

SME0827 - Estruturas de Dados

Recursividade Aula 05

Professor: André C. P. L. F. de Carvalho, ICMC-USP
PAE: Moisés Rocha dos Santos
Monitor:

© André de Carvalho - ICMC/USP

1

1

Aula de hoje

- Recursão
- Função fatorial
- Função de Fibonacci
- Outros exemplos recursivos
- Trabalhando com recursão
- Conclusão

© André de Carvalho - ICMC/USP

2

2

Introdução

- Muitas estratégias de programação têm contraexemplos fora da computação
 - Iteração (*for*, *while*)
 - Usada quando alguém esta realizando uma tarefa repetidamente
 - Controle condicional (*if*)
 - Usado quando alguém toma uma decisão,
 - Uma estratégia muito poderosa tem poucos contraexemplos no dia-a-dia: a recursão

© André de Carvalho - ICMC/USP

3

3

Introdução

CARRIAGEPOST.COM.AU SATURDAY JANUARY 25 2020

NEWS 05

Crypto collapse costly

Cairns investors among those caught out by scheme

PETE MARTINELLI
A PONDZI scheme that managed to lure in thousands of investors has collapsed in Cairns, leaving many investors with significant losses. The scheme, known as the 'Pondzi' scheme, was run by a man known as 'Pondzi' who claimed to be a wealthy investor. He had been in Cairns for some time, and had been seen at various events. The scheme was based on the idea of 'Pondzi' investing in various assets, and then passing on the profits to his investors. However, the scheme was based on a lie, and the investors were left with nothing. The scheme was exposed by a local newspaper, and the investors were left with significant losses. The scheme was based on the idea of 'Pondzi' investing in various assets, and then passing on the profits to his investors. However, the scheme was based on a lie, and the investors were left with nothing. The scheme was exposed by a local newspaper, and the investors were left with significant losses.

Pyramid scheme

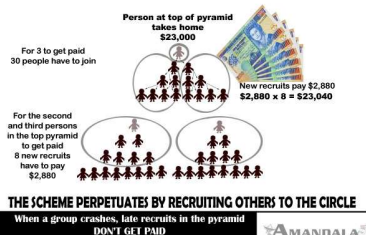
© André de Carvalho - ICMC/USP

4

4

Introdução

DREAM CATCHERS



© André de Carvalho - ICMC/USP

5

5

Introdução



East Oregonian,
November 16, 1916.
Chronicling
America archives at
the Library of
Congress


© André de Carvalho - ICMC/USP

6

6

Recursão

- Estratégia de solução de problemas complicados
 - Redução deles para problemas menores **do mesmo tipo**
- Permite escrever programas complexos em uma forma simples e elegante



© André de Carvalho - ICMC/USP 7

7

Coleta de recursos

- Você precisa coletar R\$ 1.000.000,00 para uma ONG
- Possíveis alternativas:
 - Encontrar alguém que concorde em doar toda esta quantia?
 - Quase impossível
 - Encontrar 100.000 pessoas que doem R\$ 10,00 cada?
 - Muito difícil

© André de Carvalho - ICMC/USP 8

8

Exemplo

- O que fazemos quando uma tarefa que excede nossa capacidade?
 - Pedimos ajuda

© André de Carvalho - ICMC/USP 9

9

Exemplo

- Possível solução:
 - Dividir a tarefa entre 10 colaboradores
 - Cada colaborador deve arrecadar R\$ 100.000,00
 - Cada colaborador pode dividir sua tarefa chamando 10 novos colaboradores
 - O processo pode ser repetido até que cada colaborador precise arrecadar apenas R\$ 10,00
 - Não será mais necessário delegar tarefas para outros

© André de Carvalho - ICMC/USP 10

10

Exemplo em pseudocódigo

```
def ColetaContrib (n):
  if (n <= 10):
    Coletar o dinheiro de um simples doador
  else:
    Encontrar 10 colaboradores
    Conseguir que cada colaborador arrecade n/10 Reais
    Juntar o dinheiro arrecadado pelos colaboradores
```

© André de Carvalho - ICMC/USP 11

11

Exemplo

```
def ColetaContrib (n):
  if (n <= 10):
    Coletar o dinheiro de um simples doador
  else:
    Encontrar 10 colaboradores
    ColetaContrib (n/10)
    Juntar o dinheiro arrecadado pelos colaboradores
```

© André de Carvalho - ICMC/USP 12

12

Exemplo

- Observações
 - Função *ColetaContrib ()* chama ela mesma se a contribuição for maior que R\$ 10,00
 - Cada vez que a função é chamada, valor a ser arrecadado, *n*, é dividido por 10
 - Em programação, uma função chamar ela mesma define uma recursão
 - Função recursiva
 - Estratégia recursiva

© André de Carvalho - ICMC/USP 13

13

Estratégia recursiva

Def função recursiva:
if (problema é simples):
 Computar uma solução sem usar recursão
else:
 Quebrar o problema em subproblemas do mesmo tipo
 Resolver cada subproblema, chamando função recursiva
 Combinar as soluções dos subproblemas para a solução do problema inicial

© André de Carvalho - ICMC/USP 14

14

Estratégia recursiva

- Para usá-la em problemas de computação é necessário identificar:
 - Caso(s) simples
 - Para os quais a resposta é facilmente determinada
 - Decomposição recursiva
 - Que permite quebrar um problema complexo em problemas mais simples do mesmo tipo

© André de Carvalho - ICMC/USP 15

15

Estratégia recursiva

- Soluções recursivas
 - Problemas são divididos em subproblemas cada vez menores
 - Até que se tornem simples o suficiente para serem resolvidos imediatamente
 - Sem recorrer a nova subdivisão
 - Exemplos de estratégias de "divisão e conquista"
 - Resolvem problemas difíceis dividindo eles em problemas mais simples

© André de Carvalho - ICMC/USP 16

16

Função fatorial

```
def fatorial(n):
    if (n == 0) or (n == 1):
        return 1
    elif n < 0:
        return "Não existe"
    else:
        fat = n
        while n > 1:
            n = n - 1
            fat = fat*n
        return fat
```

```
>>> x = fatorial(4)
>>> print(x)
24
```

© André de Carvalho - ICMC/USP 17

17

Função fatorial

- Função fatorial pode ser descrita de forma recursiva: $n! = n*(n-1)!$
 - Ex.: $4! = 4*3!$

$$n! = \begin{cases} 1 & \text{Se } n = 0 \text{ ou } n = 1 \\ n*(n-1)! & \text{Caso contrário} \end{cases}$$

© André de Carvalho - ICMC/USP 18

18

Função fatorial

```
def fatorial (n):
    if (n == 0) or (n == 1):
        return 1
    elif n < 0:
        return "Não existe"
    else:
        return n*fatorial (n-1)
```

```
>>> x = fatorial(4)
>>> print (x)
24
```

© André de Carvalho - ICMC/USP 19

19

Função fatorial

- Solução iterativa X Recursiva

```
def fatorial (n):
    if (n == 0) or (n == 1):
        return 1
    elif n < 0:
        return "Não existe"
    else:
        fat = n
        while n > 1:
            n = n - 1
            fat = fat*n
        return fat
```

```
def fatorial (n):
    if (n == 0) or (n == 1):
        return 1
    elif n < 0:
        return "Não existe"
    else:
        return n*fatorial (n-1)
```

© André de Carvalho - ICMC/USP 20

20

Função fatorial

- Solução iterativa X Recursiva

```
def fatorial (n):
    if (n == 0) or (n == 1):
        return 1
    else:
        fat = n
        while n > 1:
            n = n - 1
            fat = fat*n
        return fat
```

```
def fatorial (n):
    if (n == 0) or (n == 1):
        return 1
    else:
        return n*fatorial (n-1)
```

© André de Carvalho - ICMC/USP 21

21

Função fatorial

- Rastreando o processo recursivo
 - Rastrear a função *fatorial* (*n*) para *n* = 4
 - fatorial* (4)

```
fatorial
n
4
if (n == 0) or (n == 1):
    return 1
else:
    return n*fatorial (n - 1)
```

© André de Carvalho - ICMC/USP 22

22

Chamada

```
fatorial
n
4
if (n == 0) or (n == 1):
    return 1
else:
    return n*fatorial (n - 1)
```

© André de Carvalho - ICMC/USP 23

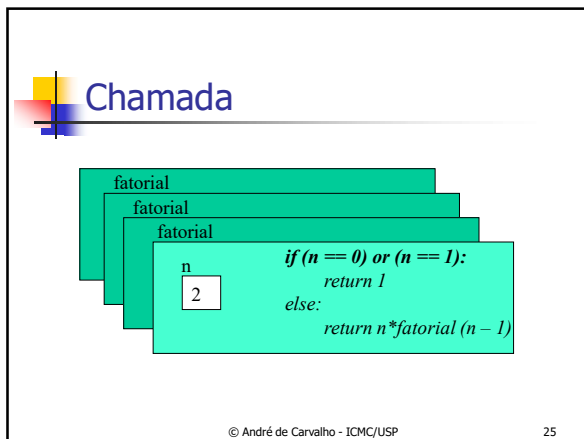
23

Chamada

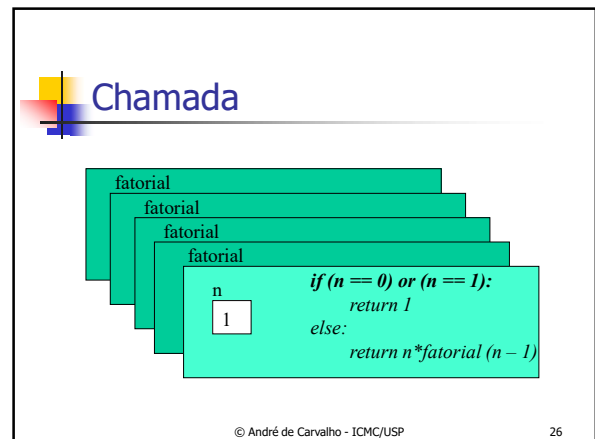
```
fatorial
fatorial
n
3
if (n == 0) or (n == 1):
    return 1
else:
    return n*fatorial (n - 1)
```

© André de Carvalho - ICMC/USP 24

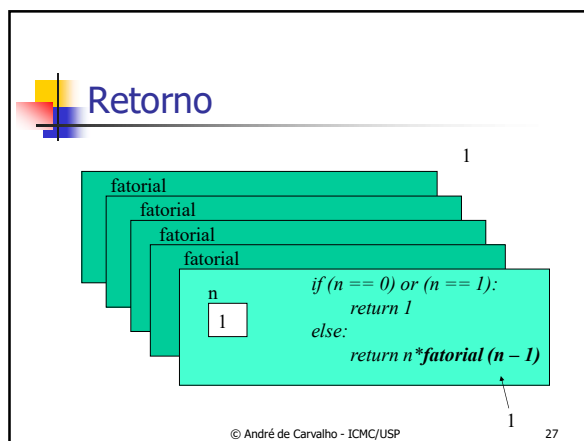
24



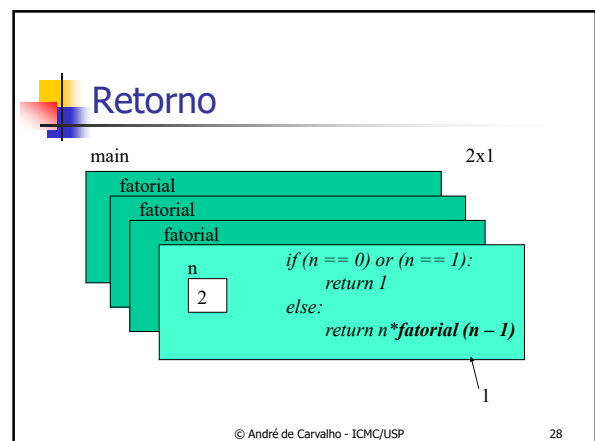
25



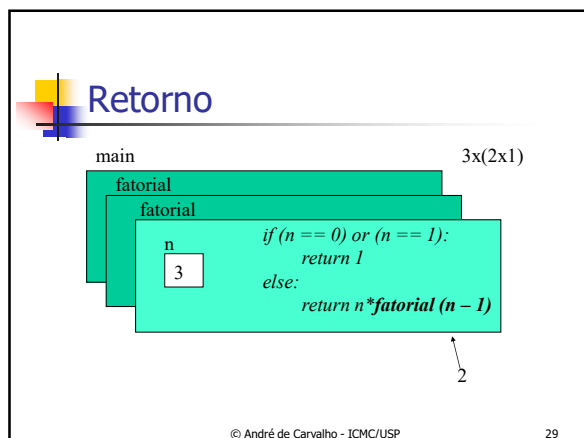
26



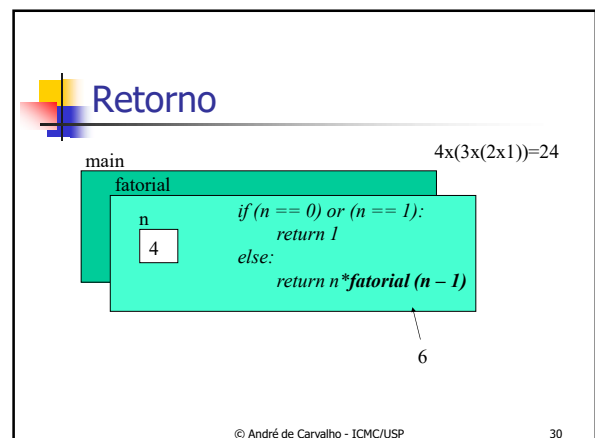
27



28



29



30

Voto de confiança recursivo

- Voto de confiança recursivo
 - Por de lado os detalhes e focar a atenção na operação sendo realizada
 - Assumir que a chamada recursiva com argumentos mais simples que os argumentos originais funcionará corretamente
 - Ex. fatorial (4) = 4 x fatorial (3)
 - Como fatorial (3) é mais simples que fatorial (4), assume-se que fatorial (3) funcionará corretamente (3! = 6)
 - Resultado de fatorial (4) = 4x6 = 24

© André de Carvalho - ICMC/USP 31

31

Função de Fibonacci

- Outro exemplo de função recursiva
 - Gera a sequência de Fibonacci
 - 0 1 1 2 3 5 8 13 21 34 55 89 ...
 - $t(n) = t(n-1) + t(n-2)$ (Relação de recorrência)
 - Relação de recorrência
 - Cada elemento de uma sequência é definido em termos de elementos anteriores

© André de Carvalho - ICMC/USP 32

32

Função de Fibonacci

- Só a relação de recorrência não é suficiente para definir a sequência de Fibonacci
 - É necessário também definir um ponto inicial (casos simples)
 - Pelo menos os dois termos iniciais: $t(0)$ e $t(1)$

$$t(n) = \begin{cases} n & \text{Se } n = 0 \text{ ou } n = 1 \\ t(n-1) + t(n-2) & \text{Caso contrário} \end{cases}$$

© André de Carvalho - ICMC/USP 33

33

Função de Fibonacci

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

```
>>> x = fibonacci(6)
>>> print(x)
24
```

© André de Carvalho - ICMC/USP 34

34

Função de Fibonacci

A recursion tree for calculating Fib(5). The root is Fib(5), which branches into Fib(4) and Fib(3). Fib(4) branches into Fib(3) and Fib(2). Fib(3) branches into Fib(2) and Fib(1). Fib(2) branches into Fib(1) and Fib(0). The tree shows the sequence of recursive calls.

© André de Carvalho - ICMC/USP 35

35

Função de Fibonacci

A recursion tree for calculating Fib(5), similar to the one in slide 35. However, several subproblems are circled with dashed lines to highlight overlapping subproblems: Fib(3), Fib(2), and Fib(1). These subproblems are repeated multiple times in the tree, illustrating the inefficiency of naive recursion.

© André de Carvalho - ICMC/USP 36

36

Função de Fibonacci

- Implementação ineficiente
 - Problema não é da recursão, mas de como a recursão foi utilizada
 - Sequência de Fibonacci é uma sequência aditiva
 - Cada novo número é formado pela soma de números anteriores
 - Existe uma forma mais eficiente de implementar sequências aditivas

© André de Carvalho - ICMC/USP 37

37

Função de Fibonacci

- Valor de termos em uma sequência aditiva
 - Seja a sequência:

t(0)	t(1)	t(2)	t(3)	t(4)	t(5)	t(6)	t(7)
0	1	1	2	3	5	8	13

 - Computar t(0) e t(1) é trivial
 - Como definir o sexto termo na sequência?
 - Valor do n-ésimo termo em uma sequência aditiva
 - É o (n-1)-ésimo termo na sequência que começa um passo depois de t(0)
 - Começam t(1)

© André de Carvalho - ICMC/USP 38

38

Função de Fibonacci

```
def fibonacci(n):
    return sequenciaAditiva(n, 0, 1) } Função
                                     empacotadora

def sequenciaAditiva(n, t0, t1):
    if n == 0:
        return t0
    if n == 1:
        return t1
    return sequenciaAditiva(n-1, t1, t0 + t1)
```

```
>>> x = fibonacci(5)
>>> print(x)
5
```

© André de Carvalho - ICMC/USP 39

39

Exemplo

- fibonacci (5)

```
fibonacci (5) = SequenciaAditiva (5, 0, 1)
               = SequenciaAditiva (4, 1, 1)
               = SequenciaAditiva (3, 1, 2)
               = SequenciaAditiva (2, 2, 3)
               = SequenciaAditiva (1, 3, 5)
               = 5
```

© André de Carvalho - ICMC/USP 40

40

Outros exemplos de recursão

- Recursão também é aplicada em funções não matemáticas
 - Detecção de palíndromos
 - Busca binária
 - Recursão mútua

© André de Carvalho - ICMC/USP 41

41

Detecção de palíndromos

- Palíndromos
 - Strings em que a leitura da esquerda para a direita é igual à leitura da direita para a esquerda
 - Exemplos: level, noon, oco, ana, boscoocsob, ilaiali, socorram-me, subi no onibus em marrocos
 - Para descobrir se um string é um palíndromo
 - Checar se o primeiro e o último caracteres são iguais
 - Checar se o sub-string gerado retirando o primeiro e o último caracteres é um palíndromo

© André de Carvalho - ICMC/USP 42

42

Detecção de palíndromos

- Exemplos (ignorar pontuações e espaços)
 - Madam, I'm Adam.
 - Able was I ere I saw Elba.
 - A man, a plan, a canal; Panama.
 - Go hang a salami, I'm a lasagna hog.
 - Doc note, I dissent, a fast never prevents a fatness; I diet on cod.
 - Satan oscillate my metallic sonatas.
 - No sir, away, a papaya war is on!

© André de Carvalho - ICMC/USP 43

43

Detecção de palíndromos

- Implementação recursiva
 - Casos simples
 - Palavra vazia é um palíndromo
 - Palavra com apenas um caracter é um palíndromo
 - Decomposição
 - Se o primeiro e último caracteres são iguais
 - Retirar o primeiro e último caracteres e testar o *string* resultante

© André de Carvalho - ICMC/USP 44

44

Detecção de palíndromos

- Implementação eficiente
 - Calcular o tamanho do argumento do tipo *string* apenas uma vez
 - Nas chamadas recursivas, basta subtrair 2 do tamanho do *string*
 - Utilizar operadores para manipulação de *strings*

© André de Carvalho - ICMC/USP 45

45


Recursão mútua

- Recursão pode ocorrer de forma indireta
 - Função f chama uma função g que chama f ... (recursão mútua)
 - Ex.: definir se um número natural n é par ou ímpar
 - n é par se seu predecessor for ímpar
 - n é ímpar se seu predecessor for par
 - 0 é par (caso simples)

© André de Carvalho - ICMC/USP 46

46

Perguntas



© André de Carvalho - ICMC/USP 47

47

Exercício

- Escrever um método que:
 - Dado um número inteiro positivo N
 - Retorne a soma dos números de 1 a N elevados ao quadrado
 - $1^2 + 2^2 + 3^2 + \dots$
 - Duas versões
 - Utilizar iteração
 - Utilizar recursão

© André de Carvalho - ICMC/USP 48

48

Exercício

- Escrever uma função recursiva para calcular o valor de x^y
 - Tanto x quanto y podem ser valores reais ou inteiros
 - Se y for real, ele é um valor com parte fracionária (n.0)

49

Exercício

```
def potencia (base,exp):  
    if (0 < exp < 1):  
        return base**exp  
    if exp == 0:  
        return 1  
    else:  
        return base*potencia(base,exp-1)
```

```
>>> x = potencia(2.0,3.1)  
>>> print (x)  
8.574187700290345
```

50

Exercício

- Escrever um método que:
 - Dado um número inteiro positivo N
 - Retorne a soma dos números de 1 a N elevados ao quadrado
 - Duas versões
 - Utilizar iteração
 - Utilizar recursão

51

Exercício iterativo

```
def soma (n):  
    res = 1  
    for i in range(2,n+1):  
        res = res + i*i  
    return res
```

```
>>> x = soma(5)  
>>> print (x)  
55
```

52

Exercício recursivo

```
def soma (n):  
    if n == 1:  
        return 1  
    return n*n + soma (n-1)
```

```
>>> x = soma(5)  
>>> print (x)  
55
```

53

Exercício

- Utilizando recursão mútua, escrever programa que define se um número é par ou ímpar

54



Exercício

```
def par(n):  
    if n == 0:  
        return True  
    else:  
        return impar (n-1)  
  
def impar(n):  
    if n == 0:  
        return False  
    else:  
        return par(n - 1)  
  
x = int(input("Escreva um numero: "))  
print (par(x))
```

© André de Carvalho - ICMC/USP

55