Análise da Complexidade de Algoritmos e P vs NP vs NP-Completo

Rafael Amauri Diniz Augusto

Novembro 2019

1 Análise de Complexidade

A teoria da complexidade é uma área de estudo da computação voltado à análise matemática da eficiência de um algoritmo. Como o poder de processamento dos computadores está crescendo cada vez mais, não vale a pena medir esse tempo em função do tempo propriamente dito, mas em função do tamanho da entrada, caso contrário seria uma comparação injusta entre computadores atuais e computadores mais antigos. Essa análise é feita "convertendo" um algoritmo para uma função polinomial, com a variável sendo uma parte do código que se repete.

Instruções repetitivas, como um for, caem nessa categoria, enquanto instruções como uma atribuição ou uma definição de variável não são contadas, por terem custo constante.

Essa separação e desconsideração de alguns custos gera imprecisão na hora de contar o tempo, msa essa imprecisão é muito pequena, e, por isso, não vale a pena tentar calcular ela. Se em uma análise de complexidade nos depararmos com uma função de custo que tenha 2 ou mais polinômios [exemplo: O(n^3 x n^2)], é considerado apenas o polinômio de maior grau, pois ele é responsável por ter a maior influência no gasto do algoritmo. Portanto, pode-se dizer que O(n^3 x n^2) = O(n^3).

2 O problema P vs NP

Com base nessa definição, em 1971 foi criada a definição do problema P vs NP. Os algoritmos P são definidos como algoritmos que podem ser executados em tempo polinomial em uma máquina determinística de Turing. Esses são os algoritmos que são simples o suficiente para resolver em um tempo considerado rápido. Um exemplo de de algoritmo em P são os algoritmos de ordenação, já que eles não são executados com base em tentativa-e-erro e apresentam um tempo de execução que seja um polinômio.

Ao entrar nessa discussão, também teremos que definir o que são algoritmos NP. Se um algoritmo P é um algoritmo que pode achar uma solucão em um tempo considerado rápido, os algoritmos NP são aqueles que são lentos para achar uma solução, mas são rápidos o bastante para verificar se uma solução é válida. Um exemplo de algoritmo NP é um algoritmo que verifica se uma chave criptografada é válida. É demorado para achar a chave que quebra uma senha, mas é rápido para se verificar se a chave está certa.

Além desses dois, temos os algoritmos NP-completo, que são considerados os mais difíceis do conjunto NP; Para ser considerado NP-completo, ao ser reduzido, o algoritmo-solução deve servir para resolver todos outros problemas NP. Os problemas NP-Completo são tão complexos que, ao se resolver um deles, alguns pedaços podem ser usados para resolver outros problemas NP-Completo e NP (isso é reduzir o problema). Com isso, ao se resolver um problema NP-Completo, pode-se provar que P = NP; Como os problemas NP-Completo, ao serem reduzidos, conseguem ser usados para resolver todos problemas NP, ao se resolver um problema NP-Completo de maneira rápida, resolveremos todos problemas NP de maneira rápida, integrando-os ao conjunto P, e assim resolvendo um problema que existe na computação há quase metade de um século. Mas, caso se consiga provar que algum algoritmo NP-Completo é fundamentalmente impossível de se resolver de maneira rápida, ao aplicar a mesma lógica de reduzir um problema NP-Completo a NP, teremos que todos problemas NP são fundamentalmente difíceis e impossíveis de resolver de maneira rápida, o que também resolveria a questão, ao estabelecer que P != NP.