Verifica Algo

Pedro Vinícius da Silva Ribeiro

O algoritmo

```
// Codigo Base
// int VerificaAlgo (n: int);
// i, j, k, l: int;
// para l := 1 TO 10.000 faça
//  para i := 1 TO n-5 faça
//      para j := i+2 TO n/2 faça
//          para k := 1 TO n faça
//              {Inspecione elemento}
```

Em C

```c
int VerificaAlgo(int n) {
    int i, j, k, l, cont = 0;
    for (l = 1; l <= 10000; l++) {
        for (i = 1; i <= n-5; i++) {
            for (j = i+2; j <= n/2; j++) {
                for (k = 1; k <= n; k++) {
                    cont++;
                }
            }
        }
    }
    return 0;
}
```

Explicação

- O algoritmo *“verificaAlgo”* recebe um número inteiro n como entrada e executa quatro loops aninhados. O loop externo é executado por 10000 iterações, e os três loops internos têm intervalos variáveis dependendo do valor de n.
- A variável *“cont”* é incrementada toda vez que o loop mais interno é executado, a variável foi adicionada ao código só para ter uma operação O(1) que não afetasse o algoritmo.

Explicação

No caso da função VerificaAlgo, podemos analisar sua complexidade da seguinte forma:

- O primeiro loop for executa 10.000 vezes, o que é uma complexidade constante O(1);
- O segundo loop for executa n-5 vezes, o que é uma complexidade O(n);
- O terceiro loop for executa (n/2) - (i+2) + 1 = (n/2) - i - 1 vezes, o que é uma complexidade O(n);
- O quarto loop for executa n vezes, o que é uma complexidade O(n);

Função de custo e Complexidade

- Função de custo:

$$\sum_{l=1}^{10000}\sum_{i=1}^{n-5}\sum_{j=i+2}^{\frac{n}{2}}\sum_{k=1}^{n}1 \qquad \sum_{k=1}^{n}1=n-1+1=n \qquad \sum_{j=i+2}^{\frac{n}{2}}n=\sum_{j=2}^{\frac{n}{2}-i}n+i=\sum_{j=2}^{\frac{n}{2}-i}n+\sum_{j=2}^{\frac{n}{2}-i}i=n\left(\frac{n}{2}-i-2+1\right)+i\left(\frac{n}{2}-i-2+1\right)=(n+i)\left(\frac{n}{2}-i-1\right)=\frac{1}{2}(n+i)(n-2i-2)$$

$$\sum_{i=1}^{n-5}\frac{1}{2}(n+i)(n-2i-2)=\frac{1}{2}\sum_{i=1}^{n-5}n^2-2ni-2n+ni-2i^2-2i=\frac{1}{2}\left[\sum_{i=1}^{n-5}n^2-2n+\sum_{i=1}^{n-5}-ni-2i+\sum_{i=1}^{n-5}-2i^2\right]$$

$$=\frac{1}{2}\left[(n-5)(n^2-2n)-\frac{(n+2)(n-6)(n-5)}{2}-\frac{2(n-5)(n-4)(2n-9)}{6}\right]=-\frac{n^3-39n^2+206n-180}{12}$$

$$\sum_{l=1}^{10\,000}-\frac{n^2-39n^2+206n-180}{12}=10\,000\left(-\frac{n^2-39n^2+206n-180}{12}\right)$$

Função de custo e Complexidade

- Função de custo:

$$=-\frac{2500n^3-97500n^2+515000n-270000}{3}$$

- Complexidade:

$$O(n^3)$$

Testes

- Durante o experimento, foram realizadas 10 rodadas de testes, cada uma consistindo de 13 ciclos de processamento. O algoritmo utilizado tinha uma complexidade $O(N^3)$, o que limitou o maior valor de entrada a ser analisado em 1000. O tempo médio de execução para o maior valor de entrada foi de 1676.30 segundos, o que equivale a aproximadamente 27 minutos.

Medias de tempo

- Tempo médio de execução para n=100: 1.78 segundos
- Tempo médio de execução para n=200: 13.73 segundos
- Tempo médio de execução para n=300: 46.94 segundos
- Tempo médio de execução para n=400: 107.87 segundos
- Tempo médio de execução para n=500: 209.72 segundos
- Tempo médio de execução para n=600: 361.39 segundos
- Tempo médio de execução para n=700: 574.07 segundos
- Tempo médio de execução para n=800: 856.47 segundos
- Tempo médio de execução para n=900: 1219.40 segundos
- Tempo médio de execução para n=1000: 1676.30 segundos

Gráfico de relação Tempo e N

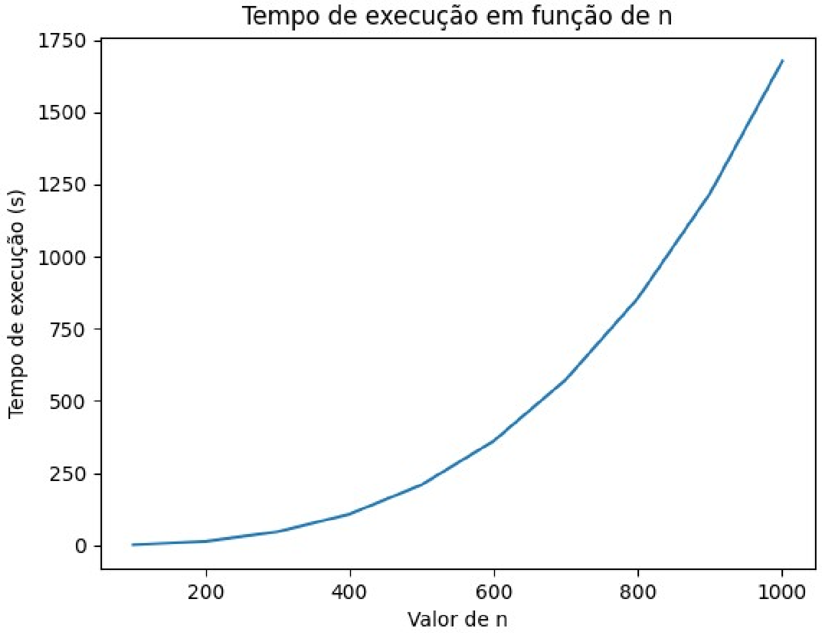
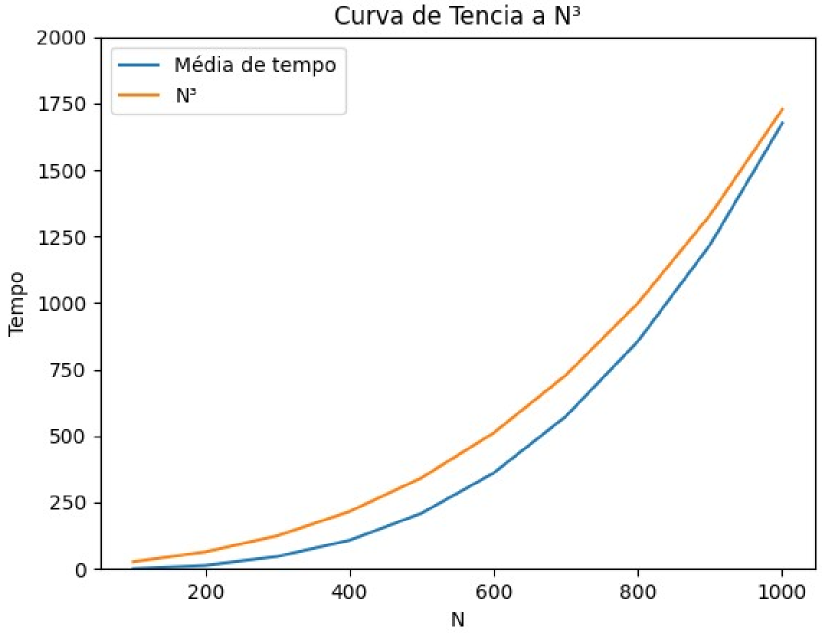


Gráfico de tendencia N³



Algoritmo melhor

- Uma forma melhorar o codigo é encontrar uma solução que não dependa do loop aninhado. Uma abordagem seria usar uma fórmula matemática que calcula diretamente o número de iterações. Nesse caso, podemos observar que o loop em k não afeta o número total de iterações.
- Essa solução tem complexidade de tempo O(n^2), o que é ainda melhor do que a solução anterior. Além disso, não há loops aninhados, o que torna a função mais clara e fácil de entender.

Algoritmo melhor

```c
int VerificaAlgo(int n) {
    int num_iteracoes = 0;
    for (int l = 1; l <= 10000; l++) {
        for (int i = 1; i <= n-5; i++) {
            for (int j = i+2; j <= n/2; j++) {
                num_iteracoes++;
            }
        }
    }
    return num_iteracoes * n;
}
```

Obrigado pela atenção!