MC102 – Aula 25
Recursão

Instituto de Computação – Unicamp

9 de Outubro de 2015

Roteiro

Recursão – Indução

- Devemos criar uma algoritmo para resolver um determinado problema.
- Usando o método de recursão/indução, a solução de um problema pode ser expressa da seguinte forma:
  - Primeiramente, definimos a solução para casos básicos;
  - Em seguida, definimos como resolver o problema para um caso geral, utilizando-se de soluções para instâncias menores do problema.

Recursão – Indução

- Devemos criar uma algoritmo para resolver um determinado problema.
- Usando o método de recursão/indução, a solução de um problema pode ser expressa da seguinte forma:
  - Primeiramente, definimos a solução para casos básicos;
  - Em seguida, definimos como resolver o problema para um caso geral, utilizando-se de soluções para instâncias menores do problema.

Recursão – Indução

- Devemos criar uma algoritmo para resolver um determinado problema.
- Usando o método de recursão/indução, a solução de um problema pode ser expressa da seguinte forma:
  - Primeiramente, definimos a solução para casos básicos;
  - Em seguida, definimos como resolver o problema para um caso geral, utilizando-se de soluções para instâncias menores do problema.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- **Indução:** Técnica de demonstração matemática onde algum parâmetro da proposição a ser demonstrada envolve números naturais.
- Seja T uma proposição que desejamos provar como verdadeira para todos valores naturais n.
- Ao invés de provar diretamente que T é válida para todos os valores de n, basta provar as duas condições 1 e 3 a seguir:

1. **Passo base:** PROVAR que T é válido para $n = 1$.
2. **Hipótese de Indução:** Assumimos que T é válido para $n - 1$.
3. **Passo de Indução:** Sabendo que T é válido para $n - 1$ devemos PROVAR que T é válido para n.

Indução

- Por que a indução funciona? Por que as duas condições são suficientes?
  - Mostramos que T é valida para um caso base como $n = 1$.
  - Com o passo da indução, automaticamente mostramos que T é válida para $n = 2$.
  - Como T é válida para $n = 2$, pelo passo de indução, T também é válida para $n = 3$, e assim por diante.

Exemplo

Teorema

A soma $S(n)$ dos primeiros n números naturais é $n(n+1)/2$

Prova.

Base: Para $n = 1$ devemos mostrar que $n(n+1)/2 = 1$. Isto é verdade:

$$1(1+1)/2 = 1.$$

Hip. de Indução: Vamos assumir que é válido para $(n-1)$, ou seja, $S(n-1) = (n-1)((n-1)+1)/2$.

Passo: Devemos mostrar que é válido para n, ou seja, devemos mostrar que $S(n) = n(n+1)/2$. Por definição, $S(n) = S(n-1) + n$ e por hipótese $S(n-1) = (n-1)((n-1)+1)/2$, logo

$$\begin{aligned} S(n) &= S(n-1) + n \\ &= (n-1)((n-1)+1)/2 + n \\ &= n(n-1)/2 + 2n/2 \\ &= n(n+1)/2 \end{aligned}$$

Exemplo

Teorema

A soma $S(n)$ dos primeiros n números naturais é $n(n+1)/2$

Prova.

Base: Para $n = 1$ devemos mostrar que $n(n+1)/2 = 1$. Isto é verdade:

$$1(1+1)/2 = 1.$$

Hip. de Indução: Vamos assumir que é válido para $(n-1)$, ou seja, $S(n-1) = (n-1)((n-1)+1)/2$.

Passo: Devemos mostrar que é válido para n, ou seja, devemos mostrar que $S(n) = n(n+1)/2$. Por definição, $S(n) = S(n-1) + n$ e por hipótese $S(n-1) = (n-1)((n-1)+1)/2$, logo

$$\begin{aligned} S(n) &= S(n-1) + n \\ &= (n-1)((n-1)+1)/2 + n \\ &= n(n-1)/2 + 2n/2 \\ &= n(n+1)/2 \end{aligned}$$

Exemplo

Teorema

A soma $S(n)$ dos primeiros n números naturais é $n(n+1)/2$

Prova.

Base: Para $n = 1$ devemos mostrar que $n(n+1)/2 = 1$. Isto é verdade:

$$1(1+1)/2 = 1.$$

Hip. de Indução: Vamos assumir que é válido para $(n-1)$, ou seja, $S(n-1) = (n-1)((n-1)+1)/2$.

Passo: Devemos mostrar que é válido para n, ou seja, devemos mostrar que $S(n) = n(n+1)/2$. Por definição, $S(n) = S(n-1) + n$ e por hipótese $S(n-1) = (n-1)((n-1)+1)/2$, logo

$$\begin{aligned} S(n) &= S(n-1) + n \\ &= (n-1)((n-1)+1)/2 + n \\ &= n(n-1)/2 + 2n/2 \\ &= n(n+1)/2 \end{aligned}$$

Recursão

- Definições recursivas de funções funcionam como o *princípio matemático da indução* que vimos anteriormente.
- A idéia é que a solução de um problema pode ser expressa da seguinte forma:
  - Definimos a solução para casos básicos;
  - Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.

Fatorial

Problema: Calcular o fatorial de um número ($n!$).
Qual o caso base e o passo da indução?

- Se n é igual a 1, então o fatorial é 1.

Qual seria o passo indutivo?

- Temos que expressar a solução para $n > 1$, supondo que já sabemos a solução para algum caso mais simples.
- $n! = n * (n-1)!$.

Este caso é trivial pois a própria definição do fatorial é recursiva.

Fatorial

Problema: Calcular o fatorial de um número ($n!$).
Qual o caso base e o passo da indução?

- Se n é igual a 1, então o fatorial é 1.

Qual seria o passo indutivo?

- Temos que expressar a solução para $n > 1$, supondo que já sabemos a solução para algum caso mais simples.
- $n! = n * (n - 1)!$.

Este caso é trivial pois a própria definição do fatorial é recursiva.

Fatorial

Problema: Calcular o fatorial de um número ($n!$).
Qual o caso base e o passo da indução?

- Se n é igual a 1, então o fatorial é 1.

Qual seria o passo indutivo?

- Temos que expressar a solução para $n > 1$, supondo que já sabemos a solução para algum caso mais simples.
- $n! = n * (n-1)!$.

Este caso é trivial pois a própria definição do fatorial é recursiva.

Fatorial

Portanto, a solução do problema **pode ser expressa de forma recursiva** como:

- Se $n = 1$ então $n! = 1$.
- Se $n > 1$ então $n! = n * (n - 1)!$.

Note como aplicamos o princípio da indução:

- Sabemos a solução para um caso base: $n = 1$.
- Definimos a solução do problema geral $n!$ em termos do mesmo problema só que para um caso menor ($(n - 1)!$).

Fatorial em Python

```
def fatorial(n):
   if n==1:
      return 1
   else:
      x = n-1
      r = fatorial(x)
      return n*r
```

Fatorial

- Para solucionar o problema, é feita uma chamada para a própria função, por isso, esta função é chamada *recursiva*.
- Recursividade geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema é recursivo por natureza.

Fatorial

- Para solucionar o problema, é feita uma chamada para a própria função, por isso, esta função é chamada *recursiva*.
- Recursividade geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema é recursivo por natureza.

O que acontece na memória

- Precisamos entender como é feito o controle sobre as variáveis locais em chamadas recursivas.
- A memória de um sistema computacional é dividida em três partes:
  - **Espaço Estático**: Contém as variáveis globais e código do programa.
  - **Heap:** Para alocação dinâmica de memória.
  - **Pilha:** Para execução de funções.

O que acontece na memória

O que acontece na pilha:

- Toda vez que uma função é invocada, suas variáveis locais são armazenadas no topo da pilha.
- Quando uma função termina a sua execução, suas variáveis locais são removidas da pilha.

Considere o exemplo:

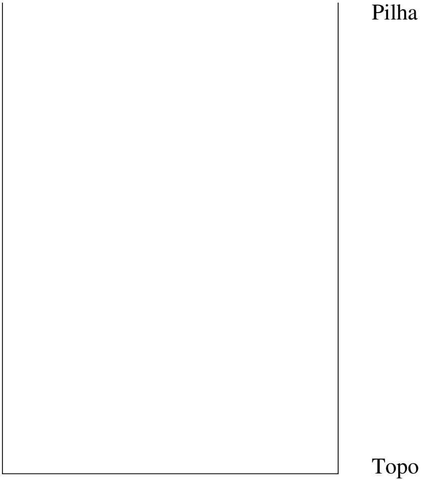
```
def f1(a, b):
   c=5
   return (c+a+b)

def f2(a, b):
   c = f1(b, a)
   return c

f2(2, 3)
```

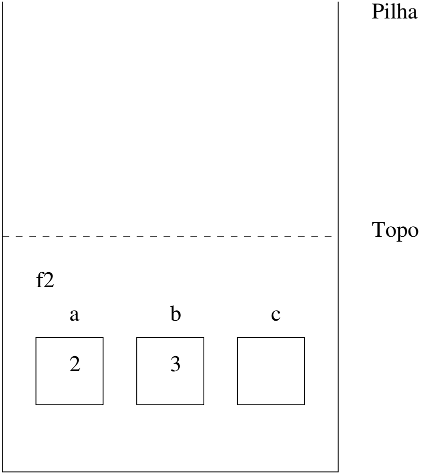
O que acontece na memória

Inicialmente a pilha está vazia.



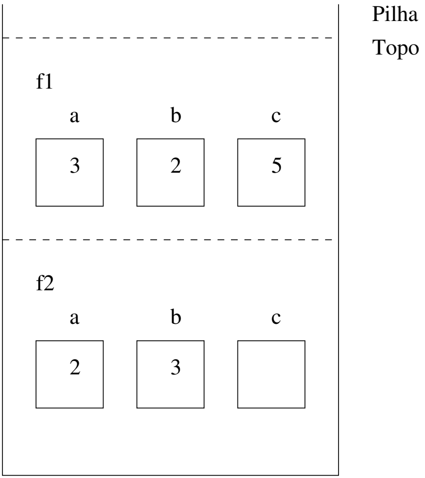
O que acontece na memória

Quando **f2(2,3)** é invocada, suas variáveis locais são alocadas no topo da pilha.



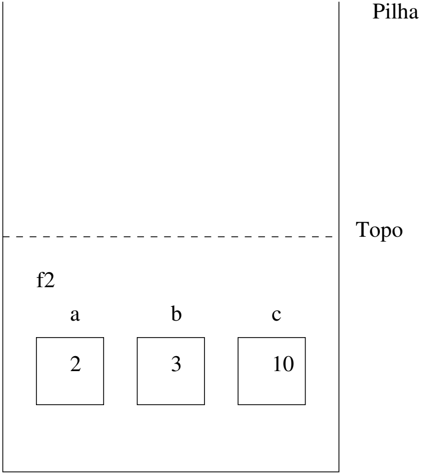
O que acontece na memória

A função **f2** invoca a função **f1(b,a)** e as variáveis locais desta são alocadas no topo da pilha sobre as de **f2**.



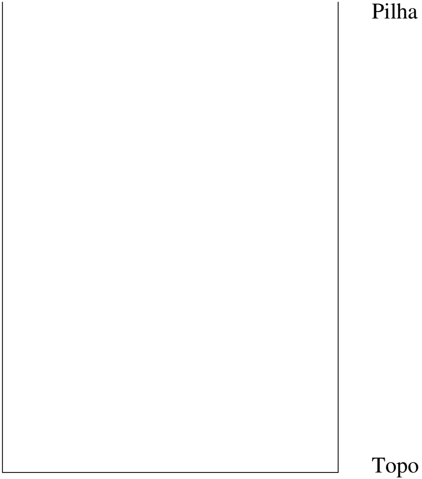
O que acontece na memória

A função **f1** termina, devolvendo 10. As variáveis locais de **f1** são removidas da pilha.



O que acontece na memória

Finalmente **f2** termina a sua execução devolvendo 10. Suas variáveis locais são removidas da pilha.



O que acontece na memória

No caso de chamadas recursivas para uma mesma função, é como se cada chamada correspondesse a uma função distinta.

- As execuções das chamadas de funções recursivas são feitas na pilha, assim como qualquer função.
- O último conjunto de variáveis alocadas na pilha, que está no topo, corresponde às variáveis da última chamada da função.
- Quando termina a execução de uma chamada da função, as variáveis locais desta são removidas da pilha.

Usando recursão em programação

Considere novamente a solução recursiva para se calcular o fatorial e assuma que seja feito a chamada **fatr(4)**.

```
def fatr(n):
    if n == 1: #Passo Básico
      return 1
    else:
      x = n-1
      r = fatr(x) //Sabendo o fatorial de (n-1)
      return n * r  //calculamos o fatorial de n
```

O que acontece na memória

- Cada chamada da função *fatr* cria novas variáveis locais de mesmo nome (n, x, r) .
- Portanto, várias variáveis *n*, *x* e *r* podem existir em um dado momento.
- Em um dado instante, o nome *n* (ou *x* ou *r*) refere-se à variável local ao corpo da função que está sendo executada naquele instante.

O que acontece na memória

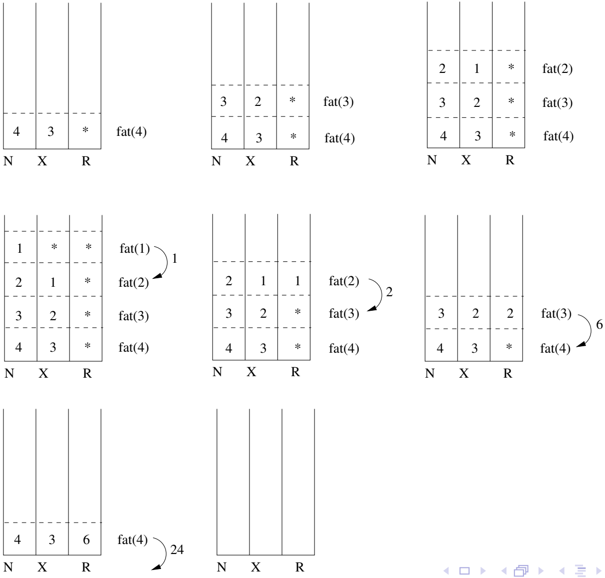
- Cada chamada da função *fatr* cria novas variáveis locais de mesmo nome (n, x, r) .
- Portanto, várias variáveis n, x e r podem existir em um dado momento.
- Em um dado instante, o nome n (ou x ou r) refere-se à variável local ao corpo da função que está sendo executada naquele instante.

O que acontece na memória

- Cada chamada da função *fatr* cria novas variáveis locais de mesmo nome (n, x, r) .
- Portanto, várias variáveis n, x e r podem existir em um dado momento.
- Em um dado instante, o nome n (ou x ou r) refere-se à variável local ao corpo da função que está sendo executada naquele instante.

O que acontece na memória

Estado da Pilha de execução para *fatr(4)*.



O que acontece na memória

- É claro que as variáveis x e r são desnecessárias.
- Você também deveria testar se n não é negativo!

```
def fatr (n):
    if n <= 1: #Passo Básico
      return 1
    else:
      return n* fatr(n-1)
```

Recursão × Iteração

- Soluções recursivas são geralmente mais concisas que as iterativas. Programas mais simples.
- Soluções iterativas em geral têm a memória limitada enquanto as recursivas, não.
- Cópia dos parâmetros a cada chamada recursiva é um custo adicional para as soluções recursivas.

Recursão × Iteração

Neste caso, uma solução iterativa é mais eficiente. Por quê?

```
def fat(n):
  r = 1
  for i in range(1,n+1):
      r = r * i
  return r
```

Exemplo: Soma de elementos de uma lista

- Suponha que temos uma lista v de números de tamanho $n+1$, e queiramos saber a soma dos seus elementos da posição 0 até n.
- Como podemos descrever este problema de forma recursiva? Isto é, como podemos descrever este problema em função de si mesmo?
- Vamos denotar por $S(n)$ a soma dos elementos das posições 0 até n da lista. Com isso temos:
  - Se $n = 0$ então a soma é igual a $v[0]$.
  - Se $n > 0$ então a soma é igual a $v[n] + S(n-1)$.

Exemplo: Soma de elementos de uma lista

- Suponha que temos uma lista v de números de tamanho $n + 1$, e queiramos saber a soma dos seus elementos da posição 0 até n.
- Como podemos descrever este problema de forma recursiva? Isto é, como podemos descrever este problema em função de si mesmo?
- Vamos denotar por $S(n)$ a soma dos elementos das posições 0 até n da lista. Com isso temos:
  - Se $n = 0$ então a soma é igual a $v[0]$.
  - Se $n > 0$ então a soma é igual a $v[n] + S(n - 1)$.

Exemplo: Soma de elementos de uma lista

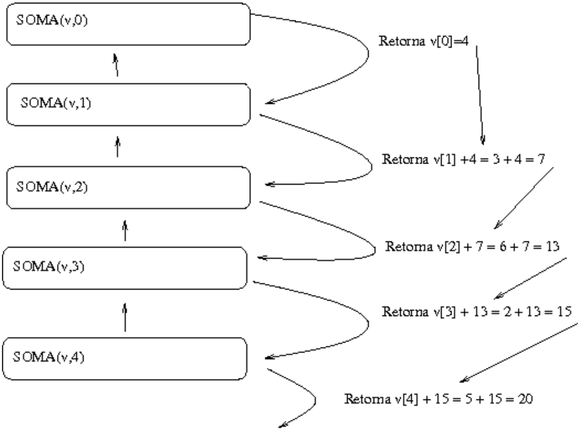
- Suponha que temos uma lista v de números de tamanho $n+1$, e queiramos saber a soma dos seus elementos da posição 0 até n.
- Como podemos descrever este problema de forma recursiva? Isto é, como podemos descrever este problema em função de si mesmo?
- Vamos denotar por $S(n)$ a soma dos elementos das posições 0 até n da lista. Com isso temos:
  - Se $n = 0$ então a soma é igual a $v[0]$.
  - Se $n > 0$ então a soma é igual a $v[n] + S(n-1)$.

Algoritmo em Python

```
def soma(v,  n):
  if n == 0:
     return v[0]
  else:
     return v[n] + soma(v,n-1)
```

Exemplo de execução

É claro que o você deve se certificar de usar valores válidos de n.
$V = [4, 3, 6, 2, 5]$



Soma do vetor recursivo

- O método recursivo sempre termina:
  - Existência de um caso base.
  - A cada chamada recursiva do método temos um valor menor de n.

Algoritmo em Python

Neste caso, a solução iterativa também seria melhor (não há criação de variáveis das chamadas recursivas):

```
def calcula_soma(v,n):
    soma=0
    for i in range(n+1):
        soma = soma + v[i]
    return soma
```

Fibonacci

- A série de fibonacci é a seguinte:
  - $1, 1, 2, 3, 5, 8, 13, 21, \ldots$.
- Queremos determinar qual é o n-ésimo (fibo(n)) número da série.
- Como descrever o n-ésimo número de fibonacci de forma recursiva?

Fibonacci

- No caso base temos:
  - Se $n = 1$ ou $n = 2$ então $\text{fibo}(n) = 1$.
- Sabendo casos anteriores podemos computar $\text{fibo}(n)$ como:
  - $\text{fibo}(n) = \text{fibo}(n-1) + \text{fibo}(n-2)$.

Programa

A definição anterior é traduzida diretamente em um algoritmo em Python:

```
def fibo(n):
  if n <= 2:
    return 1
  else:
    return fibo(n-1) + fibo(n-2)
```

Relembrando

- Recursão é uma técnica para se criar algoritmos onde:
  1. Devemos descrever soluções para casos básicos.
  2. Assumindo a existência de soluções para casos menores, mostramos como obter a solução para o caso maior.
- Algoritmos recursivos geralmente são mais claros e concisos.
- Implementador deve avaliar clareza de código $\times$ eficiência do algoritmo.

Exercício

Mostre a execução da função recursiva **imprime** abaixo:
O que será impresso?

```
def imprime(v,i,n):
   if i==n:
     print(v[i],end=" ")
   else:
     imprime(v,i+1,n)
     print(v[i],end=" ")

vet = [1,2,3,4,5,6,7,8,9,10]
imprime(vet, 0, 9)
print()
```

Exercício

- Mostre o estado da pilha de memória durante a execução da função **fibo** com a chamada **fib(5)**.
- Qual versão é mais eficiente para se calcular o *n*-ésimo número de fibonacci? A recursiva ou iterativa?