Asesoría 6: Reducciones Polinomiales

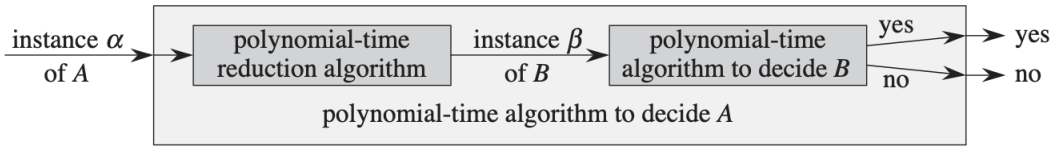
gabriel.spranger@utec.edu.pe

July 5, 2022

1 Introducción

Las reducciones polinomiales nos ayudan a **resolver un problema a partir de otro**, y por lo tanto nos ayuda a establecer **la dificultad relativa entre dos problemas**. Usualmente, esta "dificultad relativa" se lee como *el problema X es al menos tan difícil como el problema Y*. Ahora, pasemos a definir lo que es una **reducción**: si tenemos una **caja negra** que puede resolver X y podemos usar esa caja negra para resolver a Y entonces X es tan difícil como Y. Esta técnica es relevante no solo en el campo de complejidad computacional, sino también para la resolución de problemas, donde modificamos un problema de tal manera que se convierte en una instancia de otro problema cuya solución conocemos, por lo tanto, podríamos usar esa solución que ya conocemos para resolver el problema original. Procedamos a dar algunas definiciones:

1. *Dados dos problemas A y B, decimos que $A \preceq_P B$ ("A se reduce en tiempo polinomial a B") si cada instancia del problema A puede ser resuelta usando un* ***número polinomial de pasos*** *más un* ***número polinomial de llamadas*** *a una caja negra que resuelve a B. En otras palabras, "resolvemos A a partir de B". Por lo tanto, podemos decir que B es tan difícil como A.*



En la figura anterior, recibimos una instancia α de un problema A. Mediante un algoritmo polinomial, transformamos la instancia α en una instancia β de un problema B. Luego, usamos la **caja negra** que sabemos que resuelve cualquier instancia β de B para que resuelva esa misma instancia que acabamos de transformar. Finalmente, usamos la respuesta de β como respuesta de α, pero hay que tener cuidado acá. Como nos concentramos en problemas de **decisión** (SÍ o NO), puede ser que la caja negra bote SÍ, pero puede pasar que para que sea una válida respuesta de α, tengamos que negarla (ransformarla a NO) o dejarla como está (dejarla como SÍ). Esto depende de A y B.

2. *Dados dos problemas X e Y tales que $Y \preceq_P X$, si X puede ser resuelto en tiempo polinomial, entonces Y puede ser resuelto en tiempo polinomial.*

3. *Dados dos problemas X e Y tales que $Y \preceq_P X$, si Y no puede ser resuelto en tiempo polinomial, entonces X no puede ser resuelto en tiempo polinomial.*

2 NP

Los problemas que pertenecen a la clase **NP** tienen una característica importante:

- Se puede **verificar** una posible solución a este problema, en tiempo polinomial.

Para esto, se define un **certificador eficiente** para un problema X. Este certificador eficiente es un algoritmo polinomial (de ahí viene el nombre *eficiente*) que recibe dos argumentos: un input x y un certificado y. Este certificador eficiente **verifica** un input string x si existe un certificado y tal que dados ambos argumentos, el certificador eficiente devuelva "SÍ". Formalmente, sea el certificador eficiente B, un input string $x \in L$ y un certificador y, entonces si dado x, existe un y tal que $B(x, y) = 1$, entonces podemos decir que B **verifica a** L **en tiempo polinomial**. En otras palabras, x sería una instancia del problema X, y y sería una posible solución para una instancia x. En los libros, verán que representan al conjunto de todas las instancias como un lenguaje L y dicen que $x \in L$, es decir, tratan a x como una cadena y L representaría un problema.

Ahora, dejando eso en claro, definiremos a **NP** como la clase de problemas que pueden ser verificados por un algoritmo polinomial.

Contrastando con la clase **P** (existe un algoritmo polinomial que encuentra la solución de ese problema) con la clase **NP** surge una gran pregunta: ¿P = NP? Nótese que cualquier problema en **P** también pertenece a **NP**, pero, ¿lo reverso también se cumple? Si es así, esto significaría que verificar una posible solución a un problema, tiene la misma dificultad que encontrar la solución a ese problema, hablando en términos de reducciones polinomiales, lo cual no suena tan convincente, pero igual todavía no podemos afirmarlo (que P $\neq$ NP).

3 NP-Completo

Un problema X es **NP-Completo** sí:

1. $X \in NP$
2. Para todo problema $Y \in NP$, se cumple que $Y \preceq_P X$

La primera condición dice que el problema tiene que tener un certificador eficiente. La segunda, dice que cada problema $Y \in NP$ se reduce en tiempo polinomial a X, es decir, podemos usar el algoritmo que **decide** a X, para decidir a cada problema en NP. Esto suena muy tedioso, pero veremos que no es así, pero primero, unas propiedades.

4. *Si $X \in$ NP-Completo, entonces X es resoluble en tiempo polinomial si y solo si $P = NP$.*

Proof. Suponga que $X \in P$. Sea $Y \in NP$, como X es NP-Completo se cumple que $Y \preceq_P X$. Como $X \in P$, entonces $Y \in P$(ver la 1era definición). Suponga que $P = NP$, entonces $X \in NP = P$. □

Ahora, la definición que nos ayudará a demostrar de una manera más fácil el segundo punto para demostrar que un problema X pertenece a NP-Completo.

5. *Si Y es NP-Completo, y $X \in NP$, tal que $Y \preceq_P X$, entonces X es NP-Completo.*

Proof. Como $X \in NP$, basta probar solo el segundo punto. Sea $Z \in NP$. Tenemos que $Z \preceq_P Y$, ya que Y es NP-Completo (primer punto). Como $Y \preceq_P X$, por transitividad, se tiene que $Z \preceq_P X$. Entonces X es NP-Completo, ya que probamos que cualquier $Z \in NP$, se reduce en tiempo polinomial a X, lo cual demuestra el segundo punto y como ya teníamos el primer punto ($X \in NP$), X es NP-Completo. □

Esto quiere decir que en vez de tener que demostrar que cada problema $Z \in NP$ se puede reducir en tiempo polinomial a X (queremos mostrar que X es NP-Completo y ya sabemos que $X \in NP$), dado un problema Y que ya sabemos que es NP-Completo, basta solo demostrar que $Y \preceq_P X$ para concluir que X es también NP-Completo.

4 NP-Difícil

Un problema X es NP-Difícil si y solo si cumple el segundo punto, es decir, si:

- Para todo problema $Y \in NP$, se cumple que $Y \preceq_P X$.
- Puede que $X \in NP$ o que $X \notin NP$.

Dada esta definición, podemos decir que todo problema NP-Completo, también pertenece a NP-Difícil, pero no todo NP-Difícil pertenece a NP-Completo. A los NP-Difíciles también se les llama como "los problemas que son al menos tan difíciles como el problema más difícil en NP".

5 Probar que un problema es NP-Completo

Dado un problema X, si queremos probar que X pertenece a NP-Completo, entonces tenemos que probar que:

1. $X \in NP$.
2. Seleccionar un problema Y que es NP-Completo.
3. Mostrar que $Y \preceq_P X$.

El primer problema NP-Completo fue el SAT. A grandes rasgos, la idea de la prueba es que cualquier algoritmo implementado en una computadora física, puede ser reducido a las operaciones $\vee$, $\wedge$ y $\neg$. Por lo tanto, cualquier problema podría ser reducido a SAT. Una vez que ya se demostró que SAT pertenecía a NP-Completo, bastaba usar la propiedad 5 para demostrar que otros problemas pertenecen a NP-Completo. Esto está más detallado en la sección 8.4 del Tardos.

6 Material de Apoyo

- **Link al Jamboard:** `https://jamboard.google.com/d/1xdBeApjqZHJlByaOI-LJij-8vQsgtPhF9idUnG7S1m0/edit?usp=sharing`
- **Tardos:** Capítulo 8.
- **Cormen:** Capítulo 34.

7 Bibliografía

- Cormen capítulo 34.
- Material de clase (Complejidad Computacional) del Profesor Juan Gutiérrez del ciclo 2020-1 del curso de ADA.