Practica 3 – Clases 5 – 6 – 7

Métricas complejidad

Complejidad computacional, ya dentro de la clase R. Métricas de complejidad computacional que vamos a considerar:

- Tiempo = cantidad de pasos ejecutados por una MT.
- Espacio = cantidad de celdas ocupadas por una MT.

Complejidad temporal

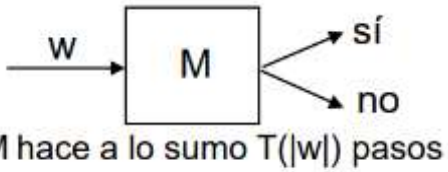
Una MT tarda más a medida que sus cadenas de entrada w son más grandes. Se usan funciones temporales T(n) definidas en términos de |w| = n.

Funciones temporales T(n) típicas: 5n, $3n^3$, 2^n , n^{log2n} , $7^{n\text{^}1/2}$, n!, $6^{n\text{^}5}$.

Dos grandes grupos de funciones:

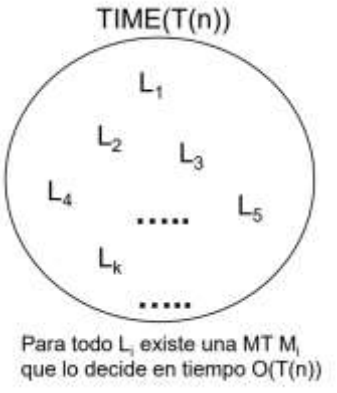
- Polinomiales o poly(n): $T(n) = c.n^k$
- Exponenciales o exp(n): el resto (cuasipolinomiales, subexponenciales, exponenciales, etc).

Tiempo tratable = tiempo poly(n)



Una MT M tarda o se ejecuta en tiempo T(n), si y solo si a partir de toda entrada w, con |w| = n, M hace a lo sumo T(n) pasos.

Una función T1(n) es del orden de una función T2 (n), es decir T1 (n) = $O(T_2(n))$, sii para todo n se cumple $T1(n) \leq c.T_n(n)$, con c > 0.



Un lenguaje L ∈ TIME(T(n)) si y solo si existe una MT M que lo decide en tiempo O(T(n))

Se considera el peor caso (cota superior). Por ejemplo, si a partir de la mayoría de las cadenas de un lenguaje el tiempo de ejecución es O(n), y sólo en algunos casos es $O(n^3)$, el tiempo de ejecución queda castigado en $O(n^3)$.

Si T1(n) = O(T2(n)), entonces TIME(T1(n)) ⊆ 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)).

Clase P

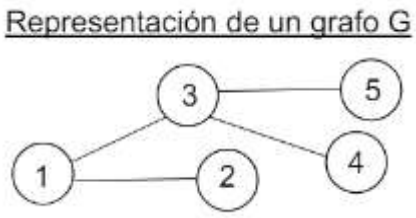
Un lenguaje está en P si es aceptado por una MT en tiempo polinomial.

Si una MT M1 con K1 cintas tarda tiempo poly(n), una MT M2 equivalente con K2 cintas tarda tiempo poly(n). El retardo es a lo sumo cuadrático.

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. 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.

Ejemplo de lenguaje en la clase P.

ACCES = {G | G es un grafo no dirigido con m vértices y tiene un camino del vértice 1 al vértice m}



G = (V, E), con V el conjunto de vértices y E el conjunto de arcos

G = ({1, 2, 3, 4, 5}, {(1,2), (1,3), (3,4), (3,5)})

Existe una MT que decide a ACCES en tiempo poly(n) haciendo un recorrido DFS (en profundidad), en donde en el peor caso M recorre los arcos 2 veces T(n) = O(|E|) = O(|G|) = O(n). Por lo tanto, ACCES ∈ P

Ejemplo de lenguajes que no estaría en P

SAT = {ϕ | ϕ es una fórmula booleana sin cuantificadores, con m variables, y es satisfactible}

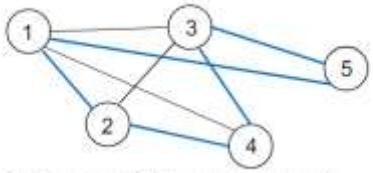
Una MT M que decide SAT emplea una tabla de verdad (no se conoce otro algoritmo). En el caso de una formula booleana que no es satisfactible con ninguna asignación se tiene el

peor caso puesto que hay que probar con todas las asignaciones posibles (2^m) si alguna la hace satisfactible.

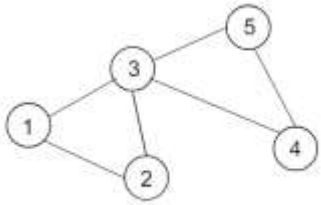
Por lo tanto, M puede llegar a ejecutar exp(n) pasos

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.



Grafo con un CdeH, p.ej. (1, 2, 4, 3, 5)



Grafo sin CdeH

Una MT M que decide CH prueba con todas las permutaciones de vértices (no se conoce otro algoritmo).

Dados m vértices existen m! permutaciones. Por ej., si m = 5: (1,2,3,4,5), (1,2,3,5,4), (1,2,4,3,5), etc.

Por otro lado, chequear si una permutación es un CdeH lleva tiempo $O(|V|.|E|) = O(|G|^2) = O(n^2)$

Como $m! = m.(m-1).(m-2).(m-3)...2.1 = O(m^m) = O(n^n)$, M puede alcanzar los $O(n^n .n^2) =$ exp(n) pasos.

Primera versión de la jerarquía temporal

Dentro de R, P es la clase de los lenguajes decidibles en tiempo poly(n) (lenguajes tratables). EXP es la clase de los lenguajes decidibles en tiempo $O(c^{poly(n)})$. Hay lenguajes fuera de P y fuera de EXP.

La tesis fuerte de Church-Turing afirma la robustez de la clase P: Si un lenguaje es decidible en tiempo poly(n) por un modelo computacional razonable (realizable), también es decidible en tiempo poly(n) por una MT

La codificación razonable (realizable) de números: es cualquiera menos la de base unaria

Ejemplo:

Sea DIV-3 = {N | N es un número natural que tiene un divisor que termina en 3}

Dependiendo el N se ejecutarán ciertas iteraciones que siempre llevarán a O(N).

Cada división se puede hacer en tiempo $O(n^2)$.

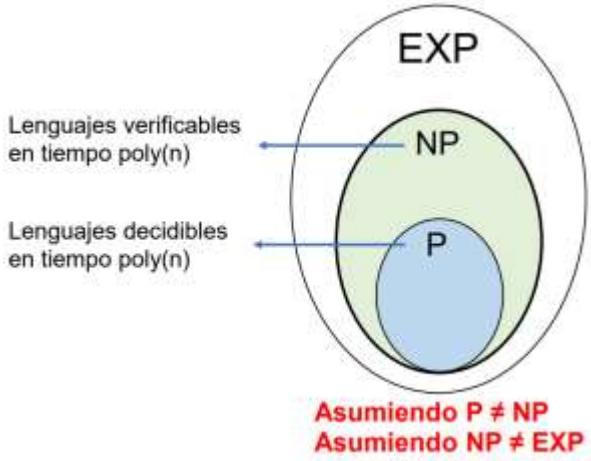
Por lo tanto, el algoritmo ejecuta $O(N.n^2)$ pasos.

Dependiendo de la codificación de N:

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

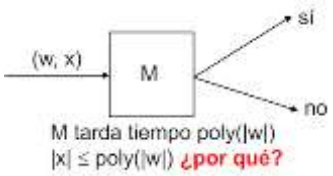
Tomando el peor caso, se tiene un tiempo exp(n), por lo tanto, el problema no está en P.

Clase NP



Un lenguaje L está en NP si existe una MT M tal que: w ∈ L sii existe x tal que M acepta (w,x) en tiempo poly(n). En otras palabras: toda cadena w de L cuenta con un certificado o prueba x que permite verificar eficientemente que pertenece a L. Si este certificado se puede verificar en tiempo polinomial se lo conoce como certificado **sucinto** (o prueba sucinta) de w.

La MT M es un verificador eficiente de L. x es un certificado sucinto (o prueba sucinta) de w. El certificado es una solución posible. Respuesta al ¿por qué?: la solución debe medir poly(n) porque recorrer una cadena exp(n) tarda tiempo exp(n).



Hasta hoy día no se ha encontrado una prueba de P ≠ NP.

Las resoluciones de los problemas de (NP – P) solo se puede realizar empleando la fuerza bruta (búsqueda exhaustiva en el espacio de todos los posibles certificados).

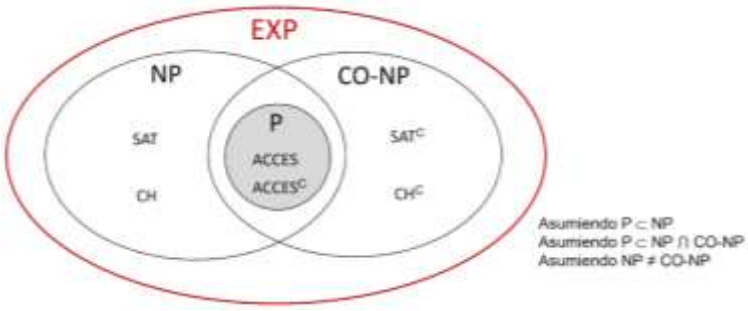
SAT no estaría en P: si ϕ tiene m variables, hay 2^m asignaciones A para chequear.

SAT está en NP: dada una fórmula booleana ϕ y una asignación A, se puede verificar en tiempo poly(n) si A satisface ϕ.

CH no estaría en P: si G tiene m vértices, hay m! secuencias C de m vértices para chequear.

CH está en NP: dado un grafo G y una secuencia C de m vértices, se puede verificar en tiempo poly(n) si C es un CdeH de G.

Tercera versión de la jerarquía temporal. La clase CO-NP.



CO-NP tiene los complementos de los lenguajes de NP. La conjetura aceptada es NP ≠ CO-NP. Es decir, NP no sería cerrada con respecto al complemento. P sí lo es, lo que refuerza P ≠ NP. 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.

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

Un clique de tamaño K en un grafo G es un subgrafo completo de G con K vértices

La mejor MT conocida para decidir si un grafo G tiene un clique de tamaño K, consiste en recorrer una a una todas las secuencias C de K vértices de G y chequear en cada caso si C determina un clique. Esto se realiza en un tiempo exp(n) ya que se deben hacer todas las combinaciones por lo que CLIQUE no estaría en P.

Por otro lado, se puede verificar en tiempo poly(n) si una secuencia C de K vértices define un clique de tamaño K de un grafo G: se chequea que todos los pares de vértices de C son arcos de G, lo que tarda poly(n). CLIQUE está en NP. M es un verificador eficiente de CLIQUE, con certificados C de tamaño O(n).

K-CLIQUE = {G | G es un grafo que tiene un clique de tamaño K}

Ahora el tamaño K del clique no forma parte de las instancias del problema, es una constante.

Como K no forma parte de las entradas del algoritmo descripto previamente, el problema ahora tiene resolución polinomial, está en P: $O(n^K)$ secuencias de K vértices en V para chequear. K ahora es una constante, no depende de la entrada. Cada chequeo se puede hacer en tiempo $O(n^3)$. Total: $O(n^K.n^3)$ = poly(n) pasos. *Nota: podría discutirse de todos modos si para un K muy grande, por ejemplo 1000, el tiempo de la MT construida puede considerarse aceptable*.

K-CLIQUE está en P.

NO-CLIQUE = {(G, K) | G es un grafo que no tiene un clique de tamaño K}

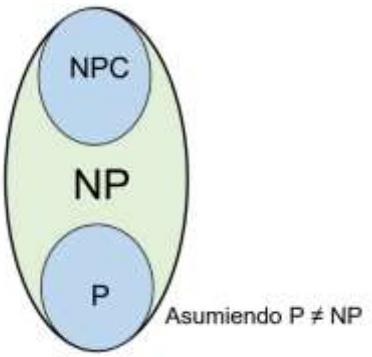
Un certificado debe incluir a todas las secuencias de K vértices de V, porque se tiene que chequear que ninguna determina un clique de tamaño K de G. El certificado x tiene que incluir las exp(n) secuencias de K vértices de G. Básicamente se necesita probar con todas las secuencias de K vértices en el grafo para chequear que ninguna determine un clique de tamaño K de G y esto implica una verificación en tiempo exponencial.

De esta manera, NO-CLIQUE no estaría en NP

- Se conjetura que la clase NP no es cerrada con respecto al complemento.

Lenguajes NP-completos

Asumiendo la conjetura P ≠ NP, hay una manera de establecer que un lenguaje L de NP no está en P que es probando que L es NP-completo, o que está en la clase NPC. Los lenguajes NP-completos son los más difíciles de la clase NP.



Para definir a los problemas NP-completos, hay que utilizar reducciones pero polinomiales (son computables en tiempo poly(n))

L1 ≤p L2 establece que existe una reducción polinomial de L1 a L2 .

Estas (al igual que en el caso general) son:

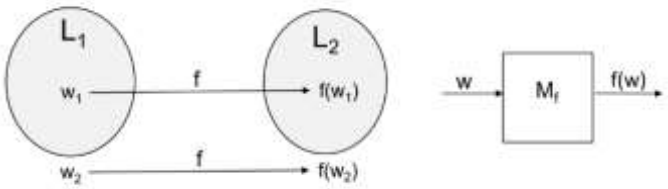
- Reflexivas: ¿ L1 ≤P L1?

    - 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.
- Transitivas: ¿L1 ≤P L2 ∧ L2 ≤P L3 ⇒ L1 ≤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 ≤P L2 y L2 ≤P L3. Esta MT va a trabajar en tiempo polinomial también, porque la suma de polinomios da otro polinomio.
- No simétricas: ¿L1 ≤P L2 ⇒ L2 ≤P L1?
    - No, o al menos no por ahora. Contraejemplo: L1 ∈ P y L2 ∈ NP, se puede construir una MT que compute la función de reducción tiempo polinomial de L1 a L2, pero no se puede construir una MT que realice lo mismo pero de L2 a L1, porque esto significaría que P = NP (ya que dado que L1 ∈ P, por teorema L2 también ∈ P) y eso todavía aún no se ha demostrado.

Las reducciones polinomiales permiten relacionar lenguajes.

Teorema

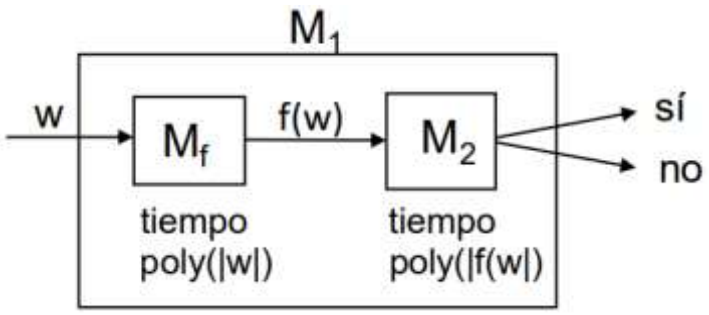
- L1 ≤p L2 entonces (L2 ∈ P ⟶ L1 ∈ P)
    - Si L1 ≤p L2 entonces (L1 ∉ P ⟶ L2 ∉ P)
- L1 ≤p L2 entonces (L2 ∈ NP ⟶ L1 ∈ NP)
    - Si L1 ≤p L2 entonces (L1 ∉ NP ⟶ L2 ∉ NP)



Si L1 ≤p L2 , L2 es tan o más difícil que L1. No puede ser que L1 ∉ P y L2 ∈ P o que L1 ∉ NP y L2 ∈ NP 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.

<u>Idea general de la prueba</u>

- M2 decide si f(w) ∈ L2 y así M1 decide si w ∈ L1
- M2 verifica si f(w) ∈ L2 y así M1 verifica si w ∈ L1



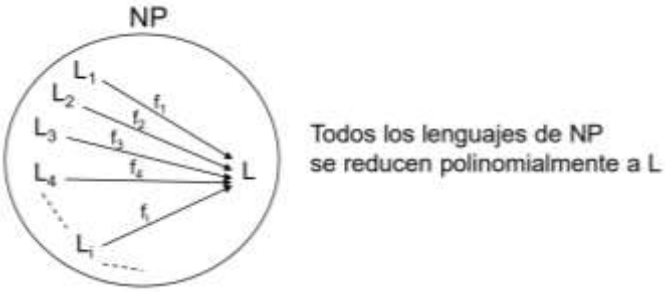
Tiempo de M1 = poly(|w|) + poly(|f(w)|)

Como |f(w)| = poly(|w|) (COMO Mf traba en tiempo polinomial su salida no puede dar más que eso) entonces tiempo de M1 = poly(|w|) + poly(poly(|w|)) = poly(|w|)

Definición NPC

Un lenguaje L es NP-completo, o L ∈ NPC, sii:

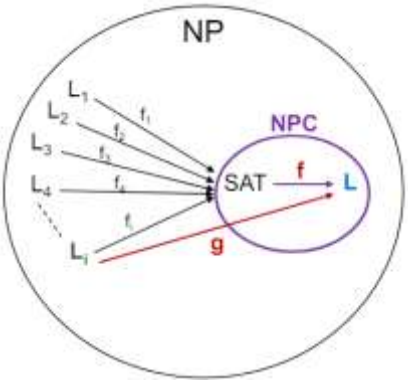
- L ∈ NP
- Para todo L' ∈ NP se cumple L' ≤p L (se dice que L es NP-difícil)



Si L estuviera en P, entonces todos los lenguajes de NP estarían en P y de esta manera, la relación entre P y NP sería: P = NP.

El primer lenguaje NP-completo encontrado es SAT = {ϕ | ϕ es una fórmula booleana sin cuantificadores y es satisfactible}. Utilizando reducciones polinomiales a partir de SAT podemos poblar la clase NPC:

1. Sea L un lenguaje de NP
2. Sea f una reducción polinomial de SAT a L
3. Sea g la reducción polinomial obtenida componiendo una de las reducciones polinomiales fi con f
4. Como lo anterior vale para todo Li se cumple que L es NP-completo



Resumiendo:

$L_1 \in NPC$

$L_2 \in NP$

$L_1 \leq_p L_2$

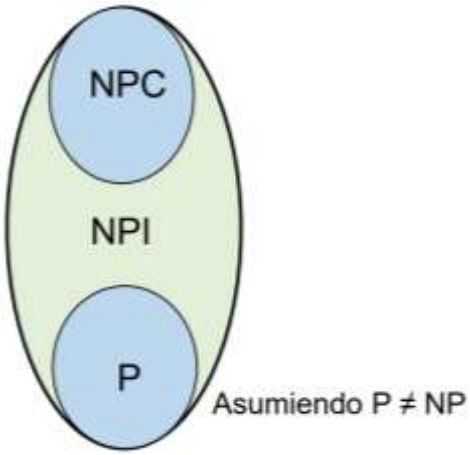
$L_2 \in NPC$

CV = {(G, K) | G es un grafo y tiene un cubrimiento de vértices de tamaño K}

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

CH = {G | G tiene un Circuito de Hamilton}.

Son ejemplos clásicos para encontrar lenguajes NP-completos



- Una visión aún más detallada de NP:
- Asumiendo P ≠ NP, se prueba que además de P y NPC, NP incluye una tercera clase de lenguajes: NPI.
- Los lenguajes de NPI no son ni tan fáciles como los de P ni tan difíciles como los de NPC.
- ISO = {(G1 , G2) | G1 y G2 son grafos isomorfos, es decir son idénticos salvo por el nombre de sus arcos} es candidato para estar en NPI:
    - ISO está en NP
        - los certificados suscintos son permutaciones de V. Es una sola permutación cada certificado.
    - ISO no estaría en P
        - en el peor caso hay que probar con todas las permutaciones de V (ejercicio)
    - ISO no estaría en NPC
        - no se ha encontrado un lenguaje NP-completo L tal que L ≤p ISO
- FACT = {(N, M1 , M2) | N tiene un divisor primo entre M1 y M2} también es candidato para estar en NPI.

Jerarquía de lenguajes (de menor a mayor dificultad)

1) P
2) (NP ∩ CO-NP) – P
3) NPI - CO-NP
4) NPC
5) CO-NPI – NP
6) CO-NPC

ISO sería más fácil que SAT

FACT sería más fácil que ISO

Los lenguajes de CO-NP que son CO-NP-completos se definen como los lenguajes de NP que son NP completos: todos los lenguajes de CO-NP se reducen polinomialmente a ellos. Los complementos de los lenguajes NP-completos son CO-NP-completos, es decir, está en CO-NP y todos los lenguajes de CO-NP se reducen polinomialmente a ellos:

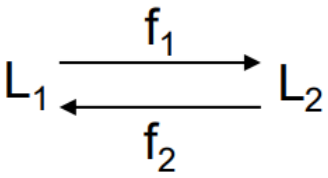
Las reducción polinómicas cuentan con la siguiente propiedad: L1 ≤P L2

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 $L2^C$).

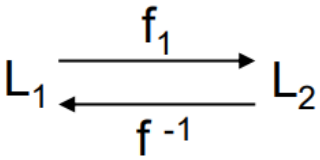
En base a esta propiedad si todo lenguaje L_i perteneciente a NP puede ser reducido a un lenguaje L_A también perteneciente a NP (haciéndolo NPC), entonces todos los complementos de esos lenguajes L_i de NP (que pertenecen a CO-NP), puede ser reducido a $L_A^C \in$ CO-NP (haciéndolo CO-NPC).

Dos características de los problemas NP-completos

- Para todo par de lenguajes L1 y L2 de NPC se cumple $L1 \leq_p L2$ y $L2 \leq_p L1$



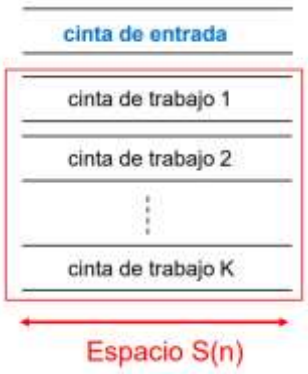
  - Si $L1 \in$ NPC, entonces es NP, por lo tanto como L2 es NPC, L1 tiene una reducción polinomial a L2 (por definición de NP-completo), y viceversa.
  - No necesariamente f2 es la función inversa de f1. Sin embargo, todos los lenguajes NP-completos conocidos cumplen dicha propiedad



- Todos los lenguajes NP-completos conocidos son densos.

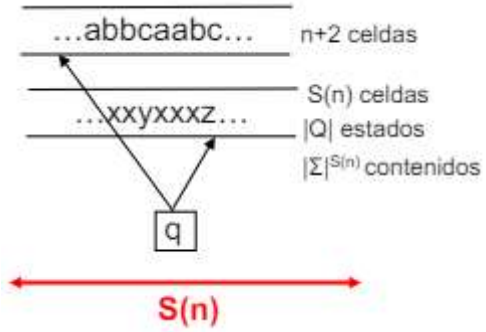
Ver los ejemplos de la practica

Introducción a la complejidad espacial



- Se consideran MT tales que la cinta de entrada es de sólo lectura. El resto son cintas de trabajo.
- Una MT ocupa espacio S(n) sii en todas sus cintas de trabajo (no cuenta la cinta de entrada) ocupa a lo sumo S(n) celdas, siendo n como siempre el tamaño de las cadenas de entrada.
- La cinta de entrada de sólo lectura permite espacios menores que O(n).

Una MT M que ocupa espacio S(n) puede no parar, pero dada M, existe una MT M' equivalente que ocupa espacio S(n) y para siempre.



P.ej., una MT con 1 cinta de entrada de sólo lectura y 1 cinta de trabajo entra en loop luego de: $(n+2).S(n).|Q|.|\Sigma|^{S(n)} = O(c^{S(n)})$ pasos. **Tener en cuenta entonces que: espacio S(n) implica tiempo $O(c^{S(n)})$.**

Un lenguaje L pertenece a la clase SPACE(S(n)) sii existe una MT que decide L en espacio O(S(n)).

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.

Jerarquía espacial

- LOGSPACE es la clase de los lenguajes aceptados en espacio $O(\log_2 n)$. Para que ocupe ese espacio básicamente lo que se hace es reusarlo.
- PSPACE es la clase de los lenguajes aceptados en espacio poly(n)
- EXPSPACE es la clase de los lenguajes aceptados en espacio exp(n)

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 $\subseteq$ P.

Los problemas tratables en espacio son los que pertenecen a la clase LOGSPACE. Existe una clase NLOGSPACE (homóloga a la clase NP), también incluida en P.

Tiempo T(n) → espacio S(n).

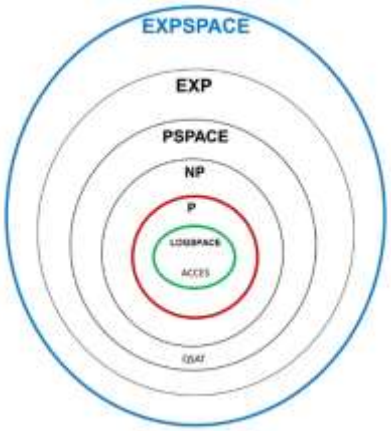
Espacio S(n) → tiempo $O(c^{S(n)})$. Una MT no puede ejecutar más de $c^{S(n)}$ pasos sin repetir alguna configuración, siendo c una constante que depende de la MT.

Justificación de por qué una MT que se ejecuta en tiempo poly(n) ocupa espacio poly(n), y 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.

Jerarquía espacio-temporal



- QSAT = {ϕ | ϕ es una fórmula booleana con cuantificadores, no tiene variables libres, y es verdadera}
- QSAT está entre los lenguajes más difíciles de PSPACE. Consume espacio $O(n^2)$
- En efecto, QSAT es PSPACE-completo, todos los lenguajes de PSPACE se reducen polinomialmente a QSAT.
- QSAT es una instancia de un problema más general: la búsqueda de una estrategia ganadora en una competencia entre dos jugadores J_1 y J_2 (ajedrez, damas, go, hexágono, geografía, etc.)

Leer sobre QSAT (QBF) en el libro.

<u>El problema de los palíndromos está en SPACE(n)</u>

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).

<u>QSAT no pertenecería a P ni a NP</u>

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

Espacio S(n) implica tiempo O(cS(n)), con c constante. Si una MT M ocupa espacio n2, entonces M tarda tiempo O(cn^2). Como cn^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.

NP ⊆ PSPACE

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.