Complejidad Computacional

Relación entre las clases P y NP

Autor: Fausto David Hernández Jasso

2 de octubre de 2022

1. Relación

P $\subseteq$ **NP**, es decir, todo problema resuelto por un algoritmo determinista en **tiempo polinomial**, también puede ser resulto por un algoritmo no-determinista en **tiempo polinomial**, ésto se cumple ya que el **determinismo** es un caso particular del **no-determinismo**. La formalización del enunciado anterior es la siguiente:
Recordemos que un **algoritmo no-determinista** cuenta con dos fases:

- Fase adivinadora.
- Fase verificadora.

Entonces sea Π un problema en la clase **P**, y sea A un **algoritmo determinista** que resuelve a Π en tiempo polinomial, entonces construimos un **algoritmo no-determinista** para Π de la siguiente manera:
Utilizamos a A como la fase verificadora del **algoritmo no-determinista** e ignoramos la fase adivinadora. Así $\Pi \in$ **NP**.
A pesar de que es sencillo convertir un **algoritmo determinista** en un **algoritmo no-determinista**, lo inverso no lo es, de hecho no se conoce un método general para convertir un **algoritmo no-determinista** en tiempo polinomial en un **algoritmo determinista** en tiempo polinomial. De hecho lo único que podemos garantizar es lo siguiente:

Teorema 1.1. *Si $\Pi \in$ **NP**, entonces existe un polinomio p tal que Π puede ser resulto por un **algoritmo determinista** de complejidad $O\left(2^{p(n)}\right)$*

Demostración. Sea Π un problema tal que Π pertenece a la calse **NP**, entonces por definición existe un **algoritmo no determinista** A que resuelve a Π en **tiempo polinomial**. Ahora sea $q(n)$ una cota polinomial superior para la **complejidad en tiempo** de A. Sin pérdida de generalidad asumiremos que $q(n)$ puede ser evaluado en **tiempo polinomial** ya que podemos tomar a $q(n) = c_1 n^{c_2}$ con $c_1, c_2 \in \mathbb{Z}^+$ lo suficientemente grandes para que $q(n)$ acoten superiormente la **complejidad en tiempo** de A. Recordemos que resolver cualquier problema puede ser visto como un problema de reconocimiento de lenguaje, la transformación del primero al último se hace de la siguiente forma *(no entraremos en muchos detalles)*:

- Codificar el ejemplar del problema en un **esquema de codificación estándar**.
- Determinar sí dicha cadena pertenece al lenguaje dado por el problema.

Es fácil ver que sí tenemos un algoritmo A para resolver a Π entonces el problema de **reconocimiento de lenguaje** asociado a Π se puede diseñar una **máquina de Turing** que reconozca dicho lenguaje. En éste contexto como A es un algoritmo no determinista, entonces la **Máquina de Turing** que reconoce al lenguaje asociado a Π es también **no-determinista**, recordemos su especificación:

- Un conjunto finito de símbolos $\mathit{\Gamma}$.
- Un conjunto finito de símbolos de entrada Σ, tal que $\Sigma \subset \mathit{\Gamma}$.
- Un símbolo especial $b \in \mathit{\Gamma}$, llamado blanco.
- Un conjunto finito de estados Q, un estado inicial q_0 y dos estados de paro: q_0 y q_N.
- Una función de transición $d : (Q \setminus \{q_0, q_N\}) \times \mathit{\Gamma} \longrightarrow Q \times \mathit{\Gamma} \times \{-1, +1\}$

Recordemos que la **Máquina de Turing No-Determinista** utiliza su "módulo adivinador" que se encarga de formar cadenas arbitrarias que pertenecen $\mathit{\Gamma}^\star$ y después se hace la verificación sí dicha cadena formada por el "módulo adivinador" es aceptada.
Por la explicación anterior sabemos que para toda entrada i de longitud n, entonces debe de existir una cadena s_i de longitud a lo más $q(n)$ formada por el "módulo adivinador" que es reconocida por la fase verificadora de A y ésta verificación no tomará más de $q(n)$ *(ya que $q(n)$ es cota de la complejidad en tiempo)*. Por la definición de la **Máquina de Turing No-Determinista** sabemos que las cadenas formadas por su "módulo adivinador" pertenecen $\mathit{\Gamma}^\star$, por lo tanto para una cadena de longitud a lo más $q\,(n)$ debemos

considerar a lo más $|\mathit{\Gamma}|^{q(n)}$, ya que en el peor de los casos tenemos que $|\mathit{\Gamma}| = n$, las cadenas que tengan longitud menor que $q(n)$ trivialmente las podemos hacer de longitud igual a $q(n)$, agregándosles el símbolo blanco b. Ahora podemos determinar de manera **determinista** cuando A acepta una entrada de longitud n, simplemente aplicamos la fase verificadora de A hasta que se detenga en los $q(n)$ pasos o haga $q(n)$ pasos de cada una de las $|\mathit{\Gamma}|^{q(n)}$ posibles cadenas. La entrada es aceptada sí en alguna de las cadenas formadas por el módulo "adivinador" lleva a un cómputo aceptado dentro de la cota de tiempo en otro caso no es aceptada. Así hemos formado un algoritmo determinista ya que explícitamente hemos considerado todas las cadenas posibles y entre ellas vamos probando todas a ver cuál es aceptada sí es que existe alguna que es aceptada. Sin embargo la complejidad de éste algoritmo determinista está dada por $O\left(q(n) \cdot |\mathit{\Gamma}|^{q(n)}\right)$ ya que cada cadena tiene longitud a lo más $q(n)$ y existen a los más $|\mathit{\Gamma}|^{q(n)}$ sin embargo eligiendo un polinomio $p(n)$ adecuado resulta que $O\left(q(n) \cdot |\mathit{\Gamma}|^{q(n)}\right) \subset O\left(2^{p(n)}\right)$ lo que implica que el **algoritmo determinista** tenga complejidad $O\left(2^{p(n)}\right)$. $\square$