UNIVERSIDAD SIMON BOLIVAR **24 de febrero de 2010**
DEPARTAMENTO DE COMPUTACION
MAESTRIA EN CIENCIAS DE LA COMPUTACION **Ing. Angarita Arocha Rafael**
ASIGNATURA: CI-7541 - Teoria de la computación. **Lic. Claro Mosqueda Alejandro**

TAREA N^o 2

Ejercicio 5.1: Muestre que EQ_{CFG} no es decidible.

Solución:

Demostración. Si se parte de la suposición de que existe una máquina de Turing R que decide EQ_{CFG}, entonces es posible construir una máquina de Turing S,

$S =$ "En entrada $\langle G_1 \rangle$:

1. Construir un CFG G_{ALL}, donde $L(G_{\text{ALL}}) = \Sigma^*$.
2. Correr R en $\langle G_1, G_{\text{ALL}} \rangle$.
3. Si R acepta, ***aceptar***.
4. De lo contrario, ***rechazar***.",

que decide $ALL_{\text{CFG}} = \{\langle G \rangle \,|G$ es un CFG y $L(G) = \Sigma^*\}$. Evidentemente, si R decide a EQ_{CFG}, entonces S decide a ALL_{CFG}. Sin embargo, es conocido que ALL_{CFG} no es decidible [1, Teorema 5.13] y por lo tanto la suposición de que existe R es falsa. Con lo cual queda demostrado que EQ_{CFG} no es decidible.

□

Ejercicio 5.3: Encuentre una correspondencia en la siguiente instancia del problema de correspondencia de Post (PCP):

$$\left\{ \left[\frac{\text{ab}}{\text{abab}}\right], \left[\frac{\text{b}}{\text{a}}\right], \left[\frac{\text{aba}}{\text{b}}\right], \left[\frac{\text{aa}}{\text{a}}\right] \right\}$$

Solución:

La siguiente secuencia es una correspondencia para esta instancia de PCP:

$$\left[\frac{\text{aa}}{\text{a}}\right] \left[\frac{\text{aa}}{\text{a}}\right] \left[\frac{\text{b}}{\text{a}}\right] \left[\frac{\text{ab}}{\text{abab}}\right]$$

Tanto en la parte superior como inferior se observa la cadena **aaaabab**.

Ejercicio 5.13: Un ***estado inútil*** en una máquina de Turing es aquel que nunca es alcanzado sin importar la entrada proporcionada. Considere el problema de determinar cuando una máquina de Turing tiene algún estado inútil. Formule este problema como un lenguaje y muestre que no es decidible.

Solución:

Demostración. Sea $USELESS_{\text{TM}} = \{\langle M, q\rangle \mid M \text{ es una } \mathsf{TM} \text{ con el estado inútil } q\}$. Si existe una TM R que decide $USELESS_{\text{TM}}$, entonces es posible construir una TM S,

$S =$ "En entrada $\langle M\rangle$:

1. Simular R en $\langle M, q_a\rangle$. Donde q_a es el estado de aceptación de M.
2. Si R acepta, ***aceptar***.
3. Si R rechaza, ***rechazar***."

que decide $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ es una } \mathsf{TM} \text{ y } L(M) = \emptyset\}$ ya que R decide si el estado de aceptación de M es inútil y en consecuencia $L(M) = \emptyset$. Sin embargo, es conocido que E_{TM} no es decidible [1, Teorema 5.12] y en consecuencia la suposición de que existe R es falsa. Con lo cual queda demostrado por contradicción que $USELESS_{\text{TM}}$ no es decidible.

□

Ejercicio 5.20: Pruebe que existe un subconjunto no decidible de $\{1\}^*$.

Solución:

Demostración. Considere el lenguaje

$$U_{A_{TM}} = \{1^k \mid 1^k \text{ es la representación unitaria de } \langle G, w\rangle \in A_{TM}\}.$$

Claramente toda cadena unitaria $1^k \in U_{A_{TM}}$ pertenece también a $\{1\}^*$ (i.e., $U_{A_{TM}} \subseteq \{1\}^*$). Además, por construcción de $U_{A_{TM}}$ existe la reducción $f : \Sigma^* \longrightarrow \Sigma^*$, donde para cada $\langle G, w\rangle$,

$$\langle G, w\rangle \in A_{TM} \iff f(\langle G, w\rangle) \in U_{A_{TM}},$$

y por lo tanto $A_{TM} \leq_m U_{A_{TM}}$ [1, Definición 5.20]. Dado esto se concluye que $U_{A_{TM}}$ es un subconjunto no decidible de $\{1\}^*$ [1, Corolario 5.23].

□

Ejercicio 7.11: Dos grafos G y H son ***isomorfos*** si los nodos de G pueden ser reordenados de tal manera que esté sea idéntico a H. Sea $ISO = \{\langle G, H\rangle \mid G \text{ y } H \text{ son grafos isomorfos}\}$. Muestre que $ISO \in \mathbf{NP}$.

Solución:

Demostración. Sean $G = (V_G, E_G)$ y $H = (V_H, E_H)$ dos grafos no dirigidos. Si un mapa $f : V_G \to V_H$ es un mapa biyectivo tal que dos vértices u y v, pertenecientes a V_G, son adyacentes sii $f(u)$ y $f(v)$ son vértices adyacentes pertenecientes a V_H, entonces se dice que G y H son grafos isomorfos ($G \simeq H$). Es posible construir una máquina de Turing no-determinista N,

$N =$ "En entrada $\langle G, H\rangle$, donde G y H son grafos no dirigidos:

1. Si G y H no tienen la misma cardinalidad, ***rechazar***.
2. De manera no-determinista construir una permutación f de todos vértices de G.
3. Verificar que para cada arista $(u, v) \in E_G$ existe la arista $(f(u), f(v))$ en E_H.
4. Si la verificación es negativa, ***rechazar***.
5. De lo contrario, ***aceptar***."

que decide ISO. El paso 1 es claramente computable en tiempo polinomial; una permutación de n valores es posible construirla en tiempo polinomial, por lo tanto el paso 2 también es computable en tiempo polinomial si se construyen todas las permutaciones de forma no-determinista; y finalmente el paso 3 son n búsquedas que son claramente posibles de computar en tiempo polinomial. De este razonamiento queda evidenciado que N decide ISO en tiempo polinomial y por lo tanto $ISO \in \mathbf{NP}$. □

Ejercicio 7.13: Una ***permutación*** en el conjunto $S = \{1, \cdots, k\}$ es una función biyectiva de esté conjunto en si mismo. Si p es una permutación, p^t es la composición de p consigo mismo t veces. Sea

$$PERM\text{-}POWER = \{\, \langle p, q, t\rangle \mid p = q^t \text{ donde } p \text{ y } q \text{ son permutaciones en } S = \{1, \cdots, k\} \text{ y } t \text{ es un entero binario} \,\}$$

Muestre que $PERM\text{-}POWER \in \mathbf{P}$. (Observe que el tiempo de ejecución del algoritmo más obvio no es polinomial. Pista: Intente primero con t igual a potencias de 2).

Solución:

Como lo indica el enunciado, una ***permutación*** en el conjunto $S = \{1, \cdots, k\}$ es una función biyectiva de esté conjunto en si mismo. Por ejemplo considérese el conjunto $S = \{1, 2, 3\}$, una posible permutación p de la secuencia $(1, 2, 3)$ es la secuencia $(2, 3, 1)$. Es posible computar tanto la permutación como la composición de 2 permutaciones en tiempo polinomial en la cardinalidad del conjunto S. Pero, por otro lado, si se desean componer p consigo mismo t veces (un ejemplo se muestra en el cuadro 1), a primera vista, parece necesario realizar sucesivamente la permutación justamente t veces. Si t esta representado en base 2, el tamaño de la representación de t es $n = O(\log_2 t)$ y por lo tanto el tiempo de ejecución de dicha composición de permutaciones es exponencial $O(2^n)$.

s_i	$p(s_i)$	$p^2(s_i)$	$p^3(s_i)$
1	2	3	1
2	3	1	2
3	1	2	3

Cuadro 1: Ejemplo de tres permutaciones sobre el conjunto $S = \{1, 2, 3\}$. La permutación p^3 no altera el orden de la secuencia de entrada (Esta permutación es llamada ***permutación identidad***).

La demostración que sigue a continuación se basa en otro algoritmo para computar la composición de p consigo mismo t veces en tiempo polinomial, aprovechando el hecho de que la representación de un número en base 2 se puede construir a partir de la combinación lineal de las potencias de 2.

Demostración. Para demostrar que $PERM\text{-}POWER \in \mathbf{P}$ basta con mostrar que existe un algoritmo que compute q^t y verifique que $p = q^t$ de forma determinista en tiempo polinomial. A continuación se muestra dicho algoritmo E:

$E =$ "En entrada $\langle p, q, t\rangle$, donde p y q son permutaciones y t un número binario:

1. $g \leftarrow$ permutación identidad.
2. Para cada bit en t desde el menos significativo hasta el más significativo, realizar:
3. Si el bit es igual a 1, entonces computar la composición $g \leftarrow q \cdot g$.
4. Computar la composición $q \leftarrow q^2$.
5. Si g y p son iguales, ***aceptar***.
6. De lo contrario, ***rechazar***."

No se dan detalles de la verificación $g = p$ debido a que es claramente computable en tiempo polinomial en la cardinalidad del conjunto S. Los pasos del 2 al 4 se lleva a cabo $n = O(\log_2 t)$ veces, por lo tanto es computable en tiempo polinomial en el tamaño de la entrada. De esta manera queda demostrado que $PERM\text{-}POWER \in \mathbf{P}$.

□

Ejercicio 7.20: Sea G un grafo no dirigido. Sean también

$$SPATH = \{\, \langle G, a, b, k\rangle \mid \begin{array}{l} G \text{ contiene un camino simple de} \\ \text{longitud a lo sumo } k \text{ desde } a \text{ hasta } b \end{array} \}$$

y

$$LPATH = \{\, \langle G, a, b, k\rangle \mid \begin{array}{l} G \text{ contiene un camino simple de} \\ \text{longitud al menos } k \text{ desde } a \text{ hasta } b \end{array} \}$$

a) Muestre que $SPATH \in \mathbf{P}$.

b) Muestre que $LPATH$ es **NP**-completo. Se puede asumir la **NP**-completitud de $UHAMPATH$, el problema del camino Hamilotoniano para grafos no dirigidos.

Solución:

a) Demostración. La siguiente modificación del algoritmo M para $PATH$ [1, Teorema 7.14]:

M = "En entrada $\langle G, a, b, k\rangle$, donde G es un grafo no dirigido con vértices a y b:

1. Marcar en el vértice a con el número 0.
2. Repetir lo que sigue hasta que no sean marcados más vértices:
3. Recorrer todas las aristas de G. Si es encontrada una arista (u, v) donde u es un vértice marcado con el número i y v es un vértice no marcado, entonces:
4. Marcar el vértice v con el número $i + 1$.
5. Si el número con el cual se marco v es mayor que k, ***rechazar***.
6. Si b esta marcado, ***aceptar***. De lo contrario, ***rechazar***."

decide $SPATH$ claramente en tiempo polinomial. □

b) Demostración. Partiendo de la definición de **NP**-completo [1, Definicion 7.34], el lenguaje $LPATH$ es **NP**-completo si:

1.- $LPATH \in \mathbf{NP}$. Para demostrar esto, basta con observar que es posible construir un verificador V en tiempo polinomial para $LPATH$.

V = "En entrada $\langle \langle G, a, b, k\rangle , C\rangle$, donde C es un camino simple:

1. Si la longitud de C es menor que k, ***rechazar***.
2. Si los extremos de C son distintos de a y b, ***rechazar***.
3. Recorre todas las aristas $(u, v) \in C$, si alguna de estas arista no pertenece al conjunto de aristas de G, ***rechazar***.
4. De lo contrario, ***aceptar***."

2.- Todo A en **NP** es reducible en tiempo polinomial a $LPATH$. Para demostrar esto, es suficiente con mostrar que existe una función computable en tiempo polinomial $f(\langle G, a, b\rangle) \equiv \langle G, a, b, k\rangle$ tal que $UHAMPATH \leq_P LPATH$. La siguiente TM M calcula la reducción en tiempo polinomial,

M = "En entrada $\langle G, a, b\rangle$, donde G es un grafo no dirigido con vértices a y b:

1. Recorre la cinta contando los vértices de G.
2. Imprime al final de la cinta el número total de vértices k.
3. ***Aceptar***."

El contenido de la cinta al finalizar M es una posible codificación de $\langle G, a, b, k\rangle$. Si $\langle G, a, b\rangle \in UHAMPATH$, entonces G tiene un camino Hamiltoniano de longitud k que contiene a los vértices a y b, por lo tanto $\langle G, a, b, k\rangle \in LPATH$. Ahora en sentido contrario, si $\langle G, a, b, k\rangle \in LPATH$, entonces G contiene un camino simple de longitud k desde a hasta b. Pero G tiene cardinalidad k, así que en este camino es hamiltoniano, $\langle G, a, b\rangle \in UHAMPATH$.

□

Ejercicio 7.36: Muestre que si **P** = **NP**, existe un algoritmo en tiempo polinomial que produce una asignación satisfactoria cuando se da una fórmula booleana satisfacible. (Nota: El algoritmo que se pide calcula una función, pero **NP** contiene lenguajes, no funciones. La suposición **P** = **NP** implica que SAT esta en **P**, por lo tanto probar la satisfactibilidad es computable en tiempo polinomial. Pero la suposición no dice como se realiza la prueba, y la prueba puede que no revele asignaciones satisfactorias. Se debe mostrar que es posible encontrarlas de cualquier manera. Pista: Use la prueba de satisfactibilidad repetidamente para encontrar la asignación bit-por-bit.)

Solución:

Considérese la función Booleana $f : B^k \to B$ que es satisfacible por cierta cadena binaria $x^{(s)} = x_1 x_2 \dots x_k$ (i.e., k variables). El siguiente algoritmo S,

$S =$ "En entrada $\langle f \rangle$, donde f es una función Booleana $f : B^k \to B$ satisfacible:

1. Construir una cadena binaria x de longitud k.
2. $\langle g \rangle \leftarrow \langle f \rangle$
3. Para cada bit x_i en x, realizar:
4. $\quad \langle g_0 \rangle \leftarrow$ Sustituir la i-ésima variable por 0 en $\langle g \rangle$.
5. $\quad \langle g_1 \rangle \leftarrow$ Sustituir la i-ésima variable por 1 en $\langle g \rangle$.
6. $\quad$ Si g_0 es satisfacible: (Observe que el número de variables en g_0 es uno menos que en g)
7. $\quad\quad x_i \leftarrow 0$.
8. $\quad\quad \langle g \rangle \leftarrow \langle g_0 \rangle$.
9. $\quad$ Si no, como g_1 debe ser satisfacible:
10. $\quad\quad x_i \leftarrow 1$.
11. $\quad\quad \langle g \rangle \leftarrow \langle g_1 \rangle$.
12. Imprimir en la cinta x.
13. ***aceptar***."

claramente computa en tiempo polinomial una asignación $x^{(s)}$ que satisface f si la satisfactibilidad de f, paso 6, es computable en tiempo polinomial como supone el enunciado.

Ejercicio 7.44: Dos fórmulas booleanas son ***equivalentes*** si tienen el mismo conjunto de variables y son verdad para el mismo conjunto de asignaciones para esas variables (i.e., describen la misma función booleana). Una fórmula booleana es ***minimal*** si no tiene una fórmula booleana más pequeña equivalente. Sea $MIN\text{-}FORMULA$ la colección de fórmulas booleanas minimales. Muestre que, si **P** = **NP**, entonces $MIN\text{-}FORMULA \in$ **P**.

Solución:

Demostración. Sea

$$MIN\text{-}FORMULA = \{\langle \phi \rangle \,|\, \phi \text{ es una función booleana } \textbf{\textit{minimal}}\}.$$

Demostraremos que $\overline{MIN\text{-}FORMULA} \in \mathbf{NP}$, construyendo un verificador V en tiempo polinomial de esté lenguaje.

$V =$ "En entrada $\langle \phi, c \rangle$, donde ϕ y c (certificado) son formulas booleanas:

1. Probar que $|c| < |\phi|$.
2. Probar que $c \equiv \phi$.
3. Si ambas pruebas pasan, ***aceptar***. Si no, ***rechazar***."

La existencia de este verificador implica que $\overline{MIN\text{-}FORMULA} \in \mathbf{NP}$. Por lo tanto,

$$MIN\text{-}FORMULA \in \mathbf{coNP}.$$

Por otro lado, es posible demostrar que si $\mathbf{P} = \mathbf{NP}$, entonces $\mathbf{NP} = \mathbf{coNP}$ [2, Lemma 2]. En consecuencia,

$$MIN\text{-}FORMULA \in \mathbf{P}$$

□

Referencias

[1] Sipser, Michael. *Introduction to the Theory of Computation.* International Thomson Publishing, 2nd Edition, 2006.

[2] Kolokolova, Antonina. *Lecture 9: NP as games, co-NP, proof complexity.* Department of Computer Science, Memorial University, 2007. Tomado el 23 de febrero de 2010 de: http://www.cs.mun.ca/ kol/courses/6743-f07/lec9.pdf