Algoritmos e Estruturas de Dados

Disciplina 301477

Programa de Pós-graduação em
Computação Aplicada

http://www.nickgentry.com/

Prof. Alexandre Zaghetto
http://alexandre.zaghetto.com
zaghetto@unb.br

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Módulo 12
Recursividade

1. Recursividade

• É uma das ferramentas de programação mais poderosas e menos entendidas pelo principiantes em programação.

• Como vocês já não são mais principiantes, suponho eu, esse capítulo será mamão com açúcar.

• Após termos trabalhando com funções, alguns devem ter se perguntado:

> Uma função pode chamar ela mesma?

• A resposta é sim, e esta técnica se chama RECURSIVIDADE.

• Algumas vezes ela facilita a interpretação de certos problemas.

1. Recursividade

- Uma questão fundamental é:

> Quando parar de chamar a função?

- A recursão requer a repetição explícita de um processo até que determinada condição seja satisfeita.

- Se você não garantir isto, o algoritmo entrará num laço infinito.

1. Recursividade

- Exemplo:

```
#include <stdio.h>
#include <stdlib.h>

int loop(int);

int main()
{
  int n;

  n = loop(5);

  system("PAUSE");
  return 0;
}

int loop(int x){

    if(x < 10)

          return loop(x);

}
```

1. Recursividade

- Somatório:

  ✓ De forma iterativa:

somatorio (n) = 1 +2 +3 .... +n

  ✓ De forma recursiva:

somatorio (n) = n + somatorio (n – 1),
somatorio (1) = 1

1. Recursividade

- Solução:
  - ✓ Iterativa:

```
int somatorio(int N)
{
   int i, resp = 0;

   for( i = 1; i <= N; i++ )
          resp += i;

   return resp;

}
```

1. Recursividade

- Solução:
  - ✓ Recursiva:

```
int somatorio(int N)
{
   if( N == 1 )
          return 1;
   else
          return N + somatorio(N - 1);
}
```

1. Recursividade

- A função fatorial:

$$n! = \begin{cases} 1 & se\ n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1 & se\ n > 0 \end{cases}$$

  - Exemplos

$$0! = 1$$
$$1! = 1$$
$$2! = 2 \cdot 1 = 2$$
$$3! = 3 \cdot 2 \cdot 1 = 6$$

1. Recursividade

- A função fatorial:

  ✓ De forma iterativa:

fatorial (n) = 1 * 2 * 3 .... n

  ✓ De forma recursiva:

fatorial (n) = n * fatorial (n – 1),
fatorial (0) = 1

1. Recursividade

- Solução:

  ✓ De forma iterativa:

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
  int n, i, fat = 1;

  printf("Digite n:");
  scanf("%d", &n);

  for(i = 1; i<=n;i++) fat = fat*i;

  printf("%d \n", fat);

  system("PAUSE");
  return 0;
}
```

1. Recursividade

- Solução:
  - ✓ De forma recursiva:

```
#include <stdio.h>
#include <stdlib.h>

int fatorial (int);

int main()
{
  int n, fat;

  printf("Digite n:");
  scanf("%d", &n);

  fat = fatorial(n);
```

1. Recursividade

- Solução:
  - ✓ De forma recursiva:

```
  printf("%d \n", fat);

  system("PAUSE");
  return 0;
}

int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1*fatorial(1-1);
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1*fatorial(1-1);
                                  [fatorial(0)]
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1*fatorial(1-1);
                                  [fatorial(0)]
                                  return 1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1*fatorial(1-1);
                                  return 1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1*1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        [fatorial(1)];
                        return 1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*fatorial(2-1);
                        return 1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2*1;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              [fatorial(2)];
              return 2;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*fatorial(3-1);
              return 2;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 3*2;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

```
N = fatorial(3);
    return 6;
```

1. Recursividade

- Exemplo - fatorial(3):

```
int fatorial (int n){

    if(n==0)
        return 1;
    else
        return n*fatorial(n-1);
}
```

N = **6;**

1. Recursividade

- Escrevendo programas recursivos:

  ➢ Primeiro podemos reconhecer um grande número de casos a solucionar: 0!, 1!, 2!, 3!, 4! etc.

  ➢ Podemos também identificar um caso "trivial", não-recursivo, diretamente solucionável: 0!=1.

  ➢ Encontramos um método para solucionar um caso "complexo" em termos de um caso "mais simples": n! = n*(n-1)!

  ➢ A transformação do caso mais complexo no caso mais simples deve ocasionalmente resultar num caso "trivial".

1. Recursividade

- Fibonacci e razão áurea:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

`0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...`

- Razão Áurea:

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\,033\,989\,.$$

`55/34 ≈ 1,61765`

1. Recursividade

- Fibonacci e razão áurea:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
```

Pro lar, implementar
a solução recursiva em C.

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$
- $4^2 = \underbrace{1+3+5}_{3^2}+7$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$
- $4^2 = \underbrace{1+3+5}_{3^2}+7$
- $3^2 = \underbrace{1+3}_{2^2}+5$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$
- $4^2 = \underbrace{1+3+5}_{3^2}+7$
- $3^2 = \underbrace{1+3}_{2^2}+5$
- $2^2 = \underbrace{1}_{1^2}+3$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$
- $4^2 = \underbrace{1+3+5}_{3^2}+7$
- $3^2 = \underbrace{1+3}_{2^2}+5$
- $2^2 = \underbrace{1}_{1^2}+3$
- $1^2 = 1$

1. Recursividade

Exercício: O quadrado de um número natural n é dado pela soma dos n primeiros números ímpares consecutivos. Por exemplo, $1^2=1$, $2^2=1+3$, $3^2=1+3+5$, $4^2=1+3+5+7$, etc. Dado um número n, escreva uma função recursiva que calcula seu quadrado usando a soma de ímpares.

- $5^2 = 1+3+5+7+9 = 25$
- $5^2 = \underbrace{1+3+5+7}_{4^2}+9$
- $4^2 = \underbrace{1+3+5}_{3^2}+7$
- $3^2 = \underbrace{1+3}_{2^2}+5$
- $2^2 = \underbrace{1}_{1^2}+3$
- $1^2 = 1$

$n^2 = (2*n-1) + (n-1)^2$

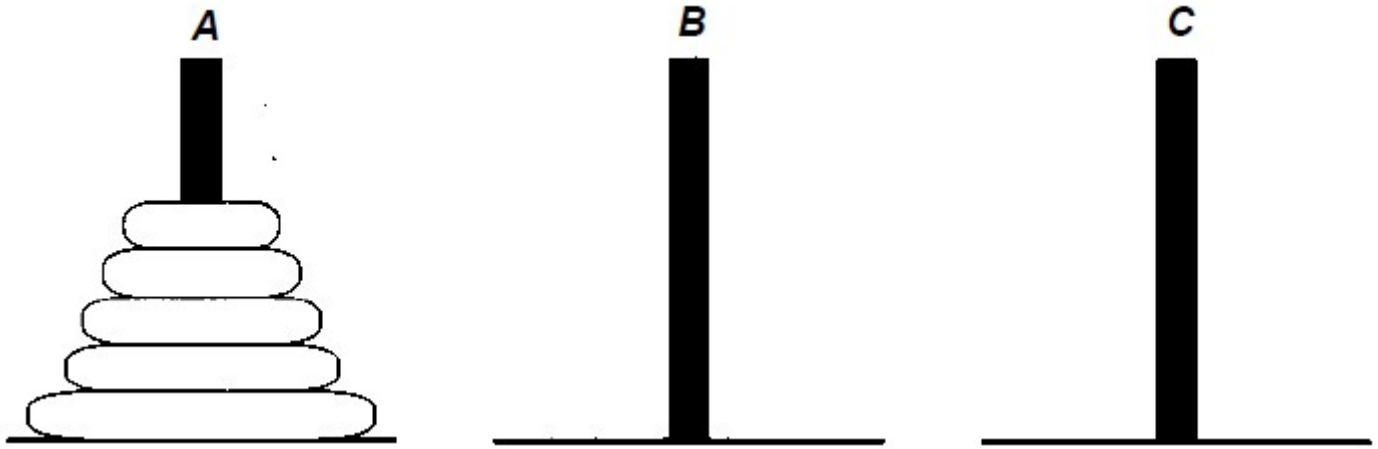
1. Recursividade

- Solução:

```
#include <stdio.h>
#include <stdlib.h>

int quadrado (int);

int main ()
{
    int n = 6, result;

    result = quadrado(n);
    printf("%d \n", result);

    system("PAUSE");
    return 0;
}

int quadrado (int n){

    if (n==1)
       return 1;
    else
       return (2*n-1) + quadrado(n-1);
}
```

1. Recursividade

- O problema das Torres de Hanoi:



➢ O objetivo é deslocar os cinco discos para a estaca C, usando B como auxiliar.

➢ Somente o primeiro disco pode ser deslocado.

➢ Um disco maior não pode ser posicionado sobre um menor.

1. Recursividade

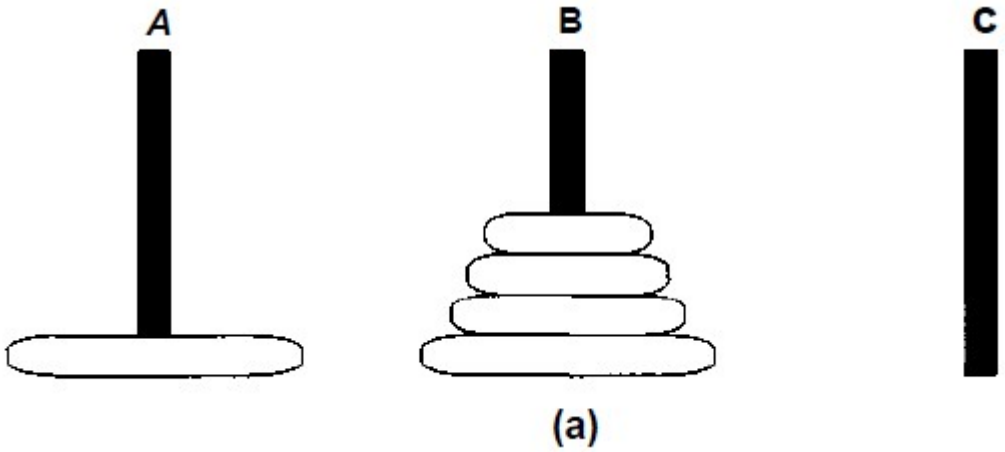
- O problema das Torres de Hanoi:
  - Considerando o caso geral, para *n* discos:
    - ✓ Suponha que tivéssemos uma solução para *n-1* discos e pudéssemos dar a solução para *n*, em função da solução para *n-1* discos.
    - ✓ No caso trivial, no qual temos apenas um único disco, basta deslocá-lo de A para C.

1. Recursividade

- O problema das Torres de Hanoi:
  - Se n = 5:
    - ✓ Vamos supor que eu pudesse movimentar os quatro primeiros discos da estaca A para a estaca B, usando C como auxiliar.

1. Recursividade

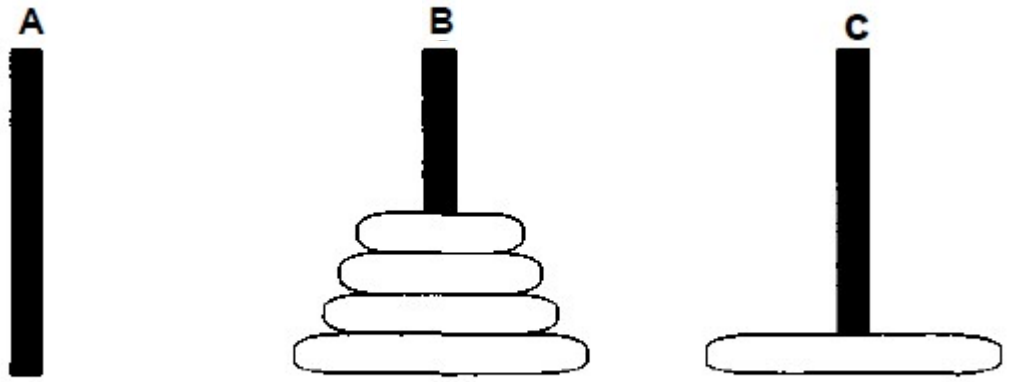
- O problema das Torres de Hanoi:
  - Se n = 5:
    - ✓ Vamos supor que eu pudesse movimentar os quatro primeiros discos da estaca A para a estaca B, usando C como auxiliar.



(a)

1. Recursividade

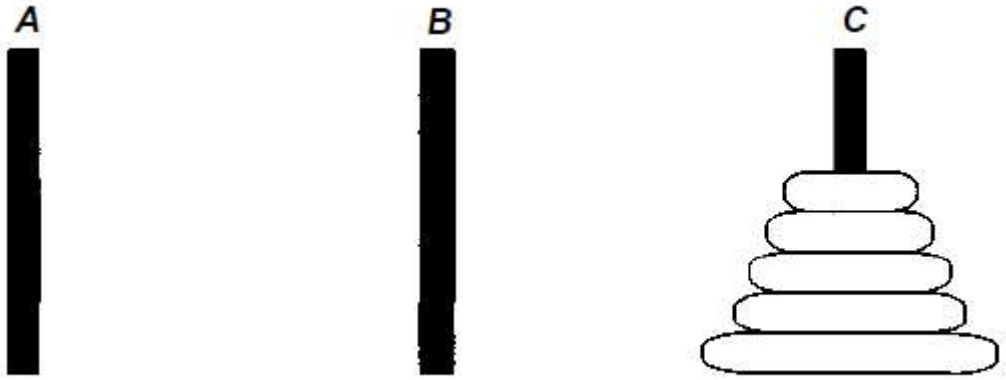
- O problema das Torres de Hanoi:
  - Se n = 5:
    - ✓ Poderíamos deslocar o maior disco de A para C.



(b)

1. Recursividade

- O problema das Torres de Hanoi:
  - Se n = 5:
    - ✓ E, por último, aplicar novamente a solução aos quatro discos, movendo-os de B para C, usando A como auxiliar.



(c)

1. Recursividade

- O problema das Torres de Hanoi:

  ➢ Para mover *n* discos de A para C, usando B como auxiliar:

  1. Se n=1, desloque o único disco de A para C e pare.
  2. Desloque os *n-1* primeiros discos de A para B, usando C como auxiliar.
  3. Desloque o último disco de A para C.
  4. Mova os *n-1* discos de B para C, usando A como auxiliar.

1. Recursividade

- Solução:

```
#include <stdio.h>
#include <stdlib.h>

void hanoi(int, char, char, char);

int main()
{

  int n = 3;

  hanoi(n, 'A', 'B', 'C');

  return 0;
}

void hanoi(int n, char origem, char intermediario, char destino){

    if(n==1){
      printf("Mover %c -> %c \n",origem, destino);
      exit;
    }else{
      hanoi(n-1, origem, destino, intermediario);
      printf("Mover %c -> %c \n",origem, destino);
      hanoi(n-1, intermediario, origem, destino);
    }
}
```

1. Recursividade

• Em termos gerais, não há por que procurar uma solução recursiva para um problema.

• A maioria dos problemas pode ser solucionada usando métodos não-recursivos.

• Uma solução recursiva ocupa mais memória e é mais lenta que a solução iterativa para um mesmo problema.

  ➢ Em cada instância, todos os parâmetros e variáveis locais são criados novamente, independentemente dos que já existiam antes.

• Nem sempre é fácil reconhecer situações apropriadas para a utilização de recursividade.

• Porém, há certos problemas cuja natureza permite uma solução recursiva bem mais elegante e intuitiva do que a solução iterativa.