# Como Implementar Recursividade

...

30 de setembro de 2023

Aluno: Jônatas Gomes Lima
Disciplina: Algoritmos e Complexidade
Professor: Heleno Cardoso

# Visão Geral

## Definição

Mais do que uma ideia ou conceito, a recursividade é um mecanismo fundamental na programação onde uma função ou objeto definido refere-se ao próprio objeto sendo definido.

## Importância

A importância da recursividade na programação é inegável, pois esse conceito desempenha um papel fundamental em muitas áreas computacionais.

# Objetivo da apresentação

O objetivo desta apresentação é fornecer uma compreensão abrangente da recursividade na programação, abordando alguns pontos como:

- Compreensão Conceitual
- Exploração da Importância
- Aplicação Prática

# Entendendo em detalhes a importância da recursividade

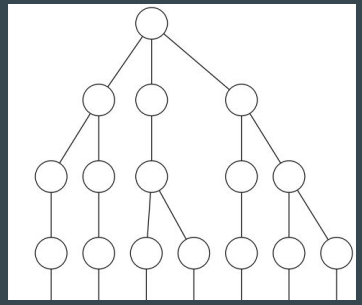
# Detalhes da Importância

## Solução Elegante de Problemas Complexos

O primeiro ponto a se destacar no que diz respeito à importância da recursividade é que ela oferece uma maneira intuitiva e elegante de resolver problemas. Ela oferece a possibilidade de quebrar um problema em subproblemas menores que serão resolvidos de maneira recursiva, o que possibilita a simplificação da lógica do programa.

## Manipulação de Estruturas de Dados Complexas

Em estruturas de dados como árvores, grafos e listas encadeadas, a recursão é uma ferramenta poderosa para navegar e manipular os elementos de forma eficiente.



# Detalhes da Importância

## Legibilidade do Código

A recursão auxilia também ao manter o código mais legível e compreensível, pois facilita a colaboração nos projetos e a manutenção do código.

- **Reutilização de Código**
    - As funções recursivas podem ser reutilizadas em várias partes do programa. O que economiza tempo, esforço e auxilia ainda mais na legibilidade.

## Modelagem Natural de Problemas

Alguns problemas têm uma estrutura recursiva por natureza, por exemplo o cálculo de fatorial:

```c
#include <stdio.h>

// Função recursiva para calcular o fatorial
unsigned long long int calcularFatorial(int n) {
    // Caso base: fatorial de 0 ou 1 é 1
    if (n == 0 || n == 1) {
        return 1;
    } else {
        // Caso recursivo: fatorial de n é n multiplicado pelo fatorial de n-1
        return n * calcularFatorial(n - 1);
    }
}

int main() {
    int numero;

    printf("Digite um número inteiro positivo para calcular o fatorial: ");
    scanf("%d", &numero);

    if (numero < 0) {
        printf("Não existe fatorial para números negativos.\n");
    } else {
        unsigned long long int resultado = calcularFatorial(numero);
        printf("%d! = %llu\n", numero, resultado);
    }

    return 0;
}
```

# Tipos de Recursividade

| | |
|---|---|
| Recursividade de Cauda | • Neste tipo de recursão, a chamada recursiva é a última operação executada dentro da função<br>• Isso significa que não há cálculos ou operações adicionais após a chamada recursiva |
| Recursividade Mútua | • A recursividade mútua ocorre quando duas ou mais funções chamam umas às outras de forma recursiva<br>• Essa técnica é usada para resolver problemas que naturalmente envolvem duas ou mais entidades relacionadas |
| Recursividade Aninhada | • A recursividade aninhada ocorre quando uma função recursiva chama a si mesma dentro de uma função diferente |

# Exemplo de Recursividade

## Recursividade de Cauda (Linear)

```c
#include <stdio.h>

// Função recursiva para calcular o fatorial
unsigned long long int calcularFatorial(int n) {
    // Caso base: fatorial de 0 ou 1 é 1
    if (n == 0 || n == 1) {
        return 1;
    } else {
        // Caso recursivo: fatorial de n é n multiplicado pelo fatorial de n-1
        return n * calcularFatorial(n - 1);
    }
}

int main() {
    int numero;

    printf("Digite um número inteiro positivo para calcular o fatorial: ");
    scanf("%d", &numero);

    if (numero < 0) {
        printf("Não existe fatorial para números negativos.\n");
    } else {
        unsigned long long int resultado = calcularFatorial(numero);
        printf("%d! = %llu\n", numero, resultado);
    }

    return 0;
}
```

```
Digite um número inteiro positivo para calcular o fatorial: 5
5! = 120

[Program finished]
```

## Recursividade Simples

```c
int fibonacci(int n) {
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

# Como Implementar a Recursividade

## Definindo o Problema Recursivo

Antes de implementar a recursividade, é fundamental compreender o problema que deseja resolver de forma recursiva. O desenvolvedor deve se perguntar se o problema pode ser dividido em subproblemas menores, semelhantes ao problema original. Isso é essencial para determinar se a recursividade é apropriada.

## Identificando o Caso Base

Todo algoritmo recursivo precisa de um caso base, que é a condição que determina quando a recursão deve parar. O caso base é a resposta direta e geralmente é uma situação trivial que não requer recursão. É importante identificar claramente o caso base para evitar recursões infinitas.

# Como Implementar a Recursividade

## Definindo a Chamada Recursiva

Após identificar o caso base, você precisa definir como a função chama a si mesma com um problema menor. Essa chamada recursiva deve se aproximar do caso base a cada iteração, para que a recursão termine eventualmente.

## Garantindo a Convergência

Certifique-se de que, em cada iteração da recursão, o problema esteja se movendo em direção ao caso base. Isso é chamado de convergência e é fundamental para evitar recursões infinitas

# Como Implementar a Recursividade

## Implementando a Função Recursiva

Após os passos anteriores você pode implementar a função recursiva usando a definição do caso base e a chamada recursiva. Apenas certifique-see de que a função está retornando resultados corretos e está se aproximando do caso base.

## Chamando a Função Recursiva

Chame a função com os parâmetros apropriados no código onde você precise usar a função.

## Otimizando a Recursão

A recursividade consome muitos recursos computacionais, o que pode diminuir a eficiência do programa, por isso é interessante que se otimize o código sempre que possível.

# Obrigado!!!