Practica 3

Agustina Sol Rojas y Antonio Felix Glorioso Ceretti

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 $O(n^2)$ 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 utilizadas porque el tiempo sigue siendo polinomial, por lo tanto, están en el mismo espacio dentro de la jerarquía temporal
   definida (P).

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?
   Cada división se puede hacer en tiempo $O(n^2)$.
   Por lo tanto, el algoritmo ejecuta $O(N.n^2)$ pasos.

   Dependiendo de la codificación de N:

   - Si N se codifica en unario, n = N. Así, M tarda tiempo $O(n.n^2) = O(n^3)$ = poly(n).
   - Si N se codifica en binario, $n = O(\log_2 N)$, y por lo tanto $N = O(2^n)$. Así, M tarda tiempo $O(2^n.n^2)$ = exp(n).
   - Si N se codifica en cualquier otra base, M tarda tiempo exp(n).

   Tomando el peor caso, se tiene un tiempo exp(n), por lo tanto, el problema no está en P.

c. Probar que si $T1(n) = O(T2(n))$, entonces $TIME(T1(n)) \subseteq TIME(T2(n))$.

Si T1(n) = O(T2(n)), todas las funciones de O(T1(n)) son también de O(T2(n)) dado que existe un c > 0 tal que T1(n) ≤ c.T2(n). Así, las soluciones que tarden T1(n) tardarán menos que el mejor caso de T2(n), cualquier lenguaje L que use una solución que tarda T1(n) podrá resolverse en una MT M en menos de T2(n). De esta forma es contenido dentro del conjunto de TIME(T2(n)) o TIME(T1(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 quiere decir que NP no es cerrada respecto al complemento, mientras que P si lo es. Si P fuera igual a NP, entonces NP también seria cerrada respecto al complemento, y se asume que no lo es.

e. ¿Qué significa que si L1 $\leq_P$ L2, entonces L2 es tan o más difícil que L1, en el marco de la complejidad temporal?

Como L1 se puede reducir en tiempo polinomial a L2, sabemos que:

a) Si L2 ∈ P entonces L1 ∈ P
b) SI L2 ∈ NP entonces L1 ∈ NP
c) Si L1 ∉ P entonces L2 ∉ P
d) Si L1 ∉ NP entonces L2 ∉ NP

No puede ser que L2 ∈ P y L1 ∉ P ya que, como L1 se puede reducir en tiempo polinomial a L2, se puede utilizar la MT que decide si un w ∈ L2 para decidir si un w ∈ L1. Por lo anterior, siempre 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: Existe una MT CHV que recibirá un grafo G y una secuencia C de m vértices y esta verificara en tiempo poly(|G|) si C es un Circuito de Hamilton de

G. Tarda un tiempo polinomial dado que solo se está recorriendo una secuencia m de vértices y no m!. Se certifica el propio circuito.

- SAT: Existe una MT SATV que recibirá una fórmula booleana ϕ y una asignación A y esta verifica en tiempo poly($|\phi|$) si A satisface la formula ϕ. Tarda un tiempo polinomial dado que solo se tiene que reemplazar las variables por las asignaciones y chequear si da verdadero, y todo esto se puede hacer en tiempo lineal.
- NOSAT: No se puede verificar en tiempo polinomial puesto que se necesita probar con todas las asignaciones posibles para chequear que ninguna haga satisfactible la formula y esto implica una verificación en tiempo exponencial.
- ISO: Existe una MT ISOV que recibe una cadena de pares donde cada vértice de un grafo esta emparejado con uno del otro grafo. La MT compara la cantidad de arcos de ambos pares y si todas las cantidades corresponden con la de su par, entonces los grafos son isomorfos. Esta verificación se hace en tiempo poly($|V|$).
- NOCLIQUE: No se puede verificar en tiempo polinomial puesto que se necesita probar con todas las secuencias de K vértices en el grafo para chequear que ninguna determine un clique de tamaño K de G y esto implica una verificación en tiempo exponencial.

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:

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

      Por el teorema $L1 \leq_P L2$ y $L2 \in NP$: $L1 \in NP$:

      - Al ser TSP NP-completo,
        TSP ∈ NP (por definición de NP-completo) : CH ∈ NP.

      Como TSP es NP-completo, es de los lenguajes más difíciles de NP. Por lo tanto, como CH se puede reducir en tiempo polinomial a TSP, este último va a ser tan o más difícil que CH.

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

Como CH es NP-completo, por definición también es NP-difícil, es decir, todos los lenguajes de NP se pueden reducir en tiempo polinomial a CH. Por lo tanto, como CH tiene una reducción polinomial a TSP, por la propiedad transitiva de la reducción polinomial todos los lenguajes de NP se pueden reducir en tiempo polinomial a TSP, haciéndolo NP-difícil.

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

   Porque es uno de los lenguajes más difíciles de NP y todos los lenguajes pertenecientes a NP tienen una reducción polinomial a este. Esto quiere decir que si pertenecería a la clase P, tendríamos P = NP, los cuales asumimos que no son iguales en la práctica (aunque no está completamente comprobado).

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

   Demostraría que existe una reducción polinomial del lenguaje CLIQUE al lenguaje que quiero agregar a la clase NPC. Por propiedad transitiva, el lenguaje que quiero agregar seria NP-difícil, y a su vez debería cumplir con la pertenencia a NP.

j. Probar que si L1 ∈ NPC y L2 ∈ NPC, entonces L1 δ_P L2 y L2 δ_P L1.

   Si L1 ∈ NPC, entonces es NP, por lo tanto como L2 es NPC, L1 tiene una reducción polinomial a L2 (por definición de NP-completo), y viceversa.

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

   Espacio S(n) implica tiempo $O(c^{S(n)})$, con c constante. Si una MT M ocupa espacio $\log_2 n$, entonces M tarda tiempo $O(c^{\log_2 n})$. Como $c^{\log_2 n} = n^{\log_2 c}$ = poly(n), haciendo que LOGSPACE ⊆ P.

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.

   Cuando un lenguaje necesita ser decidido con un espacio lineal mínimo, es posible utilizar una MT con una cinta de entrada de lectura y escritura. Esto se debe a que, como mínimo, se utilizará un espacio de O(n), donde n es la longitud del input. Ya que el input w, con una longitud de |w| = n, se escribirá en la cinta, ocupando así, al menos, n celdas.

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.

   Si tenemos un tiempo polinomial, entonces eso implicaría que no podemos escribir más de poly(n) celdas en poly(n) pasos, ya que, por definición de MT, siempre se escribe un símbolo por paso. Es decir, como máximo, tenemos espacio polinomial.

   Espacio S(n) implica tiempo $O(c^{S(n)})$, con c constante. Si una MT M ocupa espacio poly(n), entonces M tarda tiempo $O(c^{poly(n)})$. Como $c^{poly(n)}$ = exp(n), haciendo que una MT que ocupa espacio poly(n) ejecute como máximo exp(n) pasos.

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. $(x_1 \lor x_2) \land x_4 \land (\neg x_3 \lor x_5 \lor x_6)$.

SMALL-SAT ∈ P ya que puedo construir una MT SMALL-SAT que lo decide en tiempo poly(n).

MT:

- La MT chequea la sintaxis del input recibido, si no es una formula booleana sin cuantificadores en FNC rechaza. Esto tarda un tiempo poly(n).
- Si es una formula booleana sintácticamente valida, la MT tendrá que chequear la formula con a lo sumo 3 valores verdaderos, esto nos dará una cantidad de combinaciones:

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

  Siendo C:

  $$C(n, k) = \frac{n!}{k!(n-k)!}$$

- Esta fórmula se puede reescribir:

  $$\frac{1}{6}(n^3 + 5n + 6)$$

- Estas son las cantidad de asignaciones que realizara la MT y se le sumara un tiempo B que serán las comprobaciones de satisfacción para cada asignación. Todo esto tarda siempre un tiempo poly(n) de $O(n^3)$, haciendo que SMALL-SAT sea perteneciente 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-SET^C ∈ NP?

DOM-SET ∈ NP , la siguiente MT DOM-SETV verifica DOM-SET en tiempo poly(n):

1) Se le envía a la MT el input (G,K,C), donde G es el grafo, K la cantidad de vértices y C el certificado que tiene el grafo con el subconjunto dominante resaltado.
2) Primero verificara que la cantidad de vértices resaltados sea igual a K esto tardara un tiempo poly(n).
3) Luego calcula con un recorrido DFS, el cual lleva un tiempo poly(n), 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 la MT para decidir si un grafo G tiene un conjunto dominante de K, debe recorrer una a una todas las secuencias C de K vértices de G y chequear en cada caso si C es un conjunto dominante, esto se realiza en un tiempo exp(n) ya que se deben hacer todas las combinaciones.

DOM-SET^C ∉ NP porque sería similar a lo que ocurre en el caso de arriba, habría que recorrer y chequear una a una todas las secuencias C de K vértices de G para verificar que no tiene un conjunto dominante de K vértices. Es decir, no existe un sucinto.

Ejercicio 4.

Probar que el lenguaje FACT = {(N, M1, M2) | N tiene un divisor primo en el intervalo [M1, M2]} está tanto en NP como en CO-NP. Ayuda: Todo número natural N se descompone de una única manera en factores primos, los cuales concatenados no ocupan más de poly(|N|) símbolos.

FACT ∈ NP, la siguiente MT FACT verifica FACT en tiempo poly(n):

1) Se le envía a la MT FACT el input (N,M1,M2,P) donde N es un número, M1 y M2 son el intervalo y P es el numero primo.
2) Primero se chequea que la cadena sea una cadena valida. Esto tarda un tiempo poly(n).
3) Segundo verifica si P está dentro del intervalo [M1, M2]. Si no está en el intervalo, la maquina rechaza. Todo esto tarda un tiempo poly(n).
4) Luego la maquina chequea si el numero P es primo. Esto tarda un tiempo poly(n).
5) Después la MT divide a N con P y si se devuelve un numero entero la MT acepta.

FACT^c = {(N, M1, M2) | N no tiene un divisor primo en el intervalo [M1, M2]}

FACT^C ∈ NP, la siguiente MT FACTC verifica FACT en tiempo poly(n):

1) Se le envía a la MT FACTC el input (N,M1,M2,C) donde N es un número, M1 y M2 son el intervalo y C es la secuencia de divisores primos de N.
2) Primero se chequea que la cadena sea una cadena valida y que C no ocupe más de |N| símbolos. Esto tarda un tiempo poly(n).

3) Segundo verifica si cada símbolo de la secuencia C está dentro del intervalo [M1, M2]. Si no lo está, la maquina rechaza. Esto tarda un tiempo poly(n) ya que estos concatenados no ocupan más de poly(|N|) símbolos y chequear que el símbolo está en el intervalo también lleva tiempo polinomial.
4) Luego la maquina chequea que los símbolos de la secuencia C sean todos primos. Esto tarda un tiempo poly(n) ya que estos concatenados no ocupan más de poly(|N|) símbolos y chequear que un numero sea primo también tarda tiempo polinomial.
5) Después la MT multiplica todos los números de la secuencia C y si el resultado es N la MT acepta.

Como el complemento de FACT, $FACT^C \in NP$, entonces por definición FACT ∈ CO-NP. Se demostró que FACT ∈ NP, por lo tanto FACT ∈ NP ∩ CO-NP

Ejercicio 5.

Mostrar que las reducciones polinomiales son reflexivas y transitivas.

<u>Reflexiva</u>

¿ $L1 \leq_P L1$?

Se puede construir una MT Mf que computa la función de reducción de identidad en tiempo polinomial. Esta Mf básicamente recibe el input y no realiza ningún cambio sobre él.

<u>Transitiva</u>

¿$L1 \leq_P L2 \land L2 \leq_P L3 \Rightarrow L1 \leq_P L3$?

Por enunciado sabemos que se puede realizar una reducción de L1 a L2 con una MT en tiempo polinomial y una reducción de L2 a L3 con una MT en tiempo polinomial. Se puede construir una MT que componga ambas MT que realizan las reducciones de $L1 \leq_P L2$ y $L2 \leq_P L3$. Esta MT va a trabajar en tiempo polinomial también, porque la suma de polinomios da otro polinomio.

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 $w \in L \leftrightarrow E(w) \in L^C$. Sea f la función que asigna a toda cadena w su cadena espejo E(w). Responder:

a) ¿Es f una función total computable?

   Si, f es una función total computable ya que lo único que realiza es un inversión de los valores 1 y 0 de la cadena. Si w es distinto de λ y $w \in L$ entonces $E(w) \in L^c$ y si $w \notin L$ entonces $E(w) \notin L^C$, y como toda cadena es finita, la función siempre terminara.

b) ¿Cuánto tarda una MT que computa f?

   Va a tardar poly(|w|) ya que lo único que debe hacer es recorrer toda la cadena y cambiar los 1 por 0 y los 0 por 1.

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

   No es una reducción polinomial valida debido a la manera en que la función está hecha, la cadena $\lambda \in L$ se reduciría en tiempo polinomial a λ (E(λ) = λ) que $\notin L^C$. Esto incumpliría la propiedad de la reducción polinomial que dice que:

   - a partir de todo $w \in L1$, la MT Mf que computa la función genera $f(w) \in L2$
   - a partir de todo $w \notin L1$, la MT Mf que computa la función genera $f(w) \notin L2$

Ejercicio 7.

Probar:

a) Si los lenguajes A y B son tales que $A \neq \emptyset$, $A \neq \Sigma^*$ y $B \in P$, entonces $(A \cap B) \leq_P A$.

Existe una MT Mf que computa (A ⋂ B) $\leq_P$ A en tiempo polinomial, esta MT trabaja de la siguiente manera:

- Mf recibe el input w y simula la ejecución de la MT MB la cual es una máquina que acepta B y siempre termina.
    - Si la MT MB acepta a w entonces f(w) = w. Esto tarda un tiempo polinomial debido a que es escribir una cadena por lo que esto tarda poly(|w|)
        - Si w pertenece B pero no pertenece a A ⋂ B, se lo va a dejar igual y tampoco va a pertenecer a A (porque no está en la intersección).
        - Si w pertenece B y pertenece A ⋂ B, como se lo deja igual también va a pertenecer a A.
    - Si la MT MB rechaza a w entonces w → f(w) = x ∉ A. Esto tarda un tiempo polinomial debido a que es escribir una cadena por lo que esto tarda poly(|x|)
        - Si w no pertenece a B, entonces tampoco va a pertenecer a la intersección, entonces se lo va a reemplazar por una cadena que tampoco pertenezca a A.

b) Si L1 $\leq_P$ L2, L2 $\leq_P$ L1, y L1 ∈ NPC, entonces L2 ∈ NPC.

Por enunciado se sabe que L1 es NPC, que L2 es NP porque existe L2 $\leq_P$ L1 y que L1 $\leq_P$ L2.
Como L1 es NP-difícil, al ser las reducciones polinomiales transitivas, L2 también lo va a ser ya que se van a poder reducir todos los lenguajes de NP a L2.
Ya que sabemos que L2 es NP y NP-difícil, entonces L2 es 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.
Las reducción polinómicas cuentan con la siguiente propiedad:
L1 $\leq_P$ L2

En donde lo que no pertenece a L1 (lo que sería $L1^C$) tampoco va a pertenecer a L2 (va a pertenecer a $L2^C$). A su vez se cumple que lo que pertenece a L1 (por lo tanto no pertenece a $L1^C$) pertenece a L2 (por lo tanto no pertenece a $L2^C$).
En base a esta propiedad si todo lenguaje L_i perteneciente a NP puede ser reducido a un lenguaje L_A también perteneciente a NP (haciéndolo NPC), entonces todos los complementos de esos lenguajes L_i de NP (que pertenecen a CO-NP), puede ser reducido a $L_A^C \in$ CO-NP (haciéndolo CO-NPC).

Ejercicio 8.

Sea el lenguaje SH-s-t = {(G, s, t) | G es un grafo que tiene un camino (o sendero) de Hamilton del vértice s al vértice t}. Un grafo G = (V, E) tiene un camino de Hamilton del vértice s al vértice t sii G tiene un camino entre s y t que recorre todos los vértices restantes una sola vez. Probar que SH-s-t es NP-completo. Ayuda: Probar con una reducción desde el lenguaje CH, el lenguaje correspondiente al problema del circuito hamiltoniano, que es NP-completo.

Nota: creo que una mejor solución a este problema es que la función de reducción f agregue dos nuevos vértices S y T al grafo G, en donde S tiene arcos con vértices distintos a los que tiene T.

Se sabe que SH-s-t ∈ NP ya que se puede verificar en tiempo polinomial si el camino recibido como sucinto del vértice s al vértice t es un camino de Hamilton, solo hay que probar que se pasó por todos los vértices del grafo.

Existe una reducción polinomial de CH a SH-s-t, siendo:

CH = {G | G es un grafo no dirigido con m vértices y tiene un Circuito de Hamilton}

Circuito de Hamilton (CdeH): recorre todos los vértices del grafo sin repetirlos salvo el primero al final.

SH-s-t = {(G, s, t) | G es un grafo que tiene un camino (o sendero) de Hamilton del vértice s al vértice t}.

Función de reducción f de CH a SH-s-t

Dado un grafo valido G, G’ será ese mismo grafo y se le agrega a la entrada los valores s y t, donde s es cualquier vértice del grafo G’ y t será el mismo vértice que s.

Si G no es un grafo valido, se lo deja tal como está.

La función f se computa en tiempo polinomial:

Verificar el grafo tarda tiempo polinomial (se verifica la sintaxis). Seleccionar cualquier vértice tarda tiempo polinomial.

(G) ∈ CH sii (G’, s, t) ∈ SH-s-t:

- Si G tiene un circuito de Hamilton, existe un camino que recorre todos los vértices del grafo sin repetirlos salvo el primero al final. Se va a cumplir que en G’ existe un camino entre s y t (siendo t s) que recorre todos los vértices restantes una sola vez debido a las propiedades existentes en un circuito de Hamilton.
- Si G no tiene un circuito de Hamilton, no hay un camino que recorra todos los vértices del grafo sin repetirlos salvo el primero al final. En G’ no va a haber un camino entre s y t que recorre todos los vértices restantes una sola vez.

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.

Se sabe que SUB-ISO ∈ NP ya que la demostración es similar a la de ISO vista en teoría, lo único que cambia es que se deben ir haciendo todos los subgrafos de G2. La verificación es en tiempo polinomial.

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

CLIQUE = {(G, K) | G es un grafo y tiene un clique de tamaño K}

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 y un K, se va a crear un grafo G1 completo con K vértices. G2 va a ser G.

Si no es un grafo valido, se lo deja tal como está.

La función f se computa en tiempo polinomial:

Verificar el grafo tarda tiempo polinomial (se verifica la sintaxis). Crear el grafo G1 tarda tiempo polinomial. Copiar G2 tarda tiempo polinomial.

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

- Si G no tiene un clique de k vértices, entonces el grafo G1 completo de K vértices, no va a ser isomorfo a ningún subgrafo de G2 (el cual es G).
- Si G tiene un clique de k vértices, entonces el grafo G1 completo de K vértices va a ser isomorfo a un subgrafo de G2 (el cual es G).

Ejercicio 10.

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

Sea L = { wcw^R | w tiene cero o más símbolos a y b, y w^R es la cadena inversa de w }

Existe una MT PD que acepta L (SPACE(n)) esta trabaja de la siguiente manera:

1) Escribe el input de la cinta de entrada a una cinta de trabajo.
2) Mueve el cabezal de la cinta de entrada al comienzo, pero el de la cinta de trabajo se deja dónde estaba (al final del input).

3) Lee el contenido de la cinta de entrada de izquierda a derecha y el contenido de la cinta de trabajo de derecha a izquierda.
4) Si ambos leen “c” acepta y si en algún momento leen cosas diferentes rechaza. Como la cinta de trabajo con la copia del input mide O(n), L está en SPACE(n).

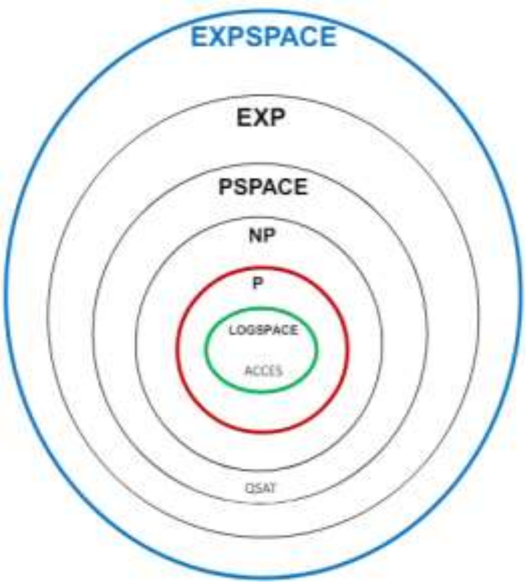
Ejercicio 11.

Justificar por qué el lenguaje QSAT no pertenecería a P ni a NP.

Se sabe que QSAT consume espacio $O(n^2)$.

Espacio S(n) implica tiempo $O(c^{S(n)})$, con c constante. Si una MT M ocupa espacio n^2, entonces M tarda tiempo $O(c^{n^{\wedge}2})$. Como $c^{n^{\wedge}2}$ = exp(n), haciendo que QSAT ⊆ PSPACE.

Esta en PSPACE-completo, todos los lenguajes de PSPACE se reducen polinomialmente a QSAT.



Se puede observar que PSPACE ≠ NP, al poderse reducir todos los lenguajes de PSPACE a QSAT, si QSAT fuera NP, entonces todos los lenguajes de PSPACE serian NP haciendo que PSPACE = NP.

Ejercicio 12.

Probar que NP ⊆ 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|).

LA MT M2 trabaja en espacio poly(n) ya que esta simula la ejecución de MT M1 de la siguiente forma:

- La M2 recibirá como entrada un w cualquiera y generará todos los sucintos posibles para este w. Se sabe que x (sucinto) tendrá siempre un tamaño poly(|w|). Se reutilizará espacio a la hora de ir generando los sucintos.
- Dado que la codificación de cualquier MT es constante y la ejecución de MT M1 es de tiempo poly(|w|) se sabe que a lo sumo usara en una cinta de trabajo un espacio poly(|w|).

MT M2 decide un L ∈ NP y tenemos una suma de espacios polinomiales, por lo tanto NP ⊆ PSPACE.