**Análise de Complexidade em Algoritmos**

Os algoritmos fazem parte do dia a dia das pessoas. Seja uma receita de bolo, de remedios, em terefas diarias,. Ou seja, é uma sequência de ações executáveis para chegar à solução de um determinado tipo de problema. Segundo *Edsger Dijkstra*, um algoritmo corresponde a uma descrição de um padrão de comportamento, expresso em termos de um conjunto finito de ações. Um algoritmo é um procedimento de passos para realizar uma determinada tarefa, este procedimento é composto de instruções que definem uma função, desta forma, a análise de algoritmos (descrita e difundida por D.E. Knuth) tem como função determinar os recursos necessários para executar um dado algoritmo. Ela estuda a correção e o desempenho através da análise de correção e da análise de complexidade. Dados dois algoritmos para um mesmo problema, a análise permite decidir qual dos dois é mais eficiente.

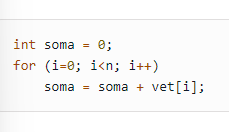
A **análise da correção** procura entender porque um dado algoritmo funciona. É preciso ter certeza de que o algoritmo está correto. Como algoritmos prometem resolver problemas, é esperado que ele cumpra essa premissa.

A **Análise da complexidade** procura estimar a velocidade do algoritmo. Prever o tempo que o algoritmo vai consumir. Verificar qual é o custo de usar um dado algoritmo para resolver um problema específico. Porém, neste aspecto, encontram-se duas dificuldades:

1. O tempo gasto depende dos dados de entrada.
2. O tempo também depende da capacidade de hardware do computador utilizado.

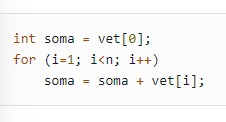
Mas, mesmo assim, é possível definir a complexidade de determinado algoritmo.

Para medir o custo de execução de um algoritmo é comum definir uma função de custo ou função de complexidade, f; f(n) é a medida do tempo necessário para executar um algoritmo para um problema de tamanho n;

Para entendermos melhor, vejamos o trecho de código abaixo:

O custo para execução deste algoritmo é f(n) = n. Pois o laço executa de acordo com o tamanho de n. Se n=10, logo o laço executará 10 vezes, se for igual a 100 executará 100 vezes.

Podemos tornar este algoritmo mais eficiente fazendo a seguinte modificação:



Assim, temos uma nova função de complexidade, f(n) = n - 1.

Os dois trechos de códigos resolvem o mesmo problema, somar os elementos de um vetor. Porém, cada um tem um custo de execução diferente do outro, a partir da análise, podemos definir qual é o algoritmo mais eficiente para resolver este problema específico.

## **Melhor caso, Pior caso, Caso Médio.**

Na análise da complexidade, definimos para um algoritmo, o **melhor**, o **pior** e o caso **médio**, como forma de mensurar o custo do algoritmo de resolver determinado problema diante de diferentes entradas.

* **Melhor caso:** é quando o algoritmo representa o menor tempo de execução sobre todas as entradas de tamanho n;
* **Pior caso:** maior tempo de execução sobre todas as entradas de tamanho n;
* **Caso médio (ou caso esperado):** é a média dos tempos de execução de todas as entradas de tamanho n.

Desta forma, considere o problema de acessar um determinado registo de um vetor de inteiro; cada registro contém um índice único que é utilizado para acessar o elemento do vetor.

**O problema:** dada uma chave qualquer, localize o registro que contenha esta chave; O algoritmo de pesquisa mais simples é o que faz a pesquisa **sequencial**.

Seja f uma função de complexidade tal que f(n) é o número de registros consultados no arquivo, temos:

* **Melhor caso:** f(n) = 1 (registro procurado é o primeiro consultado);
* **Pior caso:** f(n) = n (registro procurado é o último consultado ou não está presente no arquivo);
* **Caso médio:** f(n) = (n+1)/2 (a média aritmética entre o melhor e o pior caso);

Além disso, a análise de algoritmos estuda certos paradigmas como divisão e conquista, programação dinâmica, gula, busca local, aproximação, entre outros que se mostraram úteis na criação de algoritmos para vários problemas computacionais.

## **A Importância da Análise Matemática**

A análise matemática desempenha um papel crucial na análise de complexidade de algoritmos, uma vez que fornece um meio preciso e objetivo de descrever o desempenho de um algoritmo. Ela nos permite expressar a complexidade em termos de funções matemáticas que relacionam o tamanho da entrada com o uso de recursos. Isso facilita a comparação e a previsão do desempenho de algoritmos sob diferentes condições de entrada.

A Matemática e Estatística fornecem as bases para analisar e interpretar dados de maneira sistemática. Em um mundo onde a informação é abundante, essas disciplinas permitem que os profissionais tomem decisões baseadas em evidências sólidas. Em qualquer área, seja na tecnologia ou no mercado financeiro ou na medicina, a análise matemática e estatística permite que as decisões sejam embasadas em dados concretos.

## **Contando Instruções de um Algoritmo**

A contagem de instruções é uma técnica comum na análise de complexidade, na qual se determina o número de operações básicas (como adições, multiplicações, comparações, etc.) que um algoritmo realiza em função do tamanho da entrada. Essa contagem ajuda a quantificar o custo computacional de um algoritmo e é frequentemente usada para determinar sua complexidade temporal.

## **Comportamento Assintótico**

O comportamento assintótico se refere ao desempenho de um algoritmo à medida que o tamanho da entrada se torna infinitamente grande. Ele ignora os fatores constantes e foca nas tendências de crescimento dominantes. Três notações comuns para descrever o comportamento assintótico são:

* **Ômega (Ω)**: Descreve um limite inferior para o desempenho do algoritmo. É usado para indicar que o algoritmo não é pior do que uma determinada função em termos de complexidade.
* **O (Big O)**: Descreve um limite superior para o desempenho do algoritmo. É usado para indicar que o algoritmo não é melhor do que uma determinada função em termos de complexidade.
* **Theta (Θ)**: Descreve limites inferior e superior iguais, o que significa que a complexidade do algoritmo está estritamente ligada a uma função específica.

## **Tipos de Análise Assintótica (Ω, O, Θ)**

* A notação **Ω** é usada para descrever o melhor caso de complexidade de um algoritmo. Ela fornece um limite inferior para o desempenho e indica o cenário mais otimista.
* A notação **O** é usada para descrever o pior caso de complexidade de um algoritmo. Ela fornece um limite superior para o desempenho e indica o cenário mais pessimista.
* A notação **Θ** é usada quando a complexidade de um algoritmo é igual em seu melhor e pior caso.

**Classes de Problemas na Teoria da Complexidade:**

A teoria da complexidade se concentra em classificar problemas computacionais com base em sua dificuldade em encontrar soluções eficientes. Ela aborda classes como P, NP, NP-Completo e NP-Difícil para entender essa complexidade e seu impacto na pesquisa em algoritmos e computação.

**Problemas de Otimização:** Envolvem encontrar a melhor solução, como maximizar ou minimizar algo.

**Problemas de Busca:** Requerem encontrar objetos com certas propriedades, geralmente mais simples do que problemas de otimização. Podem ser relacionados aos problemas de otimização correspondentes.

**Problemas de Decisão:** São ainda mais simples, com apenas duas soluções possíveis: sim ou não. Podem ser vistos como versões mais restritas de problemas de busca.

**Classes de Problemas:**

**Classe P (Problemas Polinomiais):** Problemas com algoritmos polinomiais de resolução. Considerados tratáveis e fáceis. Exemplos incluem equações de segundo grau e problemas de caminho mais curto.

**Classe NP (Problemas Polinomialmente Verificáveis):** Problemas cujas soluções podem ser verificadas rapidamente por algoritmos polinomiais. Difícil determinar se eles são polinomiais. Exemplos incluem problemas de conjunto independente grande.

**Classe NP-Completo:** Problemas que são NP-difíceis e também estão em NP. Exemplos incluem problemas como Subset-Sum, Circuito Hamiltoniano e Mochila. A questão P = NP gira em torno dessa classe.

**Classe NP-Difícil:** Problemas pelo menos tão difíceis quanto os problemas em NP, mas não necessariamente em NP.

**A Questão P vs. NP:** Pergunta se todos os problemas cujas soluções podem ser verificadas rapidamente também podem ser resolvidos rapidamente. A maioria dos especialistas acredita que P ≠ NP, mas essa questão ainda não foi resolvida.

**Complexidade Relativa de Problemas:** Reduções polinomiais permitem comparar a dificuldade relativa entre problemas. Se Y é polinomialmente redutível a X e X está em P, então Y também está em P.

**Apêndice: Classe co-NP** (Problemas cujos complementos são polinomialmente verificáveis): Complemento de um problema de decisão. Não se sabe se P = co-NP ou NP = co-NP.

Em resumo, a teoria da complexidade classifica problemas em várias classes com base em sua dificuldade computacional, e a questão P vs. NP continua sendo uma questão fundamental e não resolvida nesse campo.

## **Conclusão**

A análise de complexidade em algoritmos desempenha um papel fundamental na ciência da computação. Ela nos permite compreender e quantificar o desempenho de algoritmos, tomar decisões informadas sobre a escolha de algoritmos para problemas específicos e classificar problemas com base em sua dificuldade computacional. A análise matemática, juntamente com as notações assintóticas (Ω, O, Θ), fornece as ferramentas necessárias para realizar essa análise de forma rigorosa e precisa. Portanto, é um campo de estudo essencial para qualquer profissional da área de computação.