MC102 – Aula 27
Recursão II

Eduardo C. Xavier

Instituto de Computação – Unicamp

13 de Junho de 2018

Roteiro

Recursão - Relembrando

- Definições recursivas de funções são baseadas no *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 os casos básicos;
  - Definimos como resolver o problema geral utilizando soluções do mesmo problema só que para casos menores.

Cálculo de Potências

Suponha que temos que calcular x^n para n inteiro positivo. Como calcular de forma recursiva?
x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

Suponha que temos que calcular x^n para n inteiro positivo. Como calcular de forma recursiva?
x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

Suponha que temos que calcular x^n para n inteiro positivo. Como calcular de forma recursiva?
x^n é:

- 1 se $n = 0$.
- xx^{n-1} caso contrário.

Cálculo de Potências

```
def pot1(x, n):
        assert type(n) == int and n >= 0
        if n == 0:
                return 1
        else:
                return x*pot1(x, n-1)
```

Cálculo de Potências

Neste caso a solução iterativa é mais eficiente.

```
def pot2(x, n):
        assert type(n) == int and n >= 0
        p = 1
        for i in range(n):
                p = p * x
        return p
```

- O laço é executado n vezes.
- Na solução recursiva são feitas n chamadas recursivas, mas tem-se o custo adicional para criação/remoção de variáveis locais na pilha.

Cálculo de Potências

Mas e se definirmos a potência de forma diferente?
x^n é:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note que aqui também definimos a solução do caso maior em termos de casos menores.

Cálculo de Potências

Mas e se definirmos a potência de forma diferente?
x^n é:

- Caso básico:
  - Se $n = 0$ então $x^n = 1$.
- Caso Geral:
  - Se $n > 0$ e é par, então $x^n = (x^{n/2})^2$.
  - Se $n > 0$ e é ímpar, então $x^n = x(x^{(n-1)/2})^2$.

Note que aqui também definimos a solução do caso maior em termos de casos menores.

Cálculo de Potências

Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
def pot3(x, n):
        assert type(n) == int and n >= 0
        if n == 0:
                return 1

        elif n%2 == 0:
                aux = pot3(x, n//2)
                return aux*aux

        else:
                aux = pot3(x, (n-1)//2)
                return aux*aux*x
```

Cálculo de Potências

Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
def pot3(x, n):
        assert type(n) == int and n >= 0
        if n == 0:
                return 1

        elif n%2 == 0:
                aux = pot3(x, n//2)
                return aux*aux

        else:
                aux = pot3(x, (n-1)//2)
                return aux*aux*x
```

Cálculo de Potências

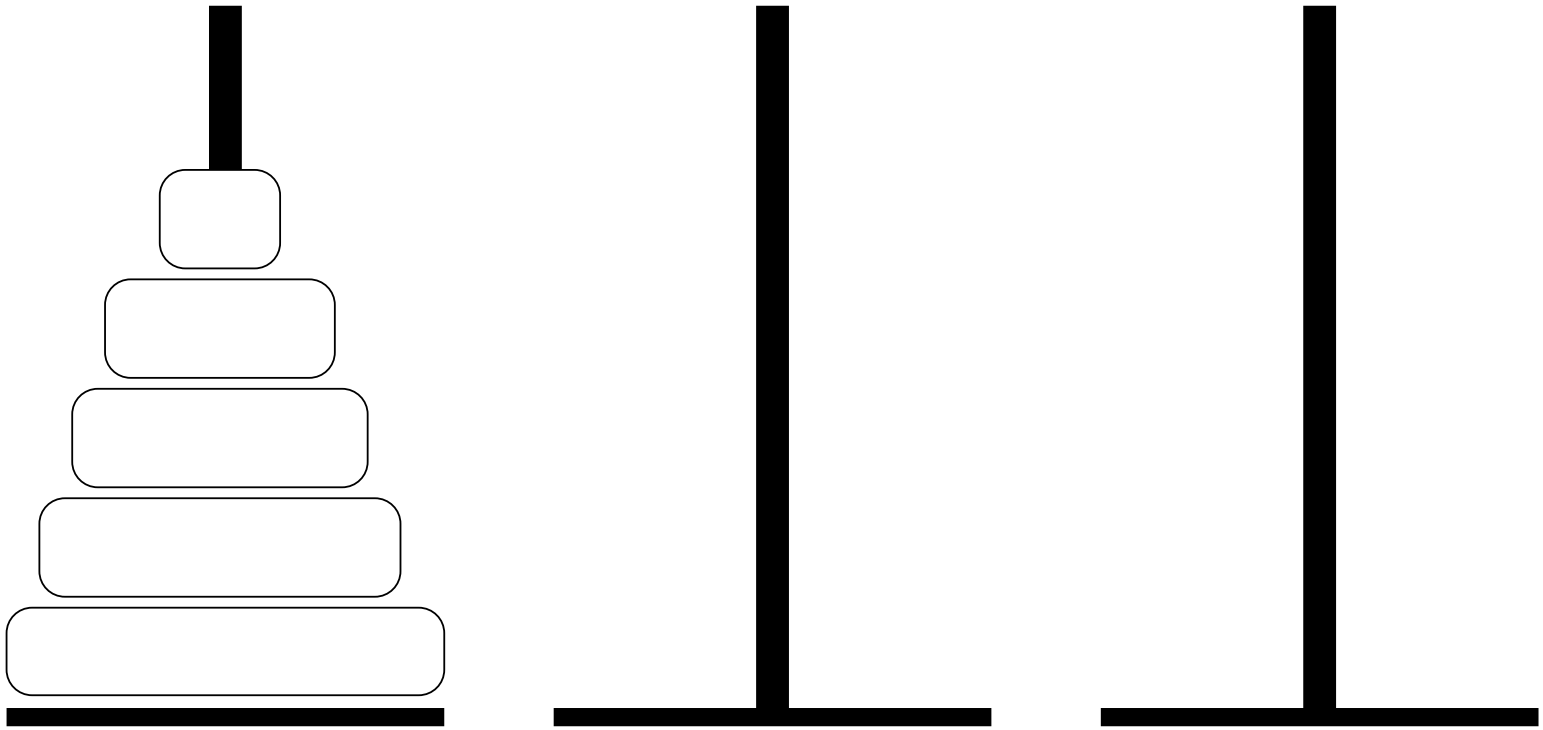
Este algoritmo é mais eficiente do que o iterativo. Por que? Quantas chamadas recursivas o algoritmo pode fazer?

```
def pot3(x, n):
        assert type(n) == int and n >= 0
        if n == 0:
                return 1

        elif n%2 == 0:
                aux = pot3(x, n//2)
                return aux*aux

        else:
                aux = pot3(x, (n-1)//2)
                return aux*aux*x
```

Cálculo de Potências

- No algoritmo anterior, a cada chamada recursiva o valor de n é dividido por 2. Ou seja, a cada chamada recursiva, o valor de n decai para pelo menos a metade.
- Usando divisões inteiras faremos no máximo $\lceil(\log_2 n)\rceil + 1$ chamadas recursivas.
- Enquanto isso, o algoritmo iterativo executa o laço n vezes.

Torres de Hanoi



Torres de Hanoi

- Inicialmente temos 5 discos de diâmetros diferentes na estaca A.
- O problema das torres de Hanoi consiste em transferir os cinco discos da estaca A para a estaca C (pode-se usar a estaca B como auxiliar).
- Porém deve-se respeitar as seguintes regras:
  - Apenas o disco do topo de uma estaca pode ser movido.
  - Nunca um disco de diâmetro maior pode ficar sobre um disco de diâmetro menor.

Torres de Hanoi

- Vamos considerar o problema geral onde há n discos.
- Vamos usar indução para obtermos um algoritmo para este problema.

Torres de Hanoi

Teorema

É possível resolver o problema das torres de Hanoi com n discos.

Prova.

- Base da Indução: $n = 1$. Neste caso temos apenas um disco. Basta mover este disco da estaca A para a estaca C.
- Hipótese de Indução: Sabemos como resolver o problema quando há $n - 1$ discos.

Torres de Hanoi

Teorema

É possível resolver o problema das torres de Hanoi com n discos.

Prova.

- Base da Indução: $n = 1$. Neste caso temos apenas um disco. Basta mover este disco da estaca A para a estaca C.
- Hipótese de Indução: Sabemos como resolver o problema quando há $n - 1$ discos.

Torres de Hanoi

Teorema

É possível resolver o problema das torres de Hanoi com n discos.

Prova.

- Base da Indução: $n = 1$. Neste caso temos apenas um disco. Basta mover este disco da estaca A para a estaca C.
- Hipótese de Indução: Sabemos como resolver o problema quando há $n - 1$ discos.

□

Torres de Hanoi

Prova.

- Passo de Indução: Devemos resolver o problema para n discos assumindo que sabemos resolver o problema com $n-1$ discos.
  - Por hipótese de indução sabemos mover os $n-1$ primeiros discos da estaca **A** para a estaca **B** usando a estaca **C** como auxiliar.
  - Depois de movermos estes $n-1$ discos, movemos o maior disco (que continua na estaca **A**) para a estaca **C**.
  - Novamente pela hipótese de indução sabemos mover os $n-1$ discos da estaca **B** para a estaca **C** usando a estaca **A** como auxiliar.
- Com isso temos uma solução para o caso onde há n discos.

□

Torres de Hanoi

Prova.

- Passo de Indução: Devemos resolver o problema para n discos assumindo que sabemos resolver o problema com $n-1$ discos.
  - Por hipótese de indução sabemos mover os $n-1$ primeiros discos da estaca **A** para a estaca **B** usando a estaca **C** como auxiliar.
  - Depois de movermos estes $n-1$ discos, movemos o maior disco (que continua na estaca **A**) para a estaca **C**.
  - Novamente pela hipótese de indução sabemos mover os $n-1$ discos da estaca **B** para a estaca **C** usando a estaca **A** como auxiliar.
- Com isso temos uma solução para o caso onde há n discos.

□

Torres de Hanoi

Prova.

- Passo de Indução: Devemos resolver o problema para n discos assumindo que sabemos resolver o problema com $n-1$ discos.
  - Por hipótese de indução sabemos mover os $n-1$ primeiros discos da estaca **A** para a estaca **B** usando a estaca **C** como auxiliar.
  - Depois de movermos estes $n-1$ discos, movemos o maior disco (que continua na estaca **A**) para a estaca **C**.
  - Novamente pela hipótese de indução sabemos mover os $n-1$ discos da estaca **B** para a estaca **C** usando a estaca **A** como auxiliar.
- Com isso temos uma solução para o caso onde há n discos.

□

Torres de Hanoi

Prova.

- Passo de Indução: Devemos resolver o problema para n discos assumindo que sabemos resolver o problema com $n-1$ discos.
  - Por hipótese de indução sabemos mover os $n-1$ primeiros discos da estaca **A** para a estaca **B** usando a estaca **C** como auxiliar.
  - Depois de movermos estes $n-1$ discos, movemos o maior disco (que continua na estaca **A**) para a estaca **C**.
  - Novamente pela hipótese de indução sabemos mover os $n-1$ discos da estaca **B** para a estaca **C** usando a estaca **A** como auxiliar.
- Com isso temos uma solução para o caso onde há n discos.

□

Torres de Hanoi: Passo de Indução

- A indução nos fornece um algoritmo e ainda por cima temos uma demonstração formal de que ele funciona!

Torres de Hanoi: Algoritmo

Problema: Mover n discos de **A** para **C**.

1. Se $n = 1$ então mova o único disco de **A** para **C** e pare.
2. Caso contrário ($n > 1$) desloque de forma recursiva os $n - 1$ primeiros discos de **A** para **B**, usando **C** como auxiliar.
3. Mova o último disco de **A** para **C**.
4. Mova, de forma recursiva, os $n - 1$ discos de **B** para **C**, usando **A** como auxiliar.

Torres de Hanoi: Algoritmo

- A função que computa a solução em Python terá o seguinte protótipo:

```
def hanoi(n, ini, fim, aux):
```

- É passado como parâmetro o número de discos a ser movido (**n**), e um caractere indicando de onde os discos serão movidos (**ini**); para onde devem ser movidos (**fim**); e qual é a estaca auxiliar (**aux**).

Torres de Hanoi: Algoritmo

A função que computa a solução é:

```
def hanoi(n, ini, fim, aux):
        if n == 1: #Caso base. Move único disco do Ini para Fim
                print('Move disco', n, 'de', ini, 'para', fim)
        else:
                #Move n-1 primeiros discos de Ini para Aux com Fim como auxiliar
                hanoi(n-1, ini, aux, fim)
                #Move maior disco para Fim
                print('Move disco', n, 'de', ini, 'para', fim)
                #Move n-1 primeiros discos de Aux para Fim com Ini como auxiliar
                hanoi(n-1, aux, fim, ini)
```

Torres de Hanoi: Algoritmo

```
def hanoi(n, ini, fim, aux):
        if n == 1: #Caso base. Move único disco do Ini para Fim
                print('Move disco', n, 'de', ini, 'para', fim)
        else:
                #Move n-1 primeiros discos de Ini para Aux com Fim como auxiliar
                hanoi(n-1, ini, aux, fim)
                #Move maior disco para Fim
                print('Move disco', n, 'de', ini, 'para', fim)
                #Move n-1 primeiros discos de Aux para Fim com Ini como auxiliar
                hanoi(n-1, aux, fim, ini)

def main():
        hanoi(4, 'A', 'C', 'B')

main()
```

Recursão e Enumeração

- Muitos problemas podem ser resolvidos enumerando-se de forma sistemática todas as possibilidades de arranjos que formam uma solução para um problema.
- Vimos em aulas anteriores o seguinte exemplo: determinar todas as soluções inteiras de um sistema linear como:

$$x_0 + x_1 + x_2 = C$$

com $x_0 \geq 0$, $x_1 \geq 0$, $x_2 \geq 0$, $C \geq 0$ e todos inteiros.

```
Para cada possível valor de x0 entre 0 e C
    Para cada possível valor de x1 entre 0 e C-x0
                Faça  x2 = C - (x0 + x1)
                Imprima solução x0 + x1 + x2 = C
```

Recursão e Enumeração

Abaixo temos o código de uma solução para o problema com $n = 2$ e constante C passada como parâmetro.

```
def solucoes(C):
        for x0 in range(0, C+1):
                for x1 in range(0, C+1 - x0):
                        x2 = C - x0 - x1
                        print(x0,'+',x1,'+',x2,'=',C)
```

Recursão e Enumeração

Como resolver este problema para o caso geral, onde n e C são parâmetros?

$$x_0 + x_1 + \ldots + x_{n-1} + x_n = C$$

- A princípio deveríamos ter n laços encaixados.
- Mas não sabemos o valor de n. Só saberemos durante a execução do programa.

Recursão e Enumeração

- A técnica de recursão/indução nos ajuda a lidar com este problema.
- Suponha o problema geral de determinar soluções do problema

$$x_0 + x_1 + \ldots + x_{n-1} + x_n = C$$

Recursão e Enumeração

- A técnica de recursão/indução nos ajuda a lidar com este problema.
- Suponha o problema geral de determinar soluções do problema

$$x_0 + x_1 + \ldots + x_{n-1} + x_n = C$$

  - Se $n = 0$ a única variável deve assumir o valor C ($x_0 = C$).
  - Se $n > 0$ para cada valor possível de x_n, resolvemos recursivamente o problema menor

$$x_0 + x_1 + \ldots + x_{n-1} = C - x_n$$

Recursão e Enumeração

A técnica de recursão pode nos ajudar a lidar com este problema:

- Construir uma função que recebe como parâmetros as variáveis x como um vetor e os valores n e C.
- Se $n = 0$ basta setar o valor da variável $x[0] = C$ e imprimir o vetor solução x.
- Se $n > 0$
  - Dentro de um laço setamos cada valor possível de x_n, e para cada um dos valores setados resolvemos o problema menor recursivamente com parâmetros (vetor x, $n - 1$, $C - x[n]$).

Recursão e Enumeração

```
função solution(x, n, C):
  Se n == 0 Então
     x[0] = C
     Imprima vetor solução x
  Senão
     Para cada valor V entre 0 e C faça
        x[n] = V
        solution(x, n, C - x[n]) #Resolvemos prob. menor recursivamen
```

Recursão e Enumeração

- Em Python teremos uma função com o seguinte protótipo:

```
def solucoes2(x, n, C):
```

- A lista **x** de tamanho $n + 1$ deve ser iniciada com zeros de **x[0]** até **x[n]**.

Recursão e Enumeração

- Primeiramente temos o caso base (quando $n == 0$):

```
def solucoes2(x, n, C):
        '''x é lista inicialmente com n zeros
        indicando valores iniciais de x0...x_n-1
        n é o indice da var. sendo setada'''
        if n == 0:
                x[n] = C
                imprime(x)
    else:
  .
  .
  .
```

- Ao chegar no caso base imprimimos o vetor solução x.

Recursão e Enumeração

- A função completa é:

```
def solucoes2(x, n, C):
        '''x é lista inicialmente com n+1 zeros
        indicando valores iniciais de x0...x_n
        n é o indice da var. sendo setada'''
        if n == 0:
                x[n] = C
                imprime(x)
        else:
                for v in range(0, C+1):
                        x[n] = v
                        solucoes2(x, n-1, C-x[n])
                        x[n] = 0
```

Recursão e Enumeração

- A função que imprime a solução é esta:

```
def imprime(x):
        C = 0
        n = len(x)-1
        for i in range(n):
                print(x[i],'+', end=' ')
                C += x[i]
        C += x[n]
        print(x[n],'=',C)
```

Exemplo de uso para 3 variáveis e $C = 5$:

```
def solucoes2(x, n, C):
        '''x é lista inicialmente com n+1 zeros
        indicando valores iniciais de x0...x_n
        n é o indice da var. sendo setada'''
        if n == 0:
                x[n] = C
                imprime(x)
        else:
                for v in range(0, C+1):
                        x[n] = v
                        solucoes2(x, n-1, C-x[n])
                        x[n] = 0

def main():
        print()
        x = [0 for i in range(3)]
        solucoes2(x, len(x)-1, 5)

main()
```

Exercício

- Defina de forma recursiva a busca binária.
- Escreva um algoritmo recursivo para a busca binária.

Exercício

- Escreva um programa que lê uma string do teclado e então imprime todas as permutações desta palavra. Se por exemplo for digitado "abca" o seu programa deveria imprimir:

```
aabc
aacb
abac
abca
acab
acba
baac
baca
bcaa
caab
caba
cbaa
```