Repaso Practica 3 Ejercicios

Ejercicio 1.

a) El lenguaje de los palíndromos se puede decidir con una MT de una cinta y también con una MT con varias cintas, en tiempo O(n2) y O(n), respectivamente. ¿Por qué es indistinta la cantidad de cintas utilizadas, considerando la jerarquía temporal que definimos?

   Es indistinta la cantidad de cintas porque en ambos casos se tarda tiempo polinomial, haciendo que estén en el mismo espacio dentro de la jerarquía.

b) Vimos que un algoritmo natural para encontrar un divisor que termine en 3 de un número N tarda O(N) pasos. ¿Esto significa que el problema está en P?.

   No, porque depende de la codificación de N.

   1) Si N se codifica en unario entonces n = N, haciendo que tarde O(n) pasos.
   2) Si N se codifica en binario entonces n = log2(N), por lo tanto $O(2^n)$ pasos, por lo tanto la MT M que decide si un numero natural tiene un divisor que termine en 3 tanto tarda tiempo exp(n).
   3) Si N esta en cualquier otra base M también tarda tiempo exp(n).

   Como se debe considerar el peor caso, se tarda tiempo exp(n), haciendo que el problema no este en P.

c) Probar que si T1(n) = O(T2(n)), entonces TIME(T1(n)) ⊆ TIME(T2(n)).

   Si T1(n) esta en el O(T2(n)) eso quiere decir que para todo n se cumple que T1(n) <= c. T2(n). Es decir que todas las MT que tardan T1(n) van a tardar menos que el mejor caso de T2(n). Si un lenguaje L es decidido por una MT M que tarda tiempo T1(n), eso quiere decir que es decidido en menos (o igual) de T2(n). Este hace que ese L pertenezca a TIME(T1(n)) y por lo tanto también a TIME(T2(n)) haciendo que TIME(T1(n)) ⊆ TIME(T2(n)).

d) Probar que la asunción NP ≠ CO-NP es más fuerte que la asunción P ≠ NP, es decir que la implica.

   NP ≠ CO-NP implica que P ≠ NP puesto que NP ≠ CO-NP quiere decir que NP no es cerrado con respecto al complemento. Si lo es P, por lo que si P = NP, entonces NP seria cerrado respecto al complemento, y esto contradice a NP ≠ CO-NP.

e) ¿Qué significa que si L1 ≤P L2, entonces L2 es tan o más difícil que L1, en el marco de la complejidad temporal?

Si L1 ≤P L2 para decidir si un w ∈ a L1 se puede usar la MT que decide si un f(w) ∈ L2. No puede suceder que L2 ∈ P y L1 ∉ P o que L2 ∈ NP y L1 ∉ NP. Por todo lo anteriormente dicho L2 es tan o más difícil que L1.

f) Mostrar certificados para los siguientes lenguajes, e indicar cuáles son sucintos: CH (el lenguaje de los grafos con un Circuito de Hamilton), SAT (el lenguaje de las fórmulas booleanas satisfactibles), NOSAT (el lenguaje de las fórmulas booleanas insatisfactibles), ISO (el lenguaje de los grafos isomorfos), NOCLIQUE (el lenguaje de los grafos que no tienen un clique de tamaño K).

CH: certificado es una secuencia C de m vértices. La MT chequea que C es Circuito de Hamilton de G en tiempo poly(n). Es sucinto.
SAT: certificado es una asignación A de valores de verdad. La MT chequea que A haga satisfactible a la formula booleana, esto se hace en tiempo poly(n). Es sucinto.
NOSAT: el certificado son todas las posibles asignaciones. Esta posible solución (certificado) es una cadena que mide exp(n), por lo que solo recorrerla tardaría exp(n), haciendo que no sea una verificación eficiente (no es polinomial). No es sucinto.
ISO: el certificado es una permutación de vértices. Es sucinto.
NOCLIQUE: el certificado son todas las posibles secuencias C de K vértices. Este certificado mide exp(n), por lo que solo recorrerla tardaría exp(n), haciendo que no sea una verificacion eficiente (no es polinomial). No es sucinto.

g) Mostramos en clase una reducción polinomial del lenguaje CH (del problema del circuito hamiltoniano) al lenguaje TSP (del problema del viajante de comercio). En base a esto justificar:

i. Como TSP es NP-completo, entonces CH ∈ NP.

Para que un lenguaje sea NP-completo debe:
1) Pertenecer a NP
2) Ser NP-hard, es decir, todos los lenguajes que pertenecen a NP se pueden reducir polinomialmente a el.

Como TSP es NP-completo (por lo tanto ∈ NP) y CH se puede reducir polinomialmente a él, por propiedad de reducción polinomial si TSP ∈ NP entonces CH ∈ NP. TSP es tan o más difícil que CH.

ii. Como CH es NP-completo, entonces TSP es NP-difícil.

Como CH es NP-completo, este entonces es NP-hard, es decir, todos los lenguajes que pertenecen a NP se pueden reducir polinomialmente a el. Como la reducción polinomial es transitiva, sea Li cualquier lenguaje ∈ NP, como se sabe que Li ≤P CH y CH ≤P TSP, entonces por transitividad Li ≤P TSP, haciendo que TSP sea NP-hard.

h) ¿Por qué si un lenguaje es NP-completo, en la práctica se considera que no pertenece a la clase P?

Porque si pertenecería a la clase P entonces P = NP, ya que por definicion para que un lenguaje sea NPC este debe ser NP-dificil, es decir, todos los lenguajes que pertenecen a NP se pueden reducir en tiempo polinomial a el. Esto haría que todos los lenguajes de NP se pudieran reducir en tiempo polinomial a un lenguaje que pertenece a la clase P, haciendo que P = NP (algo que se asume que no es cierto).

i) Explicar cómo agregaría un lenguaje a la clase NPC a partir del lenguaje CLIQUE, que se sabe que está en la clase NPC.

Para agregar un lenguaje a la clase NPC a partir del lenguaje CLIQUE demostraría que existe una reducción polinomial desde CLIQUE al lenguaje que quiero agregar. Este lenguaje debería ser NP y como existe una reducción en tiempo polinomial desde CLIQUE al lenguaje que quiero agregar, por transitividad de la reducción el lenguaje que quiero agregar seria NP-difícil, cumpliendo con los dos requisitos para que un lenguaje sea NPC.

j) Probar que si L1 ∈ NPC y L2 ∈ NPC, entonces L1 ≤P L2 y L2 ≤P L1.

Si L1 ∈ NPC y L2 ∈ NPC, eso quiere decir que L1 y L2 son NP y NP-dificil, por lo tanto existe una reducción polinomial de L1 a L2 (porque L2 es NP-dificil), y viceversa.

k) Justificar por qué se consideran tratables sólo a los problemas que se resuelven en espacio logarítmico.

Solo se consideran tratables a los problemas que se resuelven en espacio logarítmico porque espacio S(n) implica tiempo $O(c^{S(n)})$ con c constante que depende de la MT.
Si S(n) ocupa espacio $\log_2(n)$ entonces $c^{\log 2(n)} = n^{\log 2(c)}$ = poly(n) haciendo que el problema se resuelva en tiempo polinomial y por lo tanto sea tratable.
Si S(n) ocupara espacio poly(n) entonces $c^{poly(n)}$ = exp(n) haciendo que el problema se resuelva en tiempo exponencial, y por lo tanto no sea tratable.

l) Explicar por qué en el caso de que un lenguaje requiera para ser decidido mínimamente espacio lineal, se puede utilizar una MT con una cinta de entrada de lectura y escritura.

En el caso de que un lenguaje requiera para ser decidido mínimamente espacio lineal se puede utilizar una MT con una cinta de entrada de lectura y escritura porque solamente el hecho de escribir la entrada en la cinta ocuparía al menos n celdas (|w| = n) haciendo que se ocupe un espacio poly(n).

m) Justificar por qué una MT que se ejecuta en tiempo poly(n) ocupa espacio poly(n), y por qué una MT que ocupa espacio poly(n) puede llegar a ejecutar exp(n) pasos

Una MT que se ejecuta en tiempo poly(n) ocupa espacio poly(n) porque como la MT realiza a lo sumo poly(n) pasos y por definición de MT una maquina escribe un símbolo/celda por paso, se van a usar como máximo poly(n) celdas, ocupando como máximo un espacio poly(n).

Por otro lado, espacio S(n) implica tiempo $O(c^{S(n)})$ con c constante que depende de la MT, si S(n) ocupa espacio poly(n) entonces $c^{poly(n)} = exp(n)$ haciendo que el problema se resuelva en tiempo exponencial.

Ejercicio 2

Sea el lenguaje SMALL-SAT = {ϕ | ϕ es una fórmula booleana sin cuantificadores en 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: ϕ está en FNC si es una conjunción de disyunciones de variables o variables negadas, como p.ej. $(x1 \vee x2) \wedge x4 \wedge (\neg x3 \vee x5 \vee x6)$.

SMALL-SAT ∈ P puesto que se puede construir una MT SMT que la decide en tiempo polinomial, esta MT trabaja de la siguiente manera:

- Verifica que sea una formula booleana que este en FNC. Esto tarda tiempo poly(n).
- Si es una formula booleana que esta en FNC, entonces la SMT prueba todas las posibles combinaciones con 0, 1, 2, y 3 variables con valor de verdad verdadero, es decir, termina haciendo:

$$C(n,0) + C(n,1) + C(n,2) + C(n,3)$$

$$\text{Con } C(n,k) = n! / k!\,(n - k)!$$

- Probar si una combinación de valores de verdad hace satisfactible a la formula tarda tiempo poly(n), y probar con todas las posibles combinaciones con 0,1, 2 y 3 variables con valor de verdad verdadero también tarda tiempo poly(n) (de $O(n^3)$). Por lo tanto la MT SMT decide en SMALL-SAT en tiempo polinomial haciendo que este pertenezca a P.

Ejercicio 3.

El problema del conjunto dominante de un grafo se representa por el lenguaje DOM-SET = {(G, K) | G es un grafo que tiene un conjunto dominante de K vértices}. Un subconjunto de vértices de un grafo G es un conjunto dominante de G, si todo otro vértice de G es adyacente a algún vértice de dicho subconjunto. Probar que DOM-SET ∈ NP. ¿Se cumple que DOM-SET ∈ P? ¿Se cumple que DOM-SETC ∈ NP?

Para probar que DOM-SET ∈ NP se debe construir una MT DOMV que lo verifica en tiempo polinomial.

1) La MT recibe como entrada (G, K, C) siendo G el grafo K la cantidad de vértices y C una secuencia de K vértices.
2) La MT primero verifica que la cantidad de vértices recibidos sea igual a K, esto tarda tiempo poly(n).
3) Luego DOMV verifica que en efecto C sea un conjunto dominante. Para esto se debe hacer un recorrido DFS que también tarda tiempo poly(n) que permite obtener dos valores: A = cantidad de nodos adyacentes a un vértice que es perteneciente al conjunto dominante y B = cantidad de vértices no pertenecientes al conjunto dominante. Si A = B entonces G es un grafo que tiene un conjunto dominante de K vértices.

DOM-SET ∉ P puesto que se debería probar con todas las posibles secuencias C de K vértices si es un conjunto dominante, haciendo que la MT que realiza esto tarde tiempo exp(n).

DOM-SET^C ∉ NP por algo parecido a lo que se mencionó arriba. Se debería usar como certificado todas las posibles secuencias C de K vértices para determinar que no existe en el grafo un conjunto dominante de K vértices. Este certificado mide exp(n) haciendo que la MT en tan solo recorrer el certificado tarde tiempo exp(n), por lo tanto no se puede verificar en tiempo polinomial.

Ejercicio 6.

Asumiendo que Σ* sólo tiene cadenas de unos y ceros de cero o más símbolos, se define, dada la cadena w, que E(w) es su cadena espejo, obtenida reemplazando en w los unos por ceros y los ceros por unos (p.ej. E(1001) = 0110 y E(λ) = λ). Y se define que L es un lenguaje espejo si para toda cadena w distinta de λ cumple w ∈ L ↔ E(w) ∈ L^C. Sea f la función que asigna a toda cadena w su cadena espejo E(w). Responder:

a) ¿Es f una función total computable?

   Si es una función computable ya que existe una MT Mf que la computa y en una cantidad finita de pasos (se detiene). Esta MT Mf lo único que hace es, en el caso en que w sea ≠ a λ, intercambia los 0s por 1s y los 1s por 0s. Esto ultimo implica recorrer el input e ir cambiando los símbolos, lo que implica una cantidad finita de pasos.

b) ¿Cuánto tarda una MT que computa f?

   Va a tardar tiempo poly(|w|), puesto que lo único que hay que hacer es recorrer el input e ir cambiando los símbolos.

c) Si L es un lenguaje espejo, ¿se cumple que f es una reducción polinomial de L a L^C?

No, no es una función de reducción polinomial de L a L^C puesto que no se cumple la definición de reducción polinomial en el caso de que w = λ. Ya que si se aplica la función de reducción sobre λ se tendría como f(λ) = λ, y un w no puede pertenecer tanto a un lenguaje como a su complemento (por definición de complemento ya que este es Σ* - L):
Definición de reducción polinomial:

- a partir de todo w ∈ L1, la MT Mf que computa f genera f(w) ∈ L2.
- a partir de todo w ∉ L1, la MT Mf que computa f genera f(w) ∉ L2.

Ejercicio 7.

Probar:

a) Si los lenguajes A y B son tales que A ≠ ∅, A ≠ Σ* y B ∈ P, entonces (A ⋂ B) ≤P A.

Se puede construir una MT Mf que computa f. Esta Mf trabaja de la siguiente manera:

1) Simula la ejecución de la MT MB (siendo esta la MT que decide el lenguaje B en tiempo polinomial y se sabe que existe por definición de P) sobre el w recibido.
   - Si la MT MB termina en qA entonces la MT Mf deja como salida el mismo input recibido, es decir, f(w) = w, esto es así porque:
     - Si se trata de un w ∈ (A ⋂ B) entonces el w esta tanto en A como en B, por lo tanto f(w) ∈ A
     - Si se trata de un w ∉ (A ⋂ B) entonces el w esta en B, pero no en A, por lo tanto f(w) ∉ A.
   - Si la MT MB termina en qR entonces la MT Mf deja como salida un w ∉ A, es decir, f(w) = x siendo x ∉ A ^ |x| <= |w|.
     - Como el w ∉ B, entonces se sabe que w ∉ (A ⋂ B), por lo que me aseguro de dejar un f(w) que no pertenezca a A.

   Como esta MT Mf simula la ejecución de una MT que tarda tiempo polinomial (MT MB), y luego realiza una cantidad de pasos poly(n) (volver a escribir el input o escribir un x ∉ A), la MT Mf tarda un tiempo poly(n), ya que poly(n) + poly(n) = poly(n).

b) Si L1 ≤P L2, L2 ≤P L1, y L1 ∈ NPC, entonces L2 ∈ NPC.

Por enunciado se sabe que L1 ∈ NP y es NP-dificil (por ser NPC). Como L2 ≤P L1 entonces por teorema de reducción polinomial L2 ∈ NP. A su vez, como L1 ≤P L2, por propiedad transitiva de la reducción polinomial, al ser L1 NP-difícil entonces L2 también lo es, ya que se puede reducir todo lenguaje perteneciente a NP a L2. Por lo tanto al ser L2 NP y NP-dificil, es NP-completo o L2 ∈ NPC.

c) Si un lenguaje es NP-completo, entonces su complemento es CO-NP-completo, es decir, está en CO-NP y todos los lenguajes de CO-NPC se reducen polinomialmente a él.

La reducción polinomial cuenta con la siguiente propiedad:
$L1 \leq_P L2 \Leftrightarrow L1^C \leq_P L2^C$

Esto quiero decir que la función de reducción polinomial para reducir L1 a L2 también sirve para reducir $L1^C$ a $L2^C$. Esto se debe a que si un w no pertenece a L1, (pertenece a $L1^C$) f(w) no va a pertenecer a L2 (pertenece a $L2^C$), y si un w pertenece a L1 (no pertenece a $L1^C$) f(w) pertenece a L1 (no pertenece a $L1^C$).

Por lo tanto, teniendo en cuenta lo anterior, si todo $L' \in NP$ se puede reducir en tiempo polinomial a L_A también perteneciente a NP (haciendo que sea NPC). Entonces todo $L'^C \in$ CO-NP (definición de CO-NP) se puede reducir en tiempo polinomial a $L_A^C \in$ CO-NP (definición de CO-NP) haciéndolo CO-NP-completo (todo lenguaje de CO-NP se puede reducir en tiempo polinomial a él).

Ejercicio 9.

Sea el lenguaje SUB-ISO = {(G1, G2) | G1 es isomorfo a un subgrafo de G2}. Se define que dos grafos son isomorfos si son idénticos salvo por el nombre de sus arcos. Probar que SUB-ISO es NP-completo. Ayuda: Probar con una reducción polinomial desde el lenguaje CLIQUE, el lenguaje correspondiente al problema del clique, que es NP-completo.

Existe una reducción polinomial de CLIQUE a SUB-ISO, siendo:

CLIQUE = {(G, K) | G es un grafo y tiene un subgrafo completo con K vértices}

SUB-ISO = {(G1, G2) | G1 es isomorfo a un subgrafo de G2}

Función de reducción f de CLIQUE a SUB-ISO

Dado un grafo valido G (si es inválido se asigna un “1”) la función es f((G,K)) = (G1, G2) en donde G2 es exactamente el mismo grafo G y G1 es un grafo nuevo creado el cual es completo y con K vértices.

La función f se computa en tiempo polinomial:

Verificar que G sea un grafo valido tarda tiempo poly(n), solo se chequea la sintaxis, escribir un 1 tarda un tiempo constante y crear el nuevo grafo completo de K vértices también tarda tiempo poly(n).

(G, K) ∈ CLIQUE sii (G1, G2) ∈ SUB-ISO:

- Si G no tiene un clique de K vértices, entonces al ser G2 igual G, el grafo G1 completo con K vértices no va a ser isomorfo a ningún subgrafo de G2.

- Si G tiene un clique de K vértices, entonces al ser G2 igual a G, el grafo G1 completo con K vértices va a ser isomorfo a un subgrafo de G2 (justamente al clique de K vértices de G)

Ejercicio 10.

Probar que el problema de los palíndromos está en SPACE(n).

Esta MT MP que trabaja de la siguiente manera:

- La MT MP tiene dos cintas: una cinta de entrada de solo lectura y una cinta de trabajo.
    - La MT MP copia en la cinta de trabajo el input que se encuentra en la cinta de entrada luego mueve el cabezal de la cinta de entrada hacia la izquierda colocándolo en el primer símbolo de la cadena y deja el cabezal en la cinta de trabajo donde estaba (es decir, al final de la cadena).
    - Luego va recorriendo la cadena de la cinta de entrada (hacia la derecha) y la cadena de la cinta de trabajo (hacia la izquierda) comparando los símbolos y chequeando si son el mismo hasta que se llegue a la "c" y se acepta. Si en algún momento no se lee el mismo símbolo en ambas cintas, se rechaza.

Como esta MT MP ocupa en la cinta de trabajo n (|w| = n) celdas, MT MP decide el problema de palíndromos en espacio O(n), haciendo que este en SPACE(n).

Ejercicio 12.

Probar que NP $\subseteq$ PSPACE. Ayuda: Si L pertenece a NP, existe una MT M1 capaz de verificar en tiempo poly(|w|) si una cadena w pertenece a L, con la ayuda de un certificado x de tamaño poly(|w|). De esta manera, se puede construir una MT M2 que decide L en espacio poly(|w|).

Se puede construir una MT M2 que decide L en espacio poly(|w|). Esta MT M2 trabaja de la siguiente manera:

1) MT M2 va a ir generando todos los certificados x posibles en orden canónico reutilizando el espacio ocupando siempre un espacio no mayor a poly(|w|).
2) Simula la ejecución de la MT M1 sobre el w y el certificado generado en el punto 1).
3) Si MT M1 termina en qA, entonces M2 termina en qA. Si M1 termina en qR entonces M2 termina en qR.

- Como MT M1 es capaz de verificar en tiempo poly(|w|), entonces la simulación de su ejecución no va a ocupar más de poly(|w|) celdas.
- La generación de todos los certificados posibles también ocupa un espacio poly(|w|) porque se reutiliza espacio.
- El espacio ocupado por la MT M2 es la suma de espacios polinomiales por lo tanto MT M2 ocupa un espacio poly(|w|)
- Como MT M2 decide un L $\in$ NP en espacio poly(|w|), NP $\subseteq$ PSPACE.