



1º TRABALHO DE APA

Prof. Raul Fonseca Neto

**Caio Assis de Lima
Thiago Ramalho Vieira**

INTRODUÇÃO

ufjf

Linguagem Utilizada
JAVASCRIPT

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

ITERATIVO

COMPLEXIDADE

$O(N^2)$ QUADRÁTICA

```
JS AvaliacaoPolinomioIterativo.js > avaliarPolinomioIterativo
1 // implementação iterativa
2
3 Complexity is 4 Everything is cool!
4 function avaliarPolinomioIterativo(coeficientes, x) {
5     let resultado = 0;
6     const n = coeficientes.length;
7
8     for (let i = 0; i < n; i++) {
9         let potencia = 1;
10        for (let j = 0; j < i; j++) {
11            potencia *= x;
12        }
13        resultado += coeficientes[i] * potencia;
14    }
15    return resultado;
16 }
17
18 const polinomio = [-1, 2, -6, 2]; // 2x³ - 6x² + 2x - 1
19 const coeficiente = 3;
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativo.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO

```
iterativo.js
iterativoLLM.js
Recursivo.js
RecursivoLLM...
s
LLM.js
o.js
ollM.js

1  const { performance } = require('perf_hooks');
2
3  // implementação iterativa
4
5  > function avaliarPolinomioIterativo(coeficientes, x) {
17 }
18
19 > function gerarPolinomio(n) {
26 }
27
28 const grau = 1000;
29 const polinomio = gerarPolinomio(grau);
30 const coeficiente = 1.0001;
31
32 const inicioExecucao = performance.now();
33 const resultado = avaliarPolinomioIterativo(polinomio, coeficiente);
34 const fimExecucao = performance.now();
35 console.log("O resultado é: " + resultado);
36 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");

TERMINAL  PROBLEMS  OUTPUT  PORTS  DEBUG CONSOLE
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativo.js
O resultado é: -552.5813572673945
Tempo de execução: 4.37 ms
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA

COMPLEXIDADE
 $O(N)$

```
JS AvaliacaoPolinomioRecursivo.js > avaliacaoPolinomioRecursiva
1 // implementação recursiva
2
3 Complexity is 4 Everything is cool!
4 function avaliacaoPolinomioRecursiva(coeficientes,x,i = 0) {
5     const n = coeficientes.length;
6     if (i === n - 1) {
7         return coeficientes[i];
8     }
9     return coeficientes[i] + x * avaliacaoPolinomioRecursiva(coeficientes, x, i + 1);
10 }
11
12 const polinomio = [-1, 2, -6, 2]; //  $2x^3 - 6x^2 + 2x - 1$ 
13 const coeficiente = 3;
14
15 const resultado = avaliacaoPolinomioRecursiva(polinomio,coeficiente);
16 console.log("O resultado é: " + resultado);
17
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivo.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVO

```
1 // implementação recursiva
2
3 Complexity is 4 Everything is cool!
4 function avaliacaoPolinomioRecursiva(coeficientes,x,i = 0) {
5     const n = coeficientes.length;
6     if (i === n - 1) {
7         return coeficientes[i];
8     }
9     return coeficientes[i] + x * avaliacaoPolinomioRecursiva(coeficientes, x, i + 1);
10 }
11
12 Complexity is 4 Everything is cool!
13 > function gerarPolinomio(n) {
14     ...
15 }
16
17
18 const grau = 1000;
19 const polinomio = gerarPolinomio(grau);
20 const coeficiente = 1.0001;
21
22 const inicioExecucao = performance.now();
23 const resultado = avaliacaoPolinomioRecursiva(polinomio,coeficiente);
24 const fimExecucao = performance.now();
25 console.log("O resultado é: " + resultado);
26 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");
27
28
29
```

TERMINAL

PROBLEMS

OUTPUT

PORTS

DEBUG CONSOLE

```
PS C:\VESTRAO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivo.js
O resultado é: -552.581357267394
Tempo de execução: 0.28 ms
PS C:\VESTRAO\APA\PrimeiroTrabalho>
```

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

ITERATIVA (LLM)

```
JS AvaliacaoPolinomioIterativoLLM.js > ...
1 // implementação recursiva
2 // LLM utilizada:
3
4 Complexity is 3 Everything is cool!
5 function avaliarPolinomioIterativoLLM(coeficientes, x) {
6   let resultado = 0;
7   for (let i = coeficientes.length - 1; i >= 0; i--) {
8     resultado = resultado * x + coeficientes[i];
9   }
10  return resultado;
11 }
12
13 const polinomio = [-1, 2, -6, 2]; //  $2x^3 - 6x^2 + 2x - 1$ 
14 const coeficiente = 3;
15
16 const resultado = avaliarPolinomioIterativoLLM(polinomio, coeficiente);
17 console.log("O resultado é: " + resultado);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativoLLM.js
O resultado é: 5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

$O(N)$

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO ITERATIVO (LLM)

```
1  const { performance } = require('perf_hooks');
2
3  // implementação recursiva
4  // LLM utilizada: Claude
5
6  Complexity is 3 Everything is cool!
7  > function avaliarPolinomioIterativoLLM(coeficientes, x) {
8
9  }
10
11 Complexity is 4 Everything is cool!
12 > function gerarPolinomio(n) {
13
14 }
15
16 const grau = 1000;
17 const polinomio = gerarPolinomio(grau);
18 const coeficiente = 1.0001;
19
20 const inicioExecucao = performance.now();
21 const resultado = avaliarPolinomioIterativoLLM(polinomio, coeficiente);
22 const fimExecucao = performance.now();
23 console.log("O resultado é: " + resultado);
24 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\VIESTRADO\UPA\PrimeiroTrabalho> node .\AvaliacaoPolinomioIterativoLLM.js
O resultado é: -552.581357267394
Tempo de execução: 0.01 ms
PS C:\VIESTRADO\UPA\PrimeiroTrabalho>
```


ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO

RECURSIVA(LLM)

```
JS AvaliacaoPolinomioRecursivoLLM.js > avaliarPolinomio
1 // implementação recursiva
2 // LLM utilizada: DeepSeek
3
4 Complexity is 4 Everything is cool!
5 function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
6     // Caso base: quando o grau é 0, retornamos o coeficiente constante
7     if (grau === 0) {
8         return coeficientes[grau];
9     }
10    // Passo recursivo:  $a_n * x^n + a_{n-1} * x^{n-1} + \dots + a_0 = x * (a_n * x^{n-1} + \dots) + a_0$ 
11    return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
12 }
13
14 // Exemplo de uso:
15 const coeficientes = [2, -6, 2, -1]; //  $2x^3 - 6x^2 + 2x - 1$  <- ALTEREI PARA MANTER AS MESMAS ENTRADAS
16 const x = 2;
17 const resultado = avaliarPolinomio(coeficientes, x);
18 console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE powershell + ▢

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivoLLM.js
O polinômio avaliado em x = 2 é -5
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE
 $O(N)$

ALGORITMO DE AVALIAÇÃO DE UM POLINÔMIO RECURSIVA (LLM)

```
1 // Implementação recursiva
2 // LLM utilizada: DeepSeek
3
4 > function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
12 }
13
14 > function gerarPolinomio(n) {
21 }
22
23 // Exemplo de uso:
24 const grau = 1000;
25 const coeficientes = gerarPolinomio(grau);
26 const x = 1.0001;
27
28 const inicioExecucao = performance.now();
29 const resultado = avaliarPolinomio(coeficientes, x);
30 const fimExecucao = performance.now();
31 console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);
32 console.log(`Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

PS C:\MESTRADO\VAPA\PrimeiroTrabalho> node .\AvaliacaoPolinomioRecursivoLLM.js

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA

```
JS FibonacciIterativo.js > ...
1 // implementação iterativa
  Complexity is 5 Everything is cool!
2 function fibonacciIterativo(n) {
3   if (n <= 1) {
4     return n;
5   }
6   let anterior = 0;
7   let atual = 1;
8   for (let i = 2; i <= n; i++) {
9     const proximo = anterior + atual;
10    anterior = atual;
11    atual = proximo;
12  }
13  return atual;
14 }
15
16 const numero = 40; // Numero 40
17
18 const resultado = fibonacciIterativo(numero);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

- PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciIterativo.js
O resultado é: 102334155
- PS C:\MESTRADO\APA\PrimeiroTrabalho>
-

COMPLEXIDADE
 $O(N)$

ALGORITMO DA SEQUÊNCIA DE FIBONACCI ITERATIVO

```
1  const { performance } = require('perf_hooks');
2
3  // implementação iterativa
4  Complexity is 5 Everything is cool
5  function fibonacciIterativo(n) {
6      if (n <= 1) {
7          return n;
8      }
9      let anterior = 0;
10     let atual = 1;
11     for (let i = 2; i <= n; i++) {
12         const proximo = anterior + atual;
13         anterior = atual;
14         atual = proximo;
15     }
16     return atual;
17 }
18
19 const numero = 40; // Numero 40
20
21 const inicioExecucao = performance.now();
22 const resultado = fibonacciIterativo(numero);
23 const fimExecucao = performance.now();
24 console.log("O resultado é: " + resultado);
25 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\VESTIBOL\APAP\PrimeiroTrabalho> node .\FibonacciIterativo.js
O resultado é: 102334155
Tempo de execução: 0.05 ms
PS C:\VESTIBOL\APAP\PrimeiroTrabalho>
```

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

RECURSIVA

```
JS FibonacciRecursivo.js > ...
1  // implementação recursiva
  Complexity is 6 It's time to do something...
2  function fibonacciRecursiva(numero) {
3      if(numero <= 0){
4          return 0;
5      }
6      if(numero == 1){
7          return 1;
8      }
9      return fibonacciRecursiva(numero - 1) + fibonacciRecursiva(numero - 2);
10 }
11
12 const numero = 40; // Numero 40
13
14 const resultado = fibonacciRecursiva(numero);
15 console.log("O resultado é: " + resultado);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE

$O(2^n)$ EXPONENCIAL

ALGORITMO DA SEQUÊNCIA DEFIBONACCI RECURSIVO

```

1  const { performance } = require('perf_hooks');
2
3  // implementação recursiva
4  // Complexity is O(N) time to do something...
5  function fibonacciRecursiva(numero) {
6      if(numero <= 0){
7          return 0;
8      }
9      if(numero == 1){
10         return 1;
11     }
12     return fibonacciRecursiva(numero - 1) + fibonacciRecursiva(numero - 2);
13 }
14
15 const numero = 40; // Numero 40
16
17 const inicioExecucao = performance.now();
18 const resultado = fibonacciRecursiva(numero);
19 const fimExecucao = performance.now();
20
21 console.log("O resultado é: " + resultado);
22 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");

```

TERMINAL

PROBLEMS

OUTPUT

PORTS

DEBUG CONSOLE

PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js

Ln 14, Col 32

Spaces: 4

UTF-8

CSLF

(1) JavaScript

Go Live

Preview

100%

100%

2024/03/27

ALGORITMO DA SEQUÊNCIA DE FIBONACCI

ITERATIVA (LLM)

```
JS FibonacciIterativoLLM.js > ...
1 // implementação iterativa
2 // LLM utilizada:
3
4 Complexity is 5 Everything is cool!
5 function fibonacciIterativoLLM(n) {
6     if (n <= 1) return n;
7     let a = 0, b = 1;
8     for (let i = 2; i <= n; i++) {
9         [a, b] = [b, a + b];
10    }
11    return b;
12 }
13
14 const numero = 40; // Numero 40
15
16 const resultado = fibonacciIterativoLLM(numero);
17 console.log("O resultado é: " + resultado);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
• PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciIterativoLLM.js
O resultado é: 102334155
○ PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

COMPLEXIDADE
 $O(N)$

ALGORITMO DA SEQUÊNCIA DE FIBONACCI ITERATIVO (LLM)

```
4 // LLM utilizada: Claude
5
6 Complexity is 5 Everything is cool!
7 function fibonacciIterativoLLM(n) {
8   if (n <= 1) return n;
9   let a = 0, b = 1;
10  for (let i = 2; i <= n; i++) {
11    [a, b] = [b, a + b];
12  }
13  return b;
14 }
15
16 const numero = 40; // Numero 40
17
18 const inicioExecucao = performance.now();
19 const resultado = fibonacciIterativoLLM(numero);
20 const fimExecucao = performance.now();
21
22 console.log("O resultado é: " + resultado);
23 console.log("Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms");
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciIterativoLLM.js
O resultado é: 102334155
Tempo de execução: 0.00 ms
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```


ALGORITMO DE SEQUÊNCIA DE FIBONACCI

RECURSIVA(LLM)

COMPLEXIDADE

$O(2^n)$ EXPONENCIAL

```
JS FibonacciRecursivoLLM.js > [?] posicao
1 // implementação iterativa
2 // LLM utilizada: DeepSeek
3
4 Complexity is 6 It's time to do something...
5 function fibonacci(n) {
6   // Casos base: fib(0) = 0 e fib(1) = 1
7   if (n === 0) return 0;
8   if (n === 1) return 1;
9
10  // Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
11  return fibonacci(n - 1) + fibonacci(n - 2);
12 }
13
14 // Exemplo de uso:
15 const posicao = 40;
16 console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\MESTRADO\APA\PrimeiroTrabalho> node .\FibonacciRecursivo.js
O resultado é: 102334155
PS C:\MESTRADO\APA\PrimeiroTrabalho>
```

ALGORITMO DA SEQUÊNCIA DE FIBONACCI RECURSIVO (LLM)

```
4 // Implementação iterativa
5 // LLM utilizada: DeepSeek
6 Complexity is 6 It's time to do something...
7 function fibonacci(n) {
8   // Casos base: fib(0) = 0 e fib(1) = 1
9   if (n === 0) return 0;
10  if (n === 1) return 1;
11
12  // Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
13  return fibonacci(n - 1) + fibonacci(n - 2);
14 }
15
16 // Exemplo de uso:
17 const posicao = 40; // Número 40
18
19 const inicioExecucao = performance.now();
20 console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);
21 const fimExecucao = performance.now();
22 console.log(`Tempo de execução: ${(fimExecucao - inicioExecucao).toFixed(2)} ms`);
```

TERMINAL PROBLEMS OUTPUT PORTS DEBUG CONSOLE

```
PS C:\VESTRAO\APR\PrimeiroTrabalho> node .\FibonacciRecursivoLLM.js
O 40º número de Fibonacci é 102334155
Tempo de execução: 1580.64 ms
PS C:\VESTRAO\APR\PrimeiroTrabalho>
```

PROMPTS

Gere um código recursivo para avaliação de polinômios em javascript

📄 ✎ < 2 / 2 >



Código Recursivo para Avaliação de Polinômios em JavaScript

Aqui está uma implementação recursiva do método de Horner para avaliação de polinômios:

javascript

📄 Copy 📄 Download


```
/**
 * Avalia um polinômio de forma recursiva usando o método de Horner
 * @param {number[]} coeficientes - Array de coeficientes do polinômio em ordem decrescente de grau
 * @param {number} x - Valor no qual o polinômio será avaliado
 * @param {number} [grau=coeficientes.length-1] - Grau do polinômio (usado na recursão)
 * @returns {number} Valor do polinômio avaliado em x
 */
function avaliarPolinomio(coeficientes, x, grau = coeficientes.length - 1) {
  // Caso base: quando o grau é 0, retornamos o coeficiente constante
  if (grau === 0) {
    return coeficientes[grau];
  }

  // Passo recursivo: a_n * x^n + a_{n-1} * x^{n-1} + ... + a_0 = x * (a_n * x^{n-1} + ...) + a_0
  return x * avaliarPolinomio(coeficientes, x, grau - 1) + coeficientes[grau];
}

// Exemplo de uso:
const coeficientes = [2, -4, 1, 3]; // Representa 2x³ - 4x² + x + 3
const x = 2;
const resultado = avaliarPolinomio(coeficientes, x);
console.log(`O polinômio avaliado em x = ${x} é ${resultado}`);
```

PROMPTS

Gere um código recursivo para sequencia de fibonacci em javascript



Código Recursivo para Sequência de Fibonacci em JavaScript

Aqui está uma implementação recursiva clássica da sequência de Fibonacci:

javascript

Copy

Download

```
/**
 * Calcula o n-ésimo número da sequência de Fibonacci de forma recursiva
 * @param {number} n - Posição na sequência (começando de 0)
 * @returns {number} Valor do n-ésimo número na sequência de Fibonacci
 */
function fibonacci(n) {
  // Casos base: fib(0) = 0 e fib(1) = 1
  if (n === 0) return 0;
  if (n === 1) return 1;

  // Passo recursivo: fib(n) = fib(n-1) + fib(n-2)
  return fibonacci(n - 1) + fibonacci(n - 2);
}

// Exemplo de uso:
const posicao = 10;
console.log(`O ${posicao}º número de Fibonacci é ${fibonacci(posicao)}`);
```

CONCLUSÃO

AVALIAÇÃO DE POLINÔMIO CÓDIGO HUMANO

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 4.37$ MS

COMPLEXIDADE: $O(N^2) \leftarrow$ QUADRÁTICA

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 0.26$ MS

COMPLEXIDADE: $O(N) \leftarrow$ LINEAR

CONCLUSÃO

AVALIAÇÃO DE POLINÔMIO

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.08$ MS

COMPLEXIDADE: $O(N) \leftarrow$ LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 0.23$ MS

COMPLEXIDADE: $O(N) \leftarrow$ LINEAR

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI

CÓDIGO HUMANO

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.05$ MS

COMPLEXIDADE: $O(N) \leftarrow$ LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 1560$ MS

COMPLEXIDADE: $O(2^n) \leftarrow$ EXPONENCIAL

CONCLUSÃO

SEQUÊNCIA DE FIBONACCI

LLM → DEEPSEEK E CLAUDE

ITERATIVO

TEMPO DE EXECUÇÃO: $\cong 0.09$ MS

COMPLEXIDADE: $O(N) \leftarrow$ LINEAR

RECURSIVO

TEMPO DE EXECUÇÃO: $\cong 1580$ MS

COMPLEXIDADE: $O(2^n) \leftarrow$ EXPONENCIAL