Práctico 3 Parte 2 Complejidad Computacional

Julián Viera

jviera@fing.edu.uy
julviera44@gmail.com

26 de octubre de 2021

Agenda

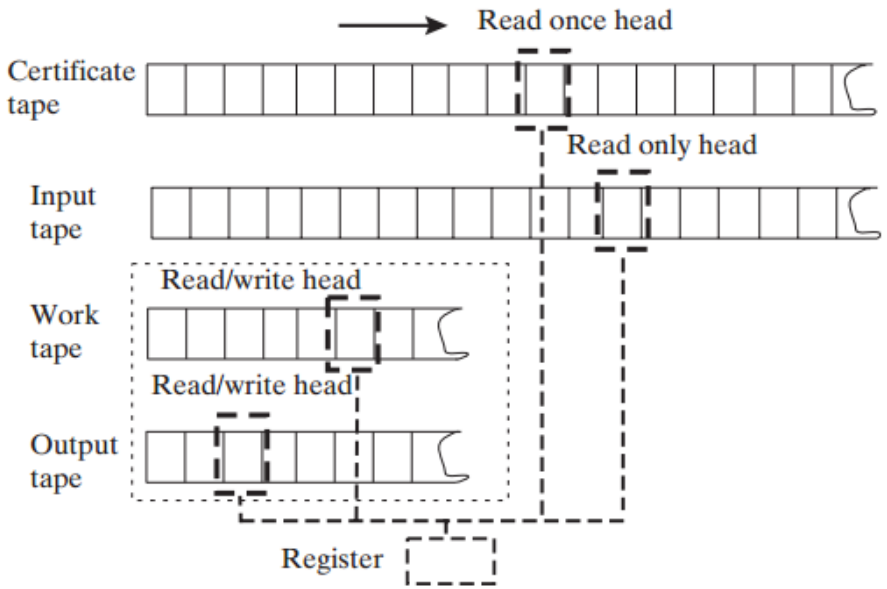
La clase **NL**

$NL = NSPACE(\log n)$

Un lenguaje L pertenece a NL si y solo si existe una MTND M y una constante $c_1 > 0$ tal que M decide L empleando a los sumo $c_1 \log n$ celdas de sus cintas de trabajo a lo largo de toda su ejecución, independientemente de las decisiones no deterministas que se tomen.

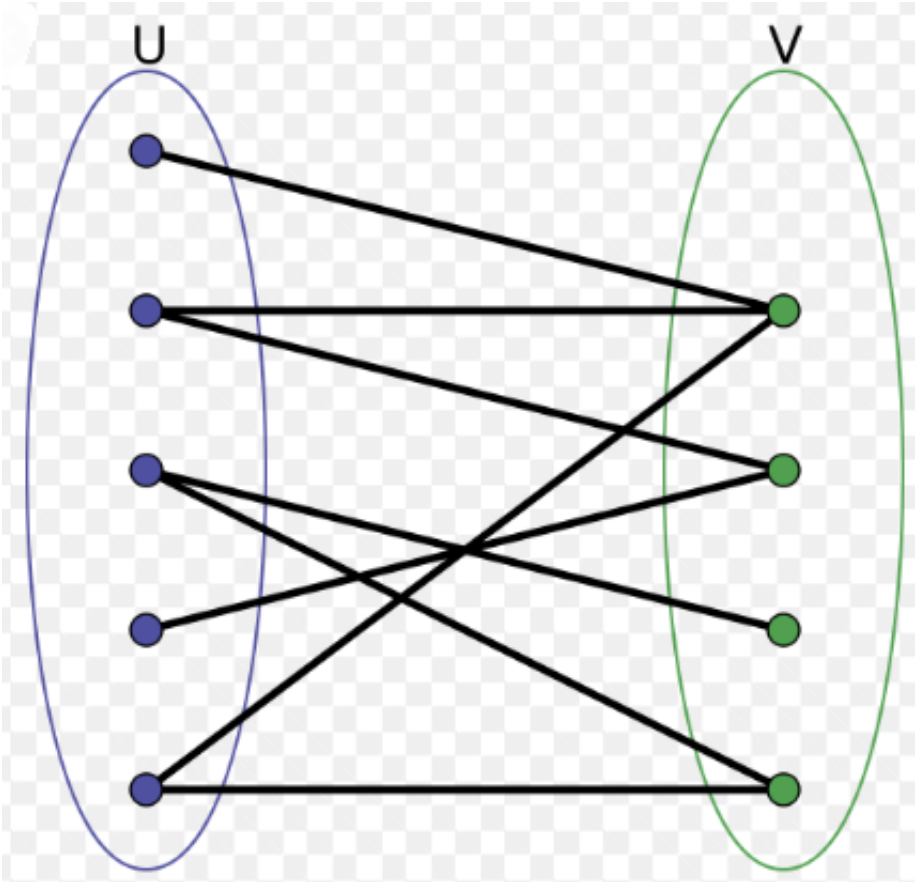
Definición alternativa de la clase **NL**

Se basa en la noción de *certificado* para lenguajes de la clase.
Un lenguaje $L \subseteq \{0,1\}^*$ está en NL si existe un polinomio $p : \mathbb{N} \longrightarrow \mathbb{N}$ y una MT determinista M (el *VERIFICADOR* de L), con una cinta de entrada especial de lectura de una sola vez (read-once tape), de modo que $\forall x \in \{0,1\}^*$:
$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)}$ que verifica $M(x,u) = 1$
En M la entrada x va en su cinta de entrada y el certificado u en la cinta de entrada especial de lectura única.



Grafo bipartito

Un grafo no dirigido es **bipartito** si sus nodos pueden ser divididos en dos conjuntos, de forma que toda arista del grafo vaya de un nodo en un conjunto a un nodo en el otro conjunto.



Propiedad que caracteriza a los grafos bipartitos

Un grafo no dirigido es bipartito si y solo si no contiene ciclos de largo impar.

Se probará aquí solamente el directo.
Ⓗ G es bipartito $\Longrightarrow$ Ⓣ G no tiene ciclos de largo impar
Sean C_1 y C_2 los dos conjuntos de nodos en los que se particiona el grafo bipartito G.
Supongamos que G tiene un ciclo y elijamos un nodo cualquiera $u \in C_1$ de ese ciclo. Las dos aristas del ciclo que inciden en u van a tener su otro extremo necesariamente en C_2, por la definición de grafo bipartito. Si recorremos el ciclo a partir de u, el primer nodo visitado está en C_2, el segundo está en C_1, el tercero en C_2 y así sucesivamente. Observamos que los caminos de largo impar a partir de u terminan en C_2, y los de largo par en $C_1 \Longrightarrow$ el ciclo de G que empieza y termina en u es de largo par $\Longrightarrow$ G no tiene ciclos de largo impar.

Planteo del problema

Sea el lenguaje $BIPARTITO = \{< G > / G \text{ es un grafo bipartito } \}$

Probar que $BIPARTITO \in NL$.

Idea para la demostración

Consideramos el lenguaje $\overline{BIPARTITO}$, complemento del lenguaje $BIPARTITO$.
$\overline{BIPARTITO} = \{< G > / G$ no es un grafo bipartito $\}$

Por la propiedad mostrada que caracteriza a los grafos bipartitos, podemos dar una definición alternativa de $\overline{BIPARTITO}$ como
$\overline{BIPARTITO} = \{< G > / G$ es un grafo que tiene al menos un ciclo de largo impar$\}$

La idea es probar que $\overline{BIPARTITO} \in NL$. ¿Por qué?
Porque para la clase NL se verifica el hecho de que NL = coNL. De esta forma, si probamos que $\overline{BIPARTITO} \in NL$ estamos probando que $BIPARTITO \in NL$.

Un algoritmo no determinista para $\overline{BIPARTITO}$

La idea del algoritmo es buscar en forma no determinista todos los posibles ciclos que pasen por cada uno de los vértices de G, sabiendo que a los sumo cada ciclo es descubierto en n pasos.

El algoritmo mantiene información acerca de la paridad del largo del camino en cada paso. Si encuentra un ciclo de largo impar acepta y en caso contrario rechaza.

Pseudocódigo del algoritmo no determinista

Dado G,
Elegir en forma no determinista un vértice u de G
$v = u$
$d = 1$
Si $grado(v) = 0$ o $grado(v) = 1$
    PARAR sin aceptar
Fin Si
Mientras $d \leq n$ hacer
    Elegir en forma no determinista un vértice w adyacente a v
    Si $w = u$ y d es impar
      ACEPTAR
    sino
      $v = w$
    Fin Si
    $d = d + 1$
Fin Mientras
PARAR sin aceptar

Espacio utilizado por el algoritmo

Sea n el número de nodos de G. El análisis de espacio requerido por el algoritmo es como sigue:

- vértice inicial del ciclo u : $\log n$
- contador de pasos del algoritmo d : $\log n$
- vértice actual del algoritmo v : $\log n$
- vértice adyacente w : $\log n$

$\Longrightarrow$ el espacio necesario es $\mathcal{O}(\log n)$.

$\Longrightarrow \overline{BIPARTITO} \in NL \xRightarrow{\text{NL=coNL}} BIPARTITO \in NL.$

Demostración alternativa de $\overline{BIPARTITO} \in NL$

Un certificado u para $\overline{BIPARTITO}$ consiste en una secuencia de k vértices $v_1 v_2 ... v_k$ que forman el ciclo de largo impar. El algoritmo determinista verificador tiene que probar, usando espacio $\mathcal{O}(\log n)$ del WT y recorriendo una única vez el certificado que:

- Existen aristas (v_i, v_{i+1}) $i = 1...k-1$ y arista (v_k, v_1)
- k es impar (ciclo de largo impar)

Función ímplicitamente computable en espacio logarítmico

Una función $f : \{0,1\}^* \rightarrow \{0,1\}^*$ es ímplicitamente computable en espacio logarítmico (ILC) si:

1. f está acotada polinomialmente ($f(x) \leq |x|^c \ \forall x \in \{0,1\}^*$)
2. Los lenguajes $L_f = \{< x, i > / f(x)_i = 1\}$ y $L'_f = \{< x, i > / i \leq |f(x)|\}$ están en la clase L

Función write-once logspace computable

Una función $f : \{0,1\}^* \to \{0,1\}^*$ se dice que es de escritura única computable en espacio logarítmico (write-once logspace computable), si es computable por una MT determinista M en espacio $\mathcal{O}(\log n)$ que tiene su cinta de salida del tipo "write-once", de modo que en cada paso de su ejecución el cabezal de la cinta de salida o bien se queda en la misma posición en que estaba o escribe un símbolo y se mueve a la posición de la derecha en la cinta.

Las celdas usadas de la cinta de salida no computan para el cálculo del espacio de ejecución de M.

Planteo del problema

Probar que f es de escritura única computable en espacio logarítmico si y solo si f es implícitamente computable en espacio logarítmico.

Prueba del directo

Ⓗ f es de escritura única computable en espacio logarítmico
Ⓣ f es implícitamente computable en espacio logarítmico
Por Ⓗ, existe una MTD M que dado $x \in \{0,1\}^*$ con $|x| = n$ calcula f en espacio $c1 \log n$ de su WT.
Mostraremos primero que $\exists c \in N,\ c > 0\ /\ |f(x)| \leq |x|^c = n^c$.
Asumiremos que M tiene una única cinta de trabajo y calcularemos las cantidad de configuraciones distintas de M. Estas se pueden obtener a partir de que existen:

- q estados diferentes de M
- n posiciones posibles para el cabezal del input
- $2^{c_1 \log n}$ configuraciones distintas de la cinta de trabajo (WT)
- $c1 \log n$ posiciones posibles para el cabezal del WT

$\Longrightarrow$ hay a lo sumo $c_1 q n^{c_1+1} \log n$ configuraciones distintas para M.
$\Longrightarrow \exists c\ /\ c_1 q n^{c_1+1} \log n \leq n^c$
$(c \geq \lceil c_1 + 1 + \frac{\log(c_1 q \log n)}{\log n} \rceil)$

Prueba del directo

Como sabemos que el tiempo de ejecución $T(n)$ de M está acotado por el número máximo de configuraciónes de M, y a su vez el espacio de la salida calculada $|f(x)|$ está acotado por el tiempo de ejecución de M,
$\Longrightarrow |f(x)| \leq T(n) \leq n^c$.

Para determinar $f(x)_i$, el bit i-ésimo de f, se simula M sin la cinta de salida "write-once" y se mantiene un contador para i que se incrementa cada vez que M escribe en el WO tape, hasta que dicho contador llegue al valor i.
El espacio que ocupa dicho contador es $\mathcal{O}(\log n)$ ya que $|f(x)|$ es polinomial en n.
Como M usa espacio $\mathcal{O}(\log n) \Longrightarrow f(x)_i$ se calcula en espacio $\mathcal{O}(\log n) \Longrightarrow L_f = \{< x, i > / f(x)_i = 1\} \in L$.

Prueba del directo

A su vez para cada par $< x, i >$ se decide si $i \leq |f(x)|$ según M pare o no antes de que el contador sobrepase el valor de i.
Como M ejecuta en espacio $\mathcal{O}(\log n)$ y el contador para i también usa espacio $\mathcal{O}(\log n) \Longrightarrow L'_f = \{< x, i > / i \leq |f(x)|\} \in L$.

Prueba del recíproco

Ⓗ f es implícitamente computable en espacio logarítmico
Ⓣ f es de escritura única computable en espacio logarítmico
Sea M la MT que computa L_f, es decir que dados $< x, i >$ calcula $f(x)_i$, y sea M' la MT que computa L'_f, es decir que dados $< x, i >$ decide si $i \leq |f(x)|$.
Consideramos una nueva MT M^* con cinta de salida del tipo WO, que mantiene un contador i.
En cada paso de su ejecución, M^*:

1. Chequea que $i \leq |f(x)|$ usando la MT M'. Si $i > |f(x)|$, M^* para.
2. Invoca a M con el valor del contador i y escribe el valor $f(x)_i$ en la cinta WO de salida

Es decir que M^* calcula y escribe en su cinta de salida WO la secuencia $M(x,1), M(x,2), ..., M(x,|f(x)|)$.

Prueba del recíproco

Tanto el chequeo de i como el cálculo de $f(x)_i$ emplean espacio $\mathcal{O}(\log n)$ pues por Ⓗ f es ILC.
A su vez el contador i ocupa espacio $\mathcal{O}(\log n)$ pues por Ⓗ $|f(x)| \leq |x|^c$ y M^* para cuando $i > |f(x)|$.
$\Longrightarrow$ M^* computa f en espacio logarítmico de su cinta de trabajo.

Reducción de espacio logarítmico y NL-completitud

Un lenguaje $B \subseteq \{0,1\}^*$ se dice que es reducible en espacio logarítmico a un lenguaje $C \subseteq \{0,1\}^*$, denotado como $B \leq_l C$, si existe una función $f : \{0,1\}^* \to \{0,1\}^*$ implícitamente computable en espacio logarítmico tal que:
$\forall x \in \{0,1\}^*, x \in B \Leftrightarrow f(x) \in C$

Se dice que C es NL-completo si $C \in NL$ y para todo $B \in NL$, $B \leq_l C$

Planteo del problema

Sea el lenguaje *STRONG-CONN*$= \{< G > / G$ es un grafo dirigido fuertemente conexo$\}$

Probar que *STRONG-CONN* es NL-completo.

Hay que probar entonces que:

1. *STRONG-CONN* $\in$ *NL*
2. *STRONG-CONN* es NL-hard

Un algoritmo no determinista para *STRONG-CONN*

Para probar que $STRONG\text{-}CONN \in NL$, la idea es construir un algoritmo no determinista que decida si hay camino dirigido entre todo par de vértices de G.
El algoritmo prueba para cada vértice s de G si hay camino a todo otro vértice t de G, eligiendo en cada paso un vértice adyacente en forma no determinista, y considerando que si hay camino éste debe ser de largo a lo sumo $n-1$, siendo n el número de nodos del grafo.
Si para algún par $< s, t >$ no se encuentra camino dirigido de s a t, el algoritmo ND para sin aceptar.
Si para todo par $< s, t >$ se encuentra un camino dirigido de s a t, el algoritmo ND acepta.

Pseudocódigo del algoritmo ND para *STRONG-CONN*

Dado G,
Calcular en número n de vértices de G
Para cada vértice s de G de 1 a n
    Para cada vértice t de G de 1 a n, con $t \neq s$
        $v = s$
        $d = 1$
        Mientras $d < n$ hacer
            Si grado(v) > 0 elegir en forma no determinista w adyacente a v
            sino Parar sin aceptar
            Si $w = t$ procesar siguiente vértice t
            $v = w$
            $d = d + 1$
        Fin Mientras
        Parar sin aceptar {no se encontró camino de s a t}
    Fin Para {t}
Fin Para {s}
Aceptar {hay camino dirigido entre todo par de vértices}

Análisis del espacio utilizado por el algoritmo

Sea n el número de nodos de G. El análisis de espacio requerido por el algoritmo es como sigue:

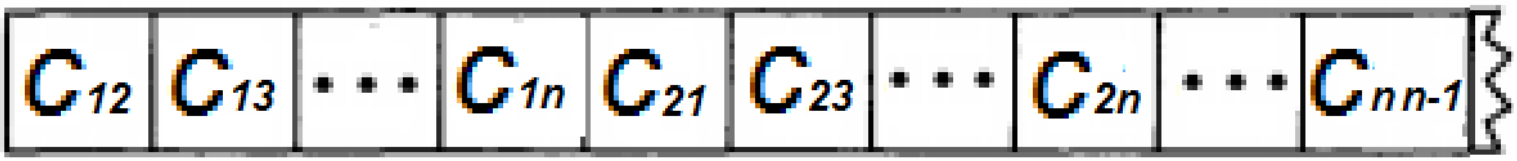
- número de nodos n y contador d: $\log n$
- variables origen s y destino t : $\log n$
- vértice actual v y adyacente w : $\log n$

$\Longrightarrow$ el espacio necesario es $\mathcal{O}(\log n)$.

$\Longrightarrow$ $STRONG\text{-}CONN \in NL$

Prueba alternativa de que $STRONG\text{-}CONN \in NL$

Un certificado u del tipo *read-once* para $STRONG\text{-}CONN$ consiste en una secuencia de caminos dirigidos C_{st} entre los vértices s y t, para todo par de vértices s y t en G.



Hay $n(n-1)$ caminos en el certificado.
Cada camino C_{st} es una secuencia de vértices que indican el camino dirigido entre s y t: $v_1, v_2 \ldots v_k$.
Para cada camino C_{st} del certificado hay que verificar que:

- $v_1 = s$ y $v_k = t$
- Existen aristas (v_i, v_{i+1}) $i = 1 \ldots k-1$

Análisis del espacio utilizado por el verificador

Sea n el número de nodos de G. El análisis de espacio requerido por el verificador del certificado es como sigue:

- contador del certificado actual que se está verificando: $\mathcal{O}(\log n^2) = \mathcal{O}(2 \log n) = \mathcal{O}(\log n)$
- variables v_1 y v_k : $\log n$
- identificadores de origen s y destino t : $\log n$

$\Longrightarrow$ el verificador ejecuta en espacio $\mathcal{O}(\log n)$.

$\Longrightarrow$ $\mathit{STRONG\text{-}CONN} \in \mathit{NL}$

Prueba de que *STRONG-CONN* es NL-hard

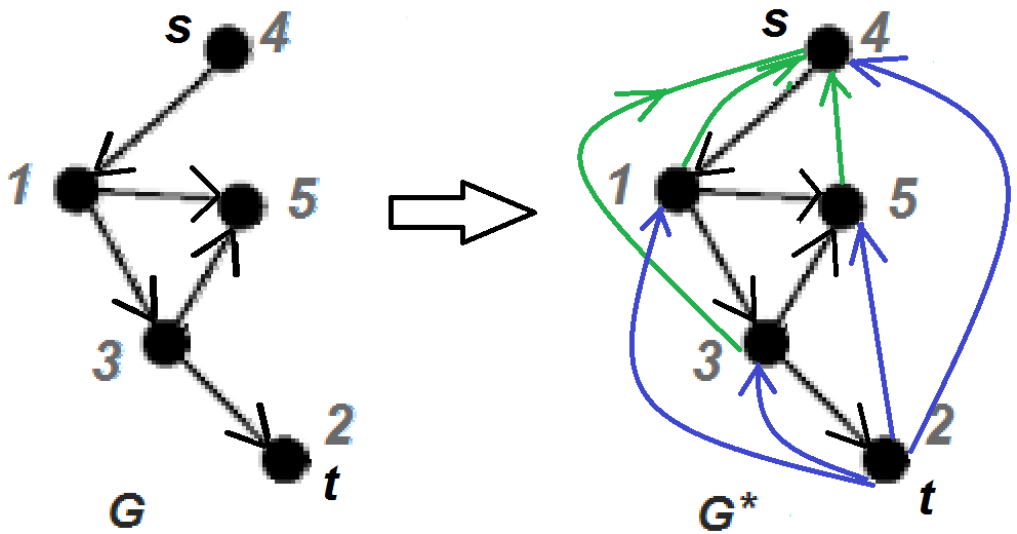
Se probará que *STRONG-CONN* es NL-hard haciendo una reducción del problema *PATH* a *STRONG-CONN*, es decir probando que $PATH \leq_l STRONG\text{-}CONN$.

Como sabemos que *PATH* es NL-completo, si se prueba que *PATH* es reducible a *STRONG-CONN* y por la propiedad transitiva de la reducción de espacio logarítmico, queda probado que *STRONG-CONN* es NL-hard.

Reducción de *PATH* a *STRONG-CONN*

Sea $< G, s, t >$ una instancia de *PATH*. La reducción propuesta mapea $< G, s, t >$ en $< G^* >$, en donde G^* se obtiene a partir G de la siguiente forma:

1. G^* contiene todos los nodos y todas las aristas de G
2. Se incluye en G^* una arista (t, u) si dicha arista no existe en G, para todo vértice $u \neq t$ de G
3. Se incluye en G^* una arista (v, s) si dicha arista no existe en G, para todo vértice $v \neq s$, $v \neq t$ de G



Prueba de que la función de reduccion es implicitamente computable en espacio logarítmico (ILC)

Sea $x =< G, s, t >$ una instancia de $PATH$, y sea $f(x) = f(< G, s, t >) =< G^* >$ su correspondiente en $STRONG\text{-}CONN$ según la reducción propuesta.
Si representamos G por su matriz de adyacencia A_{nxn}, la matriz de adyacencia de G^* es A^*_{nxn}, también de n^2 elementos.
Esto implica que $|f(x)| = n^2$, y por lo tanto es polinominal en el tamaño de la entrada (es lineal).
Para calcular cada bit de la matriz de adyacencia A^* de $< G^* >$ es suficiente con espacio logarítmico, ya que lo que hay que mantener es espacio para los índices i y j de la matriz lo cual requiere espacio $\mathcal{O}(\log n)$.
$\xrightarrow{\text{definición de función ILC}}$ f es ILC.

Correctitud de la reducción propuesta

Probaremos que hay camino dirigido de s a t en $G \Leftrightarrow G^*$ es fuertemente conexo.

$(\Longrightarrow)$

Ⓗ $< G, s, t >$ es una instancia SI de *PATH*

Ⓣ $< G^* >$ es una instancia SI de *STRONG-CONN*

Por Ⓗ sabemos que existe un camino dirigido C entre s y t en G, y por lo tanto por la definición de la reducción dicho camino se mantiene en G^*.

Sean u y v dos vértices cualquiera de G^*. Por la definición de la reducción sabemos que en G^* hay arista dirigida de u a s y de t a v.

Un camino dirigido de u a v en G^* es el siguiente:

$u \rightarrow s \xrightarrow{C} t \rightarrow v$

Como esto se cumple para cualquier par de nodos u y v en G^*,

$\Longrightarrow G^*$ es fuertemente conexo.

Correctitud de la reducción propuesta

($\Longleftarrow$)
Ⓗ $< G^* >$ es una instancia SI de *STRONG-CONN*
Ⓣ $< G, s, t >$ es una instancia SI de *PATH*

Como por Ⓗ G^* es fuertemente conexo, hay un camino dirigido C entre s y t en G^*.
Por la construcción de G^*, ese camino C debe existir también en G, pues no contiene aristas con origen en t ni aristas con destino en s, que son las que diferencian G de G^*.
$\Longrightarrow$ hay camino dirigido de s a t en G.
$\Longrightarrow$ *PATH* $\leq_l$ *STRONG-CONN* $\Longrightarrow$ *STRONG-CONN* es NL-hard

Como probamos que *STRONG-CONN* $\in$ *NL* y que *STRONG-CONN* es NL-hard $\Longrightarrow$ *STRONG-CONN* es NL-completo