Universidade de Brasília
Departamento de Ciência da Computação

Semana 14

Recursividade

CIC0004
Algoritmos e Programação de Computadores

Prof. Pedro Garcia Freitas https://pedrogarcia.gitlab.io/ pedro.garcia@unb.br

Brasilia

Objetivos

Essa aula tem como objetivo fazer com que o aluno

- familiarize-se com a ideia de recursão
- aprender a usar a recursão como uma ferramenta de programação
- familiarize-se com o algoritmo de busca binária como exemplo de recursão

1. Introdução

Como você busca um nome na lista telefônica?

1. Introdução

Como você busca um nome na lista telefônica?

```
BUSCA:
   página do meio = (primeira página + última página) / 2
   Vá a página do meio;
   SE (o nome estiver na página do meio)
        concluído; //este é o caso base
   SENÃO SE (o nome estiver antes da página do meio e em ordem alfabética)
        //redefine área de busca para a metade inicial
       última página = página do meio
       Busca //mesmo processo em um número reduzido de páginas
   SENÃO //caso o nome deve estar depois da página do meio
       //redefinir área de busca para a metade posterior
       primeira página = página do meio
       Busca //mesmo processo em um número reduzido de páginas
```

1. Introdução

Recursão: uma definição em termos de si mesma.

1. Introdução

Recursão: uma definição em termos de si mesma.

Recursão em **algoritmos**:

- Abordagem natural para **alguns** (nem todos) os problemas
- Um *algoritmo recursivo* usa **a si mesmo** para resolver um ou mais **problemas menores idênticos.**

1. Introdução

Recursão: uma definição em termos de si mesma.

Recursão em **linguagens de programação**:

- Métodos recursivos **implementam** algoritmos recursivos
- Um método recursivo **inclui uma chamada a si mesmo**.

Métodos Recursivos

Devem Terminar

Métodos Recursivos **Devem Terminar**

Um método recursivo **deve** ter pelo menos **um caso base**, ou **caso de parada.**

Métodos Recursivos **Devem Terminar**

Um método recursivo **deve** ter pelo menos **um caso base**, ou **caso de parada.**

- Um caso base **não executa** uma chamada recursiva.

Métodos Recursivos **Devem Terminar**

Um método recursivo **deve** ter pelo menos **um caso base**, ou **caso de parada.**

- Um caso base **não executa** uma chamada recursiva.
- O caso base **para** a recursão.

Métodos Recursivos **Devem Terminar**

Um método recursivo **deve** ter pelo menos **um caso base**, ou **caso de parada.**

- Cada chamada sucessiva a si mesmo deve ser uma **"versão menor de si mesma".**
  - Um **argumento** que descreve um problema menor.
  - Até atingir o **caso base**.

2. Componentes Principais do Design de um Algoritmo Recursivo

1. Qual é o menor problema **idêntico**?
   - Decomposicão
2. Como as respostas para os problemas menores são combinadas para formar a resposta para o problema maior?
   - Composição
3. Qual é o **menor problema** que pode ser resolvido facilmente (**sem mais decomposição**)?
   - Caso base (caso de parada)

3. Exemplos em Recursão

- Pode ser confuso na primeira vez.
- Comece com alguns exemplos simples
  - algoritmos recursivos podem não ser a solução óbvia.
- Posteriormente, com algoritmos inerentemente recursivos
  - mais difícil de implementar de outra forma.

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

- *N*! = N * (*N*-1)! [para *N* > 1]
- 1! = 1
- Exemplo: 3!
  = 3 * 2!
  = 3 * (2 * 1!)
  = 3 * 2 * 1

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

- *N*! = N * (*N*-1)! [para *N*
- 1! = 1
- Exemplo: 3!

  = 3 * 2!

  = 3 * (2 * 1!)

  = 3 * 2 * 1

Decomposição
(um problema idêntico menor)

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

- *N*! = N * (*N*-1)! [para *N*
- 1! = 1
- Exemplo: 3!
  = 3 * 2!
  = 3 * (2 * 1!)
  = 3 * 2 * 1

Decomposição
(um problema idêntico menor)

Composição
(como o problema atual é combinado com problema menor)

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

- *N*! = N * (*N*-1)! [para *N*
- 1! = 1
- Exemplo: 3!
  = 3 * 2!
  = 3 * (2 * 1!)
  = 3 * 2 * 1

Decomposição
(um problema idêntico menor)

Composição
(como o problema atual é combinado com problema menor)cc

Caso base
(quando o problema atinge este caso, simplesmente retorna "1" e não se decompõe mais)

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

```
def fact(n):
    return 1 if n < 1 else n * fact(n - 1)
```

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

```
def fact(n):
    return 1 if n < 1 else n * fact(n - 1)
```

Caso base

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

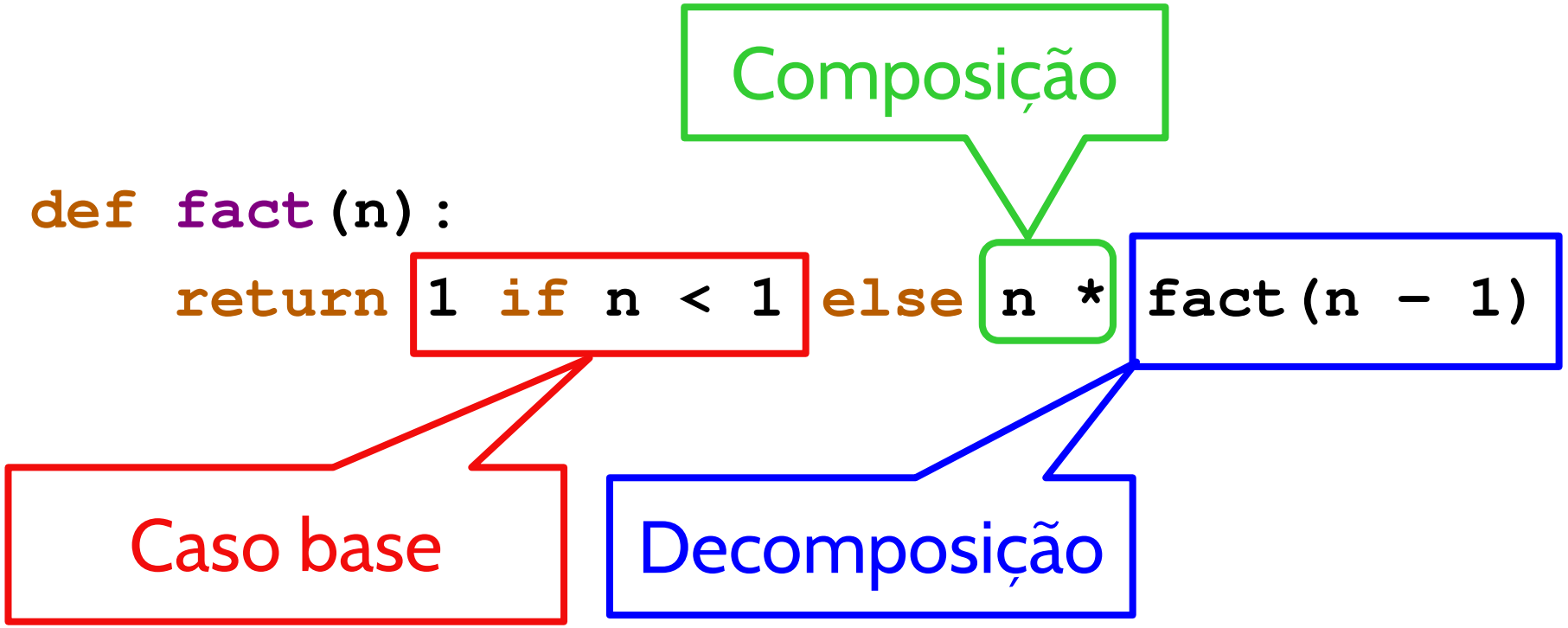
```
def fact(n):
    return 1 if n < 1 else n * fact(n - 1)
```

Caso base

Decomposição

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**



```
fact(3):
  return 1 if n < 1 else 3 * fact(2)
```

```
fact(3):
  return 1 if n < 1 else 3 * fact(2)
```

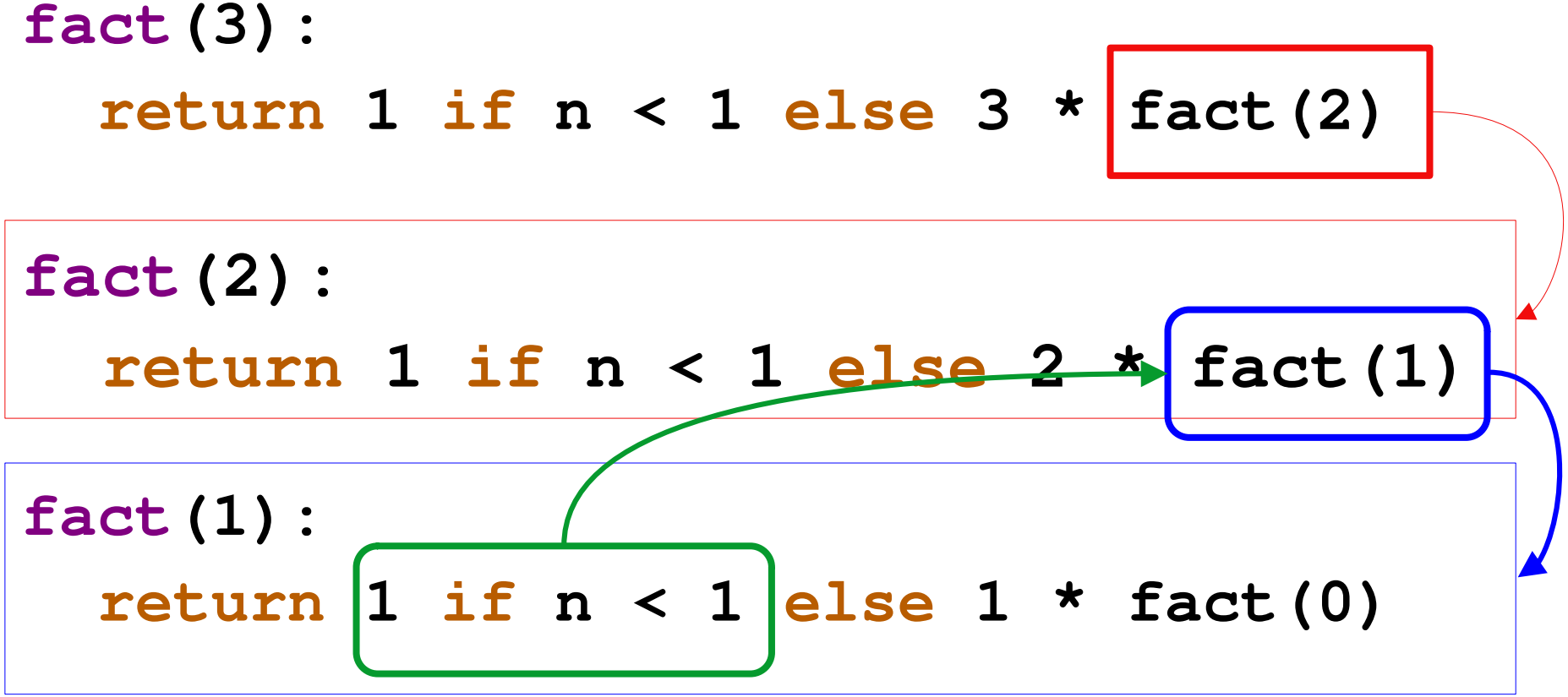
```
fact(2):
  return 1 if n < 1 else 2 * fact(1)
```

```
fact(3):
  return 1 if n < 1 else 3 * fact(2)
```

```
fact(2):
  return 1 if n < 1 else 2 * fact(1)
```

```
fact(3):
  return 1 if n < 1 else 3 * fact(2)

fact(2):
  return 1 if n < 1 else 2 * fact(1)

fact(1):
  return 1 if n < 1 else 1 * fact(0)
```

```
fact(3):
   return 1 if n < 1 else 3 * fact(2)

fact(2):
   return 1 if n < 1 else 2 * fact(1)

fact(1):
   return 1 if n < 1 else 1 * fact(0)
```


fact(3):
return 1 if n < 1 else 3 * fact(2)
fact(2):
return 1 if n < 1 else 2 * fact(1)
fact(1):
return 1 if n < 1 else 1 * fact(0)

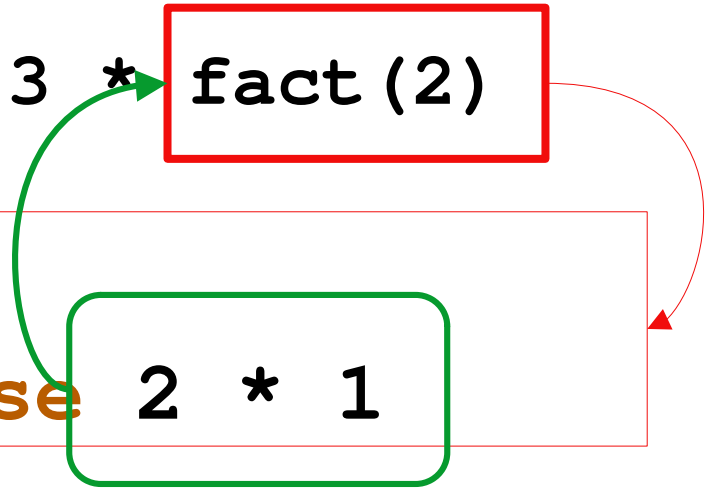
```
fact(3):
   return 1 if n < 1 else 3 * fact(2)

fact(2):
   return 1 if n < 1 else 2 * 1

fact(1):
   return 1 if n < 1 else 1 * fact(0)
```

```
fact(3):
  return 1 if n < 1 else 3 * fact(2)
```

```
fact(2):
  return 1 if n < 1 else 2 * 1
```

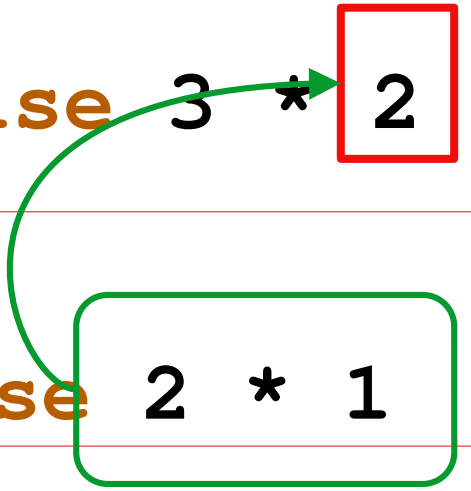
```
fact(3):
  return 1 if n < 1 else 3 * fact(2)
```

```
fact(2):
  return 1 if n < 1 else 2 * 1
```



```
fact(3):
   return 1 if n < 1 else 3 * 2
```

```
fact(2):
   return 1 if n < 1 else 2 * 1
```



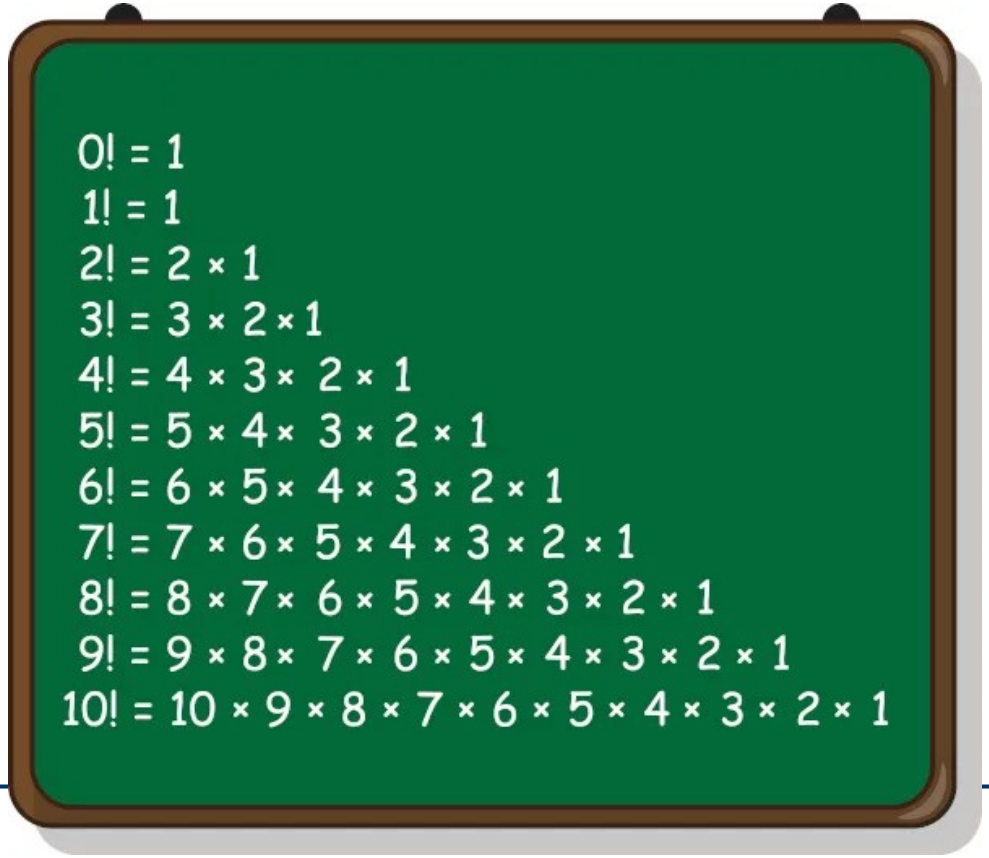
```
fact(3):
  return 1 if n < 1 else 3 * 2
```

```
fact(3):
   return 1 if n < 1 else 6
```

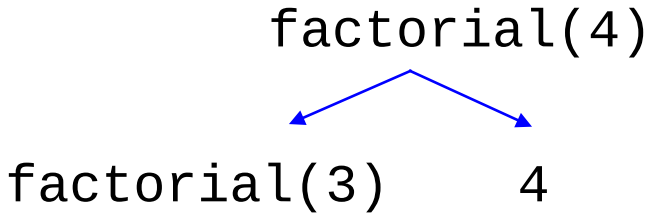
Retorno final

3. Exemplos em Recursão

Exemplo 1: Fatorial: **n !**

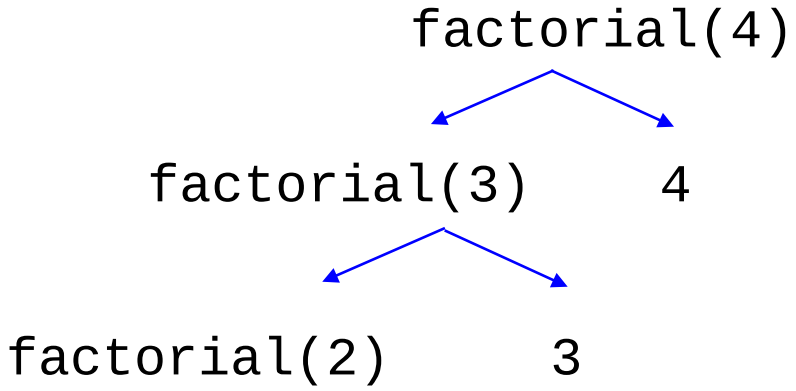


Rastreamento de execução (trace)



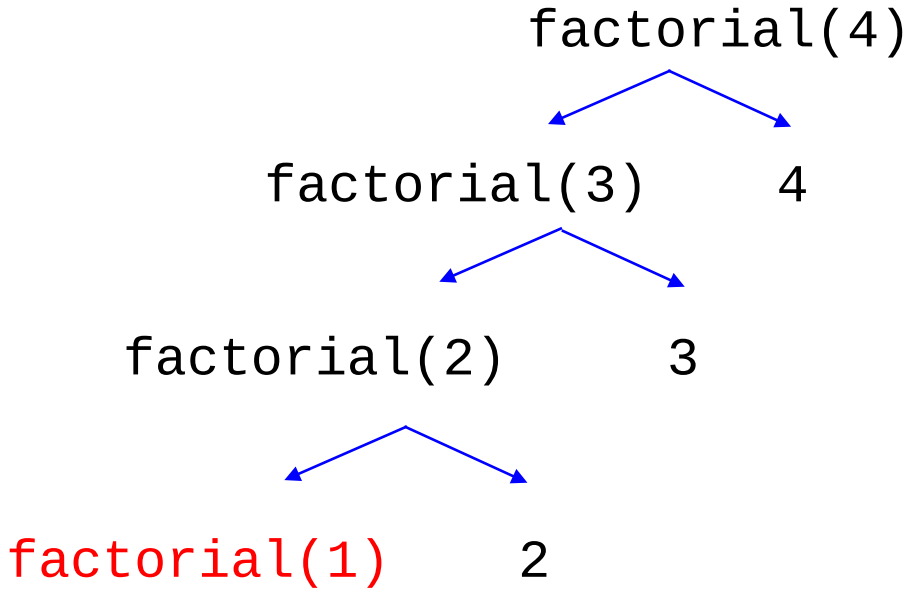
Decomposição

Rastreamento de execução (trace)



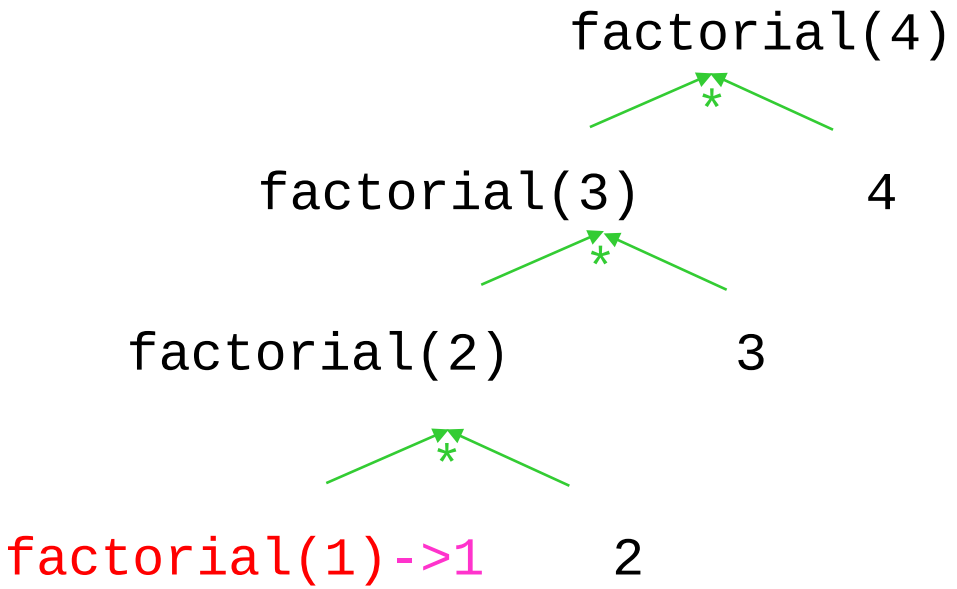
Decomposição

Rastreamento de execução (trace)



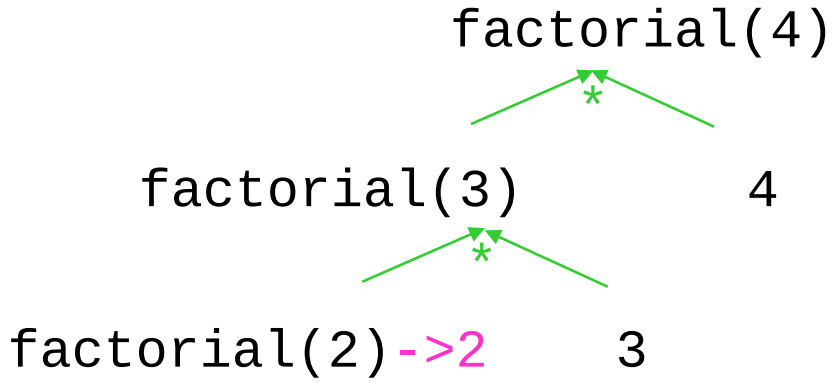
Decomposição

Rastreamento de execução (trace)



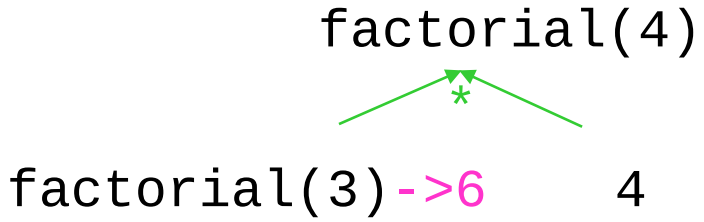
Decomposição

Rastreamento de execução (trace)



Decomposição

Rastreamento de execução (trace)



Decomposição

Rastreamento de execução (trace)

```
factorial(4)->24
```

Decomposição

3. Exemplos em Recursão

Exemplo 2: Números de Fibonacci

- O enésimo número de Fibonacci é a soma dos dois números de Fibonacci anteriores.
- 0, 1, 1, 2, 3, 5, 8, 13, ...
- Design recursivo:
  - Decomposição e composição
    - `fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)`
  - Caso base:
    - `fibonacci(1) = 0`
    - `fibonacci(2) = 1`

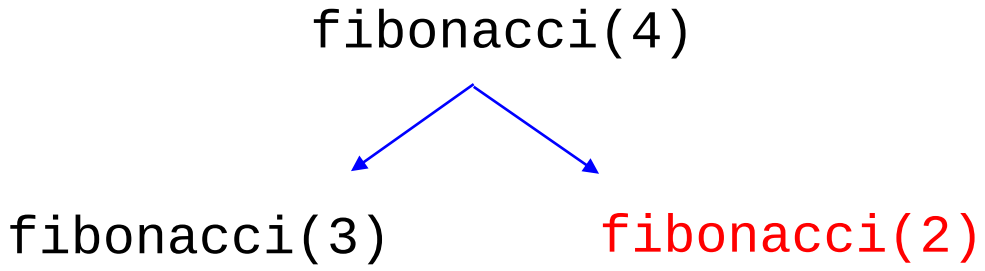
3. Exemplos em Recursão

Exemplo 2: Números de Fibonacci

```
def fibonacci(n):
    if n > 2:
        return fibonacci(n-1) + fibonacci(n-2)
    elif n == 2:
        return 1
    else:
        return 0
```

3. Exemplos em Recursão

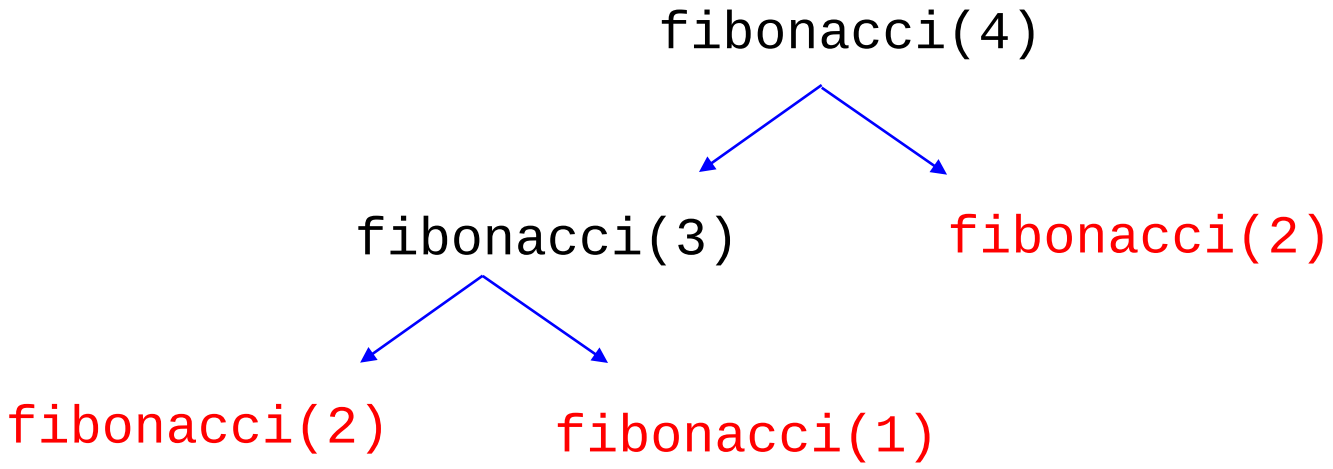
Exemplo 2: Números de Fibonacci



Decomposição

3. Exemplos em Recursão

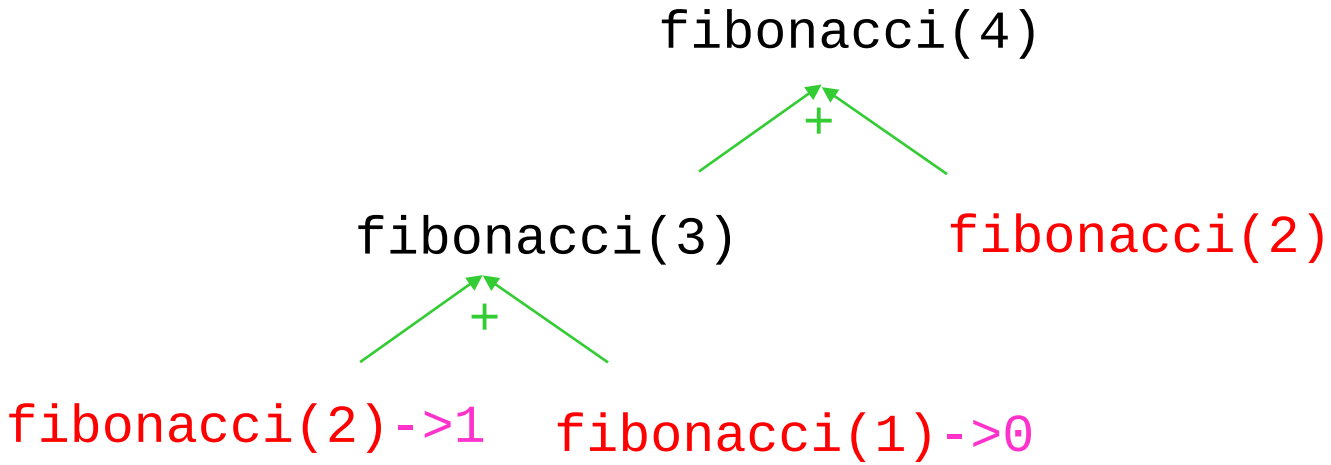
Exemplo 2: Números de Fibonacci



Decomposição

3. Exemplos em Recursão

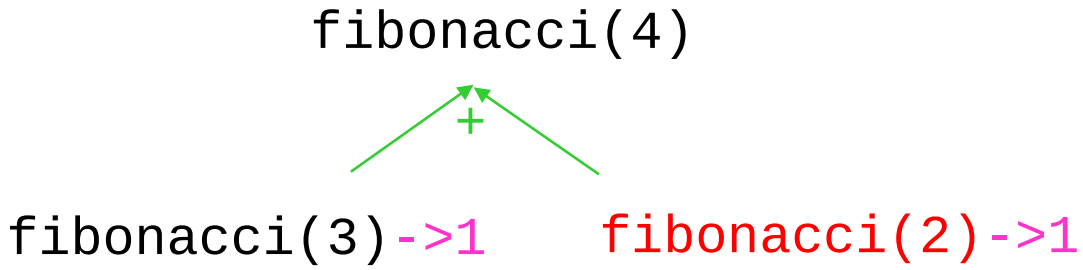
Exemplo 2: Números de Fibonacci



Composição

3. Exemplos em Recursão

Exemplo 2: Números de Fibonacci



Composição

3. Exemplos em Recursão

Exemplo 2: Números de Fibonacci

```
fibonacci(4)->2
```

Composição

4. Chave para uma recursão bem-sucedida

- Deve conter uma declaração if-else (ou alguma outra declaração de ramificação) em algum ponto.
- Alguns ramos: chamada recursiva
  - Argumentos "menores" ou resolver "versões menores" da mesma tarefa (decomposição)
  - Combinação dos resultados (composição) [se necessário]
- Outros ramos: sem chamadas recursivas
  - Casos de parada ou casos base

4. Chave para uma recursão bem-sucedida

O que acontece aqui?

```
def factorial(n):
    fact = 1
    if n > 1:
        fact = factorial(n-1) * n
    return fact
```

4. Chave para uma recursão bem-sucedida

O que acontece aqui?

```
>>> %Run factorials.py
 0 1
 1 1
 2 2
 3 6
 4 24
 5 120
 6 720
 7 5040
 8 40320
 9 362880
```

```
def factorial(n):
    fact = 1
    if n > 1:
        fact = factorial(n-1) * n
    return fact
```

4. Chave para uma recursão bem-sucedida

O que acontece aqui?

```
def factorial(n):
    return factorial(n-1) * n
```

4. Chave para uma recursão bem-sucedida

O

def

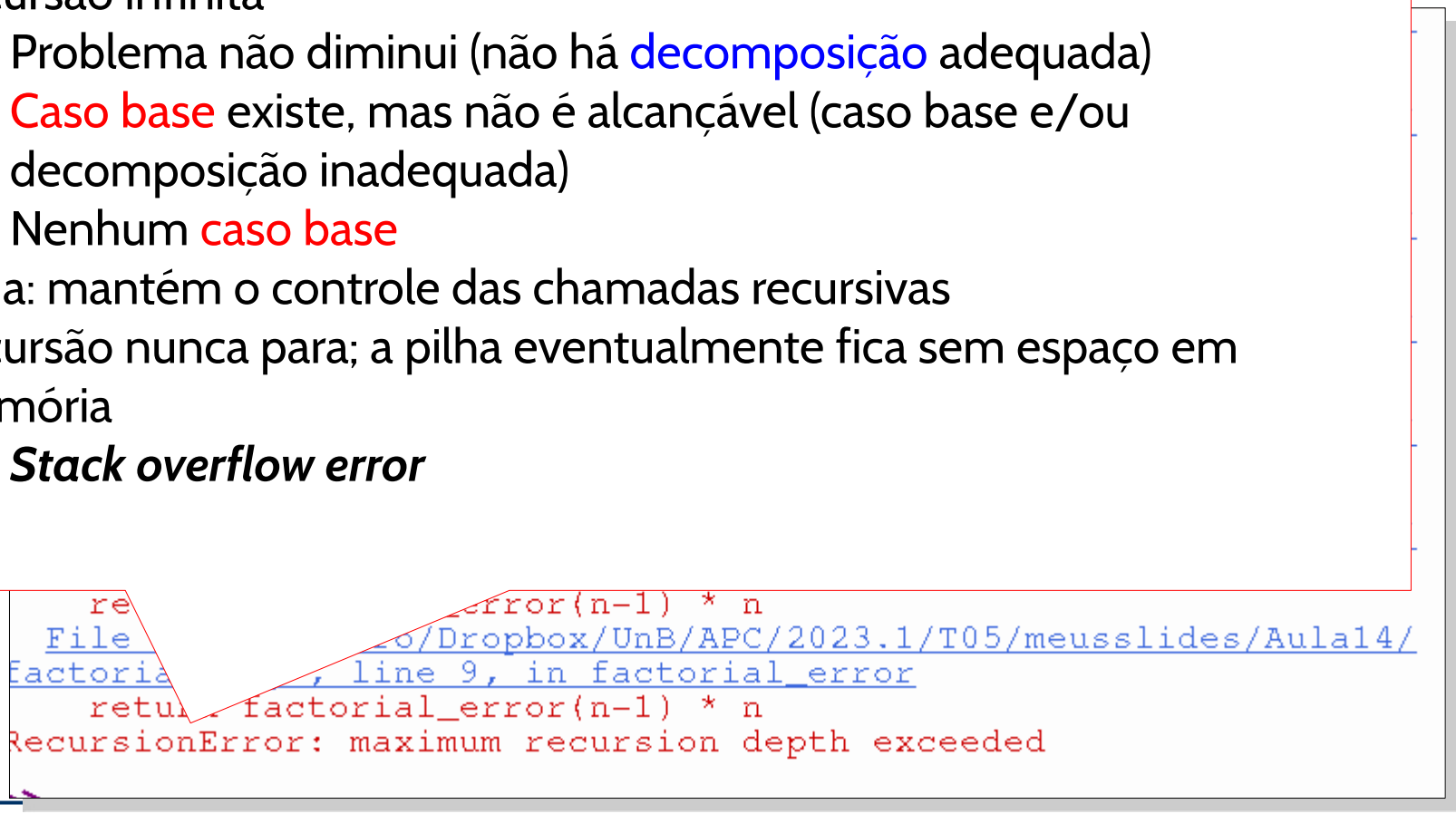
```
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
  File "/home/pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factorials.py", line 9, in factorial_error
    return factorial_error(n-1) * n
RecursionError: maximum recursion depth exceeded
```

Cuidado: Recursão infinita pode causar um erro de estouro de pilha (*Stack Overflow Error*).

```
    re                    ror(n-1) * n
  File                  Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factori                 ne 9, in factorial_error
    ret                 rial_error(n-1) * n
  File '                pedro/Dropbox/UnB/APC/2023.1/T05/meusslides/Aula14/
factoria              y", line 9, in factorial_error
    retu  factorial_error(n-1) * n
RecursionError: maximum recursion depth exceeded
```

Cuidado: Recursão infinita pode causar um erro de estouro de pilha (*Stack Overflow Error*).

- Recursão infinita
  - Problema não diminui (não há decomposição adequada)
  - Caso base existe, mas não é alcançável (caso base e/ou decomposição inadequada)
  - Nenhum caso base
- Pilha: mantém o controle das chamadas recursivas
- Recursão nunca para; a pilha eventualmente fica sem espaço em memória
  - ***Stack overflow error***



5. Erros de recursão

1) **Má decomposição:** Ao dividir o problema em subproblemas menores, é essencial garantir que cada subproblema seja significativamente menor que o problema original. Caso contrário, a recursão pode não convergir para uma solução.
2) **Falta de caso base:** É importante definir um caso base para que a recursão tenha uma condição de parada e não continue indefinidamente.
3) **Chamada recursiva incorreta:** É importante chamar a função recursiva corretamente, fornecendo os parâmetros apropriados. Erros nessa chamada podem levar a resultados incorretos ou a um comportamento indefinido.
4) **Uso excessivo de memória:** Recursões excessivamente profundas podem consumir muita memória, pois cada chamada recursiva adiciona informações à pilha de chamadas. Isso pode levar a um estouro de pilha (*stack overflow*) ou a um desempenho inadequado.

É importante compreender esses erros comuns e estar atento a eles ao projetar e implementar algoritmos recursivos para evitar resultados indesejados.

5. Erros de recursão

- **Má decomposição ?**
- **Sem decomposição:** ?

5. Erros de recursão

Exemplo 3: Números de zeros em um número

- Exemplo: 2**03****0** tem 2 zeros
- Se n tem dois ou mais dígitos:
  - O **número de zeros** é o **número de zeros** no número **n** com o **último dígito removido**
  - Mais 1 se o último dígito é zero!
- Outros exemplos:
  - O número de zeros de **2003****0** é o número de zeros em **2003** + **1**
  - O número de zeros em **2003****1** é o número de zeros em **2003**

5. Erros de recursão

Exemplo 3: Números de zeros em um número

- Exemplo: 2**0**3**0** tem 2 zeros

```
def get_zeros_naive(n: int):
    n_list = list(str(n_str))
    zeroes = [1 for i in n_list if i=='0']
    return sum(zeroes)

def get_zeros_naive2(n: int):
    n_str = str(n)
    zeroes = n_str.count('0')
    return zeroes
```

5. Erros de recursão

Exemplo 3: Números de zeros em um número

- Seja a função **`get_zeros`** que recebe o número *N*
- *K* = número de dígitos de *N*
- Decomposição:
  - **`get_zeros`** nos primeiros *K* - 1 digits
  - Último digito
- Composição: Add:
  - **`get_zeros`** nos primeiros *K* – 1 digitos
  - mais 1 se o último dígito é zero
- Caso base: Se *N* tem somente 1 dígito (i.e., *K* = 1)

5. Erros de recursão

Exemplo 3: Números de zeros em um número

```
def get_zeros(n: int):
    if n == 0:
        return 1
    elif n < 10 and n != 0:
        return 0
    elif n % 10 == 0:
        return get_zeros(n // 10) + 1
    else:
        return get_zeros(n // 10)
```

5. Erros de recursão

Exemplo 3: Números de zeros em um n

```
def get_zeros(n: int):
    if n == 0:
        return 1
    elif n < 10 and n != 0:
        return 0
    elif n % 10 == 0:
        return get_zeros(n // 10) + 1
    else:
        return get_zeros(n // 10)
```

Qual é (são) os casos base? Pq?

Qual é a Decomposição?

Qual a composição?

5. Erros de recursão

Exemplo 3: Números de zeros em um n

```
def get_zeros(n: int):
    if n == 0:
        return 1
    elif n < 10 and n != 0:
        return 0
    elif n % 10 == 0:
        return get_zeros(n // 10) + 1
    else:
        return get_zeros(n // 10)
```

Qual é (são) os casos base? Pq?

Qual é a Decomposição?

Qual a composição?

5. Erros de recursão

Exemplo 3: Números de zeros em um n

```
def get_zeros(n: int):
    if n == 0:
        return 1
    elif n < 10 and n != 0:
        return 0
    elif n % 10 == 0:
        return get_zeros(n // 10) + 1
    else:
        return get_zeros(n // 10)
```

Qual é (são) os casos base? Pq?

Qual é a Decomposição?

Qual a composição?

5. Erros de recursão

Exemplo 3: Números de zeros em um n

```
def get_zeros(n: int):
    if n == 0:
        return 1
    elif n < 10 and n != 0:
        return 0
    elif n % 10 == 0:
        return get_zeros(n // 10) + 1
    else:
        return get_zeros(n // 10) + 0
```

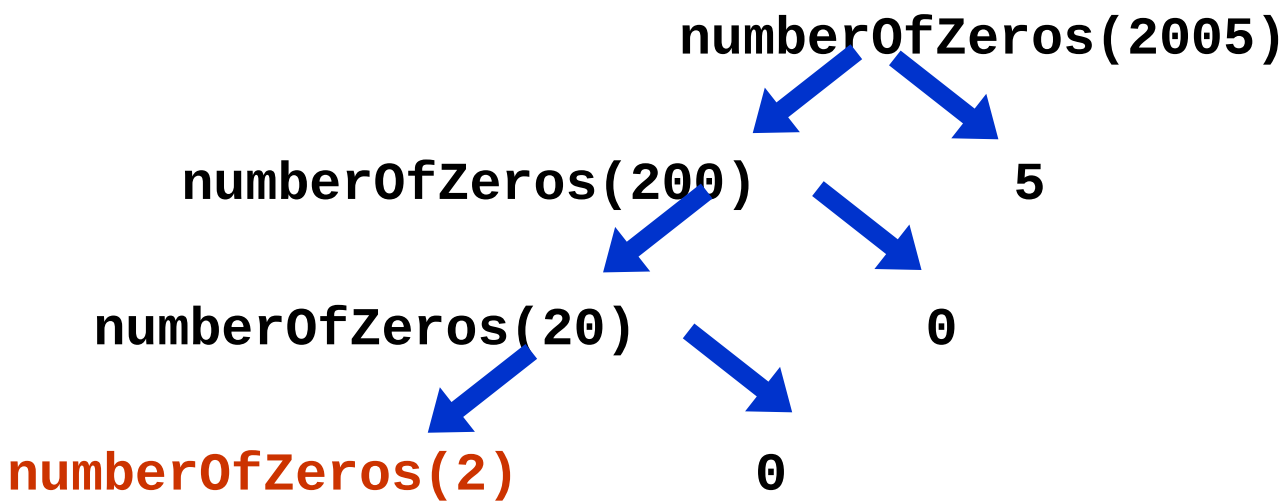
Qual é (são) os casos base? Pq?

Qual é a Decomposição?

Qual a composição?

5. Erros de recursão

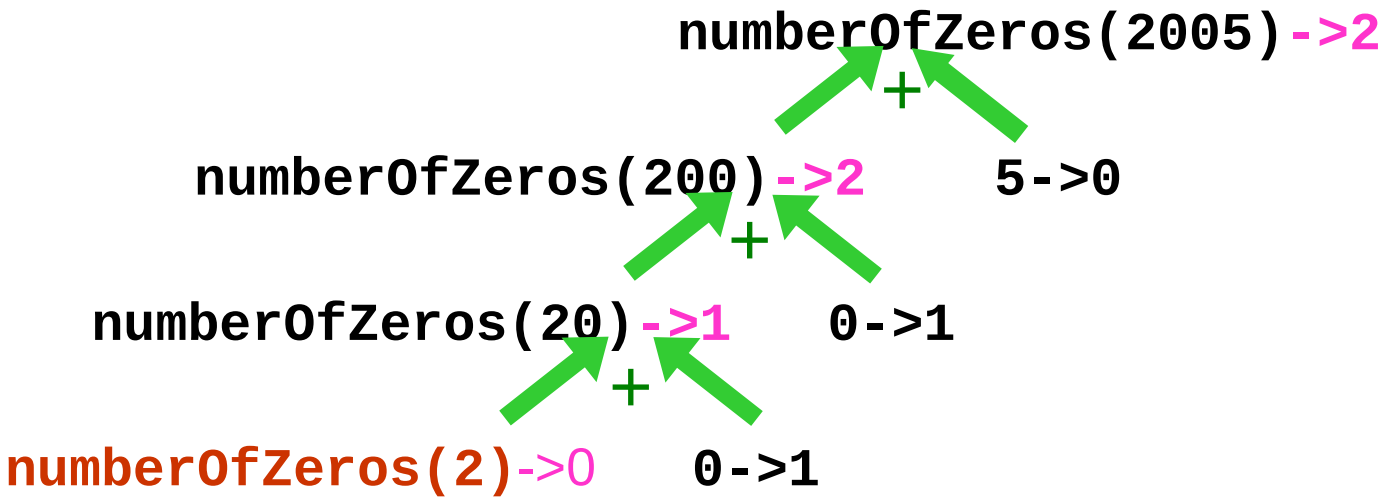
Exemplo 3: Números de zeros em um número



Decomposição

5. Erros de recursão

Exemplo 3: Números de zeros em um número



Composição

5. Erros de recursão

Exemplo 4: Números em palavras

- Processar um número inteiro e imprimir seus dígitos por extenso
- Entrada: 123
- Saída: "um dois três"

5. Erros de recursão

Exemplo 4: Números em palavras

- Processar um número inteiro e imprimir seus dígitos por extenso
- Entrada: 123
- Saída: "um dois três"

```
def int_to_words_naive(n: int):
    n_str = str(n)
    converted = [num2word(i) for i in n_str]
    return " ".join(converted)
```

5. Erros de recursão

Exemplo 4: Números em palavras

```
def int_to_words(n: int):
    if n < 10:
        return num2word(n)
    else:
        composition = " " + num2word(n % 10)
        return int_to_words(n//10) + composition
```

6. Busca binária

- Buscando por um valor em uma lista
  - **busca sequencial** e **busca binária** são dois algoritmos comuns
- Busca sequencial (conhecida como **busca linear**):
  - Não é muito eficiente
  - Fácil de entender e programar
- Busca binária:
  - mais eficiente que a busca sequencial
  - porém a **lista deve estar ordenada** primeiro!

6. Busca binária

- Compare os algoritmos de busca sequencial e busca binária:
  - Quantos elementos são eliminados da lista cada vez que um valor é lido da lista e não é o valor "alvo"?
    - Busca sequencial: *somente 1 item*
    - Busca binária: metade da lista!
- Por isso é chamada de busca binária:
  - cada teste mal sucedido para o valor buscado reduz a lista de busca restante pela metade!

6. Busca binária

Exemplo 5: Busca binária

```
def binary_search(array, target, first, last):
    if first <= last:
        mid = (first + last) // 2
        if target == array[mid]:
            return mid
        elif target < array[mid]:  # primeira metade
            return binary_search(array, target, first, mid - 1)
        else:  # segunda metade
            return binary_search(array, target, mid + 1, last)
    return -1

def search(array, value):
    return binary_search(array, value, 0, len(array)-1)

print(search([ 2, 3, 4, 10, 40 ], 2)) # 0
print(search([ 2, 3, 4, 10, 40 ], 10)) # 3
```

6. Busca binária

Exemplo 5: Busca binária

```
def binary_search(array, target,
    if first <= last:
        mid = (first + last) // 
        if target == array[mid]:
            return mid
        elif target < array[mid]:  # p          metade
            return binary_search(array        t, first, mid - 1)
        else:  # segunda metade
            return binary_search(arra         get, mid + 1, last)
    return -1

def search(array, value):
    return binary_search(array, valu   0, len(array)-1)

print(search([ 2, 3, 4, 10, 40 ], 2)) # 0
print(search([ 2, 3, 4, 10, 40 ], 10)) # 3
```

Onde está a composição?

6. Busca binária

Exemplo 5: Busca binária

- Se nenhum item for encontrado,
  - retorna "não encontrado" (-1).
- Caso contrário, se o alvo estiver no meio,
  - retorna a localização do meio.
- Senão,
  - retorna a localização encontrada pela busca (primeira metade) ou busca (segunda metade).

6. Busca binária

target é **33**

O array se parece com isso:

Indices	0	1	2	3	4	5	6	7	8	9
Contents	5	7	9	13	32	33	42	54	56	88

`Mid = (0 + 9) // 2 #` (resulta em 4)

`33 > a[mid]` (ou seja, `33 > a[4]`)

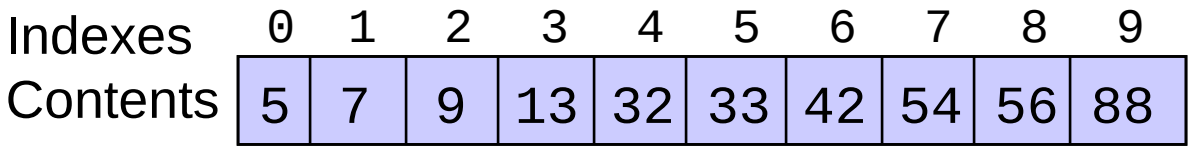
Portanto, se 33 está no array, então 33 é um dos:

					5	6	7	8	9
					33	42	54	56	88

Eliminou-se metade dos elementos restantes da consideração porque os elementos do array estão ordenados.

target is **33**

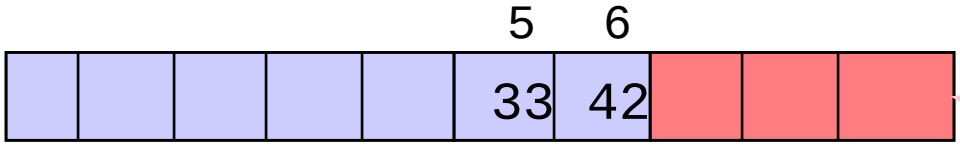
O array se parece com isso:



`mid = (5 + 9) / 2` (resulta em 7)

`33 < a[mid]` (ou seja, `33 < a[7]`)

Portanto, se 33 está no array, então 33 é um dos:

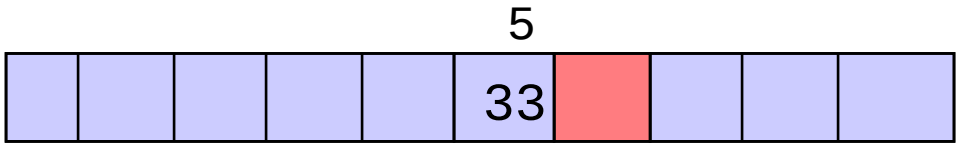


Elimine metade dos elementos restantes.

`mid = (5 + 6) / 2` (resulta em 5)

`33 == a[mid]`

Então encontramos 33 no índice 5:



Resumo

- *Chamada recursiva*: um método que chama a si mesmo
- Poderoso para o design de algoritmos
- Design de algoritmo recursivo:
  - Decomposição (problemas menores idênticos)
  - Composição (combinação de resultados)
  - Caso(s) base
- Implementação
  - Instruções condicionais (if) para separar diferentes casos

Resumo

- Implementação
  - Evite recursão infinita
    - O problema está ficando menor (decomposição)
    - Caso(s) base precisam existir e devem ser alcançáveis
  - Composição pode ser complicada


THANK
YOU
GRACIAS
ARIGATO
SHUKURIA
DANKSCHEEN
TASHAKKURATU
YAQHANYELAY
SUKSAMA
EKHMET
MEHRBANI
GRAZIE
MAAKE
KOMAPSUMNIDA
GOZAIMASHITA
EFCHARISTO
JUSPAXAR
PALDIES
BOLZÏN
MERCI
BÏYAN
SHUKRIA
TINGKI

Dúvidas?