Trabajo Práctico Nro 5 - Tiempo polinomial y no polinomial

Ejercicio 1

Responder breve y claramente los siguientes incisos:

1. ¿Por qué la complejidad temporal sólo trata los lenguajes recursivos?

La complejidad temporal solo trata lenguajes recursivos porque si una máquina de Turing no se detiene, no tiene sentido hablar de cuánto tarda, para medir el tiempo necesitamos que la máquina se detenga siempre, si no se detiene, el tiempo sería infinito o no definido.

2. Probar que $n^3 = O(2^n)$.

> **Definición Formal de la notación Big O** >
>
> "Una función $T1(n)$ es del orden de una función $T2(n)$, que se anota así: $T1(n) = O(T2(n))$, sii para todo $n \geq n_0$ se cumple $T1(n) \leq c \cdot T2(n)$, con $c > 0$."

Para probar que $n^3 = O(2^n)$, debemos demostrar que existen constantes positivas c y n_0 tales que para todo $n \geq n_0$, se cumple la desigualdad $n^3 \leq c \cdot 2^n$.

Si tomamos $c = 1$ y $n_0 = 10$ nos damos cuenta que a partir de ese punto se cumple la desigualdad $n^3 \leq c \cdot 2^n$ para todo $n \geq n_0$.

3. Probar que si $T_1(n) = O(T_2(n))$, entonces $TIME(T_1(n)) \subseteq TIME(T_2(n))$.

> **Definición de la clase TIME** >
>
> Un lenguaje L pertenece a la clase de complejidad temporal $TIME(T(n))$, denotado como $L \in TIME(T(n))$, si existe una Máquina de Turing (MT) M que decide L y su tiempo de ejecución está acotado superiormente por $O(T(n))$. Esto significa que existe una constante $c' > 0$ tal que para toda entrada w de tamaño $n = |w|$, M realiza a lo sumo $c' \cdot T(n)$ pasos antes de detenerse.

Lo que nos piden demostrar es que todo lenguaje que puede ser decidido en tiempo $T_1(n)$ también puede ser decidido en tiempo $T_2(n)$, modificado por una constante multiplicativa.

Sabiendo que $T_1(n) \leq c \cdot T_2(n)$, para todo $n \geq n_0$, con $c > 0$. Si tenemos un lenguaje
L ∈ TIME(T_1(n))⇒ \text{Existe una Máquina de Turing M que decide&nbs
. Entonces, para toda entrada w de tamaño $n = |w|$ y $n \geq n_0$, la máquina M se detiene en un $tiempo \leq T_1(n) \leq c \cdot T_2(n)$.

Como $c \cdot T_2(n) \in O(T_2(n))$, podemos decir que existe una máquina que decide L en tiempo $O(T_2(n))$, entonces, $L \in TIME(T_2(n))$ y, por lo tanto, $TIME(T_1(n)) \subseteq TIME(T_2(n))$.

4. ¿Cuándo un lenguaje pertenece a P, a NP y a EXP? ¿Por qué si un lenguaje pertenece a P también pertenece a NP y a EXP?

Un lenguaje pertenece a la clase P si existe una Máquina de Turing determinística que lo decide en tiempo **polinomial**, es decir, en tiempo $O(n^k)$ para alguna constante k.

Un lenguaje pertenece a la clase NP si tiene la siguiente propiedad:

- Si una cadena le pertenece, entonces dicha pertenencia se puede **verificar** en **tiempo polinomial**, con la **ayuda** de otra cadena conocida como **certificado** que debe ser **sucinto** (Tiene **tamaño polinomial** respecto del tamaño de la entrada).

Un lenguaje pertenece a la clase EXP (o **tiempo exponencial**) si es decidible en tiempo $O(c^{poly(n)})$, con c constante, donde $poly(n)$ es una función polinomial de la longitud de la entrada n.

¿Por qué si un lenguaje pertenece a P también pertenece a NP y a EXP?

- $P \subseteq NP$: Si un lenguaje L pertenece a P, existe una Máquina de Turing determinística M_P que lo *decide* en *tiempo polinomial*. Para demostrar que L también pertenece a NP, podemos considerar la definición de NP con un *verificador eficiente*. Si $w \in L$, la MT M_P puede determinar esto en *tiempo polinomial sin necesidad de un certificado*. Podemos considerar un certificado trivial x (por ejemplo, la cadena vacía). Un verificador M_V para L en NP podría simplemente simular la MT M_P en la entrada w (ignorando el certificado x). Como M_P decide L en tiempo polinomial, M_V también lo hará. Por lo tanto, *cualquier lenguaje decidible en tiempo polinomial por una MT determinística también puede ser "verificado" en tiempo polinomial (sin necesidad real de un certificado informativo), lo que implica que $P \subseteq NP$.*
- $P \subseteq EXP$: Si un lenguaje L pertenece a P, existe una Máquina de Turing determinística M_P que lo *decide* en *tiempo polinomial*, digamos en tiempo $O(n^k)$ para alguna constante k. Cualquier función polinomial n^k está asintóticamente acotada por una función exponencial como c^n (para alguna constante $c > 1$ y para n suficientemente grande), o más generalmente por $O(c^{n^j})$ donde j es una constante. Por lo tanto, *cualquier lenguaje decidible en tiempo polinomial también es decidible en tiempo exponencial. Por lo tanto, $P \subseteq EXP$.*

5. ¿Qué formula la Tesis Fuerte de Church-Turing?

La Tesis Fuerte de Church-Turing formula "Si L es decidible en tiempo $poly(n)$ por un modelo computacional físicamente realizable, también es decidible en tiempo $poly(n)$ por una MT *(al menos hasta que las máquinas cuánticas sean una realidad)*."

6. ¿Por qué es indistinta la cantidad de cintas de las MT que utilizamos para analizar los lenguajes, en el marco de la jerarquía temporal que definimos?

En el marco de la jerarquía temporal que definimos, la cantidad de cintas de las Máquinas de Turing (MT) que utilizamos es indistinta debido a que **una MT con múltiples cintas no tiene mayor poder computacional que una MT con una sola cinta.** Cualquier tarea que pueda realizar una MT con K cintas, también puede ser realizada por una MT con una sola cinta.

Si una MT con K_1 cintas decide un lenguaje en tiempo polinomial, digamos $O(n^k)$ para alguna constante k, entonces existe una MT equivalente con K_2 cintas (incluso una sola cinta) que también decide el mismo lenguaje en tiempo polinomial. Lo mismo se aplica para una resolución en tiempo exponencial.

7. ¿Qué codificación de cadenas se descarta en la complejidad temporal?

Se descarta la codificación unaria de cadenas ya que **no es físicamente realizable para números grandes, genera inconsistencias y tergiversa la verdadera complejidad temporal de los lenguajes.**

8. ¿Por qué si un lenguaje pertenece a P también su complemento pertenece a P?

Para demostrar que el complemento de L también pertenece a P, necesitamos construir una Máquina de Turing determinística M^C que decida L^C en *tiempo polinomial*.

Podemos construir M^C a partir de M de la siguiente manera:

1. Dada una cadena de entrada w, M^C simula la ejecución de M en w.
2. Como M decide L, sabemos que M siempre se detendrá en tiempo polinomial.
3. Si M acepta w (lo que significa que $w \in L$), entonces M^C rechaza w (lo que significa que $w \notin L^C$).
4. Si M rechaza w (lo que significa que $w \notin L$), entonces M^C acepta w (lo que significa que $w \in L^C$).

El tiempo de ejecución de M^C es esencialmente el mismo que el tiempo de ejecución de M, más un número constante de pasos adicionales para invertir la decisión de M. **Dado que el tiempo de ejecución de M es polinomial, el tiempo de ejecución de M^C también será polinomial.**

Por lo tanto, si $L \in P$, se demuestra que existe una Máquina de Turing determinística M^C que decide L^C en tiempo polinomial, lo que significa que $L^C \in P$.

9. Sea L un lenguaje de NP. Explicar por qué los certificados de L miden un tamaño polinomial con respecto al tamaño de las cadenas de entrada.

Los certificados de un lenguaje L de NP miden un tamaño polinomial con respecto al tamaño de las cadenas de entrada ya que, si la longitud del certificado $|x|$ fuera mayor que un polinomio en $|w|$ (por ejemplo, exponencial en $|w|$), entonces la MT verificadora, incluso en el caso más rápido posible donde lee un símbolo por paso, **tardaría un tiempo superpolinomial simplemente en leer el certificado completo. Esto contradiría la condición de que la verificación debe realizarse en tiempo polinomial.**

Ejercicio 2

Sea $SMALL - SAT$ = {φ | φ es una fórmula booleana sin cuantificadores en la forma normal conjuntiva (o FNC), y existe una asignación de valores de verdad que la satisface en la que hay a lo sumo 3 variables con valor de verdad verdadero}. Probar que $SMALL - SAT \in P$. Una fórmula booleana sin cuantificadores está en la forma FNC si es una conjunción de disyunciones de variables o variables negadas; p.ej. $(x_1 \vee x_2) \wedge x_4 \wedge (\neg x_3 \vee x_5 \vee x_6)$. *Ayuda: Una MT que decida $SMALL - SAT$ debe contemplar asignaciones con cero, uno, dos y tres valores de verdad verdadero.*

Idea General: Queremos saber si *existe alguna asignación* de valores de verdad a las variables de una fórmula en FNC tal que:

- La fórmula se satisface.
- A lo sumo 3 variables tienen valor verdadero.
  La segunda condición nos da la pista que *no necesitamos probar las 2^n asignaciones posibles* (como se hace en el problema SAT en general), sino *solo aquellas con 0, 1, 2 o 3 variables verdaderas*.

Cantidad de asignaciones con a lo sumo 3 variables verdaderas: Suponiendo que la fórmula tiene n variables distintas:

- Con 0 verdaderas es $\binom{n}{0} = 1$.
- Con 1 verdadera es $\binom{n}{1} = n$.
- Con 2 o más la fórmula sería $\binom{n}{k}$ con k $> 1 = \frac{n!}{k!(n-k)!}$.
  Por lo que la cantidad total nos quedaría:
  $\sum_{k=0}^{3} \binom{n}{k} = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \binom{n}{3} = 1 + n + \frac{n(n-1)}{2} + \frac{n(n-1)(n-2)}{6} \Rightarrow \sum_{k=0}^{3} \binom{n}{k} = O(n^3)$. Esto

quiere decir que hay, como mucho, $O(n^3)$ asignaciones que necesitamos probar y como podemos verificar si la fórmula es verdadera *en tiempo lineal en el tamaño de la fórmula* la máquina de Turing que decide $SMALL - SAT$ lo hará en *tiempo polinomial*, por lo tanto, $SMALL - SAT \in P$.

Ejercicio 3

Dados los dos lenguajes siguientes, (1) justificar por qué no estarían en P, (2) probar que están en NP, (3) justificar por qué sus complementos no estarían en NP:

1. El problema del conjunto dominante de un grafo consiste en determinar si un grafo no dirigido tiene un conjunto dominante de vértices. Un subconjunto D de vértices de un grafo G es un conjunto dominante de G, sii todo vértice de G fuera de D es adyacente a algún vértice de D. El lenguaje que representa el problema es $DOM - SET$ = {(G, K) | G es un grafo no dirigido y tiene un conjunto dominante de K vértices}.

El lenguaje no se encontraría dentro de P ya que para encontrar un subconjunto de vértices con una propiedad específica (como ser un conjunto dominante) y un tamaño dado, el algoritmo más directo conocido implica la exploración de un número exponencial de posibles subconjuntos (**fuerza bruta**). **Para un grafo con m vértices, el número de subconjuntos de tamaño K es $\binom{m}{K}$, que puede ser exponencial en m cuando K es una fracción fija de m, por ejemplo, $K = \frac{m}{2} \Rightarrow \binom{m}{\frac{m}{2}} \approx \frac{2^m}{\sqrt{m}}$**. No se conoce un algoritmo determinístico que resuelva el problema del conjunto dominante en **tiempo polinomial** en el tamaño de la entrada (el grafo G y el entero K).

Para probar que está en NP tenemos que demostrar que existe un **verificador eficiente** que pueda verificar un **certificado sucinto** para las **instancias positivas del problema**.

- Podemos considerar como *certificado sucinto* a un subconjunto D de K vértices de un grafo G que tiene un conjunto dominante de tamaño K. Es *sucinto* ya que el tamaño del certificado podría llegar a ser, como máximo, de tamaño v siendo v la cantidad total de vértices de G, por lo tanto, es de *tamaño polinomial* con respecto al *tamaño de la entrada*.
- Para verificar ese certificado tendríamos que recorrer todo vértice v de G que no está en D, es decir, $v \in V(G) - D$ y, para cada uno de estos vértices v, el verificador debe comprobar si v es adyacente a al menos un vértice u que pertenezca a D, es decir, $(u, v) \in E(G)$ y $u \in D$. Estas comprobaciones se pueden hacer en tiempo polinomial con un orden de complejidad $O((|V| - K) \cdot K \cdot E) \subseteq O(|V| \cdot K \cdot E)$ y como $K \leq |V|$ el orden $\subseteq O(|V|^2 \cdot E)$, siendo E el conjunto de aristas del grafo G.

El complemento de $DOM - SET$ sería **el lenguaje de los pares (G, K) tales que G no tiene un conjunto dominante de tamaño K.** Para que $DOM - SET^C$ esté en NP, debería existir un certificado sucinto que demuestre que ningún subconjunto de K vértices es un conjunto dominante.

- Tomando el ejemplo de $CLIQUE^C$, para demostrar que un grafo no tiene una cierta propiedad (en este caso, no tiene un conjunto dominante de tamaño K), *un certificado podría necesitar incluir información sobre todos los posibles subconjuntos de tamaño K, y verificar que ninguno de ellos satisface la propiedad.* Dado que el número de tales subconjuntos puede ser *exponencial*, no se cree que exista un certificado sucinto (de tamaño polinomial) que pueda ser verificado en tiempo polinomial para $DOM - SET^C$.

2. El problema de los grafos isomorfos consiste en determinar si dos grafos son isomorfos. Dos grafos son isomorfos si son idénticos salvo por la denominación de sus arcos. P.ej., el grafo $G_1 = (\{1, 2, 3, 4\}, \{(1, 2), (2, 3), (3, 4), (4, 1)\})$ es isomorfo al grafo $G_2 = (\{1, 2, 3, 4\}, \{(1, 2), (2, 4), (4, 3). (3, 1)\})$. El lenguaje que representa el problema es ISO = {(G1, G2) | G1 y G2 son grafos isomorfos}.

Podemos decir que el lenguaje ISO no se encontraría en P ya que, para poder decidir si 2 grafos G_1 y G_2 son isomorfos tendríamos que tomar uno de los grafos e ir probando todas las **permutaciones posibles de sus vértices** (modificando las aristas del mismo) y comparando la nueva lista permutada E_1 con la lista de aristas E_2 de G_2 hasta que, eventualmente, si son isomorfos lleguemos a probar una permutación que haga que las 2 listas de aristas sean idénticas. Esto es una solución por **fuerza bruta** que se decidiría en un **orden de complejidad** $O(|V|! \cdot |E|^2)$ que **supera lo exponencial**, $O(a^n) \subseteq O(n!) \text{ para toda constante a} > 0$. En 2015, se probó que ISO se puede decidir un tiempo **cuasi-polinomial** mediante el uso del algoritmo de Babai dejando un orden de complejidad más eficiente de $O(n^{(log\, n)^c})$.

Para probar que está en NP tenemos que demostrar que existe un **verificador eficiente** que pueda verificar un **certificado sucinto** para las **instancias positivas del problema**.

- Podemos considerar como *certificado sucinto* a una *permutación* de los vértices de uno de los grafos, es decir, una *función bijectiva* $f : V(G_1) \to V(G_2)$ que nos dice *"el vértice 1 de G_1 corresponde al 4 de G_2, el vértice 2 al 3, etc"*. Es *sucinto* ya que el tamaño del certificado la cantidad total de vértices de G_1, por lo tanto, es de *tamaño polinomial* con respecto al *tamaño de la entrada*.
- Para verificar ese certificado tendríamos que probar para cada par de vértices $(u, v) \in V(G_1)$ si $(u, v) \in E(G_1)$ entonces comprobamos si $(f(u), f(v)) \in E(G_2)$. *Si hay alguna diferencia, entonces no son isomorfos.* Estas comprobaciones se pueden hacer en tiempo polinomial con un orden de complejidad $O(|E|^2)$.

El complemento de ISO sería **el lenguaje de los pares (G_1, G_2) tales que G_1 y G_2 no son isomorfos.** Para que ISO^C esté en NP, debería existir un certificado sucinto que demuestre que ninguna permutación de vértices de uno de los grafos cumpla las condiciones para que sean isomorfos.

- En este caso como certificado deberíamos de usar todas las permutaciones posibles de vértices, que como vimos, serían $|V|!$, esto hace que el certificado sea de un tamaño mayor al polinomial (no se puede leer directamente), por lo tanto, *ISO^C no estaría en NP.*

Ejercicio 4

Se prueba que $NP \subseteq EXP$. La prueba es la siguiente. Si $L \in NP$, entonces existe una MT M que, para toda cadena de entrada w, verifica en tiempo $poly(|w|) \ si \ w \ \in L$, con la ayuda de un certificado x tal que $|x| \leq p(|w|)$ - p es un polinomio -, y de esta manera, se puede construir una MT M´ que decida en tiempo $exp(|w|) \ si \ w \ \in L$, sin usar ninguna cadena adicional: M´ simplemente barre todos y cada uno de los certificados posibles x de w. Se pide explicar por qué M´ efectivamente tarda tiempo $exp(|w|)$. *Ayuda: como $|x| \leq p(|w|)$ y los símbolos de x pertenecen a un alfabeto de k símbolos, ¿Cuántos certificados x tiene a lo sumo una cadena w?*

Cantidad de certificados a probar: Queremos saber cuántas cadenas x posibles hay tales que: $|x| \leq p(|w|)$, es decir, *a lo sumo tiene longitud polinomial*. Para cada longitud i entre 0 y $p(|w|)$ van a haber k^i cadenas distintas (es una *variación con repeticiones*). Sabiendo esto, el *total de certificados posibles es:* $\sum_{i=0}^{p(|w|)} k^i = \frac{1-k^{p(|w|)+1}}{1-k}$ que podemos acotar a un *orden* $O(k^{p(|w|)})$ ya que $k^{p(|w|)}$ es el término que más crece de la sumatoria. A su vez, como $p(|w|)$ es un *polinomio de un grado d*, entonces, $k^{p(|w|)} = k^{|w|^d}$. Y como una *constante* elevada a una *función polinomial es exponencial*, tenemos que $\sum_{i=0}^{p(|w|)} k^i = O(k^{p(|w|)}) = O(k^{|w|^d})$.

¿Cuánto tarda M´ en total? M´ barre todos los posibles certificados x, y para cada uno ejecuta $M(w, x)$ que tarda $poly(|w|)$, entonces, el tiempo total que tarda M´ se puede calcular como $\text{cantidad de certificados} \cdot \text{tiempo por verificación}$. Si bien el $\text{tiempo por verificación}$ es $poly(|w|)$, la $\text{cantidad de certificados}$ es de *orden exponencial* $O(k^{|w|^d})$, por lo tanto, el tiempo que tarda M´ se rige por la $\text{cantidad de certificados}$, *dejando que M´ es de orden $O(k^{|w|^d})$.*

Al haber visto lo anterior, si $L \in NP$, entonces existe una máquina M´ que decide L en tiempo exponencial, simplemente **probando todos los posibles certificados** hasta que alguno haga que $M(w, x)$ acepte. Por lo tanto, $NP \subseteq EXP$.