Pauta de Corrección

Segundo Certamen
Introducción a la Informática Teórica

19 de julio de 2014

1. Por turno.

   a) Supongamos dos lenguajes recursivamente enumerables L_1 y L_2. Como son recursivamente enumerables, hay máquinas de Turing M_1 y M_2 que los aceptan, o sea $L_1 = \mathscr{L}(M_1)$ y $L_2 = \mathscr{L}(M_2)$. Entonces $\omega \in L_1 \cup L_2$ si y solo si ω es aceptado por M_1 o por M_2. Podemos correr M_1 y M_2 en paralelo, alternadamente, para aceptar $L_1 \cup L_2$. Una descripción informal es la figura 1.

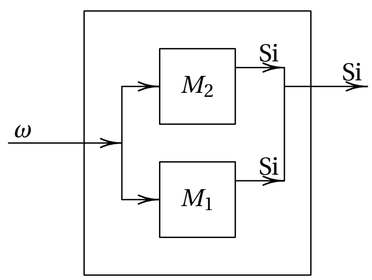

Figura 1: Reconocer unión de recursivamente enumerables

   b) Sabemos que si un lenguaje y su complemento son recursivamente enumerables, ambos son recursivos. Vimos varios ejemplos de lenguajes recursivamente enumerables no recursivos, como $L_{\texttt{ne}} = \{M \colon \mathscr{L}(M) \neq \varnothing\}$. Su complemento (en nuestro caso presente $L_{\texttt{e}} = \{M \colon \mathscr{L}(M) = \varnothing\}$) no es recursivamente enumerable.

   c) Supongamos dos lenguajes recursivamente enumerables L_1 y L_2. Como son recursivos, hay máquinas de Turing M_1 y M_2 que los aceptan y siempre se detienen, o sea $L_1 = \mathscr{L}(M_1)$ y $L_2 = \mathscr{L}(M_2)$. Ahora bien, $\omega \in L_1 \cap L_2$ si ω es aceptado por M_1 y por M_2. Una descripción informal es la figura 2.

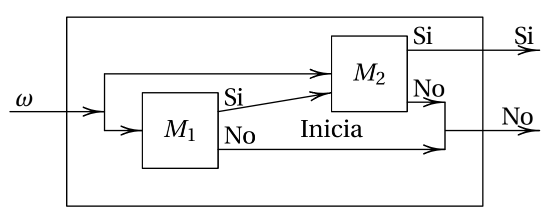

Figura 2: Reconocer intersección de recursivos

Puntajes

Total		30
a) Construcción/explicación	10	
b) Referencia a algunos de los ejemplos	10	
a) Construcción/explicación	10	

2. Podemos describir la computación mediante descripciones instantáneas:

$$
\begin{aligned}
q_0 10100 &\vdash 1 q_0 0100 \\
&\vdash 10 q_0 100 \\
&\vdash 101 q_0 00 \\
&\vdash 1010 q_0 0 \\
&\vdash 10100 q_0 b \\
&\vdash 1010 q_2 0 \\
&\vdash 101 q_Y b0
\end{aligned}
$$

Queda claro que el modus operandi es avanzar hasta hallar b, pasar sobre dos 0 y aceptar. Esto se confirma analizando los pasos luego de hallar el primer b en la tabla. El lenguaje aceptado puede describirse por la expresión regular $(0 \mid 1)^* 00$.

Puntajes

Total		15
Seguir el cómputo sobre 10100	5	
Explicar el lenguaje aceptado	10	

3. Esto corresponde a determinar si dos máquinas de Turing aceptan el mismo lenguaje. Elijamos una máquina de Turing cualquiera, llamémosle M, y sea $L = \mathscr{L}(M)$. Entonces determinar si las máquinas de Turing M y M' aceptan el mismo lenguaje corresponde a determinar si $\mathscr{L}(M') = L$, que claramente es una propiedad no trivial. Por el teorema de Rice, esto es imposible.

Puntajes

Total		25
Llevar a máquinas de Turing	10	
Aplicar teorema de Rice	10	
Conclusión	5	

4. Describimos el algoritmo en términos generales, dando cotas para cada parte. Mantenemos la lista de tareas por programar (inicialmente todas), la lista de tareas programadas en orden (inicialmente vacía) y la lista de tareas atrasadas (inicialmente vacía). Mantenemos el instante de término de la programación (inicialmente 0).

   - Se ordenan las tareas en orden de plazo fatal decreciente. Sabemos que puede hacerse en tiempo $O(r \log r)$, usando mergesort.
   - Tomamos la siguiente tarea y la programamos a continuación. Sumamos el plazo de ejecución al instante de término. Esto toma tiempo constante para cada tarea. En total, $O(r)$.
   - Si el instante de término sobrepasa al plazo fatal, elegimos una de las tareas más largas de la lista de tareas programadas, y la ponemos en la lista de tareas atrasadas. Restamos el tiempo de ejecución de la tarea eliminada del instante de término. Elegir la tarea a eliminar puede hacerse revisando la lista ($O(r)$) y transpasarla a la lista de atrasadas toma tiempo constante. En el peor caso, esto se hace para cada tarea, r veces, lo que da un total de $O(r^2)$.
   - Agregar las tareas en la lista de atrasadas a la programación significa agregar cada una de ellas y ajustar el instante de término. En el peor caso, son todas las tareas, y demanda $O(r)$ tiempo.

   En resumen, el tiempo de ejecución es:

   $$O(n \log n) + O(r) + O(r^2) + O(r) = O(r^2)$$

Puntajes

Total		15
Ordenar tareas es $O(n \log n)$	3	
Programación tentativa en total es $O(r)$	2	
Si sobrepasa, el proceso toma $O(r)$; total $O(r^2)$	5	
Programar las atrasadas toma $O(r)$	2	
Tiempo total $O(r^2)$	3	

5. El problema resuelto en la pregunta 4 puede describirse en los presentes términos mediante tiempos de ejecución $(t_1, t_2, \ldots, t_r)$, plazos fatales $(d_1, d_2, \ldots, d_r)$ y penalidades $(1, 1, \ldots, 1)$. Es un caso particular de JOB SEQUENCING. Que casos especiales tengan solución eficiente no impide que el caso más general sea mucho más difícil de resolver.

Puntajes

Total	20
Contraste y explicación	20

6. El problema más cercano es CLIQUE (¿Contiene el grafo G una clique de tamaño k, o sea, tiene un subgrafo isomorfo a K_k?). Podemos reducir CLIQUE a SUBGRAPH ISOMORPHISM de la siguiente forma: Dados el grafo $G = (V, E)$ y k, verificamos si $k \le |V|$, de no ser así no hay solución. Construimos K_k (significa agregar k vértices y $k(k-1)/2$ arcos, esto toma tiempo $O(|V|^2)$, polinomial en el tamaño de los datos de entrada), y entregamos G, K_k a SUBGRAPH ISOMORPHISM.

   Es claro que la instancia así construida tiene la misma respuesta que el problema original, lo que demuestra que SUBGRAPH ISOMORPHISM es NP-duro.

   Para demostrar que SUBGRAPH ISOMORPHISM está en NP, basta elegir para cada vértice de G' un vértice de G que es la supuesta contraparte, y luego verificar que los arcos de G' conectan los vértices en G. Suponiendo que los grafos se representan mediante matrices de adyacencia, esto toma tiempo lineal en el tamaño de los datos originales.

   Concluimos que SUBGRAPH ISOMORPHISM es NP-duro y está en NP, NP-completo.

Puntajes

Total	20
Explicar reducción, ambos la misma respuesta	
Argüir que la reducción es polinomial	
Concluir NP-duro	
Esbozar solución nodeterminista polinomial	
NP-duro y en NPes NP-completo	