf{n} - 2)$;

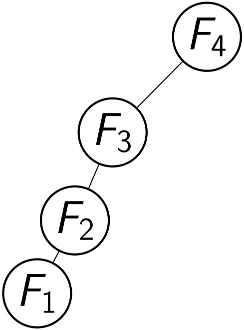


Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=1**)

---

1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;

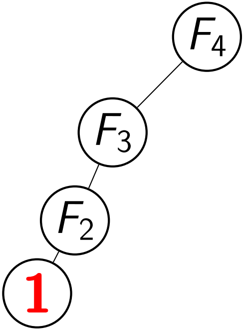


Recursão: Fibonacci

int FIB (**int n=1**)

---

1. **se n** ≤ 1 **então**
2.   **return n**; ▷
3. **return** $\text{FIB}(\mathbf{n}-1)+\text{FIB}(\mathbf{n}-2)$;



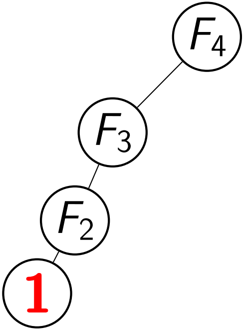
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=1**)

---

1. **se n** ≤ 1 **então**
2. **return n**;

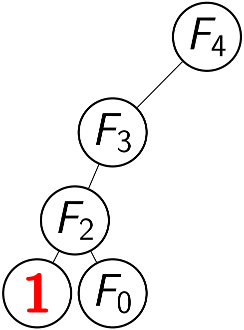
$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=0**)

1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



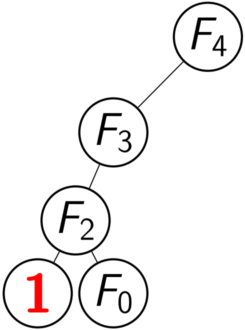
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=0**)

▷ 1. **se** **n** ≤ 1 **então**

2. **return** **n**;

3. **return** $\textsc{Fib}(\mathbf{n} - 1) + \textsc{Fib}(\mathbf{n} - 2)$;

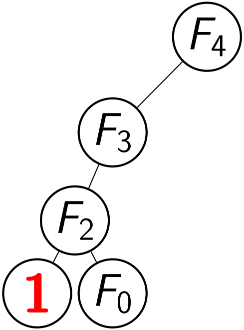


Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=0**)

---

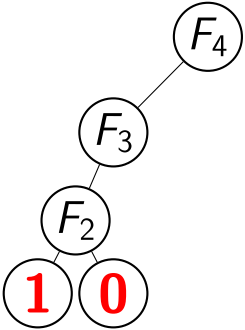
1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=0**)

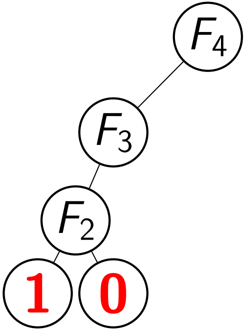
1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



int FIB (**int** $\mathbf{n=2}$)

1. **se** $\mathbf{n} \leq 1$ **então**
2. **return** $\mathbf{n}$;

$\triangleright$ 3. **return** FIB($\mathbf{n}-1$) + FIB($\mathbf{n}-2$);

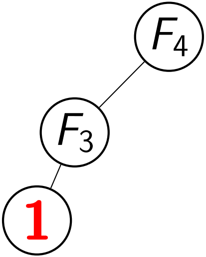


Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=2**)

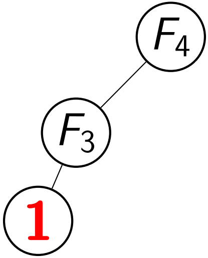
1. **se** **n** ≤ 1 **então**
2. **return** **n**;

$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

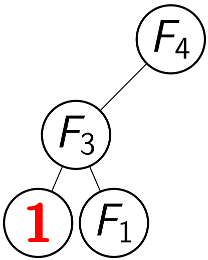
```
int FIB (int n=3)
---------------------------------
   se n ≤ 1 então
       return n;
▷  return FIB(n − 1) + FIB(n − 2);
```



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=1**)

1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



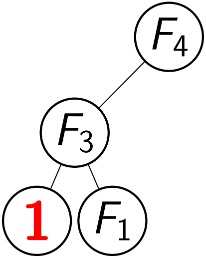
Recursão: Fibonacci

int Fib (int **n=1**)

$\triangleright$ 1. **se** **n** ≤ 1 **então**

2. **return** **n**;

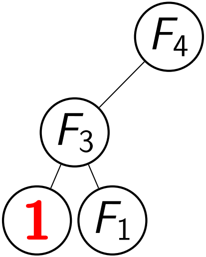
3. **return** $\text{Fib}(\mathbf{n}-1)+\text{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=1**)

1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

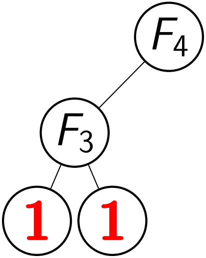
int $\textsc{Fib}$ (**int n=1**)

---

1. **se** **n** ≤ 1 **então**

$\triangleright$ 2. **return** **n**;

3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



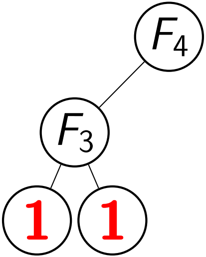
Recursão: Fibonacci

int $\textsc{Fib}$ (**int** $\mathbf{n=3}$)

---

1. **se** $\mathbf{n} \leq 1$ **então**
2. **return** $\mathbf{n}$;

$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



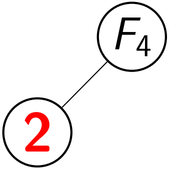
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=3**)

---

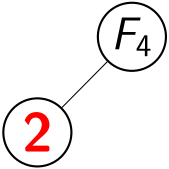
1. **se n** ≤ 1 **então**
2. **return n**;

$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

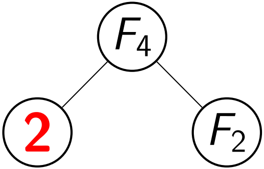
```
     int FIB (int n=4)
     ---------------------------------------
     se n ≤ 1 então
         return n;
▷    return FIB(n − 1) + FIB(n − 2);
```



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=2**)

1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



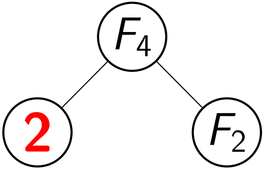
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=2**)

▷ 1. **se** $\mathbf{n} \leq 1$ **então**

2. **return** **n**;

3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



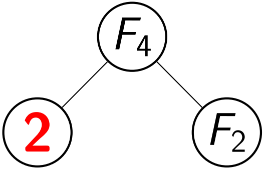
Recursão: Fibonacci

int FIB (**int** $\mathbf{n=2}$)

---

1. **se** $\mathbf{n} \leq 1$ **então**
2. **return** $\mathbf{n}$;

$\triangleright$ 3. **return** FIB($\mathbf{n} - 1$) + FIB($\mathbf{n} - 2$);

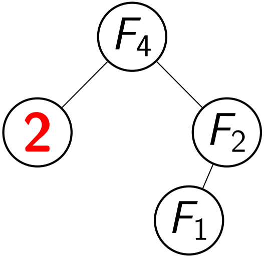


Recursão: Fibonacci

▷ **int** FIB (**int n=1**)

---

1. **se n** ≤ 1 **então**
2.   **return n**;
3. **return** $\text{FIB}(\mathbf{n} - 1) + \text{FIB}(\mathbf{n} - 2)$;



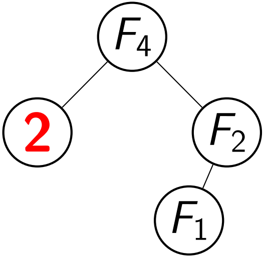
Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=1**)

$\triangleright$ 1. **se n** ≤ 1 **então**

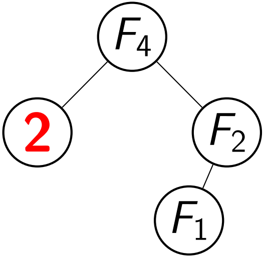
2. **return n**;

3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

```
int FIB (int n=1)
----------------------------------
  se n ≤ 1 então
▷     return n;
  return FIB(n − 1) + FIB(n − 2);
```

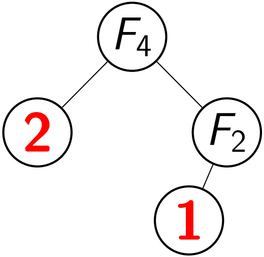


Recursão: Fibonacci

int $\textsc{Fib}$ (**int n=1**)

---

1. **se** **n** ≤ 1 **então**
2. **return** **n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



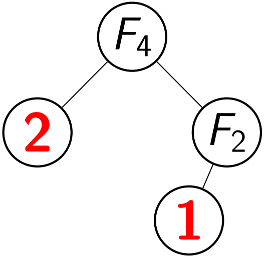
Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=2**)

---

1. **se** **n** ≤ 1 **então**
2. **return** **n**;

$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

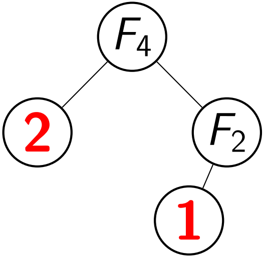
int $\text{F}\scriptstyle\text{IB}$ (**int** **n=2**)

---

se **n** ≤ 1 **então**

  return **n**;

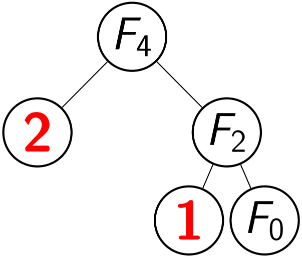
▷ **return** $\text{F}\scriptstyle\text{IB}$$(\mathbf{n}-1)+$$\text{F}\scriptstyle\text{IB}$$(\mathbf{n}-2)$;



Recursão: Fibonacci

▷ **int** $\textsc{Fib}$ (**int n=0**)

1. **se n** ≤ 1 **então**
2. **return n**;
3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



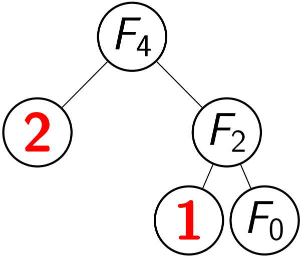
Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=0**)

▷ **se** **n** ≤ 1 **então**

  return **n**;

return $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

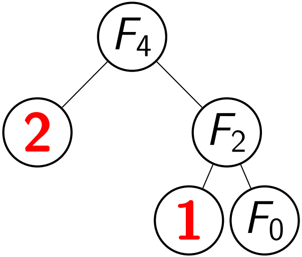
int $\textsc{Fib}$ (**int** **n=0**)

---

1. **se** **n** ≤ 1 **então**

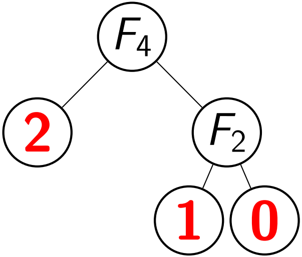
▷ 2. **return** **n**;

3. **return** $\textsc{Fib}(\mathbf{n} - 1) + \textsc{Fib}(\mathbf{n} - 2)$;



Recursão: Fibonacci

```
int FIB (int n=0)
---------------------------------
   se n ≤ 1 então
▷      return n;
   return FIB(n − 1) + FIB(n − 2);
```

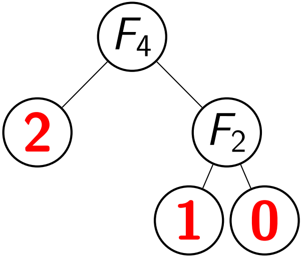


Recursão: Fibonacci

int $\textsc{Fib}$ (**int** $\mathbf{n{=}2}$)

---

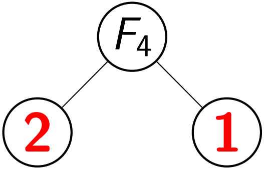
se $\mathbf{n} \leq 1$ **então**
    return $\mathbf{n}$;
$\triangleright$ **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;



Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=2**)

```
  se n ≤ 1 então
      return n;
▷ return Fib(n − 1) + Fib(n − 2);
```



Recursão: Fibonacci

int $\textsc{Fib}$ (**int** **n=4**)

---

1. **se** **n** ≤ 1 **então**
2. **return** **n**;

$\triangleright$ 3. **return** $\textsc{Fib}(\mathbf{n}-1)+\textsc{Fib}(\mathbf{n}-2)$;

3

Sumário

Considerações

De forma geral, problemas que permitem aplicar o paradigma de divisão e conquista são bons candidatos ao uso de recursão. No entanto, abordagens recursivas são geralmente mais lentas pois as variáveis locais, endereços de retorno, etc, devem ser armazenados em uma pilha na memória para permitir o correto funcionamento.

Considerações

De forma geral, problemas que permitem aplicar o paradigma de divisão e conquista são bons candidatos ao uso de recursão. No entanto, abordagens recursivas são geralmente mais lentas pois as variáveis locais, endereços de retorno, etc, devem ser armazenados em uma pilha na memória para permitir o correto funcionamento.

Considerações

No entanto, otimizações de cauda (**tail-call optimization**), podem deixar programas recursivos mais rápidos. **Definição** de **tail-recursive algorithm**: quando a última ação de uma função é chamar uma função (a si própria ou outra):

Soma de Gauss: **versão tail-recursive**

Soma-Gauss (**n**)

---

1. **se** **n** $= 1$ **então**
2. **return** **n**;
3. **return** **n** $+$ Soma-Gauss ($\mathbf{n} - 1$);

Considerações

No entanto, otimizações de cauda (**tail-call optimization**), podem deixar programas recursivos mais rápidos. **Definição** de **tail-recursive algorithm**: quando a última ação de uma função é chamar uma função (a si própria ou outra):

Soma de Gauss: **versão non-tail-recursive**

$\textsc{Soma-Gauss}\ (\mathbf{n})$

---

1. **se** $\mathbf{n} = 1$ **então**
2. **return** $\mathbf{n}$;
3. **return** $\textsc{Soma-Gauss}\ (\mathbf{n} - 1) + \mathbf{n}$;

Considerações

Algoritmos estilo **tail-recursive** são otimizados por compiladores como o gcc (opção -02) pois existe uma equivalência entre uma função recursiva de cauda e um laço. Ou seja, você pode converter função recursiva de cauda para um laço simples (e vice-versa). Assim, você evita o famoso estouro de pilha (**stack overflow**) em recursões profundas.