Complejidad Computacional
Tarea 2

Karla Adriana Esquivel Ramírez
Andrea Itzel González Vargas
Luis Pablo Mayo Vega
Carlos Gerardo Acosta Hernández

Entrega: 14/03/17
Facultad de Ciencias UNAM

Ejercicios

1. Supón que L_1, $L_2 \in$ NP. ¿Qué pasa con $L_1 \cup L_2$ y $L_1 \cap L_2$?

Primero vemos si $L_1 \cup L_2 \in$ NP:
Sea $V_1(x, c)$ un algoritmo verificador para L_1 que para una cadena x y un posible certificado c, entonces si $V_1(x, c) = 1$ significa que c verifica que $x \in L_1$ y si $V_1(x, c) = 0$ sucede lo contrario. Análogamente $V_2(x, c)$ es un verificador para L_2. Ya que L_1, $L_2 \in$ NP, sabemos que sus respectivos verificadores deben de correr en tiempo polinomial $O(|x|^k)$ para alguna constante k.
Sea $L_3 = L_1 \cup L_2$, si podemos construir un verificador V_3 para L_3 que corra en tiempo polinomial, podremos decir entonces que $L_3 \in NP$.
Ya que en L_3 estamos uniendo L_1 y L_2, tenemos que para un certificado c para L_3 sucede que $V_1(x, c) = 1$ o $V_2(x, c) = 1$, por lo tanto podemos definir a V_3 de la siguiente manera: $V_3(x, c) = V_1(x, c) \vee V_2(x, c)$. Para verificar que V_3 corre en tiempo polinomial podemos decir que si V_1 corre en tiempo $O(|x|^{k_1})$ y V_2 en tiempo $O(|x|^{k_2})$, entonces como V_3 consiste en correr a V_1 y V_2, el tiempo en que lo realizará será $O(|x|^{k_1}) + O(|x|^{k_2}) = O(|x|^{max\{k_1,k_2\}})$, que es tiempo polinomial, y por lo tanto $L_3 \in NP$.

Ahora vemos si $L_1 \cap L_2 \in$ NP:
Similarmente al caso anterior, tenemos V_1 y V_2 verificadores de L_1 y L_2 y queremos construir V_3 verificador de $L_3 = L_1 \cap L_2$. Para esto notamos que para que c sea un certificado de L_3 tiene que suceder que $V_1(x, c) = 1$ y $V_2(x, c) = 1$, entonces definimos a V_3 como: $V_3(x, c) = V_1(x, c) \wedge V_2(x, c)$, y análogamente al caso anterior, ya que se corre ambos verificadores, tenemos que V_3 corre en tiempo polinomial $O(|x|^{k_1}) + O(|x|^{k_2}) = O(|x|^{max\{k_1,k_2\}})$, y por lo tanto $L_3 \in NP$.

2. Demuestra que si P = NP, entonces NP = coNP.

Partamos de la afirmación de que **P** es cerrado bajo complemento.

Dem. (**NP** $\subseteq$ **coNP**)
Sea $L \in$ **NP**. Dado que **P** = **NP**,
$\Rightarrow L \in$ **P**,
$\Rightarrow \widetilde{L} \in$ **P**,
$\Rightarrow \widetilde{L} \in$ **NP** (por hipótesis).
Por la definición de **coNP**, $L \in$ **coNP**.

Dem. (**coNP** $\subseteq$ **NP**)
Sea $L \in$ **coNP**,
$\Rightarrow \widetilde{L} \in$ **NP**,
$\Rightarrow \widetilde{L} \in$ **P** (por hipótesis),
$\Rightarrow L \in$ **P**, pues **P** es cerrado bajo complemento.
Por el supuesto de que **P** = **NP**, $L \in$ **NP**.

Ahora que hemos probado ambas contenciones, ello demuestra la igualdad de los conjuntos bajo la condición de que **P** = **NP**. Es decir, si **P** = **NP**, entonces **NP** = **coNP**.

3. Demuestra que el lenguaje $SPACETM = \{\langle \mathbf{M} \rangle \langle \alpha \rangle 1^n \mid$ **M es una MT que acepta** α **en espacio** n $\}$ **es PSPACE-completo.**

P.D.

I) $SPACETM \in PSPACE$.

II) Cualquier $L \in PSPACE$ se puede reducir a SPACETM

Entrada $<$ M, α, $1^n >$

Construimos $M' \in TM$ para Simular M, M' con dos contadores.

I) Un contador para contar el espacio utilizado por M, el otro contador contará el número de pasos que va a ejecutar M. Cada vez después de que M ' Simule un paso de M, actualiza los contadores y comprueba si el espacio utilizado es mayor que n o si las etapas que se ejecutan en M son mayores que 2^{cn} para alguna constante c que está relacionada con M.
Si cualquiera de las anteriores ocurre entonces se rechaza inmediatamente. Observemos que el número total de configuraciones posibles de M es como máximo 2^{cn}, por lo tanto, si M no se detiene en este número de pasos, entonces debe haber entrado en un bucle infinito y M ' Rechaza, después de simular M corriendo en w, si M acepta w en espacio n, M ' Acepta, en otro caso M' rechaza. Por definición de SPACETM, M' decide a SPACE. Para simular M solamente utilzamos

espacio O(n). Entonces SPACETM $\in$ PSPACE

II) Ahora Observemos que para cualquier $L \in PSPACE$ dónde L es un lenguaje existe una Maquina de Turing Determinista M que decide L usando espacio s(n) el cual es una entrada polinomial de tamaño n. Consideremos la función f: $\left\{0,1\right\}^n \Rightarrow \left\{0,1\right\}^m$ tal que $f(x) =< M, x, 1^{s(n)} >$. Dónde m es polinomial de n. La función f puede ser computada en espacio y tiempo polinomial. Por definición de SPACETM $x \in L$ syss $f(x) \in SPACETM$. Así que SPACETM es PSPACE-hard entonces $SPACETM \in PSPACE - completo$

$\therefore EsPSPACE - Completo.$

...

4. Demuestra $2SAT \in NL$

Primero veamos que para todo F en 2-SAT podemos construir una grafica dirigida G de la siguiente manera:
V(G) son las literales de F y hay una arista de a a b en E(G) si y solo si (a' v b) esta e F
Con esta construccion veamos que:
F es satisfaceible si y solo si G no tiene un ciclo donde exista una literal y su negacion.
=>)
veamos que a' V b es equivalente a a->b
Supongamos que tenemos un ciclo que contiene a b y b' en G. Si esto pasa entonces por transitividad signfica que b→...→b' y que b'→. . . →b por lo que tendriamos b ↔ b' lo cual es una contradiccion.
Por lo que si F es satisfacible entonces G no tiene un ciclo donde exista una literal y su negacion.
<=)
Veamos estos haciend induccion sobre el numero de variables Cuando se tienen 0 varaibles
Si F no tiene variables entonces F es satisfacible y G no tiene ciclos.
Hip. Ind
Supongamos que se cumple cuando se tiene n variables
Ahora vemos que pasa cuadno se tiene un variables mas:
Tomamos una literal b que en la grafica G generada no tiene un camino de b a b'.
Ahora supongamos que hay un camino de b a c y un camino de b a c'
Por contraposicion tendriamos un camino de c' a b' y por transitividad tendiamos un camino b a b' lo cual sera una contradiccion.
Ahora hagamos todas las literales a las que llegamos a partir de b verdaderas, esta conjunto esta bien definido por lo anterior visto. Entonces todas las clausulas que contenga a un literal de este conjunto se satisfacen. Y nos queda una grafica G' dada por las clausulas restantes las cuales tampoco tienen un ciclo con una literal y su negacion y por Hipotesis de induccion esta formula tambien es satisfacible.
.: Si G no tiene ciclos donde existe una literal y su negacion F es satisfacible
En libro tenemos que NL= coNL veamos que 2-SAT' en NL implica 2-SAT en NL

Para ver si algo cumple con 2-SAT' basta con ver si hay una camino de alguna literal b a b' y de b' a b esto es que exista un ciclo en la grafica G generada. Por lo que podemos reducir el problema a PATH que de igual manera por lo visto en el libro PATH esta en NL.
.: 2-SAT' esta en NL .: 2-SAT esta en NL

5.

6. Demuestra que en todo juego finito de información perfecta con dos jugadores, uno de los dos tiene una estrategia ganadora.

Supongamos que n es la cota superior de movimientos después de los cuales se tiene un ganador en el juego. Para que uno de los jugadores, digamos el jugador 1, tenga una estrategia ganadora en un juego de información perfecta -es decir, que los jugadores conocen todos los movimientos desde que comenzó el juego-, es necesario que para toda primera jugada del jugador 2, exista una serie de jugadas siguientes por parte del jugador 1 que al termino del juego le permitan ganar a dicho jugador.
Lo anterior lo podemos modelar considerando una Fórmula Booleana Cuantificada (QBF). Primero, consideremos que cada movimiento puede ser codificado como una cadena binaria, decidir qué jugador gana al término del juego podemos expresarlo como una fórmula Booleana φ. S.p.g $\varphi = 1$, indica que 2 ha ganado el juego. Dicho esto, el jugador 2 tendrá una estrategia ganadora si y sólo si, la siguiente QBF es verdadera (aquí empieza el jugador 1):

$$QBF_1 := \forall x_1 \exists x_2 \forall x_3 \exists x_4 \ldots \exists x_{2_n} \varphi(x_1, x_2, ..., x_n) \tag{1}$$

Supongamos que QBF_1 es falsa, con lo que su negación debe ser verdadera, por la definición de una QBF (al no tener variables libres), con lo que tendríamos:

$$QBF_2 := \neg(\forall x_1 \exists x_2 \forall x_3 \exists x_4 \ldots \exists x_{2_n} \varphi(x_1, x_2, ..., x_n)) \tag{2}$$

Es decir:

$$QBF_2 := \exists x_1 \forall x_2 \exists x_3 \forall x_4 \ldots \forall x_{2_n} \neg\varphi(x_1, x_2, ..., x_n)) \tag{3}$$

Teníamos previamente que el jugador 2 ganaba, por lo que $\neg\varphi = 0$, así que el ganador en esta QBF_2 es 1 y se muestra que existe una estrategía en la que este jugador gana el juego. Por lo tanto, hemos demostrado que alguno de los dos jugadores tiene una estrategia ganadora en el juego finito de información perfecta.

Anexo

Tarea 1.1 Ejercicio 1. En clase se presentó una descripción general de cómo codificar una máquina de Turing como una cadena binaria. Completa la descripción de tal forma que una máquina universal pueda utilizarla para imitar la máquina codificada

Comencemos con la codificación de la máquina de Turing como una cadena binaria.
Sea $M = (Q, \Sigma, \Gamma, \delta, s, r, t, \vdash, \sqcup)$ una máquina de Turing, con

- $Q = i$
- $\Sigma = j$
- $\Gamma = k$
- s el m-ésimo estado
- t el n-ésimo estado
- r el p-ésimo estado
- $\vdash$ el q-ésimo caracter de Σ
- $\vdash$ el r-ésimo caracter de Σ

Dicha especificación será codificada por la siguiente cadena binaria:

$$0^i 1 0^j 1 0^k 1 0^m 1 0^n 1 0^p 1 0^q 1 0^r 1. \tag{4}$$

Continuamos con la codificación de una transición en delta de la forma $\delta(s_u, a_v) = (s_w, a_x, \rightarrow)$ como (2), mientras que una transición de la forma $\delta(s_u, a_v) = (s_w, a_x, \leftarrow)$ como (3):

$$0^u 1 0^v 1 0^w 1 0^x 101 \tag{5}$$

$$0^u 1 0^v 1 0^w 1 0^x 111 \tag{6}$$

De esta manera, podemos codificar completamente la tabla original de M.
Finalmente, una cadena de entrada en Γ^* de la forma $a_{n_1} \dots a_{n_z}$, quedará codificada por la cadena binaria:

$$0^{n_1} 1 0^{n_2} 1 \dots 0^{n_z} \tag{7}$$

Recordemos que la Máquina Universal de Turing U que simulará a M, consta de tres cintas, además de la de entrada y salida. La primera de ellas será utilizada para la descripción de M, es decir, la tabla que quedó codificada para cada transición δ como en (3); la segunda, contendrá lo que corresponde con la cinta original de M; y la tercera el estado en el que la M se encontraría.
Al iniciar U, en la primera cinta ya debe tener la tabla de M codificada, mientras que en la tercera cinta aparecerá un 0, indicando que se encuentra en el estado inicial de la máquina original.

Separados por unos, podemos considerar una cadena binaria proveniente de una codificación de transición en δ, como una quinteta, por los 5 elementos en total que componen una transición. Para emular a M, la Máquina Universal de Turing U, deberá hacer lo siguiente por cada transición de la TM original:

1. Localiza en la primera cinta aquella quinteta en la que el primer componente sea igual al que se encuentra en la tercera cinta y el segundo componente sea igual al que se encuentra en la segunda cinta.

2. Copia a la tercera cinta el tercer componente de la primera cinta (el estado al que se transfiere).

3. Sustituye la segunda cinta con lo que tiene el cuarto componente.

4. Interpetra el movimiento de la cabeza que se encuentra en el quinto componente, moviendo acorde la cabeza de la segunda cinta.