## **Problemas P, NP e NP-completo:**

O [**problema P**](https://pt.wikipedia.org/wiki/P_(complexidade)) é um problema de decisão (ou seja, a resposta pode ser sim ou não), que possa ser **resolvido** em tempo polinomial.

O [**problema NP**](https://pt.wikipedia.org/wiki/NP_(complexidade)) (Non-Deterministic Polynomial time), ou seja "Tempo polinomial não-determinístico". O Problema NP pode ser um Problema P (na verdade [o assunto é mais complexo](http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext)), com a característica de "não determinístico", ou seja, ele pode ser **provado** em tempo polinomial.

Um exemplo de NP é a Fatorização:

Dados n e m, há um inteiro f na faixa 1 < f < m que seja um fator de n (divida com resultado inteiro)? Este é um problema de decisão, pois a resposta pode ser apenas sim ou não. Porém, estamos tratando de um caso específico (o f fornecido). Para verificar, temos que testar casos concretos, verificando n / f para cada f.

O [**problema NP-completo**](https://pt.wikipedia.org/wiki/NP-completo), de acordo com a Wikipedia, "é um subconjunto de NP, o conjunto de todos os problemas de decisão os quais suas soluções podem ser verificadas em tempo polinomial; NP pode ser equivalentemente definida como o conjunto de problemas de decisão que podem ser solucionados em tempo polinomial em uma Máquina de Turing não determinística. Um problema p em NP também está em NPC Se e somente se todos os outros problemas em NP podem ser transformados em p em tempo polinomial (...)

Um problema de decisão *C* é NP-completo se:

1. *C* está em NP, e
2. Todo problema em NP-Completo é redutível para *C* em tempo polinomial.

*C* pode ser mostrado que pertence à NP demostrando que uma solução candidata para *C* pode ser verificada em tempo polinomial. Note que um problema que satisfaz a condição 2 é dito ser [NP-difícil](https://pt.wikipedia.org/wiki/NP-dif%C3%ADcil), se satisfizer a condição 1 ou não. Uma consequência dessa definição é que se tivéssemos um algoritmo de tempo polinomial para *C*, podíamos resolver todos os problemas NP em tempo polinomial."

## 

## **A "importância" dos problemas NP-completo:**

Ainda de acordo com a [Wikipedia](https://pt.wikipedia.org/wiki/NP-completo), "Problemas NP-completo são estudados porque a habilidade de rapidamente verificar soluções para um problema (NP) parece correlacionar-se com a capacidade de resolver rapidamente esse problema (P). Não é sabido se todos os problemas em NP podem ser rapidamente resolvidos - isso é chamado de problema P versus NP. Mas se qualquer problema em NP-completo pode ser resolvido rapidamente, então todo problema em NP também pode ser, por causa da definição de NP-completo afirma que todo problema em NP deve ser rapidamente redutível para todo problema em NP-completo (ou seja, pode ser reduzido em tempo polinomial). Por causa disso, é geralmente falado que os problemas NP-completo são mais difíceis que os problemas NP em geral."

Um exemplo de problema NP completo é a [Torre de Hanoi](https://pt.wikipedia.org/wiki/Torre_de_Han%C3%B3i), que é um *puzzle* em que você tem discos de tamanhos diferentes, em ordem crescente de diâmetro, com os menores em cima, sobre um pino, e tem mais dois pinos livres. O objetivo é passar todos os discos para outro pino, mas passando apenas um por vez, e de forma a nunca ficar um maior sobre o menor.



Na prática, a [grande discussão](http://cacm.acm.org/magazines/2009/9/38904-the-status-of-the-p-versus-np-problem/fulltext) é que os problemas NP-completo são a chave para se determinar se P = NP ou P ≠ NP. Se em algum momento um problema NP não puder ser resolvido em tempo P, nenhum problema NP-completo pode ser resolvido em tempo P. Por outro lado, se algum problema NP-completo puder se resolvido em tempo P, P = NP

## 

## **Algoritmos para solução de problemas NP-completo:**

* [Aproximação](https://pt.wikipedia.org/wiki/Aproxima%C3%A7%C3%A3o): Um algoritmo que rapidamente encontra uma solução não necessariamente ótima, contudo dentro de um certo intervalo de erro. Em alguns casos, encontrar uma boa aproximação é o suficiente para resolver o problema, porém nem todos os problemas NP-completos tem bons algoritmos de aproximação.
* [Probabilístico](https://pt.wikipedia.org/wiki/Probabilidade): Um algoritmo que pode obter em média uma boa solução para um problema apresentado de uma distribuição de dados de entrada.
* Restrição: Restringindo a estrutura da entrada, algoritmos mais rápidos são possíveis.
* Parametrização: Geralmente há algoritmos rápidos se certos parâmetros da entrada são fixos.
* [Heurísticas](https://pt.wikipedia.org/wiki/Heur%C3%ADstica): Um algoritmo que trabalha razoavelmente bem em muitos casos, mas não há prova de que são sempre rápidos e que produzam sempre bons resultados.

| 8votar contraaceitos | Um problema NP-Completo é um problema pertencente à [classe de problemas NP](https://pt.wikipedia.org/wiki/NP_%28complexidade%29) que pode ser *reduzido* em tempo polinomial ao [**problema da satisfatibilidade booleana**](https://pt.wikipedia.org/wiki/Problema_de_satisfatibilidade_booleana) (SAT):  Dada uma expressão booleana expressa como uma conjunção de disjunções entre n variáveis, negadas ou não, como por exemplo:  (x1 ou x2 ou não x3 ou x4) e (x1 ou não x2 ou x3 ou não x4) e ...  Encontrar um conjunto de valores para x1, x2, x3, ... xn tal que essa expressão seja verdadeira. **P vs NP** Um problema é dito "polinomial", ou [pertencente à classe P](https://pt.wikipedia.org/wiki/P_%28complexidade%29), se existe um algoritmo conhecido capaz de solucionar o mesmo cuja [ordem de complexidade](https://pt.stackoverflow.com/q/33319/215) [no pior caso] seja polinomial em relação ao "tamanho" da entrada. Caso não se conheça tal algoritmo, o problema não pertence a essa classe (para definições mais precisas, ver [a resposta do Bacco](https://pt.stackoverflow.com/a/34106/215)).  Alguns problemas não polinomiais, entretanto, possuem uma outra característica: dada uma solução candidata (i.e. um conjunto de valores que pode ou não ser solução para esse problema) pode-se *verificar* se ela é ou não uma solução em tempo polinomial. Esses problemas são ditos "polinomiais não determinísticos", ou [pertencentes à classe NP](https://pt.wikipedia.org/wiki/NP_%28complexidade%29), pois eles podem ser resolvidos em um esquema de "geração e teste" (escolha dentre o espaço de soluções uma candidata, e teste pra ver se é de fato uma solução).  Naturalmente, o espaço de soluções pode ser maior do que "polinomial em relação ao tamanho da entrada"; escolher uma candidata nesse espaço não é algo que possa ser feito, deterministicamente, em tempo polinomial. Mas é preciso que exista um meio de fazê-lo "não deterministicamente", ou seja, dadas duas escolhas A e B, "adivinhe" qual é a escolha certa e siga em frente. Se por outro lado o espaço de soluções é tão grande que mesmo adivinhando certo sempre ainda não seja possível chegar à resposta em tempo polinomial, então esse problema é mais difícil que NP. **Redução** O que significa um problema ser "reduzido" a outro? Simplesmente que existe um processo capaz de de transformar um problema em outro, solucionar o outro, e transformar a solução encontrada numa solução para o problema original. Exemplo:  Qual é o mínimo múltiplo comum entre 6 e 15?  Em vez de resolver esse problema diretamente, pode-se transformar esse problema em um problema de máximo divisor comum:  mmc(x, y) = abs(x \* y)/mdc(x, y)  Resolver o problema mdc(6, 15) (muito mais "fácil" - dado o [algoritmo de Euclides](http://pt.wikipedia.org/wiki/Algoritmo_de_Euclides) ou, se a questão for eficiência, o [MDC Binário](http://en.wikipedia.org/wiki/Binary_GCD_algorithm)) e, de posse da solução do mesmo (3), obter a solução final:  mmc(6, 15) = abs(6 \* 15)/mdc(6, 15) = 90/3 = 30  Às vezes a importância da redução é apenas teórica (mais sobre isso adiante), em outras pode-se ter benefícios concretos em fazê-lo - reaproveitar uma implementação já pronta, por exemplo. Desde é claro que o *processo* de se converter o problema e a solução não seja mais custoso que simplesmente solucionar o problema original. **SAT** O problema da satisfatibilidade, mencionado anteriormente, é de especial importância por duas razões: 1) foi o primeiro exemplo conhecido da classe NP-Completo (i.e. ele "inaugurou" a classe); e 2) ele é [auto-redutível](https://en.wikipedia.org/wiki/Boolean_satisfiability_problem#Self-reducibility), ou seja, qualquer algoritmo capaz de dizer se tal instância do SAT possui ou não uma solução também é capaz de dizer *qual é* essa solução (i.e. não precisa ser somente "sim" ou "não", para esse problema em particular pelo menos).  Esclarecendo, quando digo que ele "inaugurou a classe" isso significa que foi provado que qualquer problema pertencente à classe NP pode ser reduzido a SAT. Os problemas em P também podem ser reduzidos a SAT, claro, mas como já se conhece uma boa solução para os mesmos, esses não são considerados parte de NP-Completo (i.e. se P ≠ NP, então temos que P ⊆ NP e NP-Completo ⊆ NP mas P ∩ NP-Completo = Ø).  Em geral, pode-se dizer que um problema é NP-Completo se ele se reduz a qualquer outro problema NP-Completo, não necessariamente o SAT (o problema do [**clique**](http://pt.wikipedia.org/wiki/Clique) é muito popular nesse sentido), mas como este é de importância teórica achei interessante dar esse destaque. E caso não tenha ficado claro, acredita-se que o SAT não pertence a P (não se conhece até o presente momento uma solução determinística para SAT em tempo polinomial). **Importância Prática** Ok, mas qual a importância prática de se saber que tal problema é NP-Completo ou não? Tudo é uma questão de se decidir, rapidamente, se um dado problema pode ser satisfatoriamente resolvido dadas as técnicas atuais de resolução ou não (dependendo de sua escala, é claro). Em vez de quebrar a cabeça durante horas tentando achar uma solução para um problema, perceber que ele é equivalente (ou redutível) a um outro problema pode te poupar bastante tempo:   1. Esse problema é redutível a um outro em P, então dá pra resolver! Só falta saber a melhor maneira; 2. Esse problema é redutível a um outro NP-Completo, então não dá pra *garantidamente* achar a melhor solução, teremos que nos contentar com uma solução aproximada; 3. Esse problema é redutível a um outro que **não é** [comprovadamente] NP, então é melhor desistir de vez e tentar achar um outro problema mais simples, factível e que seja "bom o bastante" pra mim. |
| --- | --- |