Problemas P y NP

# ¿Qué es un problema P y uno NP?

La teoría de la completitud NP descansa en la fundación de rigurosas pero sutiles definiciones de la teoría de autómatas y lenguajes. Estos términos se confunden entre las personas que comienzan por el camino de las Ciencias en Computación. Pero, la pregunta real es: ¿Es P=PN?

Pensemos que la clase P, es un club exclusivo de los problemas algorítmicos de forma que: dada un problema si se demuestra que existe un **tiempo polinomial en el que se resuelvan** entonces pertenece a la clase P (P significa *polynomial-time*). Un club menos exclusivo es para los problemas que pueden ser **verificados** en tiempo polinomial, siendo que todos los problemas en P también pertenecen a esta clase, si puedes resolver el problema en tiempo polinomial **es seguro que puedes verificarlo en esa misma velocidad**, este club se llama clase NP (significa *nondeterministic polynomial-time*, algo como *not-necessarily polynomial-time*).

# Problemas NP-Completos

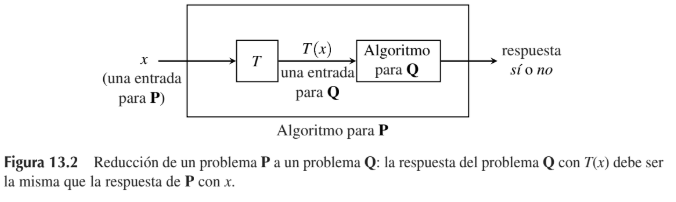
Como en muchas cosas, las clasificaciones tienen clasificaciones, tal es el caso de la clase NP, que se divide en NP-difícil (o NP-duro) y NP-completo, existe una sutil (y generalmente irrelevante) diferencia entre ellos: decimos que un problema es NP-difícil si es al menos tan difícil como cualquier problema en NP, mientras que, decimos que un problema es NP-completo si es NP-difícil y también es NP en sí mismo.

# El tiempo polinomial: verificación y descubrimiento

Se trata de, literalmente, una función con respecto al número de datos a manejar que retorna una cantidad de tiempo, en términos polinomiales, por ejemplo, t = f(n) = n+1. Pero, hay que tener en cuenta dos conceptos totalmente diferentes: el descubrimiento y la verificación. Si una persona descubre la solución dado un problema, entonces, **es seguro que lo puedes verificar en ese tiempo o menor**. De aquí la diferencia, obtener la solución es más largo.

# NP-completos: reducción

La definición formal del NP-completo emplea reducciones o transformaciones de un problema a otro. Si queremos resolver el problema P, pero ya conocemos la solución para cierto problema Q. Entonces, sea T una función que tome una entrada x para P y produzca una T(x), que sea una entrada de Q tal que la respuesta correcta para P con x es Sí, **si y sólo si** la respuesta en Q con T(x) también es Sí.



# Demostración de NP-completos

En realidad, más allá de demostrar que un problema es NP-completo y no sólo NP-duro, en la práctica lo que se requiere es resolverlo, es decir, es evidente que el problema lo queríamos resolver desde un inicio. Pero ¿por qué se demuestran?, bueno, porque con eso, al menos sabemos que tan difícil es: tan difícil como otro problema NP (al menos). Pero, podemos confiar en tres conceptos claves en la demostración de que un problema es NP-completo:

## Problemas de decisión y problemas de optimización

Muchos problemas de interés son los de optimización, justamente, por ejemplo, el encontrar un camino que sea más corto que todos. Aquí entra una cosa importante: los problemas NP-completos **no aplican directamente a problemas de optimización**, de hecho, es a problemas de decisión, donde la respuesta es 1 o 0, o, sí o no.

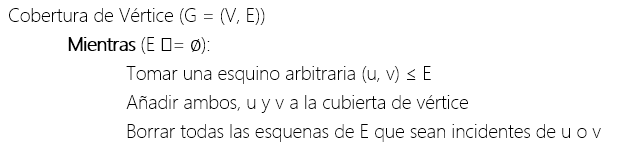
## Reducción del problema NP

Como se habló arriba, si existe una transformación T, que pase el problema P a Q, entonces P es NP-completo, de hecho, esta idea es la que generalmente se usa para demostrar que cierto problema P es NP-completo. (P, en el sentido de Problema, no de *polynomial-time*).

# Ejemplos

## Aproximación de la cubierta de vértices (NP-completo)

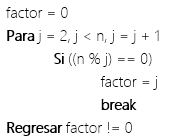
Encontrar la cubierta de vértices dado un grafo, es un problema NP-completo,



!

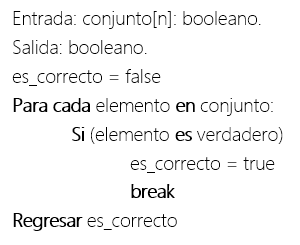
## Determinar si un número entero no es primo

Dado un entero positivo n, ¿existen enteros j, k > 1 tales que n = j \* k? (O sea, ¿es n un no primo?):



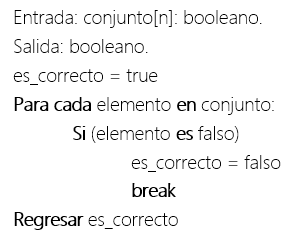
## Operación OR de booleanos (NP-completo)

Dado un conjunto de booleanos, ¿al menos uno de ellos tiene el valor de verdadero?, como existe la transformación de true -> 1, false -> 0, entonces, esto, es un problema NP-completo:



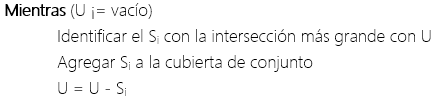
## Operación AND de booleanos (NP-completo)

Dado que también existe una transformación de la misma forma que el OR en el problema anterior, este problema también es un problema NP-completo, el problema sería, dado un conjunto de valores booleanos, ¿no existe algún elemento que sea falso?



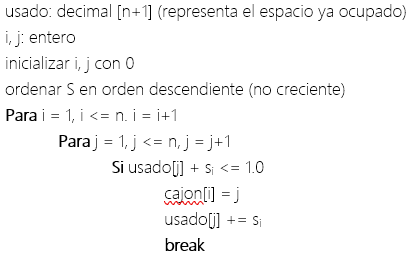
## Cubierta de conjunto (NP-completo)

Dada una colección de subconjuntos S del conjunto Universal U, ¿Cuál es la salida más pequeña T, tal que T es subconjunto de S y su unión es igual al conjunto U?



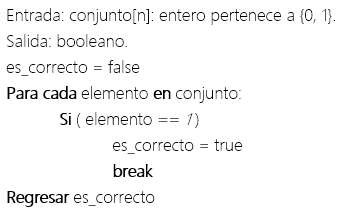
## Llenado de cajones: primer ajuste decreciente

Sea S una sucesión de números decimales, donde 0 < si <= 1, para todo si que pertenece a S, donde S representa el tamaño de los objetos {1, …, n} que se colocan en cajones con capacidad de 1.0 cada uno. Encontrar un arreglo donde para 1 <= i <= n, donde arreglo en i es el número del cajón en el que se colocó el objeto i.



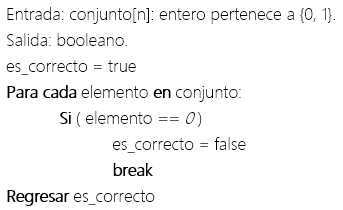
## Función OR en binario

Como se vio arriba, la función del OR en booleano tiene una transformación a Q, donde Q tiene un lenguaje {0, 1}:



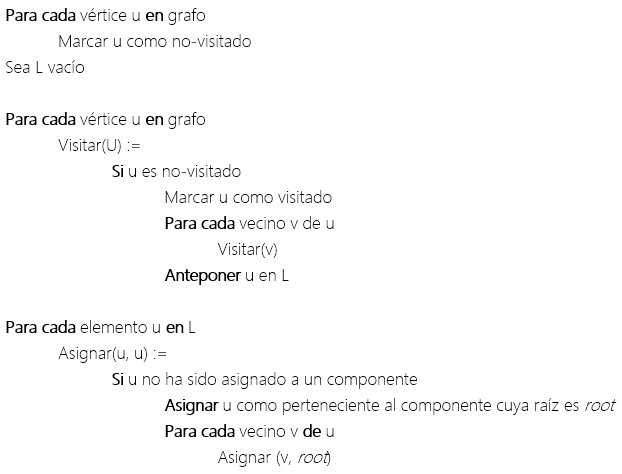
## Función AND en binario

En el mismo sentido, el AND binario también es el problema al que se puede transforma el AND booleano, por tanto, ambos, son problemas NP-completos:



## Encontrar componentes fuertes en un grafo

Dado un grafo dirigido, encontrar los componentes fuertemente conectados. Un componente es fuertemente conectado si todos los vértices son alcanzables desde cualquier otro vértice siendo que un componente es un subgrafo.



## 2-SAT: satisfacibilidad de dos variables

Dada una entrada en forma de fórmula booleana, digamos (p V q) V (p), encontrar las tuplas de valores p, q, tales que la entrada al ser evaluada sea verdadera. Este problema se resuelve con el algoritmo de arriba, de hecho, la fórmula se convierte en un grafo

# Referencias

Baase, S., Van Gelder, A., Escalona García, R. and Torres, S., 2002. Algoritmos Computacionales. 3rd ed. México: Pearson Educación.

Cormen, T., Leiserson, C., Rivest, R. and Stein, C., 2014. *Introduction To Algorithms*. 3rd ed. Cambridge, Massachusetts: MIT Press.

Skiena, S., 2009. The Algorithm Design Manual. 2nd ed. Dordrecht: Springer.