Clases de Complejidad de Problemas
Primavera 2020

0. El problema del viajante. Trabajando como ingeniero, cualquiera de nosotros podría recibir como asignación la siguiente:

Desarrollar un algoritmo que reciba un conjunto de puntos a visitar, junto con el costo de traslado entre dos cualesquiera de esos puntos, y retorne el circuito de costo mínimo que partiendo de uno de los puntos pase por todos los otros y retorne.

No sería improbable que nuestro jefe añadiera: "La solución obvia puede ser un poco ineficiente. Por favor investiguen algoritmos de tiempo óptimo de ejecución".

Enfrentados a esta (no demasiado artificial) situación, ¿cuál debería ser nuestra respuesta?

Bueno, uno modelaría el parámetro del problema como un grafo, no dirigido, tal que entre dos vértices cualesquiera diferentes existe una arista con su costo (digamos entero positivo). Luego lo que se pide es el *circuito hamiltoniano* de mínimo costo en el grafo dado[1].

Algo de entrenamiento en algoritmos en grafos ayudaría a identificar este problema como el *problema del viajante (de comercio)* (*Traveling Salesman Problem* en inglés). Y, siendo esto más importante, a identificar rápidamente la solución obvia de la que habla nuestro jefe.

Ella no es otra que el algoritmo de "fuerza bruta", es decir: generar todos los circuitos hamiltonianos y calculando por simple sumatoria el costo de cada uno de ellos, quedarse con aquel de costo mínimo.

¿Cuán "ineficiente" es esta solución? Bien, cierto entrenamiento en Matemática Discreta nos puede ayudar con cierta predicción (es decir, cálculos *anteriores* al desarrollo de una implementación).

Un circuito hamiltoniano queda determinado por un ordenamiento de los vértices del grafo, lo cual no será otra cosa que el orden en que los vértices serán visitados –entendiéndose que luego de visitar el último debe retornarse al primero.

¿Cuńtas maneras existen de ordenar un conjunto de n vértices? Bueno, pues, tantas como permutaciones de los n elementos, es decir, $n!$. De modo que nuestra solución obvia debe calcular $n!$ costos de circuitos. ¿Es eso demasiado costoso en tiempo de máquina?

La respuesta es: *"Oh, sí, realmente!"*, como puede verse por ejemplo considerando el módico caso $n = 20$. la factorial de 20 tiene 19 dígitos. Si postulamos que el costo de cada circuito puede calcularse en un (extremadamente favorable) tiempo de un micro segundo, entonces nos queda un número de 13 dígitos que equivale al tiempo en segundos necesario para ejecutar nuestra solución. Eso da un número de *nueve* dígitos para el mismo tiempo medido

[1] *Circuito* por ser un camino cerrado, y *hamiltoniano* por pasar por todos los vértices.

en horas...y si continuamos con el cálculo llegaremos a una medida superior a los 77000 años. Realmente, decir que la solución obvia puede ser algo ineficiente es en este caso una burda subestimación de la situación. Se trata de una solución totalmente fuera de toda factibilidad práctica.

Esto es una característica de las soluciones (algoritmos) *exponenciales* y por ello una división básica entre soluciones factibles y no factibles en la práctica que se utiliza en Computación es la de algoritmos *polinómicos* y *exponenciales*.

Ahora bien, la cuestión entonces debe ser hallar una solución no obvia y práctica al problema del viajante. Quienes intenten ese camino probablemente puedan chocarse una y otra vez contra la necesidad de calcular el mínimo de un conjunto de valores sin compararlos a todos ellos. De hecho, ocurre que *nadie* hasta el momento ha:

- Desarrollado un algoritmo distinto del obvio para resilver el problema del viajante.
- Ni demostrado que tal algoritmo no puede existir.

Se trata de una de las tantas expresiones del problema abierto más célebre y relevante de la Ciencia de la Computación, uno cuyas bases teóricas vamos a introducir a continuación.

De modo que, como ingenieros o informáticos, nuestra respuesta a aquel hipotético jefe debería ser bastante negativa. Deberíamos encarar un problema menos exigente para poder arribar a soluciones factibles de ser empleadas, por ejemplo admitiendo soluciones subóptimas. Esta situación se da con muchos problemas de amplia aplicación, muy frecuentemente de optimización, y es necesario conocer la existencia, fundamentos y extensión de la clase de problemas involucrada.

1. Problemas de decisión. En vez de problemas de optimización y otras índoles es conveniente y apropiado estudiar el siguiente formato de problemas:

Un *problema de decisión* es una pareja $(\mathcal{D}, \mathcal{Q})$, donde $\mathcal{D}$ es un *tipo* (*estructura*) de datos y $\mathcal{Q}$ un *predicado* sobre $\mathcal{D}$.

La idea es que el predicado plantea un *pregunta de respuesta binaria* (sí o no) sobre ciertos *parámetros* que conforman la estructura de datos del problema. El predicado puede también vere como una *función booleana total* sobre los parámetros del problema. Habitualmente se usa Π (posiblemente con primas o subíndices) para designar problemas genéricos.

Ejemplos.

1. *TSP* es el problema del viajante como problema de decisión, donde los parámetros son:
   - un grafo no dirigido G con una arista ponderada con costo entero positivo entre cualquier par de vértices diferentes, y
   - un entero B,

y el predicado es el siguiente: G contiene un circuito hamiltoniano de costo total $\leq B$.

Notar que teniendo una solución cualquiera al problema de optimización tenemos una de complejidad computacional equivalente inmediata al problema de decisión, la cual surge de simplemente comparar el costo mínimo con la cota B dada. Luego, si llegamos a conclusiones negativas sobre existencia de soluciones eficientes al problema de decisión podemos concluir lo mismo respecto del problema de optimización.

2. *SAT*, cuyo parámetro es una fórmula de Lógica Proposicional (en un lenguaje fijo, digamos el determinado por letras, negación, disyunción y conjunción) y el predicado de satisfactibilidad (es decir, aquel que se cumple cuando para al menos una fila de tabla de verdad de la fórmula en cuestión, ésta toma el valor *Verdadero*).

3. *HC*, dado por un grafo no dirigido y el predicado que vale cuando el grafo en cuestión contiene al menos un circuito hamiltoniano.

4. *OBS*, dado por el tipo de las listas finitas binarias (es decir, de ceros y unos o de booleanos) y el predicado de estar ordenada en forma creciente (es decir, no existe un cero en posición posterior a la de un uno).

2. La clase P. Un problema de decisión está en la clase P ssi su predicado es computable *en tiempo polinomial* respecto del *tamaño* de los parámetros. La definición de *tamaño* es dependiente de la estructura de datos en cuestión.

Ejemplos. OBS $\in P$, siendo la medida del tamaño el *largo* de la lista en cuestión.

Sobre los otros ejemplos de la lista de arriba, la cuestión de su pertenencia a la clase P es un problema abierto.

3. La clase *NP*. No se conoce solución polinómica para el problema del viajante, pero si alguien sostuviera, para un grafo G y una cota B que la respuesta es positiva (i.e. *sí* o *Verdadero*) podría proporcionar evidencia (en este caso un circuito hamiltoniano en G con costo total $\leq B$) que sería *verificable* en tiempo polinómico.

Esta noción de *verificabilidad polinómica* caracteriza una clase de problemas, llamada *NP*:

Un problema de decisión $(\mathcal{D}, \mathcal{Q})$ está en la clase *NP* ssi existe un tipo de datos $\mathcal{E}$ y una relación $\mathcal{R}$ entre $\mathcal{D}$ y $\mathcal{E}$ tal que

1. $(\forall d)(\forall e)((d, e) \in \mathcal{R} \Rightarrow \mathcal{Q}(d))$,

2. $(\forall d)(\mathcal{Q}(d) \Rightarrow (\exists e)(d, e) \in \mathcal{R}))$, y

3. R es *decidible en tiempo polinómico* en el tamaño de $\mathcal{D}$. Es decir, existe un algoritmo polinómico en el tamaño de $\mathcal{D}$ que determina, para cualquier pareja $(d.e)$ dada, si ella pertenece o no a $\mathcal{R}$.

Los elementos de $\mathcal{E}$ que estén en la relación $\mathcal{R}$ con algún elemento de $\mathcal{D}$ son llamados *evidencias* o *certificados*.

Ejemplos.

1. $TSP \in NP$. Para verificar esto, pensemos qué es una *evidencia* de que un grafo no dirigido y ponderado G contiene al menos un circuito hamiltoniano de costo total $\leq B$ para B dado. Bueno, la evidencia en este caso no debería ser otra que cualquiera de los propios circuitos hamiltonianos en cuestión, lo cual a su vez no es más que una enumeración de los vértices del grafo tal que la suma de los costos de las aristas entre dos vértices consecutivos de la enumeración, más el costo de la arista entre el último vértice y el primero, es efectivamente $\leq B$. Vamos a detallar la verificación de que esta idea satisface la definición que acabamos de dar:

   (a) En este caso, $\mathcal{D} = (G, B)$ donde G es el grafo y B la cota elegida para el costo de las recorridas hamiltonianas.

   (b) $\mathcal{E}$ es el conjunto de enumeraciones (permutaciones) de los vértices de G. Cada una de éstas da lugar a un circuito hamiltoniano, i.e. una recorrida completa de los vértices de G realizada en el orden en el que éstos aparecen en e, agregando la arista que une el útlimo vértice al primero para volver así al lugar de origen.

   (c) Definimos la relación $\mathcal{R}$ entre $\mathcal{D}$ y $\mathcal{E}$ simplemente como las parejas $((G, B), e)$ tales que el costo del circuito e (es decir, la suma de los costos de las aristas entre vértices consecutivos de e más el costo de la arista entre el último vértice de e y el primero) es $\leq B$.

   (d) Es claro que si una estructura $d = (G, B)$ tiene asociada una evidencia es porque satisface el predicado del problema. Recíprocamente, si d satisface el predicado del problema, existe al menos una evidencia de esto tal como las hemos definido. Luego las condiciones 1. y 2. de la definición de la relación $\mathcal{R}$ entre instancias del problema y evidencias se cumplen.

   (e) $\mathcal{R}$ es claramente decidible en tiempo polinomial. Basta recorrer la permutación de vértices e ir sumando los costos de aristas entre vértices consecutivos (incluyendo la arista entre el último vértice y el primero) y luego comparar con la cota B. Esto puede hacerse en tiempo polinomial en el tamaño de G.

2. $SAT \in NP$. Para verificarlo, simplemente indicamos:

   (a) $\mathcal{D}$ es el lenguaje fijo de fórmulas proposicionales al que nos referimos arriba.

   (b) $\mathcal{E}$ es el tipo de (una representación de) "filas de tabla de verdad" o, más técnicamente hablando, de *asignaciones* de valores de verdad a todas las letras (variables) proposicionales. Basta considerar aquellas que asignan sólo una cantidad finita de *verdadero*s, porque en cada fórmula la cantidad de letras participantes es finita. Luego,

las asignaciones relevantes se pueden representar mediante listas de letras. Las letras que aparezcan en la lista serán interpretadas como verdaderas y las que no, como falsas.

(c) Una evidencia de que una fórmula es satisfactible es una asignación que la haga verdadera. Esto define la relación $\mathcal{R}$ requerida. En otras palabras, cada fórmula tiene como correspondientes en la relación $\mathcal{R}$ a todas aquellas asignaciones que la hacen verdadera.

(d) $\mathcal{R}$ es decidible mediante un algoritmo que evalúa la fórmula para la asignación dada. Esto se puede hacer claramente en tiempo polinómico en el tamaño de la fórmula.

3. $OBS \in NP$. En este caso, el hecho de que $OBS \in P$ adquiere una relevancia primordial. De hecho, podemos en este problema decidir en forma polinomial si una instancia (lista binaria) dada satisface o no el predicado (estar ordenada), con prescindencia total de cualquier otra evidencia. En otras palabras, el dato mismo del problema es toda la evidencia necesaria, porque es tratable polinómicamente. Simplemente para verificar que la definición de clase NP se aplica en este caso, tomemos $\mathcal{E}$ como un tipo de un solo elemento (sea éste $\heartsuit$) y $\mathcal{R}$ como las parejas $(ls, \heartsuit)$ tales que ls está ordenada.

4. P **versus** NP**.**

Teorema. $P \subseteq NP$.

Demostración. Generalizar la construcción precedente a un problema cualquiera de la clase P.

□

El recíproco es el ***problema abierto*** más célebre y relevante de la Ciencia de la Computación.

Puede verse como la cuestión de determinar si la factibilidad de *verificación* (es decir, su carácter polinómico) implica la factibilidad de *solución*[2].

?1. Demostrar que $HC \in NP$.

5. Porqué del nombre NP**.**

Otra caracterización de la clase NP (históricamente anterior a la dada arriba) tiene que ver con la "resolución" de los problemas en cuestión mediante *algoritmos no determinísticos*.

Un algoritmo no determinístico es la generalización del concepto de algoritmo que se obtiene cuando admitimos que, en cualquier punto dado de la ejecución del algoritmo en cuestión, éste pueda tener asociado un *conjunto* de pasos subsiguientes. En otras palabras, el paso subsiguiente de ejecución no queda en general unívocamente determinado por el estado actual de la computación sino apenas restringido a un conjunto de pasos posibles.

[2]Puesto de esta manera, uno parece ser llevado a inclinarse por la opinión de la mayoría abrumadora de los expertos, que *conjetura* que $P \neq NP$.

Una formulación más concreta del concepto exige ser realizada en el contexto de un cierto modelo de computabilidad. Así, por ejemplo, podemos pensar en Máquinas de Turing no determinísiticas como aquellas en las que el estado actual y el símbolo leído no determinan una única acción y estado subsiguiente, sino un conjunto de parejas (acción, estado subsiguiente).

Una *ejecución* de un algoritmo no determinístico es lo que se obtiene al realizar en cada estado de no determinación una *elección* arbitraria del paso subsiguiente.

En general, al estudiar algoritmos no determinísticos, nos interesarán propiedades del conjunto de sus posibles ejecuciones. Por ejemplo, querremos determinar si en *todas ellas* se alcanza determinado resultado o, alternativamente, si *al menos alguna* alcanza un resultado.

El concepto es sobre todo una abstracción útil como etapa intermedia en el desarrollo de soluciones algorítmicas de problemas. Los algoritmos realmente implementados siguen siendo determinísticos, como lo son todos los programables en los diversos modelos de computabilidad empleados.

En el presente contexto nos restringiremos a un modelo específico de algoritmo no determinístico. Éste constará de:

- Un módulo de *conjetura* (inglés: *guess*) donde no determinísticamente se selecciona una cierta estructura de datos elegida de un conjunto de posibilidades determinado por la entrada proporcionada al algoritmo.
- Un módulo de *verificación* que trabaja sobre la entrada dada y el dato conjeturado en el paso precedente para producir una salida booleana.

Ahora decimos que uno de estos algoritmos *resuelve* un problema de decisión $(\mathcal{D}, \mathcal{Q})$ ssi para cada entrada $d : \mathcal{D}$, si $\mathcal{Q}(d)$ vale, entonces existe al menos una estructura producible por el módulo de conjetura que conduce al módulo de verificación a responder *Verdadero*; y si $\mathcal{Q}(d)$ no vale, entonces ninguna de las estructuras producibles por el módulo de conjetura conduce al módulo de verificación a responder *Verdadero*.

El *costo (complejidad) en tiempo* de uno de estos algoritmos será el de su módulo de verificación.

Y, finalmente, caracterizamos a la clase *NP* de problemas de decisión como aquella cuyos miembros son los resolubles mediante un algoritmo *no determinístico polinómico*.

De esta caracterización es que surge el nombre *NP* (*N*o determinísticamente *P*olinómico).

Las dos caracterizaciones de la clase *NP* dadas son claramente equivalentes. Para verificar esto, basta identificar las estructuras $\mathcal{E}$ de la primera caracterización con aquellas generadas por el módulo de conjetura en la segunda.

Se puede demostrar también que todo problema *NP* tiene una solución determinística exponencial (aplicando "fuerza bruta").

6. *NP*-completitud.

Reducción polinómica. Podemos comparar problemas por su complejidad en tiempo mediante la siguiente relación (llamada de *reducción polinómica*):

$(\mathcal{D}_1, \mathcal{Q}_1) \leq_p (\mathcal{D}_2, \mathcal{Q}_2)$ ssi existe una función total $f : \mathcal{D}_1 \to \mathcal{D}_2$ computable en tiempo polinómico tal que $(\forall d : \mathcal{D}_1)(\mathcal{Q}_1(d) \Leftrightarrow \mathcal{Q}_2(f(d)))$.

De este modo, se tiene una transformación computable en tiempo polinómico de cada instancia de Π_1 en una de Π_2 de modo que los predicados de los problemas se respetan, es decir, instancias que satisfacen el predicado del primer problema son transformadas en instancias que satisfacen el segundo, mientras que las que no satisfacen el predicado son asimismo transformadas en instancias que no satisfacen el segundo predicado.

Leemos $\Pi_1 \leq_p \Pi_2$ así:

- Π_1 es *reducible polinómicamente* a Π_2 o, también,
- el problema Π_2 es (computacionalmente) al menos tan complejo como Π_1.

Esta última lectura queda justificada por el siguiente ejercicio:

?2. Demostrar que $\Pi_1 \leq_p \Pi_2 \Rightarrow (\Pi_2 \in P \Rightarrow \Pi_1 \in P)$.

Ejemplos. $HC \leq_p$ TSP.

Paa demostrar esto, debemos transformar cada instancia de *HC*, es decir un grafo no dirigido, en una de *TSP*, es decir, un grafo con aristas ponderadas con enteros positivos entre cualquier par de vértices distintos y un número B, de modo que el primer grafo contiene un ciclo hamiltoniano ssi el segundo contiene al menos un ciclo hamiltoniano de costo $\leq B$. Una transformación que logra el cometido es la siguiente; los vértices del segundo grafo son los mismos que los del primero; y en el segundo grafo hay, como se requiere, siempre una arista entre dos vértices diferentes a y b. El peso de esta arista será 1 si la arista ya existía en el grafo original y 2 en el otro caso. Luego el número B a emplear como cota en el *TSP* resultante es el número de vértices.

?3. Verificar la corrección de la transformación precedente, i.e. que ella cumple los requisitos para establecer $HC \leq_p TSP$.

Problemas NP-completos. Un problema Π es *NP-completo* ssi

1. Es *NP*.
2. Todo problema de la clase *NP* es reducible polinómicamente a él, es decir, $(\forall \Pi' \in NP)\ \Pi' \leq_p \Pi$.

En otras palabras, los problemas *NP*-completos son los más complejos, computacionalmente hablando, de la clase *NP*. La utilidad del concepto deriva del siguiente ejercicio:

?4. Demostrar: Sea Π un problema *NP*-completo cualquiera. Entonces $\Pi \in P \Rightarrow P = NP$.

Ahora bien, ¿Existen problemas *NP*-completos? El punto fundacional de esta teoría es el siguiente resultado:

Teorema. (Cook, 1971) *SAT* es *NP*-completo.

Y luego de ese disparador, se tiene el siguiente mecanismo de propagación:

?5. Π *NP*-completo $\Rightarrow (\forall \Pi')(\Pi \leq_p \Pi' \Rightarrow \Pi'$ *NP*-completo).

Así por ejemplo, sabiendo que *HC* es *NP*-completo, podemos concluir que también lo es *TSP*, pr la reducción polinómica efectuada arriba. A su vez, *HC* *NP*-completo se demuestra por una serie de reducciones polinómicas desde *SAT*.

Extensas listas de problemas *NP*-completos pueden consultarse en varios sitios. Muchos de ellos tienen asociadas una variedad de aplicaciones en la industria y otras actividades.