Práctica 5 Tiempo polinomial y no polinomial

Ejercicio 1. Responder breve y claramente los siguientes incisos:

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

Los lenguajes recursivos son los único donde siempre termina la ejecución.

b. Probar que n3 = O(2n). Ayuda: hay que encontrar un número natural n0 y una constante c > 0 tales que n3 □ c.2n para todo n □ n0.

Se puede probar usando limites:

$$\lim_{n\to\infty} \frac{n^3}{2^n} = 0$$

Esto que quiere decir, que 2^n crece mucho más rápido que n^3, de hecho cualquier polinomio n^k es más lento que k^n tendiendo al infinito.
Por ello, decimos que n^3 es del orden de 2^n porque es un cota mayor que cubre todos lo que crecen de igual o menor manera.

c. Probar que si T1(n) = O(T2(n)), entonces TIME(T1(n)) □ TIME(T2(n)). Ayuda: hay que probar que si un lenguaje L está en TIME(T1(n)), también está en TIME(T2(n)) - recurrir directamente a la definición de orden O de una función T y de clase TIME(T(n)) -.

Lo que me esta pidiendo es decir, si tengo un problema que se resuelve en x pasos, este esta dentro del conjunto donde se resuelve en y pasos e y es > x , ya que si por ejemplo un problema lo resuelvo en 10 pasos también lo puedo resolver en más.
Suponiendo entonces que tenemos un lenguaje que pertence a T1, y una maquina deterministica que lo resuelve en n pasos. Siempre podemos hacer una maquina M2 que ejecute M1 sobre el problema, y que luego le agregue pasos.

d. ¿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?

Cualquier lenguaje que este en P se puede resolver en tiempo polinomial con una maquina no determinística, por ello un lenguaje en P pertenece también a NP.

Cualquier lenguaje en P tendría un Time(p(n)) que ⊆TIME(2p(n)), se resuelve con la prueba del punto C.

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

La **Tesis Fuerte de Church-Turing** afirma que cualquier modelo de computación físicamente realizable no puede superar **eficientemente** a una máquina de Turing determinista. Sin embargo, la computación cuántica ha llevado a cuestionar su validez, abriendo la posibilidad de que existan modelos físicos que rompan esta barrera.

f. ¿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?

Esto se da porque usar una cantidad de cintas distinta de cintas aunque puede cambiar el tiempo, este cambio entra dentro de la constante y no es afectado por el tamaño de la entrada, cuando hacemos el O de la misma queda igual.

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

En complejidad temporal **se descarta la codificación unaria** (ej: (5 = 11111)), porque:

1. **Infla artificialmente la entrada**: Si (n) es el valor numérico, en unario su longitud es (|w| = n) (exponencialmente mayor que en binario, donde (|w| = \log n)).
2. **Distorsiona la complejidad**: Un algoritmo (O(n)) en unario sería (O(2^{\log n})) en binario (¡exponencial!), pero es el mismo problema.

h. ¿Por qué si un lenguaje L pertenece a P, también su complemento L C pertenece a P? Ayuda: hay que probar que si existe una MT M que decide L en tiempo poly(n), también existe una MT M´ que decide L C en tiempo poly(n).

Esto se debe a que simplemente hay que negar la salida de una máquina de turing que lo resuelva para resolverlo.

i. 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.

Si el tamaño de los certificados no es polinomiales y en cambio son exponenciales el simple hecho de leerlos llevaría tiempo exponencial y no sería NP.

Ejercicio 2. Sea el lenguaje 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 ∈ P. Comentario: las fórmulas φ en la forma FNC son conjunciones de disyunciones de variables o variables negadas; p.ej. (x1 □ x2) □ x4 □ (□x3 □ x5 □ x6). Ayuda: una MT que decide SMALL-SAT debe contemplar asignaciones con cero, uno, dos y hasta tres valores de verdad verdadero.

Idea general, al estar acotado el numero de variables la cantidad maxima de combinaciones se reduce:
En este caso la cantidad de asignaciones posibles de variables :
$\sum_{k=0}^{3} \binom{n}{k} = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \binom{n}{3} \in O(n^3)$

O lo que es lo mismo:

- Todas falsas
- 1 verdadera
- 2 verdaderas
- 3 verdaderas
  O sea, n tomados de a k donde k esta limitado al rango, si no esta limitado la sumatoria seria hasta n y nos daría tiempo O(2^n).

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:

a. 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 problema del conjunto dominante de un grafo no esta en P porque la única solución es hacerlo por fuerza bruta, lo que involucra probar todas las posibles combinaciones de nodos del grafo de tamaño k 1 por 1 hasta conseguir uno que sea el conjunto dominante.
La cantidad de combinaciones son n tomados de a k, y luego la revisión de si una arista se conecta con alguna dentro del grafo dominante en el peor caso es de n^2, es decir en cada arista reviso una conexión con todo el resto de la aristas.
En total, n tomados de a k = 2^n, el orden es O($2^n * n^2$).

Pertenece a NP:

- Entrada máxima del certificado es tamaño = n.
- Verificar si correcto el certificado es recorrer todos los nodos (n), y si no pertenecen al conjunto dominante (n nodos pueden no pertenecer si k=0) chequear si alguna de sus conexiones (n-1 máximo) están conectados a alguno del conjunto dominante, O(n^2).

El complemento no esta en NP:

- El tamaño del certificado es equivalente a todos los conjuntos de tamaño k y un nodo que lo hace no dominante: Leer esto es exponencial.

b. 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., dado el grafo G1 = ({1, 2, 3, 4}, {(1, 2), (2, 3), (3, 4), (4, 1)}), el grafo G2 = ({1, 2, 3, 4}, {(1, 2), (2, 4), (4, 3). (3, 1)}) es isomorfo a G1. El lenguaje que representa el problema de los grafos isomorfos es ISO = {(G1, G2) | G1 y G2 son grafos isomorfos}.

NO esta en P:

- Peor caso hay que chequear n! permutaciones del grafo (recombinaciones de las aristas).
- Peor caso hay que chequear todas los vértices, n^2.
  En resumen, la solución de fuerza bruta es de tiempo factorial, peor que exponencial.

Esta en NP:

- El certificado es de tamaño n, o lo que vendría siendo todos los nodos de G1 y su biyeccion con uno de G2.
- La verificación consiste en revisar cada arista cada arista del lado del G1 y ver si se cumplen en G2, en el peor caso son O(n^2) aristas.

El complemento no esta en NP:

- El certificado que te dice sin lugar a dudas que 2 grafos NO son isomorfos son todas las permutaciones.
- Factorial.

Ejercicio 4. Se prueba que NP □ EXP. La prueba es la siguiente. Si L ∈ NP, entonces existe una MT M que, para toda cadena de entrada w, verifica en tiempo poly(|w|) si w ∈ L, con la ayuda de un certificado x tal que |x| <= p(|w|) - p es un polinomio -, y de esta manera, se puede construir una MT M´ que decida en tiempo exp(|w|) si w ∈ 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| <= p(|w|) y los símbolos de x pertenecen a un alfabeto de k símbolos, ¿cuántos certificados x de tamaño p(|w|) tiene a lo sumo una cadena w?

Porque M' tarda tiempo exp(|w|), es decir crece de forma exponencial con el tamaño del entrada.
El certificado:

- Se verifica, para bien o para mal en tiempo polinomial. Por tanto su tamaño máximo es un polinomio de la entrada.
- Entonces como la longitud del certificado esta en un rango de 0 hasta p(|w|), y la cantidad de símbolos del alfabeto del certificado es k:
  K^0+K^1+K^2+...+K^(p(|w|)) ≈ K^(p(|w|)) (Termino dominante)
  Esto se ve simplemente si planteamos un alfabeto binario, supongamos que determinamos valores de |w| y que hace p (solo para el ejemplo):
- |w| = 5
- p(|w|) = |w|^2 = 25
- O sea que tenemos una cadena binaria de 25 bits básicamente:

Decimal calculation:

$$11111111111111111111111 = (1 \times 2^{24}) + (1 \times 2^{23}) + (1 \times 2^{22}) + (1 \times 2^{21}) + (1 \times 2^{20}) + (1 \times 2^{19}) + (1 \times 2^{18}) + (1 \times 2^{17}) + (1 \times 2^{16}) + (1 \times 2^{15}) + (1 \times 2^{14}) + (1 \times 2^{13}) + (1 \times 2^{12}) + (1 \times 2^{11}) + (1 \times 2^{10}) + (1 \times 2^{9}) + (1 \times 2^{8}) + (1 \times 2^{7}) + (1 \times 2^{6}) + (1 \times 2^{5}) + (1 \times 2^{4}) + (1 \times 2^{3}) + (1 \times 2^{2}) + (1 \times 2^{1}) + (1 \times 2^{0}) = 33554431$$

Conclusión:
El tiempo total de M' es el producto del número de certificados por el tiempo de verificación de cada uno, esto es exponencial, ya que la cantidad máxima de certificados a verificar en el peor caso es aproximadamente K^(p(|w|)) y el tiempo de verificación es p(|w|) y estos se multiplican.