

Recursão

Roland Teodorowitsch

Algoritmos e Estruturas de Dados I - Escola Politécnica - PUCRS

17 de agosto de 2023

Recursão

Leitura(s) Recomendada(s)



Seção 3.5

GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de dados e algoritmos em Java**. Tradução: Bernardo Copstein. 5. ed. Porto Alegre: Bookman, 2013. xxii, 713 p. E-book. ISBN 9788582600191. Tradução de: Data Structures and Algorithms in Java, 5th Edition. Disponível em: <<https://integrada.minhabiblioteca.com.br/#/books/9788582600191/>>. Acesso em: 01 ago. 2023.

Algoritmo Recursivo

- É um algoritmo que chamada a si próprio
- **Importante:** o algoritmo deve garantir que a recursão termine
 - Caso contrário cria-se um “laço infinito” que causará um estouro de pilha

```
void loop() {  
    loop();  
}
```

- Geralmente coloca-se uma condição de parada (situação em que a chamada recursiva NÃO é realizada) no início da implementação

Exemplo clássico: Fatorial

- O fatorial de n é normalmente definido como:

$$n! = \prod_{k=1}^n k = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1, \quad \forall n \in \mathbb{N}$$

- Por exemplo:

$$5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

- Mas também pode-se usar a sua definição recursiva:

$$n! = \begin{cases} 1, & \text{se } n = 0 \text{ ou } n = 1 \\ n \times (n-1)!, & \text{se } n > 1 \end{cases}$$

Implementação do Fatorial

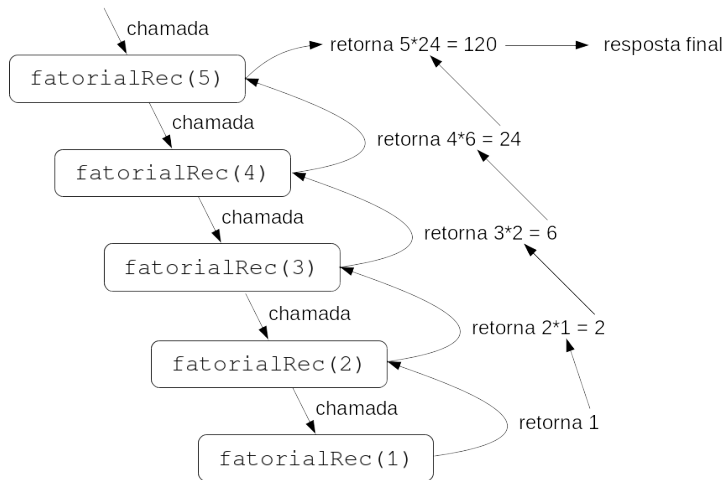
- Iterativo

```
unsigned int fatorial(unsigned int n) {  
    unsigned int res = 1;  
    for (unsigned int i=2; i<=n; ++i)  
        res *= i;  
    return res;  
}
```

- Recursivo

```
unsigned int fatorialRec(unsigned int n) {  
    if (n <= 1) return 1;  
    return n * fatorialRec(n-1);  
}
```

Rastreamento Recursivo



Exemplo: Contagem

- Iterativo

```
#include <iostream>
void contagem(int n) {
    for (int i=1; i<=n; i++)
        std::cout << i << std::endl;
}
```


Exemplo: Contagem

- Iterativo

```
#include <iostream>
void contagem(int n) {
    for (int i=1; i<=n; i++)
        std::cout << i << std::endl;
}
```

- Recursivo

```
#include <iostream>
void contagemRec(int n) {
    if (n == 0) return;
    contagemRec(n-1);
    std::cout << n << std::endl;
}
```

Exercício: Contagem Regressiva

- Iterativo

```
#include <iostream>
void contagemRegressiva(int n) {
    for (int i=n; i>=1; i--)
        std::cout << i << std::endl;
}
```

Exercício: Contagem Regressiva

- Iterativo

```
#include <iostream>
void contagemRegressiva(int n) {
    for (int i=n; i>=1; i--)
        std::cout << i << std::endl;
}
```

- Recursivo

```
#include <iostream>
void contagemRegressivaRec(int n) {
    if (n == 0) return;
    std::cout << n << std::endl;
    contagemRegressivaRec(n-1);
}
```

Exercício: Geração de Índices de uma Matriz

- Iterativo

```
#include <iostream>
void indicesMatriz(int lin, int col) {
    for (int i=0; i<lin; ++i)
        for (int j=0; j<col; ++j)
            std::cout << i << ", " << j << std::endl;
}
```

Exercício: Geração de Índices de uma Matriz

- Iterativo

```
#include <iostream>
void indicesMatriz(int lin, int col) {
    for (int i=0; i<lin; ++i)
        for (int j=0; j<col; ++j)
            std::cout << i << ", " << j << std::endl;
}
```

- Recursivo

```
#include <iostream>
void indicesMatrizRec(int lin, int col) {
    if (lin < 1) return;
    indicesMatrizRec(--lin, col);
    for (int j=0; j<col; ++j)
        std::cout << lin << ", " << j << std::endl;
}
```

Exercício: Geração de Índices de uma Matriz

- Iterativo

```
#include <iostream>
void indicesMatriz(int lin, int col) {
    for (int i=0; i<lin; ++i)
        for (int j=0; j<col; ++j)
            std::cout << i << ", " << j << std::endl;
}
```

- Recursivo

```
#include <iostream>
void indicesMatrizRec(int lin, int col) {
    if (lin < 1) return;
    indicesMatrizRec(--lin, col);
    for (int j=0; j<col; ++j)
        std::cout << lin << ", " << j << std::endl;
}
```

- Desafio:** faça uma versão recursiva do algoritmo acima sem usar nenhum laço!

Recursividade [*]

- Poderosa ferramenta de programação
- Apesar de bastante empregada, nem sempre ela deve ser aplicada
 - É preciso analisar o problema e ver se necessita de uma solução recursiva
- Quando bem empregada pode tornar a solução de um problema clara, simples e consisa

Vantagens [*]

- Rotinas mais concisas
- Relação direta com uma prova por indução matemática
 - Indução matemática: metodo de prova matemática usado para demonstrar a verdade de um número infinito de proposições
 - Válida se funciona para n igual a 0 ou 1
 - Válida se vale para n igual a k e $k + 1$
 - Facilita verificar a correção

Desvantagens [*]

- Cada chamada recursiva implica em um custo (tempo e espaço)
 - Informações são armazenadas na pilha
 - Para cada chamada realizada, um conjunto de variáveis locais é alocado (criado)
 - Cada chamada requer
 - O empilhamento de parâmetros e endereços de retorno da função que chama
 - A alocação de variáveis locais da função
 - Cada retorno requer
 - A desalocação das variáveis locais da função
 - O desempilhamento do endereço de retorno

Algoritmos de Pesquisa

Algoritmos de Pesquisa

- Localizam um elemento dentro de uma coleção
- Podem retornar
 - **Valor booleano** (true se o elemento existir na coleção ou false, em caso contrário) ou
 - **Índice** (posição) do elemento na coleção (ou -1 se não encontrar)
- Para coleções **desordenadas**, deve-se usar um algoritmo de **pesquisa linear**
- Quando a coleção está **ordenada**, pode-se usar um algoritmo de **pesquisa binária**

Pesquisa Linear

- Serve para coleções **desordenadas**
- Estratégia: comparar o elemento procurado com cada item da coleção até encontrar ou até chegar ao fim
- Melhor caso: o elemento procurado é o primeiro da lista
- Pior caso: o elemento procurado **NÃO** existe na lista
- Também poderia ser aplicado sobre coleções ordenadas

Pesquisa Linear

- Serve para coleções **desordenadas**
- Estratégia: comparar o elemento procurado com cada item da coleção até encontrar ou até chegar ao fim
- Melhor caso: o elemento procurado é o primeiro da lista
- Pior caso: o elemento procurado **NÃO** existe na lista
- Também poderia ser aplicado sobre coleções ordenadas
- Complexidade: $O(n)$

Implementação da Pesquisa Linear

- Iterativo

```
int pesquisaLinear(int *dados, int tam, int valor) {  
    for (int i=0; i<tam; i++)  
        if (valor == dados[i])  
            return i;  
    return -1;  
}
```

Implementação da Pesquisa Linear

- Iterativo

```
int pesquisaLinear(int *dados, int tam, int valor) {  
    for (int i=0; i<tam; i++)  
        if (valor == dados[i])  
            return i;  
    return -1;  
}
```

- Recursivo

```
int pesquisaLinearRec(int *dados, int tam, int valor) {  
    if (tam == 0) return -1;  
    --tam;  
    if (dados[tam] == valor) return tam;  
    return pesquisaLinearRec(dados, tam, valor);  
}
```

Pesquisa Binária

- Serve **APENAS** para coleções **ordenadas**
- Estratégia:
 - Compara o valor procurado com o elemento do meio
 - Se for igual, encontrou
 - Se for menor, continua-se a busca na metade inferior (desconsidera a metade superior)
 - Se for maior, continua-se a busca na metade superior (desconsidera a metade inferior)
 - Repete-se o procedimento até que o valor procurado seja encontrado ou até que não se consiga dividir a coleção

Pesquisa Binária

- Serve **APENAS** para coleções **ordenadas**
- Estratégia:
 - Compara o valor procurado com o elemento do meio
 - Se for igual, encontrou
 - Se for menor, continua-se a busca na metade inferior (desconsidera a metade superior)
 - Se for maior, continua-se a busca na metade superior (desconsidera a metade inferior)
 - Repete-se o procedimento até que o valor procurado seja encontrado ou até que não se consiga dividir a coleção
- Complexidade: $O(\log n)$

Pesquisa Binária Iterativa

```
int pesquisaBinaria(int *dados, int ini, int fim, int valor) {  
    while (ini <= fim) {  
        int meio = (ini + fim) / 2;  
        if (valor == dados[meio])  
            return meio;  
        else if (valor < dados[meio])  
            fim = meio - 1;  
        else  
            ini = meio + 1;  
    }  
    return -1;  
}
```

Pesquisa Binária: Exemplo (valor = 18)

• Passo 1:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
↑				↑					↑
ini				meio					fim

• Passo 2:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
					↑		↑		↑
					ini		meio		fim

• Passo 3:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
								↑	↑
								ini	fim
									meio

Valor 18 encontrado no índice 8!

Pesquisa Binária: Exemplo (valor = 5)

• Passo 1:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
↑				↑					↑
ini				meio					fim

• Passo 2:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
↑	↑		↑						
ini	meio		fim						

Pesquisa Binária: Exemplo (valor = 5)

• Passo 3:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
		↑	↑						
		ini	fim						
		meio							

• Passo 4:

0	1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18	20
	↑	↑							
	fim	ini							

Valor 5 **NÃO** encontrado!

Pesquisa Binária Recursiva

```
int pesquisaBinariaRec(int *dados, int ini, int fim, int valor) {  
    if ( ini > fim ) return -1;  
    int meio = (ini + fim) / 2;  
    if (valor == dados[meio])  
        return meio;  
    else if (valor < dados[meio])  
        return pesquisaBinariaRec(dados, ini, meio-1, valor);  
    else  
        return pesquisaBinariaRec(dados, meio+1, fim, valor);  
}
```

Exercícios

Exercícios

- 1 Dado um valor inteiro e positivo (n), o valor da constante de *Euler* poder ser calculando com precisão diretamente proporcional a n através da fórmula:

$$E = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!}$$

Implemente, em C, um método recursivo que recebe n , retornando o valor de *Euler* calculado usando a fórmula acima.

- 2 Considere a função abaixo, que implementa o somatório de um vetor e implemente a sua versão recursiva.

```
int somatorio(int *dados, int tam) {  
    int soma = 0;  
    for (int i=0; i<tam; i++)  
        soma += dados[i];  
    return soma;  
}
```


Exercícios

- ③ Considere a função que implementa o algoritmo *Bubble Sort* (apresentada na unidade anterior) e implemente a sua versão recursiva).
- ④ Implemente uma função recursiva que inverte os elementos de um vetor, trocando o primeiro elemento com o último, o segundo com o penúltimo, e assim sucessivamente.
- ⑤ Implemente uma função recursiva que verifica se uma cadeia de caracteres recebida como parâmetro é um palíndromo ou não. Por exemplo, “socorrammesubinoonibusemmarrocos” é um palíndromo.

Soluções

Soluções

```
1 unsigned int fatorial(unsigned int n);

double eulerRec(unsigned int n) {
    if (n == 0) return 1;
    else return 1.0 / fatorial(n) + eulerRec(n-1);
}
```

```
2 int somatorioRec(int *dados, int tam) {
    if (tam == 0) return 0;
    --tam;
    return dados[tam] + somatorioRec(dados, tam);
}
```

```
3 void bubbleSortRec(int *dados, int tam) {
    int trocou = 0;
    --tam;
    for (int i=0; i<tam; ++i)
        if (dados[i] > dados[i+1]) {
            int aux = dados[i];
            dados[i] = dados[i+1];
            dados[i+1] = aux;
            trocou = 1;
        }
    if (trocou) bubbleSortRec(dados, tam);
}
```

Soluções

4

```
void invertVetorRec(int *dados, int ini, int fim) {  
    if (ini >= fim) return;  
    int aux = dados[ini];  
    dados[ini] = dados[fim];  
    dados[fim] = aux;  
    invertVetorRec(dados, ini+1, fim-1);  
}
```

5

```
#include <string>  
int palindromoRec(std::string palavra, int ini, int fim) {  
    if (ini >= fim) return 1;  
    else if ( palavra.at(ini) != palavra.at(fim) ) return 0;  
    else return palindromoRec(palavra, ini+1, fim-1);  
}
```

Créditos

Créditos

- Estas lâminas contêm trechos adaptados de materiais criados e disponibilizados pelo professor Iga Janicki Weber [*].