

ALGORITMOS E ESTRUTURAS DE DADOS 2023/2024 FUNÇÃO-MEMÓRIA

Armanda Rodrigues

18 de outubro de 2023

Fibonacci Recursivo

```
//Requires: n >= 0
public static long fibonacciRec( int n ){

    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return fibonacciRec(n - 1) + fibonacciRec(n - 2);
}
```

Número de Chamadas Recursivas

$$\text{numCR}(n) = \begin{cases} 0, & n = 0 \\ 0, & n = 1 \\ \text{numCR}(n - 1) + \text{numCR}(n - 2) + 2, & n \geq 2 \end{cases}$$

Fibonacci Recursivo

Número de Chamadas Recursivas

$$\text{numCR}(n) = \begin{cases} 0, & n = 0 \\ 0, & n = 1 \\ \text{numCR}(n - 1) + \text{numCR}(n - 2) + 2, & n \geq 2 \end{cases}$$

Prova-se que $\text{numCR}(n) = O(\phi^n)$, ou seja
 $\text{fibonacciRec}(n) = O(\phi^n)$, com $\phi = (1 + \sqrt{5}) / 2 \approx 1.6180\dots$

O que significa que a função recursiva de Fibonacci tem complexidade temporal exponencial

Técnica da Memorização

- Consiste em guardar todos os resultados conseguidos em chamadas recursivas, da primeira vez que a respetiva chamada for ativada
- Na ocasião da necessidade de execução de uma chamada recursiva, verifica-se, antes da execução da mesma, se o seu resultado já foi calculado anteriormente
- A ativação de uma determinada chamada recursiva só se dá uma vez
- Esta técnica reduz um algoritmo tipicamente exponencial para a dimensão da memória necessária para guardar todos os resultados gerados pelo algoritmo
- A complexidade espacial da solução cresce

Fibonacci Recursivo

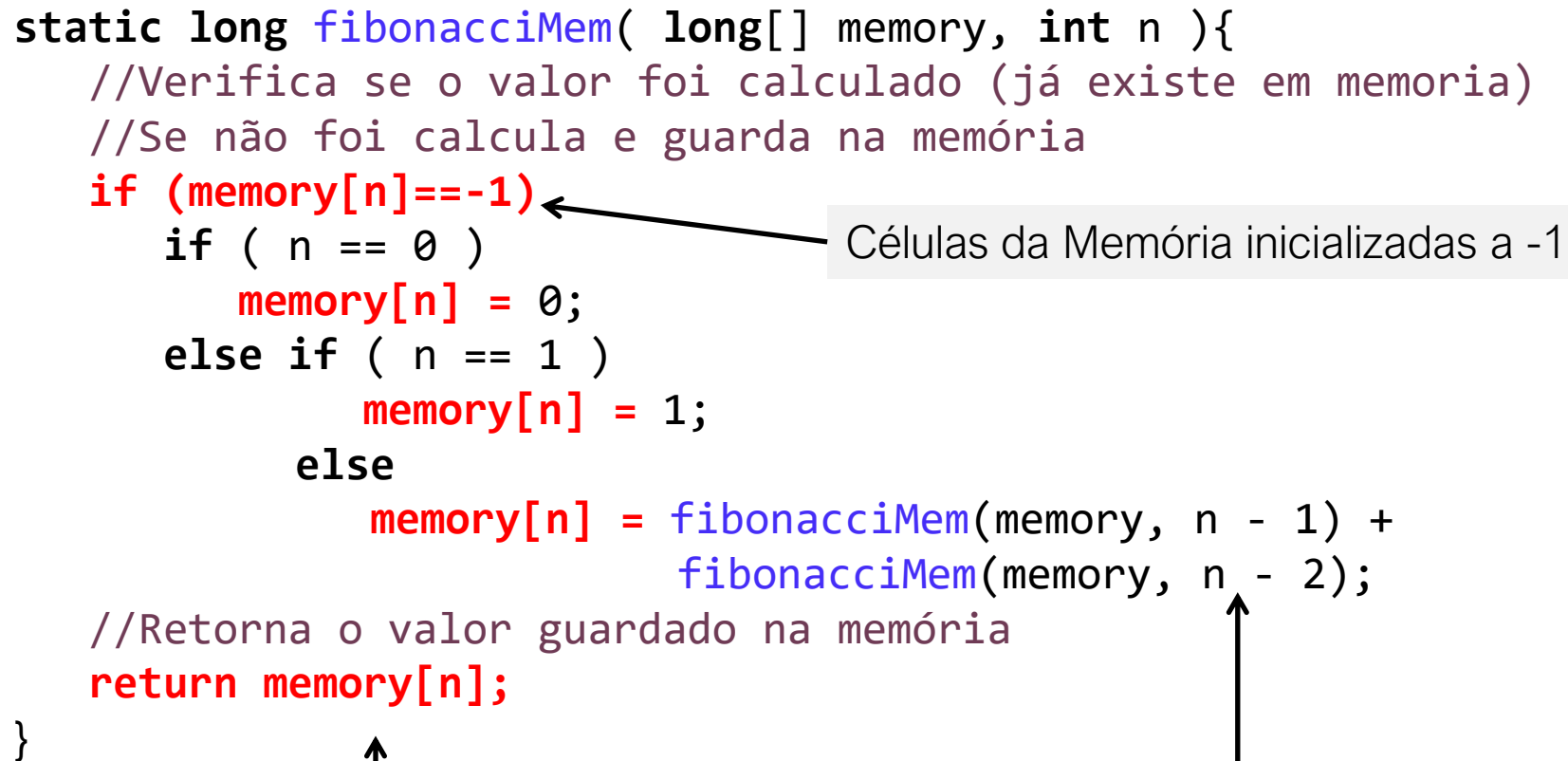
```
//Requires: n >= 0
public static long fibonacciRec( int n ){

    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
    else
        return fibonacciRec(n - 1) + fibonacciRec(n - 2);
}
```

- O método recursivo será associado a um vetor que constituirá a memória do método
- Todas as células do vetor serão inicializadas a um valor fora do conjunto de resultados possíveis do método
- Antes da execução de uma chamada recursiva, verifica-se o valor da célula em causa

Fibonacci com Função-Memória

```
static long fibonacciMem( long[] memory, int n ){  
    //Verifica se o valor foi calculado (já existe em memória)  
    //Se não foi calcula e guarda na memória  
    if (memory[n]==-1) ← Células da Memória inicializadas a -1  
        if ( n == 0 )  
            memory[n] = 0;  
        else if ( n == 1 )  
            memory[n] = 1;  
        else  
            memory[n] = fibonacciMem(memory, n - 1) +  
                        fibonacciMem(memory, n - 2);  
    //Retorna o valor guardado na memória  
    return memory[n];  
}
```



O método devolve sempre uma célula da memória

O resultado das chamadas recursivas é guardado na memória

Métodos auxiliares

```
//Método que inicializa a memória  
//todas as célula inicializadas com um valor fora dos resultados  
//possíveis do método  
static void fibonacciInitMem( long[] vector ){  
    for ( int i = 0; i < vector.length; i++ )  
        vector[i] = -1;  
}
```

```
//Método inicial que chama a inicialização da memória e  
//o método recursivo com memória  
static long fibonacciMem( int n ){  
    long[] memory = new long[n+1];  
    fibonacciInitMem(memory);  
    return fibonacciMem(memory, n);  
}
```

