Recursão

Prof. Fernando Figueira
(adaptado do material do Prof. Rafael Beserra Gomes)

UFRN

Material compilado em 26 de novembro de 2025.

Recursão


Recursão
Subproblemas
Recursão
Recursão
Subproblemas
Recursão
Subproblemas
Recursão
Rafael Beserra Gomes
Recursão
Rafael Beserra Gomes
Recursão

Recursão

Uma **recursão** ocorre quando algo é definido em função de si próprio.

Recursão

Nas artes:

Recursão

Na natureza:

Recursão

Na natureza:

[1]

[1] By Chiswick Chap - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=19619156

Recursão

Na matemática:

Fatorial

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

Recursão

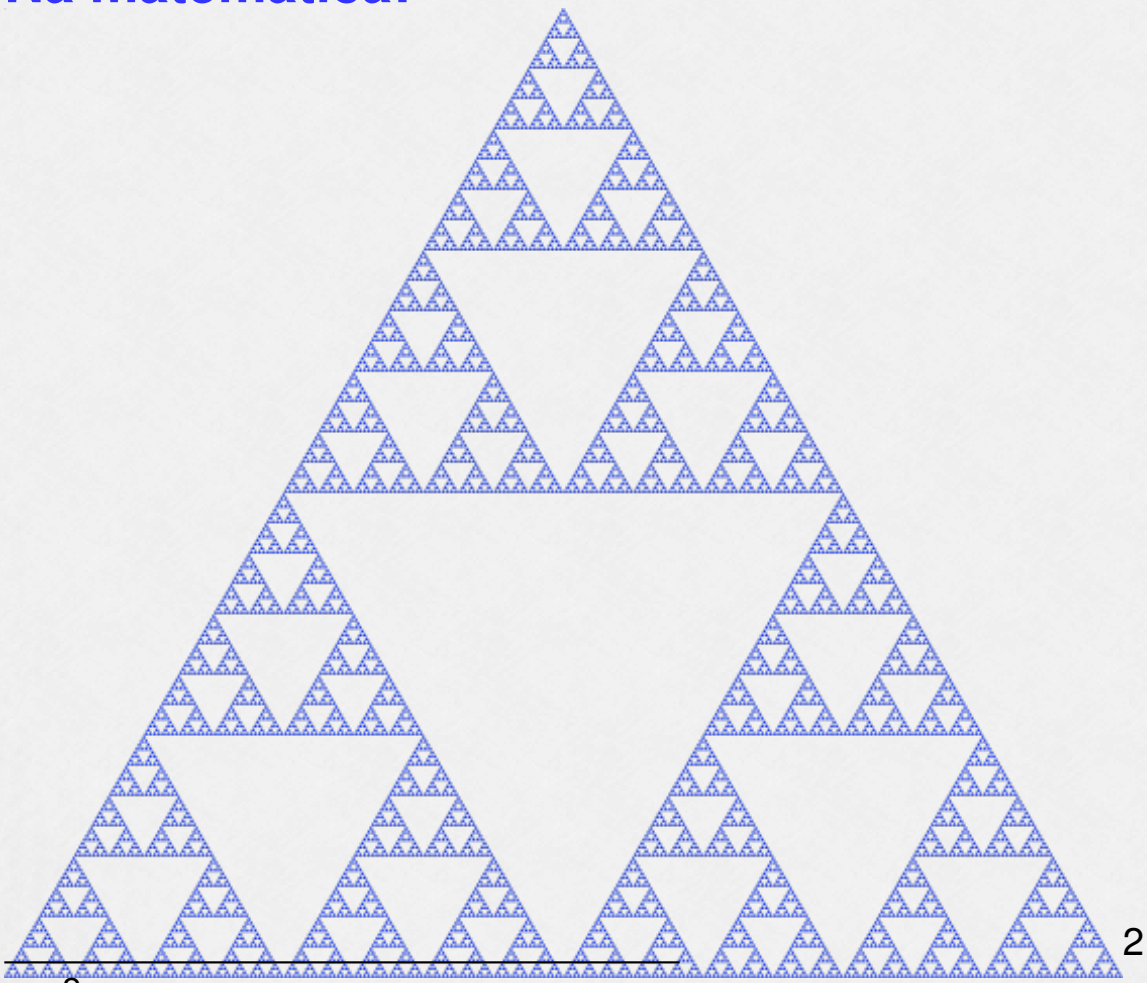
Na matemática:

Sequência de Fibonacci:

$$f_n = \begin{cases} 1 & \text{se } n = 1 \text{ou } n = 2 (\text{caso base}) \\ f_{n-1} + f_{n-2} & \text{se } n > 2 \end{cases}$$

Recursão

Na matemática:



Implementação

Podemos aplicar o conceito de **recursão** com uma função chamando a si.

```
1 void funcao(int x) {
2     printf("%d\n", x);
3     if(x >= 1)
4         funcao(x-1);
5     printf("%d\n", -x);
6 }
```

- quando durante a execução de uma função (digamos f_1), ocorrer uma chamada à outra função (digamos f_2, seja a própria ou outra), a execução de f_1 é interrompida e retomada somente **após a conclusão** da chamada f_2
- quando a execução da função é interrompida, as variáveis pertencentes à essa chamada são gravadas e recuperadas tão logo a execução seja retomada.

Veja com atenção o que ocorre quando uma função chama a si
É fundamental para implementar a recursão.

```
1 void funcao(int x) {
2     printf("%d\n", x);
3     if(x >= 1)
4         funcao(x-1);
5     printf("%d\n", -x);
6 }
```

funcao(3):
print 3
funcao(2)
print -3

A chamada funcao(3) é interrompida até que funcao(2) finalize
Saída Padrão: 3

```c
void funcao(int x) {
    printf("%d\n", x);
    if(x >= 1)
        funcao(x-1);
    printf("%d\n", -x);
}
```

funcao(3):
print 3
funcao(2)
print -3

→

funcao(2):
print 2
funcao(1)
print -2

A chamada funcao(2) é interrompida até que funcao(1) finalize
Saída Padrão: 3 2

Atenção: cada chamada de função contém sua própria cópia de parâmetros e variáveis locais
Exemplo: x = 3 em função(3), x = 2 em função(2) são x's distintos!

```c
void funcao(int x) {
    printf("%d\n", x);
    if(x >= 1)
        funcao(x-1);
    printf("%d\n", -x);
}
```

funcao(3):
print 3
funcao(2)
print -3

→

funcao(2):
print 2
funcao(1)
print -2

→

funcao(1):
print 1
print -1

A chamada funcao(1) é finalizada sem chamadas recursivas funcao(2) é retomada
Saída Padrão: 3 2 1 -1

```
1 void funcao(int x) {
2     printf("%d\n", x);
3     if(x >= 1)
4         funcao(x-1);
5     printf("%d\n", -x);
6 }
```

funcao(3):
print 3
funcao(2)
print -3

→

funcao(2):
print 2
funcao(1)
print -2

print -2 é executado
finalizando agora funcao(2)
funcao(3) é retomada
Saída Padrão: 3 2 1 -1 -2

```
1 void funcao(int x) {
2     printf("%d\n", x);
3     if(x >= 1)
4         funcao(x-1);
5     printf("%d\n", -x);
6 }
```

funcao(3):
print 3
funcao(2)
print -3

print -3 é executado
finalizando agora funcao(3)
Saída Padrão: 3 2 1 -1 -2 -3

Exemplos

Exemplo: fatorial

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

```c
#include <stdio.h>

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

int main() {

    int x;
    scanf("%d", &x);
    printf("%d! = %d\n", x, fatorial(x));
    return 0;
}
```

Exemplo: fatorial

- Professor, resolvi o mesmo problema com um for 

```c
#include <stdio.h>

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

int main() {

    int x;
    scanf("%d", &x);
    printf("%d! = %d\n", x, fatorial(x));
    return 0;
}
```

Pode resolver assim também! Mas há vários problemas em que o pensamento recursivo é essencial para a resolução, como os encontrados nos estudos em:

- backtracking
- divisão e conquista
- programação dinâmica (não confunda com alocação dinâmica)

São tópicos mais avançados que serão visto em momento oportuno no curso.

Exemplo: Fibonacci

$$f_n = \begin{cases} 1 & \text{se } n = 1 \text{ ou } n = 2 \\ f_{n-1} + f_{n-2} & \text{se } n > 2 \end{cases}$$

```c
#include <stdio.h>

int fib(int n) {
    if(n == 1 || n == 2) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}

int main() {

    int x;
    scanf("%d", &x);
    printf("%d\n", fib(x));

    return 0;
}
```

Subproblemas

Subproblemas

Subproblemas

- Geralmente utilizamos o conceito de recursão para resolver **problemas**
- O **problema** pode ser resolvido em função de **subproblemas**, os quais são resolvidos recursivamente
- **Caso base:** algum(ns) do(s) **subproblema(s)** não são resolvidos recursivamente
- Usa-se esse conceito para resolver, por exemplo:
  - ordenação
  - busca
  - vários problemas de grafos
  - exploração de um espaço solução
  - análise numérica

Subproblemas

O que são subproblemas

Subproblemas

- Um subproblema consiste em um mesmo problema, mas com parâmetros diferentes, geralmente representando uma instância menor do problema
- Por exemplo: fatorial
  fatorial(0) = 1 (**caso base**)
  fatorial(n) = fatorial(n-1)*n para n > 0
- Por exemplo: sequência de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, ...)
  fib(1) = 1 (**caso base**)
  fib(2) = 1 (**caso base**)
  fib(n) = fib(n-1) + fib(n-2) para n > 2
  O cálculo do n-ésimo elemento de Fibonacci envolve a resolução de dois **subproblemas**: o (n-1)-ésimo e o (n-2)-ésimo elemento da sequência.

Subproblemas

✎ Exercício em sala

Crie uma função **recursiva** para calcular:

$$\sum_{i=1}^{n} = 1 + 2 + ... + n$$

Sugestão para assinatura da função:
int somatorio(int n);

Subproblemas

✎ Exercício em sala

Crie uma função **recursiva** para determinar se uma string é um palíndromo:
Sugestão para assinatura da função:
*int palindromo(char *s, int indiceInicial, int indiceFinal);*

Subproblemas

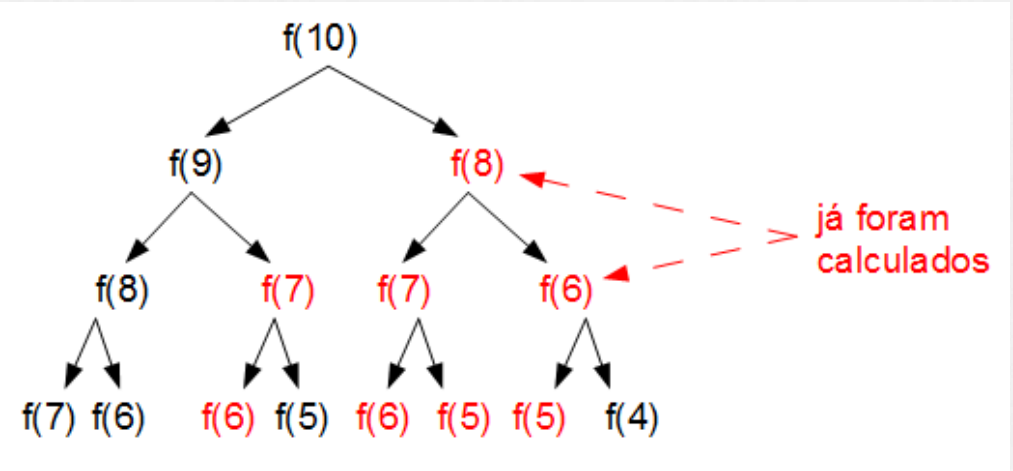
✎ Exercício em sala

Crie uma função **recursiva** para determinar se um vetor de n inteiros está ordenado:
Sugestão para assinatura da função:
*int ordenado(int *v, int n);*

Subproblemas

Sobreposição de subproblemas

- se um mesmo subproblema ocorre em várias etapas da resolução, então ocorre uma **sobreposição de subproblemas**
- no exemplo da sequência de Fibonacci:



- podemos otimizar usando a técnica chamada de **memoization**:
  - sinaliza-se com um valor específico os subproblemas ainda não calculados
  - se já tiver sido calculado, recupera-se o valor
  - caso contrário, calcula-se e armazena-se o valor