Guia 8 Complejidad Computacional 1 Cuatrimestre 2025

Juan DElia

Ejercicio 1 Un lenguaje es esparso si existe un polinomio p tq $|L \cap \{0,1\}^n \leq p(n)|$ para todo n. Probar que todo lenguaje esparso esta en P/Poly

Sea L esparso quiero ver que L pertenece a P/Poly

Si L es esparso para todo n la cantidad de palabras de longitud n de L esta acotada por un polinimio P(n).

Para ver que L pertenece a P/Poly tenemos que ver que existe una familia de circuitos C_n tal que para todo n:

- C_n tiene tamaño polinomial
- C_n tiene n bits de entrada
- $C_n(x) = 1 \iff x \in L$

Veamos que se puede construir la familia de circuitos.

cada C_n se puede construir de la siguiente manera:

El circuito C_n tiene que aceptar a las palabras del lenguaje de longitud n y rechazar al resto. Como la cantidad de palabras de longitud n esta acotada por un polinomio P(n) las podemos hardcodear en el circuito.

Luego C_n como todas las palabras hardcodeadas podemos chequear que la entrada sea igual a alguna de ellas y aceptar o rechazar si esta o no, esto se puede hacer con compuertas and y or. Vemos que caracter a caracter sea igual a alguna palabra mediante and's y devolvemos un or que este conectado a cada and (si es iguala a alguna devuelve true).

Se ve que $C_n(x) = 1 \iff x \in L$ Ademas este circuito tiene tamaño polinomial (p(n).n), porque tiene P(n) palabras de longitud n.

Ejercicio 2 Probar que existen lenguajes fuera de P/poly

$$\#\text{ circuitos polinomiales} = 2^{O(p(n)log(p(n)))} = 2^{kp(n)log(p(n))}$$

$$\#\text{ funciones} = 2^{2^n}$$

Para que todos las funciones sean P/Poly tendria que valer que:

$$\#\text{ circuitos polinomiales} \geq \ \#\text{ funciones}$$

$$2^{kp(n)log(p(n))} \geq 2^{2^n}$$

Tomando logaritmo de ambos lados:

$$k.p(n).log(p(n)) \geq 2^n$$

Que es falso pues es una exponencial contra un polinomio. de hecho:

$$k.p(n).log(p(n)) << 2^n$$

Por lo tanto existen lenguajes fuera de P/poly

Ejercicio 3 Probar que $P_{advice} = P/poly$

El consejo tiene tamaño polinomial

Veamos que $P/poly \subseteq P_{advice}$

Dado un L $\in$ P/Poly quiero ver que L $\in P_{advice}$

Como L esta en P/poly hay C_n para todo n tq: $x \in L \iff x \in C_n$ con $|x| = n$

Para que este en P_{advice} tiene que existir un adv(n) de tamaño polinomial tq: $M(x, adv(|x|)) = 1 \iff x \in L$ (el consejo solo depende del tamaño de x).

Elijo adv(n) = C_n (que existe por hipotesis).

Esto funciona porque el consejo solo depende del tamaño de la entrada. Entonces la maquina es $M(x, adv(|x|)) = M(x, C_|x|)$ y simplemente va a simular $C_{|x|}(x)$

Veamos que $P_{advice} \subseteq P/poly$

Dado L en P_{advice} qvq esta en P/poly.

Como esta en P_{advice} existe M(x,adv(|x|)) = 1 (llamemos y a adv(|x|))

por cook-levin (como sat es np-completo puedo reducir cualquier maquina poli a una formula):

$$\phi_m(x, y) = 1 \iff M(x, y) = 1$$

Esta claro que puedo escribir una formula armando un circuito. Ademas solo depende de x (porque y depende de la longitud de x).

Ejercicio 4 Definimos P/f(n) como la clase de problemas que se resuelve con un consejo de tamaño f(n). Probar que $P \neq P/1 \cap R$

Basta con dar un lenguaje de $P/1 \cap R$ que no este en P.

Propongo:

L = {x:$\{0,1\}^n$: Existe codificacion(x) una cadena que depende de x y |codificacion(x)| = 2^x. x pertenece sii $codificacion(x)_i = 1$}

Este lenguaje no pertenece a P porque ir a leer el bit puede ser exponencial. Pertenece a P/1 si usamos como consejo el valor de ese bit.

Ejercicio 5 Probar que P = NP si solo si NP $\subseteq$ P/log(n)

La ida es trivial si NP esta en P tambien va a estar en P/f(x) para cualquier f(x) los consejos "son gratis".

$\Leftarrow$)

Hipotesis: NP $\subseteq$ P/log(n)

Quiero ver que NP = P, esto pasa si solo si SAT $\in$ P

Por hipotesis se que SAT $\in$ P/log(p). Esto significa que existe un adv. |adv(n)| = O(log(n)) y existe una maquina M polinomial tal que

$$M(x, adv(|x|)) = 1 \iff x \in SAT$$

Ademas como los consejos son de tamaño $O(log(n))$ podria tenerlos todos en una lista , llamemosla a, de tamaño $O(2^{log(n))} = O(n^c)$

Es decir que tiene tamaño polinomial.

La idea va a ser entonces probar todos los consejos, podria definir la maquina $M_sat(x)$ que: 1. genera todos los consejos, esto es polinomial por lo que explique antes. 2. Corre la maquina M (

$$M(x, adv(|x|)) = 1 \iff x \in SAT$$

) con cada consejo. Esta maquina corre en tiempo polinomial |a| veces (que a su vez es polinomial). 3. Si alguna simulacion dio 1 significa que existe formula que satisface asi que devuelve 1, caso contrario devuelve 0

Ejercicio 6 Probar que si $NP \not\subseteq P/poly$ entonces $NP \neq P$

Por teorema sabemos que $P \subseteq P/poly$

Veamos por el absurdo que P = NP,

valdria: $NP = P \subseteq P/poly$

Esto es absurdo por hipotesis, NP no esta contenido en P/poly, entonces $P \neq NP$

Ejercicio 9 Probar que los lenguajes AND y OR estan en NC^1 usar esto para probar que $AC^d \subseteq NC^{d+1}$

Para ver que estan en NC^1 propongo el siguiente circuito

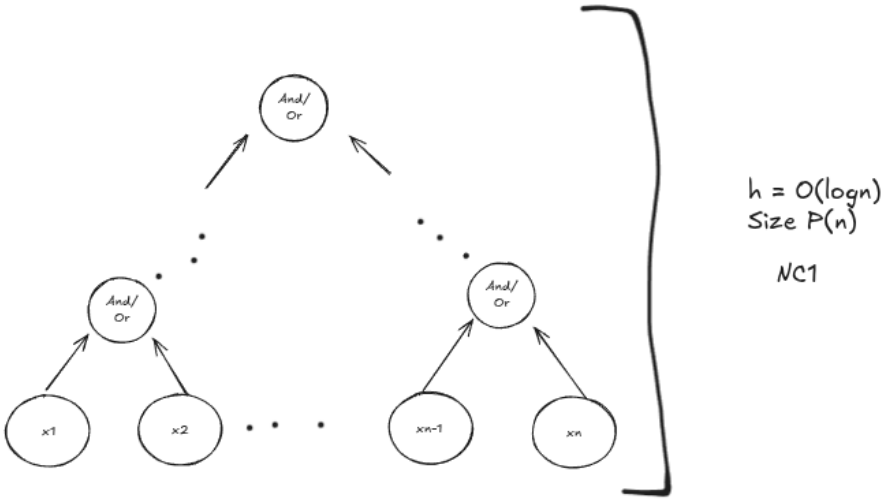

Figure 1: NCˆ1

Analogamente se puede ver que esta en AC^0

Basicamente vimos que los and u or de muchos x's se puede escribir en altura O(1) en AC por el fanning arbitrario. En NC por el fanning binario tiene altura O(log n). Veamos que cualquier C en AC^d se puede escribir en NC^{d+1}:

C tiene altura $O(log^d n)$ y usa fanning arbitrario en los or y and.

Creo un C' que simula a C.

C' reemplaza cada and/or que tiene fanning arbitrario por nodos que usan fanning binario (como vimos en los circuitos). Esto cambia la altura de esos or/and de O(1) a O(log n).

Si la altura de C es $O(log^d n)$ como en C los and/or tenian altura O(1) en C' tienen altura O(log n). Por lo tanto la altura de C' es $O(log^{d+1} n)$.

Entonces C' (como sigue siendo polinimial) pertenece a NC^{d+1}

Por lo tanto $AC^d \subseteq NC^{d+1}$

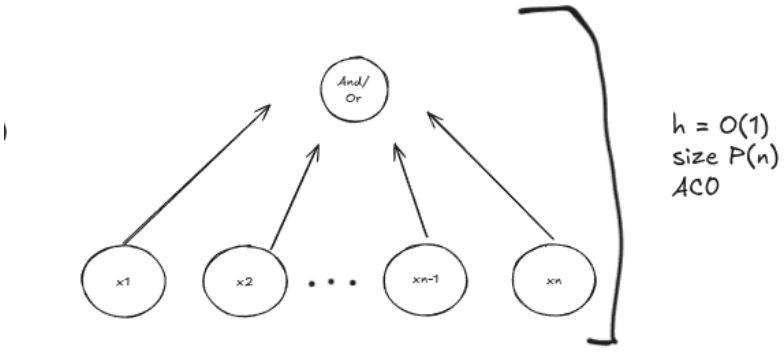

Figure 2: ACˆ0

Ejercicio 10 Probar que

$$NC^1 \subseteq L$$

(Esta demostrado en la clase teorica mucho mejor que lo que voy a escribir)

Quiero ver que cualquier lenguaje $\in NC^1$ tambien pertenece a L.

Sea $\Pi \in NC^1$:

Existe C_n una familia de circuitos polinomiales y de profundidad O(log n) que reconoce Π.

Quiero ver que una maquina M implicitamente computable en L puede simular a los circuitos.

La maquina solo puede traerse espacio logaritmico a la memoria.

Podemos pensar al circuito como un camino de las varibles al sumidero (DAG). Hay que ir viendo cada camino para eso hacemos DFS sobre el circuito. Como tiene altura logaritmica y DFS solo guarda de a una rama en memoria, se puede hacer con espacio logaritmico. De esta manera recorre todas las posibilidades recursivamente y decide Π

Ejercicio 11 Decidir si las clases AC^k y NC^k estan cerradas por la union, interseccion y complemento.

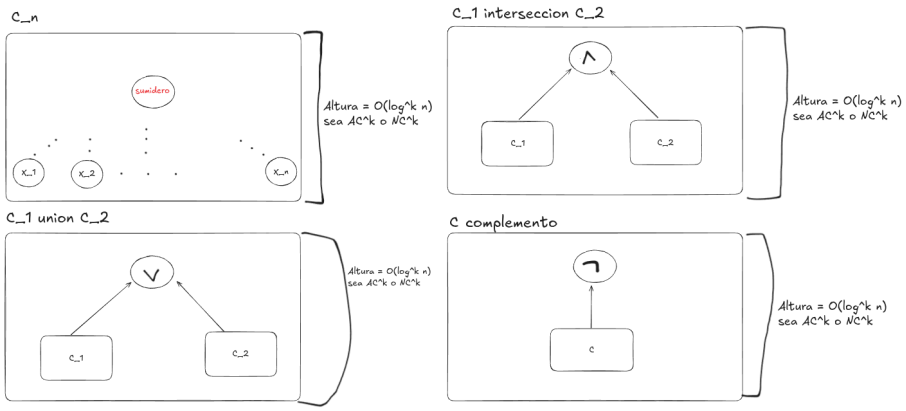

Figure 3: Circuitos

Se ve como se pueden armar circuitos que mantienen la altura O(logˆk n) y son polinomiales.

Ejercicio 15 Probar que NC *subsetneq* **PSPACE. Ayuda: probar que** $NC^k \subseteq SPACE(log^k n)$

Algunas aclaraciones antes de empezar:

- $NC = \bigcup_k NC^k$
- $\bigcup_k SPACE(log^k n) \subsetneq PSPACE$ (por teorema de jerarquia espacial es estricta)

Si logramos probar que $NC^k \subseteq SPACE(log^k n)$ vale que:

$\bigcup_k NC^k \subseteq \bigcup_k SPACE(log^k n)$

$NC \subseteq \bigcup_k SPACE(log^k n) \subsetneq PSPACE \Rightarrow NC \subsetneq PSPACE$

Que es lo que queremos demostrar.

Veamos que vale $NC^k \subseteq SPACE(log^k n)$

Tomo un $\Pi \in NC^k$ cualquiera. Esto significa que hay una familia de circuitos C_n polinomiales y de altura $O(log^k n)$

Definamos la maquina de Turing deterministica M que hace simula a C_n de la siguiente manera:

Recorre recursivamente el circuito usando DFS. Sabemos que en DFS solo necesitamos guardarnos en la memoria los nodos correspondientes a un camino. Como la altura del DAG es $O(log^k n)$ necesitamos esa cantidad de memoria.

En conclusion pordemos reconocer a Π con una Maquina deterministica con espacio $O(log^k n)$ como queriamos ver. Por lo tanto $NC^k \subseteq SPACE(log^k n)$

Queda demostrado que $NC \subsetneq PSPACE$

Ejercicio 16 Sea REG el conjunto de lenguajes regulares. Probar que:

a) $REG \not\subseteq AC^0$

Basta con dar un lenguaje que este en REG y no en AC^0.

Elijo Parity que se que no pertenece a AC^0.

Podemos hacer facilmente un AFD que reconozca Parity, es reconocer las palabras con longitud impar de 1's

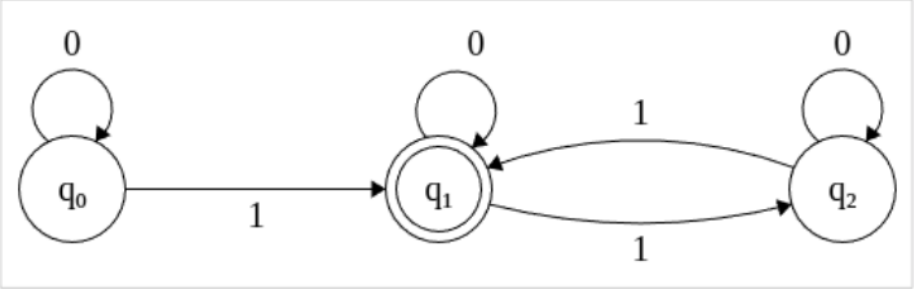

Figure 4: Automata

b) $AC^0 \not\subseteq REG$

Ahora propongo uno que este en AC^0 pero no sea regular.

$\Pi = \{\langle x, n\rangle : x_n = 1 \wedge x_n + 1 = 1\}$

Este lenguaje claramente esta en AC^0. El circuito es simplemente conectar el n-esimo y el (n+1)-esimo bit de x a un and (sumidero)

Ahora hay que ver que el lenguaje no sea regular. Para eso veamos por el contrareciproco de pumping:

obviamente no me voy a poner a hacer pumping.

Ejercicio para el lector!

c) REG $\subseteq NC^1$

Hay que probar que dado Π un lenguaje regular cualquiera, $\Pi \in NC^1$

Hay que ver que pueda reconocer a cualquier lenguaje regular con una familia de circuitos C_n polinomiales de altura O(log n) y fanning arbitrario.

Veamos que puedo simular cualquier AFD: