

### *Engenharia da Computação – 3ª série*

## *Algoritmos Recursivos*

*(E1, E2)*

**2025**

### Pergunta

- O que é Repetição de Instruções?



## Repetição de Instruções

### Resposta



- Quando se é necessário executar um conjunto de instruções várias vezes em um programa de computador;
- Recurso de programação que permite executar uma sequência de instruções repetidamente, enquanto uma determinada condição for verdadeira;

### Pergunta

- Como pode ser obtida a Repetição de Instruções?



## Repetição de Instruções

### Resposta



- Em um computador, numa linguagem de programação moderna, pode ser obtida por meio de:
  - Iteração: ação de iterar, ou repetir, processo de repetir uma sequência de instruções várias vezes, com o objetivo de atingir uma meta específica, utilizando de laços (*loops*), onde um bloco de código é executado repetidamente até que uma condição seja atendida, por exemplo:

```
public static void apresentarNumeros(int n)
{   for(int i = n; i >= 0; i--)
    {   System.out.println(n);
    }
}
```

## Repetição de Instruções

### Resposta



- Em um computador, numa linguagem de programação moderna, pode ser obtida por meio de:
  - Recursão: conceito que se refere a uma função, que chama a si mesma, repetidamente, até que uma condição de parada seja atingida, dividindo um problema em subproblemas menores e chamando a si mesma para resolvê-los.

```
public static void apresentarNumeros(int n)
{
    if(n >= 0)
    {
        System.out.println(n);
        apresentarNumeros(n - 1);
    }
}
```

### Conclusão



- Recursão ocorre quando uma função faz a chamada de si própria;
- Entretanto, a fim de gerar uma resposta, uma condição de término deve ocorrer;
- Algoritmos Recursivos são representados por **Recorrências**;
- **Recorrência** é uma expressão que fornece o valor de uma função em termos dos valores “anteriores” da mesma função.



### Exemplo



- O fatorial de um número inteiro positivo  $n$ , denotado por  $n!$  é definido por:

- Se  $n = 0$ ,  $n! = 1$ ;
- Se  $n \geq 1$ ,  $n! = n \cdot (n - 1)!$

Por exemplo:  $3! = 3 \cdot 2!$

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



# ECM306 – Tópicos Avançados em Estrutura de Dados

## Algoritmos Recursivos

### Exemplo



- Então, para  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$
- Será que a **função fatorial** pode ser definida de **forma recursiva**?

# ECM306 – Tópicos Avançados em Estrutura de Dados

## Algoritmos Recursivos

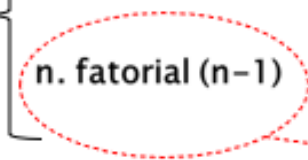
### Exemplo



- Então, para  $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$
- Será que a **função fatorial** pode ser definida de **forma recursiva**?
- $\text{fatorial}(5) = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1) = 5 \cdot \text{fatorial}(4)$

|             |   |                 |  |  |  |
|-------------|---|-----------------|--|--|--|
| fatorial(5) |   |                 |  |  |  |
| 5           | * | fatorial(5 - 1) |  |  |  |
| 4           | * | fatorial(4 - 1) |  |  |  |
| 3           | * | fatorial(3 - 1) |  |  |  |
| 2           | * | fatorial(2 - 1) |  |  |  |
| 1           | * | fatorial(1 - 1) |  |  |  |
|             |   | 1               |  |  |  |

$$\text{fatorial}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot \text{fatorial}(n-1) & \text{se } n \geq 1 \end{cases}$$

 **Recorrência**

### Pergunta

- Como devem ser as Funções Recursivas?



## Funções Recursivas

### Resposta



- Devem possuir um ou mais **casos básicos**, os quais são definidos de forma não-recursiva em termos de quantidades fixas, por exemplo, no caso da função fatorial, o caso básico é  $n = 0$ ;
- Devem possuir, também, um ou mais **casos recursivos**, os quais são definidos por meio da aplicação da definição da função;
- Pseudocódigo e código em Java da Função Fatorial:

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

```
public static int fatorial(int n)
{
    if(n == 0)
    { return 1;
    }
    else
    { return n * fatorial(n-1);
    }
}
```

### Pergunta

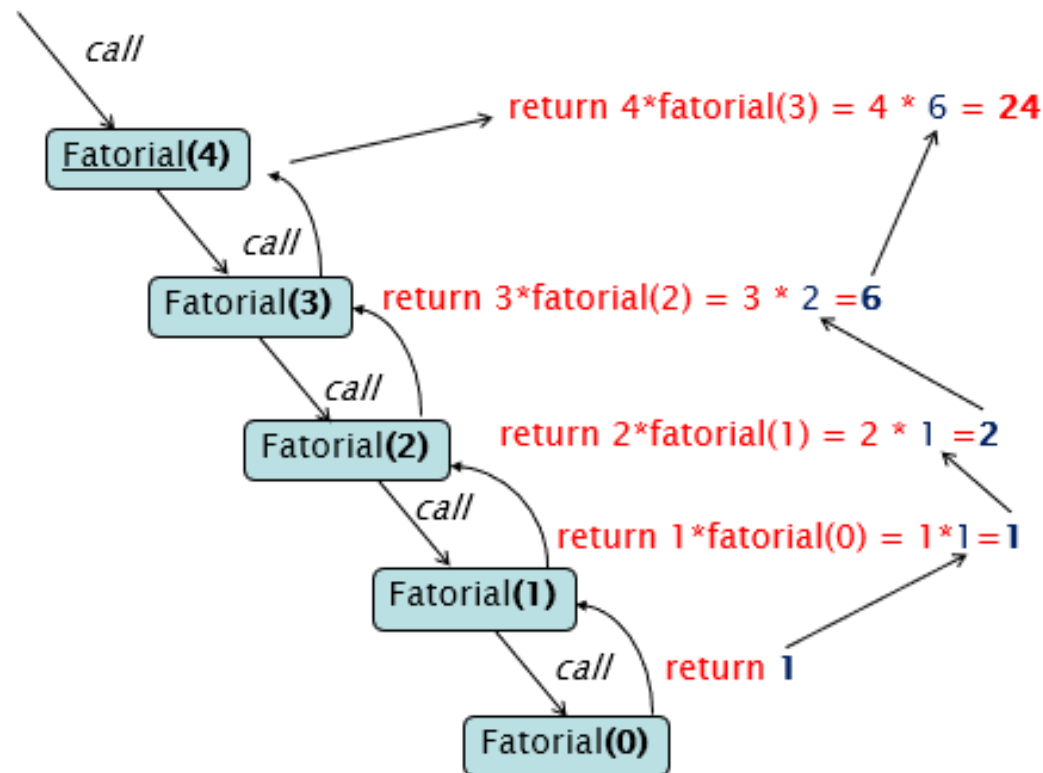
- Como é um *Trace* de Recursão?



### Resposta



- Trace de Recursão da Função Fatorial:



### Pergunta

- O que é Recursão Linear?



## Recursão Linear

### Resposta



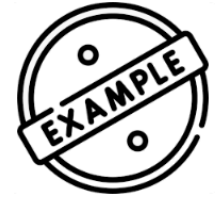
- Corresponde à forma mais simples de **recursão**;
- Neste tipo de **recursão** uma única chamada recursiva é feita de cada vez;
- **Recursão Linear** é útil quando num algoritmo se deseja obter o primeiro ou último elemento de uma lista, na qual os elementos restantes têm a mesma estrutura da estrutura original.



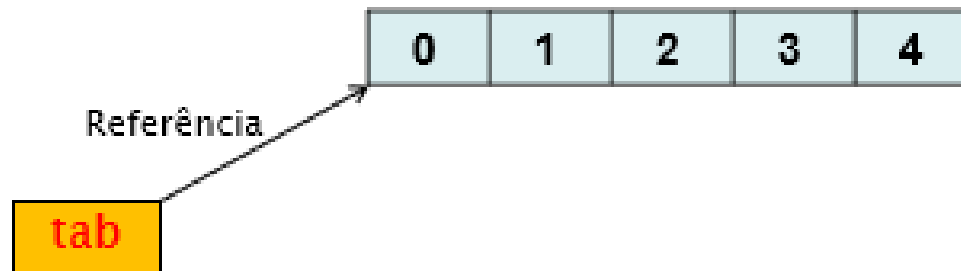
# ECM306 – Tópicos Avançados em Estrutura de Dados

## Recursão Linear

### Exemplo



- A soma dos elementos de um vetor (*array*):
- Seja um vetor **A**, de ***n*** inteiros, no qual deseja-se obter a soma;
- Pode-se resolver este problema com o uso de **recursão linear**, observando-se que a soma de todos os ***n*** inteiros no vetor **A** é igual a ***A*[0]** se ***n* = 1**, ou a soma dos primeiros **(*n*-1)** inteiros em **A** mais o último elemento em **A**.

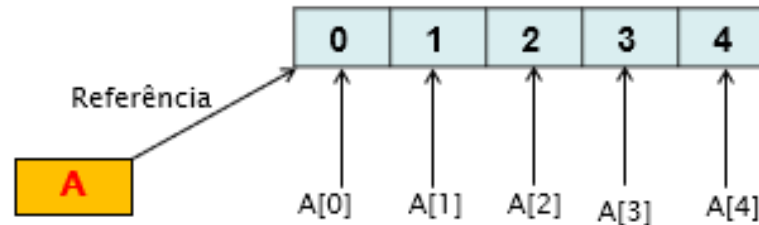


## Recursão Linear

### Exemplo



- A soma dos elementos de um vetor (*array*):



Se  $n=1$ ,  $S(n) = a[0]$

Se  $n > 1$ ,  $S(n) = S(n-1) + A[n-1]$

$$\begin{aligned} S_5 &= S_4 + A[4] \\ &= S_3 + A[3] + A[4] \\ &= S_2 + A[2] + A[3] + A[4] \\ &= S_1 + A[1] + A[2] + A[3] + A[4] \\ &= A[0] + A[1] + A[2] + A[3] + A[4] \end{aligned}$$

## Recursão Linear

### Exemplo



- A soma dos elementos de um vetor (*array*), em Pseudocódigo:

**soma\_Rec**(A,n)

if (n = 1)

return A[0];

else

return **soma\_Rec**(A,n-1) + A[n-1];

## Recursão Linear

### Exemplo



- A soma dos elementos de um vetor (*array*), em Java:

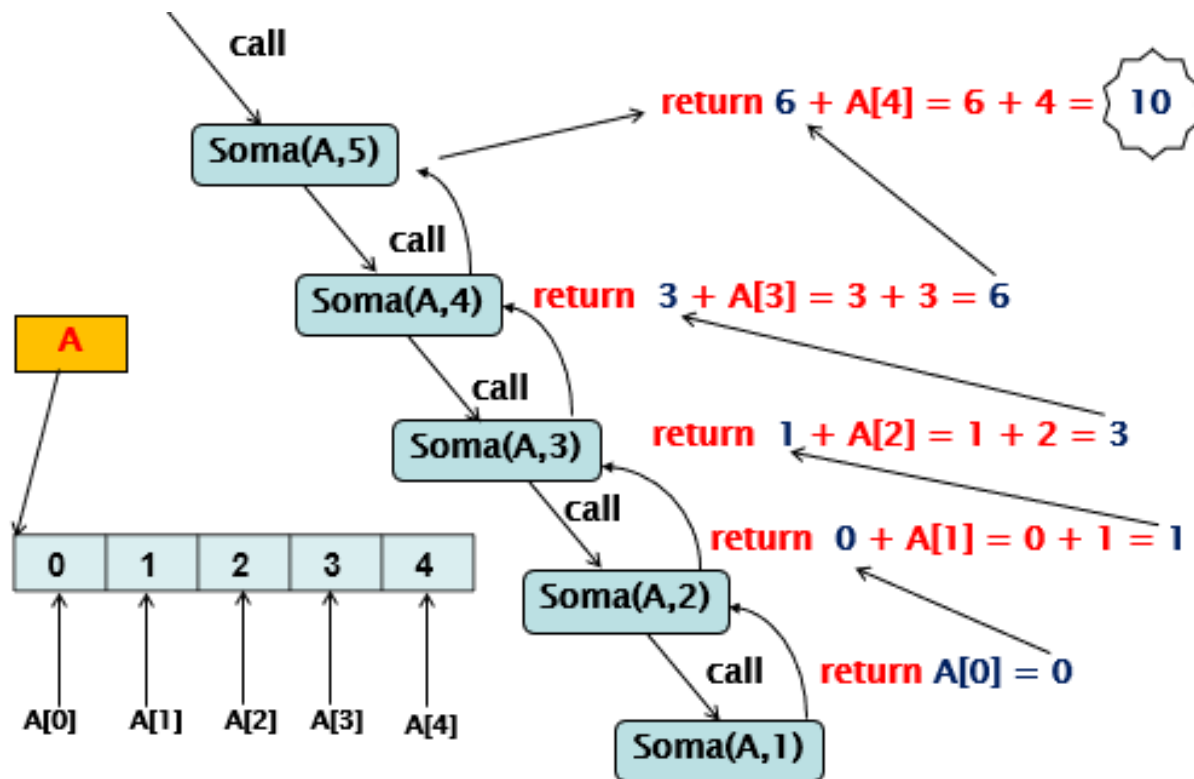
```
public class Soma_Array
{
    public static void main(String args[])
    {
        int tab[] = new int[5];
        for(int i=0; i < tab.length; i++)
        {
            tab[i] = i;
        }
        System.out.println("A soma dos elementos do array é: " + soma_Rec(tab, tab.length));
    }

    public static int soma_Rec(int A[], int n)
    {
        if(n == 1)
        {
            return A[0];
        }
        else
        {
            return soma_Rec(A, n-1) + A[n-1];
        }
    }
}
```

### Exemplo



- Trace de Recursão da função *soma\_Rec()*:



### Conclusão



- O método recursivo deve sempre assegurar que o procedimento **pare!**
- Isso é assegurado por meio da escrita do caso para  **$n = 1$** ;
- A chamada recursiva sempre é feita com um valor de parâmetro inferior a  **$n$** , no caso,  **$n - 1$** .

```
public static int soma_Rec(int A[], int n)
{
    if(n == 1)
    {
        return A[0];
    }
    else
    {
        return soma_Rec(A, n-1) + A[n-1];
    }
}
```

### Pergunta

- O que é a Série de Fibonacci?



## Série de Fibonacci

### Resposta



- Também conhecida por sucessão de **Fibonacci**, ou sequência de **Fibonacci**, é uma sequência de números naturais, na qual os **primeiros dois termos** são **0** e **1**, e cada termo subsequente corresponde à **soma dos dois precedentes**.
- Os números de **Fibonacci** são, portanto, compostos pela seguinte sequência de números inteiros:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...



## Série de Fibonacci

### Resposta



- Em termos matemáticos, a sequência de Fibonacci é definida recursivamente pela fórmula abaixo, sendo os dois primeiros termos  $F_0=0$  e  $F_1=1$ :

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

## Série de Fibonacci

### Resposta



- Série de Fibonacci, em Pseudocódigo:

```
Fibonacci(n)
    if (n <= 1)
        return n ;
    else
        return Fibonacci(n-1) + Fibonacci(n-2);
```

## Série de Fibonacci

### Resposta



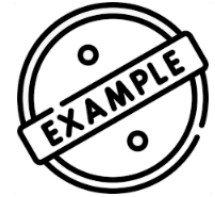
- Série de Fibonacci, em Java:

```
public class Fibo
{   public static void main(String args[])
    {   int n=10;
        System.out.println (Fibonacci(n));
    }

    public static int  Fibonacci(int n)
    {   if(n <= 1) return n;
        else return (Fibonacci(n-1) + Fibonacci(n-2));
    }
}
```

## Algoritmos Recursivos

### Exemplo 1



- Defina um algoritmo **recursivo** para encontrar o **máximo** elemento de um vetor  **$A$** , de  **$n$**  elementos.

# ECM306 – Tópicos Avançados em Estrutura de Dados

## Algoritmos Recursivos

### Exemplo 1



- Defina um algoritmo **recursivo** para encontrar o **máximo** elemento de um vetor **A**, de **n** elementos:

```
1 public class Exemplo_1
2 {   public static void main(String args[])
3     {   int tab[] = { 4, 6, 8, 1, 4, 9, 10, 4 };
4         int n = tab.length;
5         imprime(tab);
6         System.out.println("Maximo: " + max_Recursive(tab, n));
7     }
8
9     public static void imprime(int v[])
10    {   System.out.print("Vetor: ");
11        for(int i = 0; i < v.length; i++)
12        {   System.out.print (v[i] + " ");
13        }
14        System.out.println("");
15    }
16
17    public static int max_Recursive(int A[], int n)
18    {   if(n == 1) return A[0];
19        else
20        {   int x = max_Recursive(A,n-1);
21            if(x < A[n-1]) return A[n-1];
22            else return x;
23        }
24    }
25 }
```

### Exemplo 2

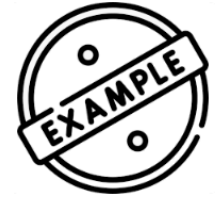


- Considerando a função abaixo, em pseudocódigo, responda:

**O que faz esta função?**

```
Func (int a)
    if (a < 2 )
        return 1
    else
        return (a-1) * Func(a-1)
```

### Exemplo 2

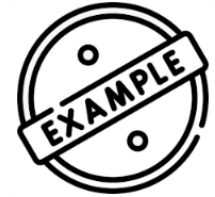


- A função **Func**, em Java:

```
public class Exemplo_2
{   public static void main(String args[])
    {   int n = 5;
        System.out.println("\n" + Func(n));
    }

    public static int Func(int a)
    {   if(a < 2 )   return 1;
        else return (a-1) * Func(a-1);
    }
}
```

### Exemplo 2



- Considerando a função abaixo, em pseudocódigo, responda:

**O que faz esta função?**

```
Func (int a)
    if (a < 2 )
        return 1
    else
        return (a-1) * Func(a-1)
```

**Resposta:** Esta função calcula  $(n-1)!$  de forma RECURSIVA



# ECM306 – Tópicos Avançados em Estrutura de Dados

## Algoritmos Recursivos

### Exemplo 3



- Escreva uma implementação em Java que recebe do usuário uma *String* e chama uma função **recursiva** que retorna um valor **booleano** representando *true* se a *String* corresponder a um **Palíndromo**, ou *false* caso contrário.

Exemplo de Palíndromo:

A B C C B A

X Y Z A Z Y X

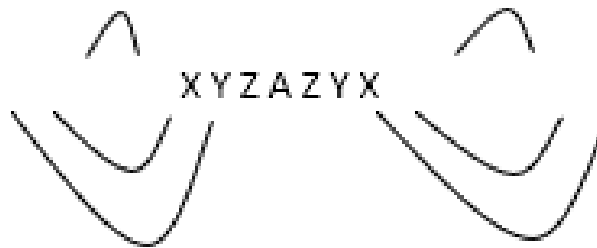
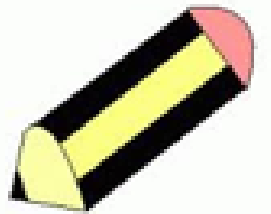
TATTARRATTAT

RACECAR

ROTATOR

PULLUP

REDDER ...



### Exemplo 3



- Função recursiva *isPalindrome()* , em Java:

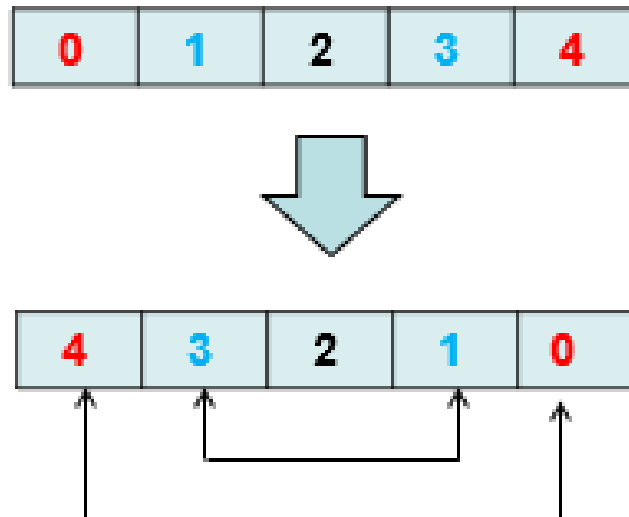
```
import java.util.*;
public class Exemplo_3
{   public static void main(String args[])
    {   Scanner sc = new Scanner(System.in);
        System.out.println("Digite uma String: ");
        String x = sc.nextLine();
        if(isPalindrome(x)) System.out.println("A String " + x + " é um PALÍNDROMO...");
        else System.out.println("A String " + x + " não é um PALÍNDROMO...");
    }

    public static boolean isPalindrome(String s)
    {   if(s.length() <= 1) return true; //caso base
        if(s.charAt(0) != s.charAt((s.length() - 1))) return false;
        else return isPalindrome(s.substring(1,s.length()-1));
    }
}
```

### Exemplo 4



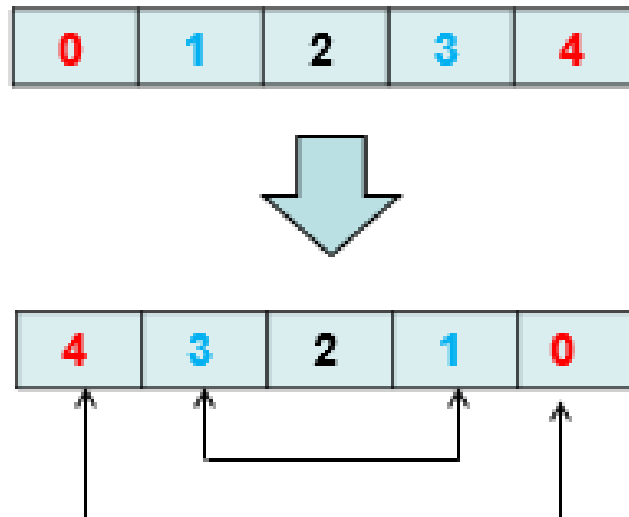
- Escreva uma implementação em Java que consiste em reverter os  $n$  elementos de um **vetor**, de modo que o primeiro elemento torna-se o último, o segundo elemento o penúltimo, e assim por diante.



### Exemplo 4



- O problema consiste em reverter os  $n$  elementos de um **vetor**, de modo que o primeiro elemento torna-se o último, o segundo elemento o penúltimo, e assim por diante:



### Exemplo 4



- Revertendo os  $n$  elementos do **vetor** com **Recursão Linear**, em Pseudocódigo:

```
reverte_array(A,i,j)
{
    if (i<j)
    {
        swap(v,i,j);
        reverte_array(v, i+1, j-1);
    }
}
```

### Exemplo 4



- Revertendo os  $n$  elementos do **vetor** com **Recursão Linear**, em Java:

```
1 public class Exemplo_4
2 {   public static void main(String args[])
3     {   int tab[] = new int[6];
4         for(int i = 0; i < tab.length; i++)
5         {   tab[i] = i;
6             }
7         imprime(tab);
8         reverte_array(tab, 0, tab.length-1);
9         imprime(tab);
10    }
11
12    public static void reverte_array(int v[], int i, int j)
13    {   if(i<j)
14        {   swap(v,i,j);
15            reverte_array(v, i+1, j-1);
16        }
17    }
18 }
```

```
19     public static void swap(int v[], int i, int j)
20     {   int trab = v[i];
21         v[i] = v[j];
22         v[j] = trab;
23     }
24
25     public static void imprime(int v[])
26     {   System.out.print("Vetor: ");
27         for(int i = 0; i < v.length; i++)
28         {   System.out.print(v[i] + " ");
29             }
30         System.out.println("");
31     }
32 }
33 }
```

# ECM306 – Tópicos Avançados em Estrutura de Dados

## Referências bibliográficas



- CORMEN, T.H. et al. Algoritmos: Teoria e Prática (Caps. 13). Campus. 2002.
- ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (Cap. 1). 2.ed. Thomson, 2004.
- FEOFILOFF, P. Minicurso de Análise de Algoritmos, 2010. Disponível em:  
<http://www.ime.usp.br/~pf/livrinho-AA/>
- DOWNEY, A.B. *Analysis of algorithms* (Cap. 2), Em: *Computational Modeling and Complexity Science*. Disponível em:  
<http://www.greenteapress.com/compmo/html/book003.html>
- ROSA, J.L. Notas de Aula de Introdução a Ciência de Computação II. Universidade de São Paulo. Disponível em:  
<http://coteia.icmc.usp.br/mostra.php?ident=639>

# ECM306 – Tópicos Avançados em Estrutura de Dados

## Referências bibliográficas

- GOODRICH, Michael T. et al: *Algorithm Design and Applications*. Wiley, 2015.
- LEVITIN, Anany. *Introduction to the Design and Analysis of Algorithms*. Pearson, 2012.
- SKIENA, Steven S. *The Algorithm Design Manual*. Springer, 2008.
- Série de Livros Didáticos. *Complexidade de Algoritmos*. UFRGS.
- BHASIN, Harsh. *Algorithms – Design and Analysis*. Oxford University Press, 2015.
- FREITAS, Aparecido V. de – 2022 – Estruturas de Dados: Notas de Aula.
- CALVETTI, Robson - 2015 – Estruturas de Dados: Notas de Aula.





# ECM306 – Tópicos Avançados em Estrutura de Dados

## Aula 07

FIM