Práctica: complejidad

27 de junio de 2016

Guido Tagliavini Ponce

Ejercicio

Demostrar detalladamente que alguno de los siguientes problemas es NP-completo usando que el otro lo es. Indicar claramente cuál problema se intenta demostrar que es NP-completo y cuál se supone que lo es.

Π_1: CAMINO HAMILTONIANO
Entrada: grafo G.
Pregunta: ¿existe un camino que pasa exactamente una vez por cada vértice de G?

Π_2: CAMINO MÁXIMO
Entrada: grafo G de n vértices; $k \in \mathbb{Z}$ tal que $0 \leq k \leq n-1$.
Pregunta: ¿tiene G un camino simple de k o más ejes?

Solución

Asumimos que Π_1 es NP-completo, y probamos que Π_2 es NP-completo.

$\Pi_2 \in$ **NP**

Representación del input:

- $A \in \{0,1\}^{n \times n}$ (la matriz de adyacencias de G)
- $k \in \mathbb{Z}$, $0 \leq k \leq n-1$

Certificado:

- secuencia $\langle u_1, \ldots, u_{t+1} \rangle$, con cada $u_i \in \mathbb{Z}$, $1 \leq u_i \leq n$

Verificación:

1. Verificar que el camino tenga k o más ejes:
   si $t < k$ devolver NO
2. Verificar que $C = \langle u_1, \ldots, u_{t+1} \rangle$ sea un camino en el grafo:
   para cada $i \leftarrow 1, \ldots, t$
   si $A[u_i][u_{i+1}] = 0$ devolver NO
3. Verificar que C sea simple:
   sea B un arreglo de n booleanos
   para cada $i \leftarrow 1, \ldots, n$

$B[i] \leftarrow false$

para cada $i \leftarrow 1, \ldots, t+1$

si $B[u_i]$ devolver NO

$B[u_i] \leftarrow true$

4. Devolver SI.

Tenemos que demostrar dos cosas:

1. **Verifica correctamente.** Concretamente, que dada una instancia $I \in D_{\Pi_2}$,

$$I \in Y_{\Pi_2} \Leftrightarrow \textit{para el input } I \textit{ existe un certificado tal que el verificador devuelve SI}$$

Escribimos $I = (G, k)$.

($\Rightarrow$) Si $(G, k) \in Y_{\Pi_2}$, entonces existe un camino simple $\langle v_1, \ldots, v_{s+1} \rangle$ de $s \geq k$ ejes en G. Si el oráculo escribe $\langle v_1, \ldots, v_{s+1} \rangle$ como certificado, el mismo pasará todos los chequeos del verificador, que terminará respondiendo SI.

($\Leftarrow$) Si devuelve SI, es porque $C = \langle u_1, \ldots, u_{t+1} \rangle$ pasó todos los chequeos que, en conjunto, implican que C es un camino simple de G con k o más ejes. En particular, existe un camino simple de k o más ejes en G, con lo cual $(G, k) \in Y_{\Pi_2}$.

2. **Corre en tiempo polinomial.** Esto es, que

para un input y un certificado que hacen que el verificador devuelva SI, el tiempo de ejecución es polinomial en el tamaño del input

Notar que no nos interesa lo que sucede cuando el input es una instancia de N_{Π_2} o cuando el certificado es fruta. Esto es porque $\Pi_2 \in NP$ si se puede verificar *eficientemente* las instancias *afirmativas* de Π_2.

Llamemos T al tiempo que toma el verificador en una ejecución que devuelve SI. Calculemos T, sumando el costo de cada paso. El paso 1 toma tiempo $O(1)$. El paso 2 es $O(t) \subset O(n)$. El paso 3 es $O(n)$. En total, el verificador toma tiempo $T \in O(n)$.

Llamemos S al tamaño del input. Se puede ver que $S \in \Omega(n^2)$, pues necesitamos al menos este espacio para guardar la matriz A. Equivalentemente, $n^2 \in O(S)$. Luego, $T \in O(n) \subset O(n^2) \subseteq O(S)$, es decir que T es polinomial en S.

$\Pi_2 \in$ **NP-Hard**

Como $\Pi_1 \in$ NP-completo, basta ver que $\Pi_1 \leq_p \Pi_2$, luego por transitividad tenemos que $\forall \pi' \in$ **NP**, $\pi' \leq_p \Pi_2$.

Sea $f : D_{\Pi_1} \to D_{\Pi_2}$ tal que $f(G) = (G, n(G) - 1)$.

1. f **mapea instancias de** Π_1 **en instancias de** Π_2**.**

   Por un lado, las instancias de Π_1 son grafos arbitrarios, y coincidentemente a f le estamos pasando un grafo arbitrario G.

   Por otro lado, asumiendo que G es una instancia de Π_1, debemos ver que $f(G) = (G, n(G) - 1)$ es una instancia de Π_2. Sobre G no hay ninguna restricción. Sobre $k = n(G) - 1$, nos piden que $0 \leq k \leq n(G) - 1$, que obviamente vale para tal k.

2. f **es polinomial.** Esto es,

   hay un algoritmo que computa f *en tiempo polinomial en el tamaño del input*

   Representación del input:

   - $A \in \{0,1\}^{n \times n}$ (la matriz de adyacencias de G)

   Algoritmo:

   1. Devolver: $G =$ el grafo representado por A; $k = n - 1$.

   Falta mostrar que f toma tiempo polinomial. El tamaño del input es $S \in \Omega(n^2)$, y el algoritmo toma tiempo $O(1)$, por lo que el algoritmo es polinomial.

3. Para todo $I \in D_{\Pi_1}$,

   $$I \in Y_{\Pi_1} \textit{ si y sólo si } f(I) \in Y_{\Pi_2}$$

   ($\Rightarrow$) Escribimos $I = G$. Se tiene que

   $$\begin{aligned} G \in Y_{\Pi_1} &\Rightarrow G \text{ tiene un camino hamiltoniano} \\ &\Rightarrow G \text{ tiene un camino simple de } n(G) - 1 \text{ ejes} \\ &\Rightarrow G \text{ tiene un camino simple de } n(G) - 1 \text{ o más ejes} \\ &\Rightarrow (G, n(G) - 1) \in Y_{\Pi_2} \\ &\Rightarrow f(G) \in Y_{\Pi_2} \end{aligned}$$

   ($\Leftarrow$) Cada una de las implicaciones vale en el otro sentido.

   En este caso, la demostración de ida y de la vuelta son similares, aunque en general esto puede no ser así.

Un par de cuestiones técnicas

Sobre modelo de cómputo

Cuando calculamos el costo de los algoritmos, supusimos que todas las operaciones con números toman tiempo $O(1)$, independientemente del tamaño del número. Por ejemplo, hicimos ésto en el verificador, al incrementar el índice i de las estructuras de repetición. También asumimos que acceder a cualquier posición de la matriz A y del arreglo B toma tiempo $O(1)$. Esto se debe a que estamos asumiendo que nuestro algoritmo corre sobre una RAM (random-access machine) con costos uniformes. Entre otras cosas, este modelo de cómputo considera que cualquier número (por más grande que sea) entra en una sola celda de memoria, y las operaciones básicas sobre ellos toman tiempo $O(1)$. Las indirecciones de posiciones de memoria también cuestan $O(1)$.

En una DTM (deterministic Turing machine), en cambio, esto no es tan así. Para empezar, este tipo de máquinas no provee acceso aleatorio sobre las celdas de la cinta. Además, un número puede requerir varias celdas para ser representado, con lo cual las operaciones sobre éstos pueden tomar tiempo proporcional a la longitud de la cantidad de celdas sobre las que se extiendan.

Sin embargo, estos dos modelos son polinomialmente equivalentes, bajo ciertas condiciones razonables[1]. En otras palabras, puedo simular una RAM con una DTM y viceversa, con un costo polinomial. Esto justifica que podamos hacer los cálculos de complejidad sobre una RAM, y aún así concluir polinomialidad sobre una DTM.

Sobre el tamaño del input

La definición de tamaño del input está sujeta a interpretación. En esta materia tomamos a este tamaño como la cantidad de símbolos necesarios para representar el input. Por ejemplo, un número natural n puede ser representado con $O(\lg n)$ símbolos (su representación binaria). Un valor booleano puede ser representado con 1 símbolo (0 o 1). Una cadena de n caracteres puede ser representada con n símbolos (cada uno de los n caracteres).

Usaremos esta definición de tamaño del input tanto para RAM como para DTM. Para RAM, uno podría interpretar que un número del input tiene tamaño 1, pues entra en una celda de memoria. Sin embargo, según la definición considerada, el tamaño del input es independiente de las características del modelo de cómputo.

Aclarada la definición de tamaño del input que usamos, sepan que **en ningún ejercicio van a necesitar calcularlo exactamente.** Les va a alcanzar con dar una cota inferior, tal como hicimos al decir que $S = \Omega(\cdot)$.

Por ejemplo, supongamos que tenemos un algoritmo que recibe como input un arreglo de n naturales, y que su costo está dominado por un ordenamiento del arreglo.

[1] Dichas condiciones son: (1) las operaciones primitivas de la RAM son polinomiales en la DTM; (2) todos los números que genera el programa de la RAM a lo largo de su ejecución tienen tamaño polinomial en el tamaño del input.

El tamaño del input es $S = \Omega(n)$. El costo del algoritmo es $T = O(n \lg n)$. Este algoritmo es polinomial en el tamaño del input, porque $T = O(n \lg n) = O(n^2) = O(S^2)$.

Notar que el costo del algoritmo no tiene que ser *chico* para ser polinomial. Por ejemplo, si el tamaño del input es $S = \Omega(n^2)$, y el algoritmo toma tiempo $T = O(n^{700})$, entonces $T = O(S^{350})$, que es polinomial en S.