Guía pre-certamen 2 INF-155

Ayudante R.B 2024-2

November 8, 2024

1 Resolución problemas "tipo"

1. Para demostrar que está en NP, damos un certificado: dadas dos asignaciones de valores a las variables de ϕ, podemos verificar que son distintas y calcular el valor de ϕ para ambos, corroborando que resultan verdaderos. Esto claramente puede hacerse en tiempo polinomial en el largo de la expresión.

   Para demostrar que es NP-duro, basta reducir polinomialmente a DOUBLE SAT un problema NP-*completo* cualquiera. El más cercano es SAT: ¿puede satisfacerse la fórmula lógica $\phi(x_1, \ldots, x_n)$? La idea es modificar ϕ para asegurar que tenga al menos dos maneras de satisfacerse si tiene alguna. Agregamos una nueva variable x_{n+1} y escribimos:

   $$\phi'(x_1, \ldots, x_n, x_{n+1}, \neg x_{n+1}) = \phi(x_1, \ldots, x_n) \wedge (x_{n+1} \vee \neg x_{n+1})$$

   Si ϕ es satisfacible por valores dados de $(x_1, \ldots, x_n)$, entonces ϕ' se satisface de dos formas por $(x_1, \ldots, x_n, T)$ y también por $(x_1, \ldots, x_n, F)$. Si ϕ' puede satisfacerse de dos formas, es con alguno de los valores indicados de x_{n+1} (en realidad, x_{n+1} no hace diferencia), y puede satisfacerse ϕ. Realmente es una reducción, y es claro que la modificación de ϕ a ϕ' puede hacerse en tiempo polinomial.

2. Sabemos que un problema es NP-duro si todos los problemas en NP se reducen polinomialmente a él, y es NP-completo si es NP-duro y pertenece a NP. De acá dedujimos que para demostrar que un problema es NP-duro basta reducir a él desde un problema NP-completo cualquiera. Hacemos notar que reducir (polinomialmente o no) a un problema NP-completo nada nos dice, el problema original puede incluso ser trivial.

   Como premisa, nos dan $X \leq_p Y$. Veamos cada caso:

   (a) Por lo discutido, que Y sea NP-completo nada permite concluir sobre X. Es falso.

   (b) Si X es NP-completo nos permite concluir que Y es NP-duro. Falta determinar si Y está en NP para demostrar que es NP-completo. Es falso.

   (c) Hay reducciones polinomiales entre problemas en P (una reducción obvia entre A y B, ambos en P, es elegir un par de instancias de B, llamémoslas B_v con respuesta «verdadero» y B_f con respuesta «falso»; dada la consulta ¿$x \in A$? resolvemos ese problema — en tiempo polinomial, por hipótesis —, y lo asociamos a la instancia B_v o B_f según la respuesta). Es falso.

   (d) Ya lo respondimos en la respuesta 4b. Es verdadero.

3. (a) No es decidible.
   Reducimos el HALTING PROBLEM a este problema. Dada una máquina de Turing M, que no incluye # ni $ en su alfabeto de cinta, la modificamos de la siguiente forma:
   - Modificamos los estados finales de M para ser no-finales, agregamos un estado final q_f y a los estados finales de M agregamos movidas que sobrescriben el símbolo leído con # y se mueven a la derecha, pasando a q_f.
   - Agrega un nuevo símbolo $ antes de la cinta de entrada. Si la máquina modificada lee $ (M se salió de la cinta por la izquierda), lo sobreescribe con #.

Llamemos M' al resultado. Es claro que M' escribe # en la cinta al procesar σ si y solo si M se detiene al procesar σ. Decidir este problema es decidir el HALTING PROBLEM, cosa que sabemos imposible.
Demostraciones similares son posibles partiendo de la no decidibilidad del lenguaje universal.

(b) Es decidible.
Este es un conjunto finito, podemos revisar todas las palabras para determinar si pertenecen a $\mathcal{L}(G)$ (podemos transformar G a la forma normal de Chomsky; sabemos que en esta forma la derivación de σ toma a lo más $2|\sigma| - 1$ pasos; basta revisar todas las derivaciones de largo a lo más $2 \cdot 2020 - 1 = 4039$ pasos).

(c) Es decidible, incluso trivial.
Dada la máquina de Turing M podemos crear infinitas máquinas de Turing que aceptan el mismo lenguaje agregando «relleno» en forma de estados inalcanzables.

(d) No es decidible.
Basta demostrar que algún caso particular no puede decidirse. La propiedad aceptada en el lenguaje $\mathcal{L}^2(M_2)$ es no trivial, por el teorema de Rice no puede decidirse si M_1 lo acepta.

4. La demostración es incorrecta:

   - No demuestra que **Quadratic Knapsack** está en NP.
   - Falta indicar que **Knapsack** es un problema NP-completo.
   - Debe reducir un problema NP-completo al problema entre manos, acá dice hacerlo al revés. Demostrar que **Quadratic Knapsack** $\leq_p$ **Knapsack** demuestra que **Quadratic Knapsack** está en NP (todos los problemas en NP se reducen polinomialmente a cualquier problema NP-completo) que no es lo que se busca.
   - No indica que es una reducción polinomial.
   - La «reducción» es incorrecta: una reducción debe tomar cualquier instancia de **Quadratic Knapsack** y entregar una instancia de **Knapsack**. Lo que demuestra es **Knapsack** $\leq_p$ **Quadratic Knapsack** (agregar los $a_{ij} = 0$ a la instancia de **Knapsack** tomará tiempo proporcional a n^2 si son n ítem, polinomial en el tamaño de la representación del problema).

   La conclusión es correcta (un certificado para **Quadratic Knapsack** que demuestra que está en NP es un conjunto de ítem que cumple las restricciones; la discusión del último error reportado provee una reducción correcta **Knapsack** $\leq_p$ **Quadratic Knapsack**).

5. a) **recursivamente enumerable:** si es aceptado por una TM. Alternativamente, existe una TM determinista con multcinta, con una cinta de salida que va enumerando las palabras del lenguaje separadas por algún símbolo en especial.

   b) **problema en NP:** está en NP si hay un polinomio p y una TM no-determinista M tal que si $w \in L$ hay una computación de M de largo a lo más p(—w—) que acepta w. Alternativo, para todo $w \in L$ hay un certificado c tal que hay una TM que acepta w,c en a lo más p(—w—) pasos.

   c) **problema NP-completo:** es completo si está en NP y es NP-duro... todos los problemas NP pueden reducirse a él.

   d) **reducción polinomial:** una rp del problema A al problema B es una función f que traduce la instancia w del problema A (?‘$w \in A$?) a una instancia del problema B (?‘$f(w) \in B$?) tal que ambos tengan la misma respuesta y tal que f(w) pueda ser computado por una TM que siempre se detiene y está acotado por un polinomio —w—. Si tenemos cómo resolver las instancias de B en tiempo polinomial, podemos resolver A en tiempo polinomial a través de las instancias de B.