Universidad Simón Bolívar
Teoría de la Computación
Tarea II

Fabiola Di Bartolo
Carnet: 09-87324

25 de febrero de 2010

Los siguientes ejercicios fueron resueltos consultando los libros [1] [2] [3] [4]

1. **Ejercicio 5.1.** Mostrar que EQ_{CFG} no es decidible.

   Para mostrar que EQ_{CFG} es indecidible, se realiza una demostración por contradicción utilizando el Teorema 5.13 que dice que ALL_{CFG} es indecidible y haciendo una reducción de ALL_{CFG} a EQ_{CFG}.

   Donde:

   $ALL_{CFG} = \{\langle G\rangle : G \text{ es una CFGs y } L(G) = \sum^*\}$
   $EQ_{CFG} = \{\langle G_1, G_2\rangle : G_1 \text{ y } G_2 \text{ son CFGs y } L(G_1) = L(G_2)\}$.

   Sea R un decider para EQ_{CFG}, construimos la siguiente máquina de Turing S para decidir ALL_{CFG}.

   S = En entrada $\langle G\rangle$, siendo G una CFG.

   1. Correr R en entrada $\langle G, G_1\rangle$, donde G_1 es una CFG tal que $L(G_1) = \sum^*$.
   2. Si R acepta, ACEPTAR. Si rechaza, RECHAZAR.

   Claramente S es un decider para el lenguaje ALL_{CFG}, porque todos sus pasos terminan, pero por el Teorema 5.13 ALL_{CFG} es indecidible, lo que es una contradicción, por lo tanto EQ_{CFG} es indecidible.

2. **Ejercicio 5.3.** Encontrar una correspondencia a la siguiente instancia de PCP

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

La correspondencia puede ser observada en la Figura 1, el string es el mismo arriba y abajo:

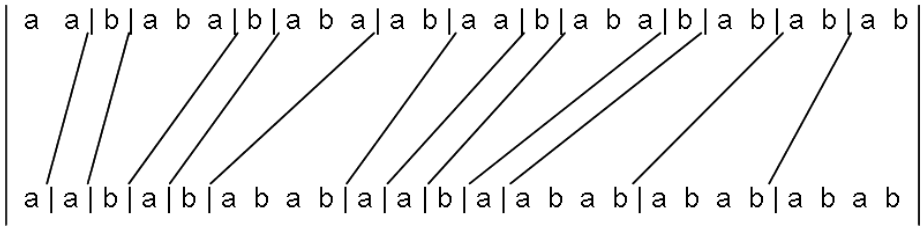

Figura 1: Correspondencia de la instancia dada

3. **Ejercicio 5.13.** Un estado inútil es uno en el que nunca entra la máquina de Turing en cualquier entrada. Considere el problema de determinar si una máquina de Turing tiene algún estado inútil. Formula este problema como un lenguaje y muestra que es indecidible.

Definimos el lenguaje L de la siguiente manera:
$L = \{\langle M, q\rangle : M$ es una máquina de Turing que tiene el estado inútil $q\}$

Una máquina de Turing tiene un estado inútil q si en la secuencia de configuraciones $C_1, C_2, ...$ en cualquier entrada x, q no aparece en la configuración C_i para todo i.

La demostración la haremos por contradicción, realizando una reducción de A_{TM} a L. Suponemos que R es un decider para L y utilizaremos

R en la construcción la siguiente máquina de Turing S que decide A_{TM}.

$S =$ En entrada $\langle M, w\rangle$.

1. Utilizar M para construir la siguiente máquina de turing T:

   $T =$ en entrada x:

   1.1. Ignorar la entrada. Escribe w en la cinta.
   1.2. Simular M en w.
   1.3. Si M acepta, pasar al estado de aceptación q_{TA} y ACEPTAR. Sino, RECHAZAR.

2. Correr R en $\langle T, q_{TA}\rangle$ para determinar si q_{TA} es un estado inútil.
3. Si R rechaza, ACEPTAR. Sino, RECHAZAR.

El estado de aceptación q_{TA} de la máquina T sera inútil si y sólo si M no acepta a w. Por lo tanto, si R rechaza, es porque M acepta a w y T en cualquier entrada, entra en estado q_{TA}, y en consecuencia, S acepta. Por otro lado, si R acepta, es porque q_{TA} es un estado inútil, lo que significa que M no acepta a w, y T rechaza todas las entradas y en consecuencia, S rechaza.

Como A_{TM} es indecidible, la existencia de la máquina S nos lleva a una contradicción, por lo tanto R no existe y L es indecidible.

4. **Ejercicio 5.20.** Probar que existe un subconjunto de $\{1\}^*$ que es indecidible.

   Para demostrarlo, construiremos el lenguaje L (subconjunto de $\{1\}^*$) tal que exista una reducción de un lenguaje que sabemos es indecidible, A_{TM}, al lenguaje L. La idea es hallar una función f biyectiva tal que, para todo x, $x \in A_{TM}$ si y sólo si $f(x) \in L$, la forma de hacer esto es codificando las palabras en unario.

Sea $s_1, s_2, ..., s_i, ...$ tal que $s_i \in \sum^*$ ($A_{TM} \subseteq \sum^*$) una lista de palabras ordenadas lexicográficamente por un enumerador, definimos el subconjunto L de $\{1\}^*$ de la siguiente forma: $L = \{1^i : s_i \in A_{TM}\}$.

Suponiendo que R es un decider para L, construimos la siguiente máquina S que decide a A_{TM}.

$S =$ En entrada x.

1. Chequear que x sea de la forma $\langle M, w\rangle$, sino RECHAZAR.
2. Encontrar la posición i, tal que $x = s_i$.
3. Correr R en 1^i.
4. Si R acepta, ACEPTAR. Sino, RECHAZAR.

Claramente S es un decider para A_{TM}. Para todo s_i chequea si 1^i pertenece o no a L, lo cual es equivalente a la definición de la función de reducción f que se definió anteriormente. Sin embargo, como A_{TM} es indecidible, entonces S no puede ser un decider, por lo tanto R no existe y L es indecidible.

5. **Ejercicio 7.11.** G y H son grafos isomórficos si los nodos de G pueden ser reordenados tal que G sea idéntico a H. Sea $ISO = \{\langle G, H\rangle : G$ y H son grafos isomórficos$\}$. Probar $ISO \in NP$

Para demostrar que $ISO \in NP$ se construye el siguiente verificador polinomial V para ISO. El certificado será una lista de pares ordenados que indiquen la correspondencia entre los vértices de G (V_G) y los de H (V_H). La correspondencia estará dada por una función biyectiva f, tal que para un vértice $x \in V_G$ devuelve su correspondiente $f(x) \in V_H$.

S = En entrada $\langle\langle G, H\rangle, C\rangle$, donde G y H son grafos.

1. Chequear que $\mid V_G \mid = \mid V_H \mid$, sino RECHAZAR.
2. Chequear que el certificado C es de la forma $\{(x_1, f(x_1)), (x_2, f(x_2)), ...\}$, tal que para todo i: $x_i \in V_G$ y $f(x_i) \in V_H$, ningún vértice se repite en C y $\mid C \mid = \mid V_G \mid$, sino RECHAZAR.
3. Chequear que para cada par de vértices adyacentes u y v en V_G, $f(u)$ y $f(v)$ son adyacentes en V_H
4. Si los tests pasan, ACEPTAR. Sino, RECHAZAR.

V es un verificador polinomial para ISO, en cada paso corre en tiempo polinomial en función de los vértices de G y H. Por lo tanto, V corre en tiempo polinomial, más aún, en $O(n^2)$ donde n es igual al número de vertices de los grafos.

6. **Ejercicio 7.13.** Una permutación en el conjunto $\{1, ..., k\}$ es una función uno a uno en el mismo conjunto. Cuando p es una permutación, $p^{(t)}$ es la composición de p consigo mismo t veces. Sea $PERM\_POWER = \{\langle p, q, t\rangle : p = q^{(t)}$ donde p y q son permutaciones en $\{1, ..., k\}$ y t es un entero binario $\}$. Mostrar que $PERM\_POWER \in P$. (Nota: el algoritmo más obvio no corre en tiempo polinomial. Ayuda: Intentar primero con t potencia de 2)

   Para mostrar que $PERM\_POWER$ está en P, se construye la máquina de Turing polinomial M que realiza la composición de funciones. Para que corra en tiempo polinomial es necesario descomponer t en potencias de 2 para evitar que el valor de t sea exponencial en la longitud de la entrada.

   Se utilizará para hacer la composición la función q de entrada y la función identidad I, la cual no altera el orden, es decir, devuelve la misma permutación. Asumimos que la composición de estas funciones de permutación es polinomial con respecto a la codificación de la permutación.

   Utilizando el hecho de que $q^{(2i)} = q^{(i)} \circ q^{(i)}$, la permutación $q^{(t)}$ puede ser hallada en $log\ t$ pasos. Entonces, la idea para obtener esta composi-

ción de funciones es tener dos funciones temporales, una que tendrá la composición de funciones de resultados anteriores, sólo si la potencia en cuestión es impar, y otra que tendrá únicamente composiciones consigo misma. De esta forma se obtendrá $q^{(t)}$.

M = En entrada $\langle p, q, t\rangle$.

1. Verificar que p y q son permutaciones válidas en $\{1, ..., k\}$. Sino, RECHAZAR.
2. Inicializar g con la permutación identidad I.
3. Para $t > 0$:
   3.1. Si t es impar: actualizar g con $g \circ q$ (composición con la función de ultima iteración donde t fue impar)
   3.2. Actualizar q con $q \circ q$ (composición consigo misma)
   3.3. Actualizar t con $\lfloor \frac{t}{2} \rfloor$
4. Si p es igual a g, entonces ACEPTAR. Sino, RECHAZAR.

La máquina M es de orden polinomial debido a que en el paso 3 realiza $log\ t$ operaciones debido a la descomposición de t en potencias de 2 y los demás pasos son polinomiales. Lo que implica que $PERM\_POWER$ está en P.

7. **Ejercicio 7.20.** Sea G un grafo no dirigido y sea:

$SPATH = \{\langle G, a, b, k\rangle : G$ contiene un camino simple de longitud a lo más k desde a hasta $b\}$
$LPATH = \{\langle G, a, b, k\rangle : G$ contiene un camino simple de longitud al menos k desde a hasta $b\}$

a) Mostrar que $SPATH \in P$.

Para mostrar que $SPATH \in P$, se construye la siguiente máquina de Turing M que decide a $SPATH$ en tiempo polinomial. Donde $G = (V_G, E_G)$ es un grafo y V_G el conjunto de vértices y E_G el

conjunto de aristas.

M = En entrada $\langle G, a, b, k\rangle$, donde G es el grafo, a, b son vértices de G y k es un entero.

1. Marcar el nodo a.
2. Para $i = 1...k$ y mientras se haya marcado un nodo en la última iteración y no sea b el nodo marcado:
   2.1 Marcar todos los nodos v si la arista $\langle u, v\rangle \in E_G$ y u está marcado.
3. Si b está marcado, ACEPTAR. Sino, RECHAZAR.

La máquina M realiza una búsqueda en amplitud del camino de a lo más tamaño k, inicialmente en el paso 1, se marca el nodo a, luego en el paso 2, por cada iteración (a lo sumo son k iteraciones) marca los nodos adyacentes al nodo marcado en la iteración anterior o inicialmente, a lo sumo en el paso 2.1 se marcarán $\mid V_G \mid$ nodos. En el paso 3, si al terminar, el nodo b está marcado, entonces se encontró un camino entre a y b de a lo sumo longitud k.

Claramente M es un decider porque k es un número finito y V_G es un conjunto finito, y tomando el razonamiento anterior, se puede observar que corre en tiempo polinomial en función de la cantidad de nodos de G debido a que se están buscando caminos simples.

b) Mostrar que $LPATH$ es $NP - completo$. Puede asumir la NP-completitud de $UHAMPATH$, el problema del camino Hamiltoniano para grafos no dirigidos.

Para demostrar que $LPATH$ es $NP - completo$, utilizaremos el Teorema 7.55 en el cual se prueba que $UHAMPATH$ es $NP - Completo$.

Se debe demostrar que:

- $\star$ $LPATH \in NP$ (inclusión)
- $\star$ Para todo $UHAMPATH \in NP, UHAMPATH \leq_p LPATH$ (dificultad)

Para la demostración de la inclusión, se construye un verificador polinomial, el cual llamaremos V.

V = En entrada $\langle\langle G, a, b, k\rangle, C\rangle$, donde G es el grafo, a, b son vértices de G y k es un entero.

1. Chequear que el certificado C es una lista $P_1...P_m$ donde $P_1 = a$ y $P_m = b$, tal que $k - 1 \leq m$ y $m \leq V_G$, sino RECHAZAR.
2. Chequear que no hay repeticiones de vértices en C, sino RECHAZAR.
3. Chequear que la arista $\langle P_i, P_{i+1}\rangle$ pertenece a E_G para todo $1 \leq i \leq m$
4. Si los tests pasan, ACEPTAR. Sino, RECHAZAR.

V realiza la verificación en tiempo polinomial de que el certificado es un camino simple de G, ya que los pasos 1 y 2 se ejecutan en orden lineal en función del número de vértices y el paso 3 en orden cuadrático, porque por cada nodo en C chequea que sea adyacente del siguiente. Por lo tanto, como existe un verificador, $LPATH \in NP$.

Para demostrar la dificultad, realizamos una reducción polinomial de $UHAMPATH$ a $LPATH$. Se define la siguiente función de reducción f: $\langle G, a, b\rangle \in UHAMPATH$ si y sólo si $f(\langle G, a, b\rangle) = \langle G, a, b, \mid V \mid -1\rangle \in LPATH$ para todo $\langle G, a, b\rangle$.

f es una reducción polinomial, ya que si G tiene un camino Hamiltoniano de a a b, entonces ese es un camino simple de a a b con $k = \mid v \mid -1$ y por lo tanto $f(\langle G, a, b\rangle) \in LPATH$. Por otro lado, si existe en G un camino simple de al menos longitud $\mid V \mid -1$ de a a b, entonces este pasa por cada nodo del grafo,

y como es un camino simple, pasa por cada nodo exactamente una vez, por lo tanto, es un camino Hamiltoniano, es decir que $f(\langle G, a, b\rangle) \in UHAMPATH$.

8. **Ejercicio 7.36.** Mostrar que si $P = NP$, existe un algoritmo en tiempo polinomial que produce una asignación satisfacible para una fórmula Booleana satisfacible dada. (Nota: el algoritmo debe proporcionar una función computable, pero NP contiene lenguajes, no funciones. La suposición $P = NP$ implica que SAT esta en P, así que en tiempo polinomial puede ser probada la satisfacibilidad. Pero la suposición no dice cómo es realizada la prueba y la prueba podría no proporcionar las asignaciones que satisfacen la función. Mostrar que de todas formas, éstas se puede encontrar. Ayuda: usar verificador de satisfacibilidad repetidas veces para encontrar la asignación bit a bit)

   Sea $SAT = \{\langle\phi\rangle : \phi$ es una fórmula booleana satisfacible $\}$. Si $P = NP$, implica que SAT esta en P, lo que significa que existe un algoritmo polinomial que verifique para cualquier fórmula ϕ si se satisface. Llamaremos a este decider M_{SAT}, si M_{SAT} acepta una fórmula booleana, significa que la fórmula es satisfacible.

   Para probar que una fórmula booleana es satisfacible, lo único que se necesita es especificar una asignación de variables ($true$, o $false$) que haga la fórmula cierta. Dada esta asignación, se puede chequear en orden lineal leyendo cada variable de izquierda a derecha y evaluándola.

   Para encontrar una asignación en tiempo polinomial que satisface a ϕ, construimos el siguiente algoritmo A.

   $A =$ En entrada $\langle\phi\rangle$, donde $x_1, x_2, ..., x_n$ son variables de ϕ

   1. Chequear que ϕ sea una formula booleana, sino terminar.
   2. Simula M_{SAT} en $\langle\phi\rangle$. Si acepta M_{SAT}, continuar.
   2. Para $i = 1...n$:

2.1. Sustituir $x_i = 0$ en ϕ y simular M_{SAT} en $\langle\phi\rangle$ para verificar si es satisfacible. Si M_{SAT} acepta, fijar x_i con 0.

2.2. Si M_{SAT} no aceptó. Sustituir $x_i = 1$ en ϕ, y simular M_{SAT} en $\langle\phi\rangle$ para verificar si es satisfacible. Si M_{SAT} acepta, fijar x_i con 1.

El algoritmo termina en n pasos, donde n es el número de variables en ϕ. Al finalizar el algoritmo, se habrá encontrado una asignación para ϕ tal que sea satisfacible (si existe alguna). Cada paso corre en tiempo polinomial porque asumimos que M_{SAT} verifica en tiempo polinomial. Por lo tanto, este algoritmo corre en tiempo polinomial.

9. **Ejercicio 7.44.** Se dice que dos fórmulas Booleanas son equivalentes si ellas tienen el mismo conjunto de variables y son ciertas en el mismo conjunto de asignaciones de esas variables (es decir, describen la misma función Booleana). Una fórmula Booleana es minimal si no existe una fórmula Booleana más corta que sea equivalente. Sea $MIN\_FORMULA$ la colección de fórmulas Booleanas minimales. Mostrar que si $P = NP$, entonces $MIN\_FORMULA \in P$

   Definimos el lenguaje $MIN\_FORMULA$ de la siguiente forma:
   $MIN\_FORMULA = \{\langle\phi\rangle : \phi$ es minimal, es decir, si $\phi \equiv \phi'$, entonces $\mid \phi \mid \leq \mid \phi' \mid\}$

   Es decir, ϕ es minimal si y sólo si para cualquier fórmula ϕ' más corta que ϕ, existe una asignación de variables tal que ϕ y ϕ' tengan valores diferentes.

   Sea NE el lenguaje que indica si dos fórmulas booleanas no son equivalentes, es decir que difieren el alguna asignación que haga que las fórmulas sean ciertas: $NE = \{\langle\phi, \phi'\rangle : \phi$ no es equivalente a $\phi'\}$. Sea $\overline{NE}$ el lenguaje que indica si dos fórmulas booleanas son equivalentes, es decir que para cualquier asignación, ambas formulas tienen los mismos valores: $\overline{NE} = \{\langle\phi, \phi'\rangle : \phi$ es equivalente a $\phi'\}$

Mostramos que NE está en NP. Si $P = NP$, entonces por el Teorema 7.27 de Cook-Levin, $SAT \in P$. Por lo tanto, dada la existencia de un algoritmo polinomial para chequear la pertenencia a SAT, un verificador V_1 mediante ese algoritmo puede chequear dada una asignación v (certificado) y las fórmulas booleanas ϕ, ϕ', si las dos pertenecen a SAT o no. Entonces como asumimos que $P = NP$, entonces $NE \in P$, y como P es cerrado bajo el complemento por el Teorema 8.6.1 en [4], $\overline{NE} \in P$

Sabiendo ahora que si $P = NP$, $\overline{NE} \in P$ entonces mostramos que $\overline{MIN\_FORMULA}$ está en P. Dada una asignación v (certificado), y las fórmulas booleanas ϕ, ϕ', un verificador V_2 puede chequear si ϕ es mas corta que ϕ' ($\mid \phi \mid < \mid \phi' \mid$) y la equivalencia entre ambas (como lo hace el verificador V_1) en tiempo polinomial. Entonces como asumimos que $P = NP$, entonces $\overline{MIN\_FORMULA} \in P$, y como P es cerrado bajo el complemento por el Teorema 8.6.1 en [4], $MIN\_FORMULA \in P$

Bibliografía

[1] M. Sipser, *Introduction to the Theory of Computation; 2nd ed.* Cambridge: Thomson Course Technology, 2006.

[2] J. D. U. John E. Hopcroft, Rajeev Motwani, *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, second ed., 2001.

[3] N. Cutland, *Computability: An Introduction to Recursive Function Theory.* Cambridge: Cambridge Univ. Press, 1980.

[4] J. E. Savage, *Models of computation : exploring the power of computing.* Addison Wesley, 1998.