Desafio: Sequência de Fibonacci Recursiva

A sequência de Fibonacci é uma série de números onde cada número é a soma dos dois anteriores. A sequência começa com 0 e 1. Ou seja:

- F(0) = 0
- F(1) = 1
- F(n) = F(n-1) + F(n-2) para n > 1

Crie uma função recursiva que calcule o n-ésimo número de Fibonacci. Depois, escreva um programa que peça para o usuário inserir um número inteiro (n) e imprima o valor de F(n).

Exemplo de Entrada e Saída:

Entrada:

In [ ]:
```
Digite um número: 6
```

Saída:

In [ ]:
```
O número Fibonacci na posição 6 é: 8
```

In [ ]:

In [1]:
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("Digite um número");
            int n = scanner.nextInt();

            System.out.println("O número Fibonacci na posição " + n + " é: " + f
        }
    }

    public static int fibonacci(int number) {
        if (number == 0 | number == 1) {
            return number;
        } else {
            return fibonacci(number - 1) + fibonacci(number - 2);
        }
    }
}
```

In [ ]:

Melhorias:

1. **Uso de Memoization**: Para evitar recomputação desnecessária.
2. **Substituição por Iteração**: Iterar em vez de usar recursão reduz o risco de estouro de pilha (stack overflow) para entradas grandes.

Versão com Memoization em Java

In [1]:
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("Digite um número");
            int n = scanner.nextInt();

            // Array para memoization
            int[] memo = new int[n + 1];
            for (int i = 0; i <= n; i++) {
                memo[i] = -1; // Inicializar com -1 para indicar valores não cal
            }

            System.out.println("O número Fibonacci na posição " + n + " é: " + f
        }
    }

    public static int fibonacci(int number, int[] memo) {
        if (number == 0 || number == 1) {
            return number;
        }

        // Verificar se o valor já foi calculado
        if (memo[number] != -1) {
            return memo[number];
        }

        // Calcular e armazenar no memo
        memo[number] = fibonacci(number - 1, memo) + fibonacci(number - 2, memo)
        return memo[number];
    }
}
```

In [ ]:
```
```

Versão Iterativa (mais eficiente)

Esta versão evita a recursão completamente.

In [ ]:
```java
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        try (Scanner scanner = new Scanner(System.in)) {
            System.out.println("Digite um número");
            int n = scanner.nextInt();
```

```
            System.out.println("O número Fibonacci na posição " + n + " é: " + f
        }
    }

    public static int fibonacci(int number) {
        if (number == 0) return 0;
        if (number == 1) return 1;

        int prev1 = 0, prev2 = 1, current = 0;
        for (int i = 2; i <= number; i++) {
            current = prev1 + prev2;
            prev1 = prev2;
            prev2 = current;
        }

        return current;
    }
}
```

Comparação:

1. Memoization:

- Fácil de adaptar ao seu código atual.
- Evita cálculos redundantes.
- A complexidade é reduzida para O(n).

2. Iterativa:

- Mais eficiente em termos de memória (O(1)).
- Evita problemas de pilha em números grandes.
- Boa escolha para aplicações práticas.

In [ ]: