MC102 – Aula 22
Recursão II

Instituto de Computação – Unicamp

22 de Maio de 2014

Roteiro

Aviso

Não haverá aula dia 27/05 (avaliação de cursos).

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 queiramos 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

```
long pot(long x, long n){
  if(n == 0)
    return 1;
  else
    return x*pot(x,n-1);
}
```

Cálculo de Potências

Neste caso a solução iterativa é mais eficiente.

```
long pot(long x, long n){
    long p = 1, i;
    for( i=1; i<=n; i++)
       p = p * x;
    return p;
}
```

- O laço é executado n vezes.
- Na solução recursiva são feitas n chamadas, 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 como no caso geral 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?

```
long pot(long x, long n){
    double aux;
    if(n == 0)
       return 1;

    else if(n%2 == 0){ //se n é par
      aux = pot(x, n/2);
      return aux * aux;
    }

    else{ //se n é impar
      aux = pot(x, (n-1)/2);
      return x*aux*aux;
    }
}
```

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.

Recursão com várias chamadas

- Não há necessidade da função recursiva ter apenas uma chamada para si própria.
- A função pode fazer várias chamadas para si própria.
- A função pode ainda fazer chamadas recursivas indiretas. Neste caso a função 1, por exemplo, chama uma outra função 2 que por sua vez chama a função 1.

Soma de um vetor

- Dado um vetor, vamos definir $S(i, n)$ como a soma de n elementos a partir da posição i.
- Com isso temos a seguinte definição recursiva para a soma dos elementos de um vetor:
  - Se $n = 1$ então $S(i, n) = v[i]$.
  - Se $n > 1$ então $S(i, n) = S(i, \lceil n/2 \rceil) + S(i + \lceil n/2 \rceil, \lfloor n/2 \rfloor)$.
- Observações
  - $n = \lceil n/2 \rceil + \lfloor n/2 \rfloor$.
  - Dada uma posição i e quantidade $x = \lceil n/2 \rceil$ de elementos, a próxima posição a ser considerada será $(i + x)$.
- Para computarmos a soma de todos os elementos de um vetor com n elementos, devemos calcular $S(0, n)$.

Soma de um vetor

O código recursivo segue abaixo. Basta implementarmos funções para calcular o teto e o piso da divisão de dois números.

```
int soma(int v[], int i, int n){
  if(n == 1)
    return v[i];
  else{
    return soma(v, i, teto(n,2)) +
      soma(v, i+teto(n,2), piso(n,2));
  }
}
```

Soma de um vetor

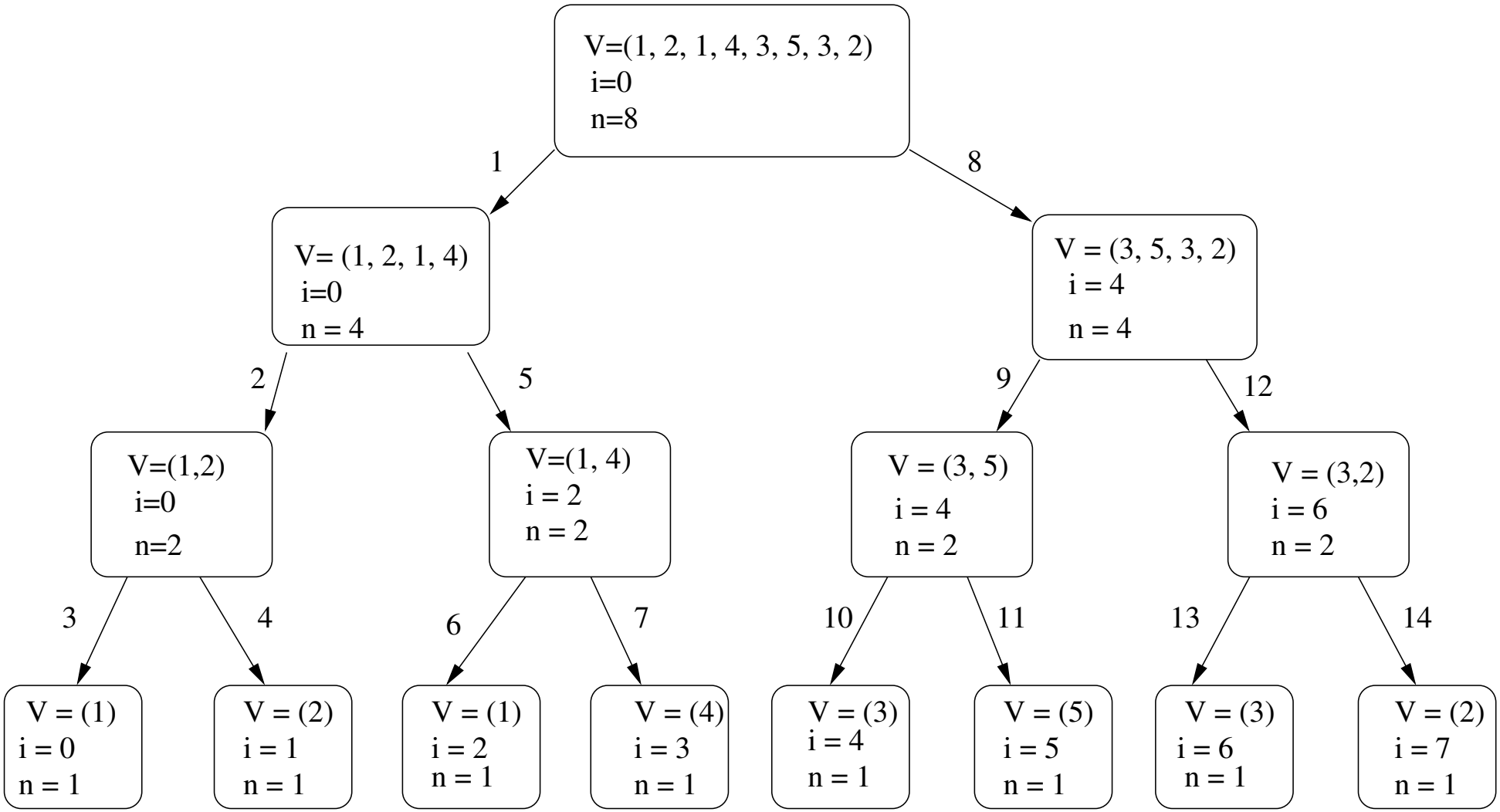
```
int teto(int numerador, int denominador){
  if(numerador % denominador == 0) //se divisão for inteira
    return (numerador/denominador);
  else
    return (numerador/denominador + 1);
}


int piso(int numerador, int denominador){
    return (numerador/denominador);
}
```

Soma de um vetor

- Abaixo temos um exemplo de execução da função para o vetor v=[1, 2, 1, 4, 3, 5, 3, 2].
- Há uma indicação da ordem em que ocorrem as sucessivas chamadas recursivas.
- Em cada balão é apresentada apenas a parte do vetor que está sendo considerada pela função naquele momento.
- A chamada da função deve ser: **somar(v, 0, 8)**.

Soma de um vetor



Torres de Hanoi

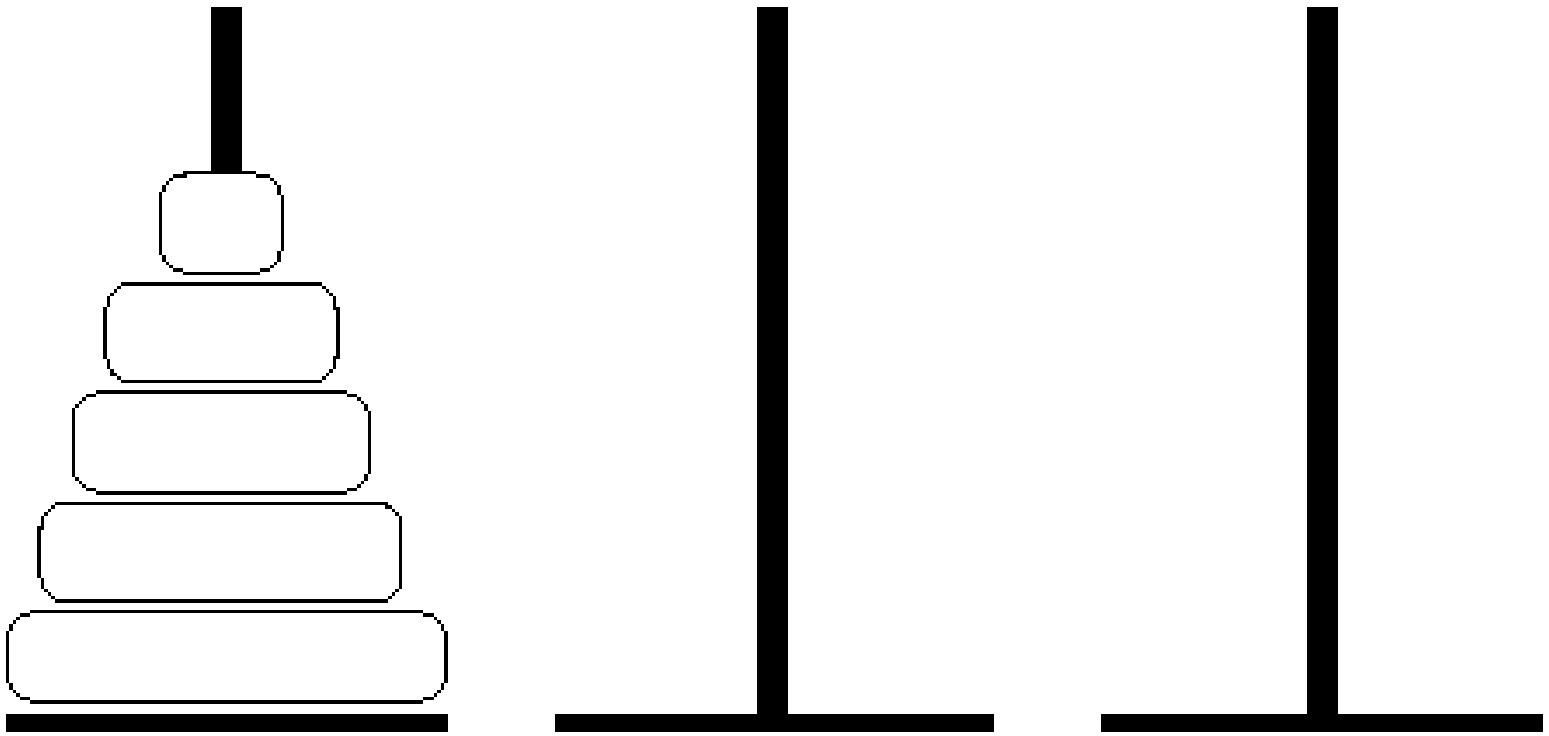
- Problema inventado pelo matemático francês Édouard Lucas em 1883.
- Também conhecido como Torres de Brahma.

No grande templo de Brahma em Benares, numa bandeja de metal sob a cúpula que marca o centro do mundo, três agulhas de diamante servem de pilar a sessenta e quatro discos de ouro puro. Incasavelmente, os sacerdotes transferem os discos um de cada vez, de agulha para agulha, obedecendo sempre à lei imutável de Brahma: nenhum disco se poderá sobrepor a um menor.

Torres de Hanoi

No início do mundo, todos os sessenta e quatro discos de ouro foram dispostos na primeira das três agulhas, constituindo a Torre de Brahma. No momento em que o menor dos discos for colocado de tal modo que se forme uma vez mais a Torre de Brahma numa agulha diferente da inicial, tanto a torre como o templo serão transformados em pó e o ribombar de um trovão assinalará o fim do mundo.

Torres de Hanoi



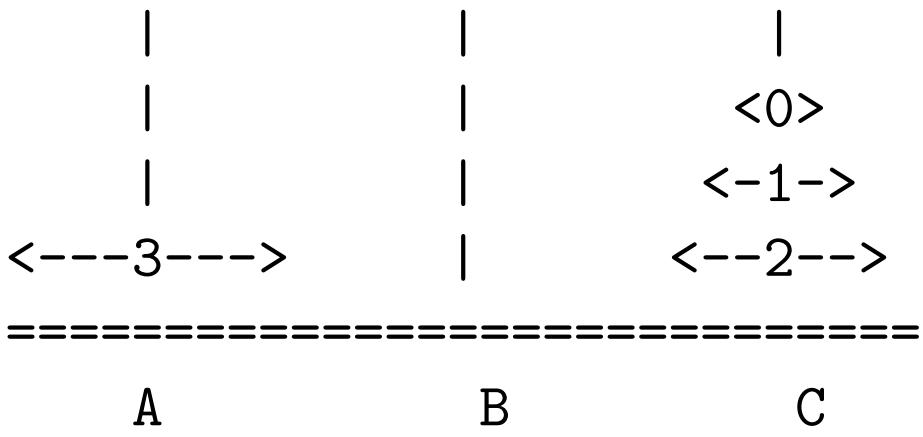
Torres de Hanoi

- 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 B (pode-se usar a estaca C como auxiliar).
- Porém deve-se respeitar algumas 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

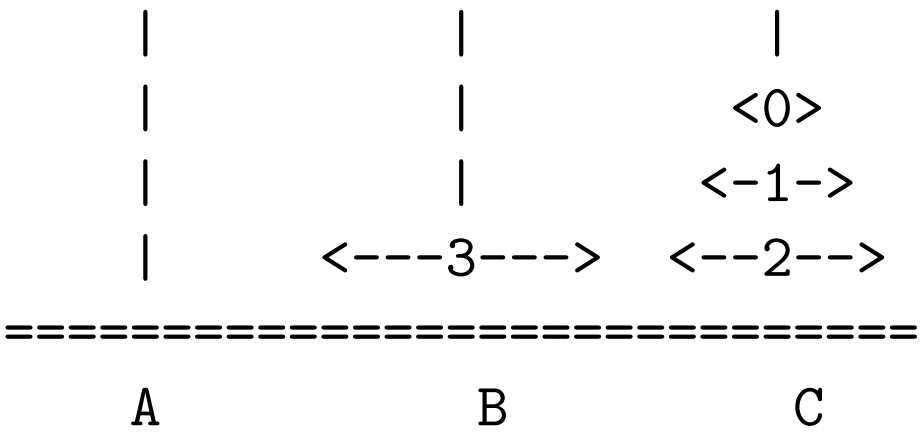
- Vamos considerar o problema geral onde há n discos.
- Vamos usar indução para obter um algoritmo para este problema.

Torres de Hanoi



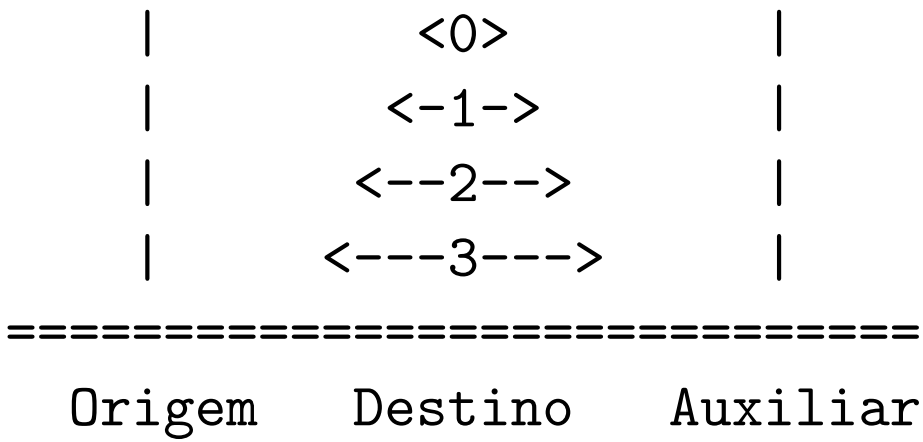
- Podemos mover n-1 discos para a torre C

Torres de Hanoi



- Podemos mover o disco maior para a torre B

Torres de Hanoi



- Podemos mover n-1 discos da torre C para a torre B.

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

□

Torres de Hanoi: Passo de Indução

- Olhem que maravilha que é 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 B:

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

Torres de Hanoi: Algoritmo

```
#include <stdio.h>

void hanoi(int n, char estacaIni, char estacaFim, char estacaAux);

int main(){
  hanoi(3, 'A', 'B' , 'C');
  printf("\n");
}

//Discos são numerados de 1 até n

void hanoi(int n, char estacaIni, char estacaFim, char estacaAux){
 if(n==1)
    printf("\nMova disco %d da estaca %c para %c.", n,  estacaIni, estacaFim);
  else{
    hanoi(n-1,estacaIni,estacaAux,estacaFim);
    printf("\nMova disco %d da estaca %c para %c.", n,  estacaIni, estacaFim);
    hanoi(n-1,estacaAux,estacaFim,estacaIni);
  }
}
```

Torres de Hanoi

Seja $T(n)$ o número de movimentos necessários para resolver o problema com n discos.
Pelo algoritmo anterior, $T(n)$ é dado pela seguinte recursão:

$$T(n) = \begin{cases} 1 & \text{se } n = 1 \\ 2T(n-1) + 1k & \text{se } n \geq 1 \end{cases}$$

Fazendo alguns cálculos, chegamos à solução seguinte:
$T(n) = 2^n - 1$, para $n \geq 1$.
Quanto tempo para resolver o problema com 64 discos?

Torres de Hanoi

Quanto tempo para resolver o problema com 64 discos? Suposições:

- Sacerdotes são semi-deuses que movem os discos na velocidade da luz (≈300.000.000 m/s) (lembre-se: só Chuck Norris consegue mover objetos mais rápido que a velocidade da luz.).
- 3 torres a 66.33 cm uma da outra. Distância média dos movimentos: $\frac{3*0.6633}{2} = 0.99495 \approx 0.995$m.
- Discos com altura mínima, tendendo a zero $\Rightarrow$ tamanho da pilha tende a zero $\Rightarrow$ Tempo para tirar o disco da pilha tende a zero.

$R = (2^{64}) \times (\frac{0.995}{300000000})/60/60/24/365$

Exercício

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