- .ipynb
-

Clases de complejidad y P vs. NP

Contenido

Clases de complejidad y P vs. NP #

IntroducciÃ³n#

Los problemas algorÃtmicos estÃ¡n clasificados en **clases de complejidad** de acuerdo al algoritmo mÃ¡s rÃ¡pido que conocemos para resolverlos. En este capÃtulo hablaremos de algunas de las clases de complejidad mÃ¡s famosas: P, NP, NP-completo y NP-difÃcil. La primera es la de los problemas que podemos responder rÃ¡pido. La segunda es de los problemas cuya soluciÃ³n podemos verificar rÃ¡pido. Las Ãºltimas dos son clases de problemas que al resolverlos rÃ¡pido podrÃamos resolver cualquier otro en NP. Esto nos llevarÃ¡ al problema abierto en teorÃa computacional mÃ¡s importante: P vs. NP.

Las clase de complejidad P#

Todo lo que discutiremos en estas secciones se refiere a problemas de decisiÃ³n. Como platicamos anteriormente, podemos pensar a los problemas de decisiÃ³n como aquellos en los cuales la respuesta es sÃ o no. Aunque de momento nos limitaremos a esa clase de problemas, hay maneras de extender lo que discutimos a otro tipo de problemas algorÃtmicos.

La **clase de complejidad** P consiste de aquellos problemas algÃ³ritmicos que podemos resolver en tiempo polinomial. Por eso la letra P. Con esto nos referimos a que para ellos debe existir un entero $k \geq 0$ y un algoritmo tal que cualquier instancia de tamaÃ±o n la resuleva en tiempo $O(n^k)$.

En el transcurso de este libro hemos encontrado varios problemas que estÃ¡n en la clase P, por ejemplo:

- Dado un conjunto de n nÃºmeros, ver si hay dos de ellos con suma igual a un nÃºmero dado M. El problema estÃ¡ en P pues un algoritmo trivial que verifica todas las $\binom{n}{2}$ parejas corre en tiempo $O(n^2)$. De hecho, ya vimos que se puede mejorar este tiempo, pues dimos un algoritmo que corre en tiempo $O(n\log n)$.
- En general, si tenemos un entero $k\geq 0$ fijo, ver si dados n nÃºmeros hay k de ellos cuya suma es igual a un nÃºmero dado M. EstÃ¡ en la clase P pues para resolverlo en tiempo $O(n^k)$ basta verificar cada una de las $\binom{n}{k}$ posibilidades de subconjuntos de k elementos.
- Dada una grÃ¡fica en n vÃ©rtices, decir si hay un camino entre dos vÃ©rtices o no. EstÃ¡ en P pues podemos hacer BFS o DFS en tiempo $O(n+m)=O(n^2)$. De hecho, ver si la grÃ¡fica es conexa tambiÃ©n estÃ¡ en P.
- Dada una lista de n nÃºmeros, saber si hay dos de ellos repetidos. La forma en la que resolvimos esto fue ordenando los nÃºmeros y viendo con una lectura lineal si en la lista ordenada hay dos iguales. Esto toma tiempo $O(n\log n)$, que tambiÃ©n es $O(n^2)$.
- Dada una grÃ¡fica en n vÃ©rtices, saber si hay dos de ellos a distancia 100. TambiÃ©n estÃ¡ en P pues el algoritmo de Floyd-Warshall nos permite calcular todas las distancias en tiempo $O(n^3)$, y

luego en tiempo \(O(n^2)\) podemos ver si alguna de ellas fue \(100\).

La clase de complejidad NP#

La **clase de complejidad** \(NP\) es algo distinta. Son aquellos problemas cuya respuesta Â«sÃÂ» se puede verificar en tiempo polinomial. Siendo un poco mÃ¡s precisos, lo que esperamos de los problemas en \(NP\) es que cada que la respuesta sea Â«sÃÂ» haya un **objeto testigo** que **testifique** que la respuesta sea Â«sÃÂ» y un **algoritmo verificador** que verifique la correctitud del testigo en tiempo polinomial. Es un poco mÃ¡s fÃ¡cil entender esto mediante problemas ejemplo.

Para empezar, es fÃ¡cil convencerse de que todos los problemas que mencionamos arriba estÃ¡n en la clase \(NP\). Argumentemos eso para algunos como primeros ejemplos.

- En el problema de dar dos nÃºmeros cuya suma sea \(M\), lo que podemos usar para poder responder Â«sÃÂ» como testigo es a dichos dos nÃºmeros \(a\) y \(b\). Un algoritmo verificador puede en tiempo \(O(1)\) sumar estos dos nÃºmeros y darse cuenta de que en efecto suman \(M\) (o no).
- En el problema de decir si una grÃ¡fica es conexa, lo que podemos usar para poder responder Â«sÃÂ» como testigo es una lista de caminos entre cualesquiera dos vÃ©rtices. Un algoritmo verificador puede tomar cada uno de estos \(\binom{n}{2}\) caminos y en tiempo \(O(n)\) ver si son caminos vÃ¡lidos, verificando que en efecto consistan de aristas de la grÃ¡fica.

De hecho, despuÃ©s de pensarlo un poco no es difÃcil convencerse de que todos los problemas que estÃ¡n en \(P\) tambiÃ©n estÃ¡n en \(NP\). Sin embargo, lo opuesto no es nada claro: si podemos verificar rÃ¡pido la respuesta, Â¿serÃ¡ que podemos resolver rÃ¡pido el problema? No lo sabemos. Hay problemas que estÃ¡n en NP, pero que **nadie sabe** si estÃ¡n en la clase P. Veamos algunos ejemplos.

Ejemplo. El problema **suma de subconjuntos** pregunta lo siguiente: Â«Dado un conjunto \(X\) de \(n\) nÃºmeros, ver si hay algunos de ellos con suma \(M\).Â»

Veamos que estÃ¡ en la clase \(NP\). Para poder responder que sÃ, podemos usar como testigo un subconjunto \(Y\subseteq X\) con suma igual a \(M\). Como \(Y\) tiene a lo mÃ¡s \(n\) nÃºmeros, un algoritmo verificador puede sumarlos en tiempo \(O(n)\) y decirnos si es un testigo vÃ¡lido o no. Entonces, es rÃ¡pido verificar soluciones.

Pero, Â¿es rÃ¡pido resolver el problema? Un algoritmo inocente recorrerÃa todos los subconjuntos de \(X\) para ver si la suma de los elementos de alguno es \(M\). Esto tomarÃa tiempo \(\Theta(2^n)\). Â¿Podemos hacerlo mÃ¡s rÃ¡pido? En esencia, nadie sabe. Los mejores algoritmos que se conocen corren en tiempo exponencial en \(n\). En particular, nadie sabe si puede responderse en tiempo polinomial.

Ejemplo. El problema **trayectoria hamiltoniana** pregunta lo siguiente: Â«Dada una grÃ¡fica, encontrar si hay una trayectoria que pase por todos los vÃ©rticesÂ». \(\square\)

El problema estÃ¡ en \(NP\). Un testigo de la respuesta Â«sÃÂ» es el orden en el que se recorren los vÃ©rtices. Un algoritmo verificador puede irla recorriendo. Si en algÃºn momento pasa por mÃ¡s de \(n\) vÃ©rtices, ya no es trayectoria. Si tiene \(n\) o menos, entonces en tiempo \(O(n)\) puede ver si en efecto se usan las aristas de la grÃ¡fica de manera vÃ¡lida, y en tiempo \(O(n\log n)\) que no se repitan vÃ©rtices.

AsÃ, verificar un testigo es rÃ¡pido pero, Â¿resolver el problema? Un algoritmo inocente serÃa probar con cada una de las \(n!\) formas de recorrer los vÃ©rtices y ver si en efecto pasan por aristas de la grÃ¡fica. Esto tarda mucho, pues toma tiempo \(\Theta(n!)\). Â¿Se podrÃ¡ bajar a tiempo polinomial? Nadie sabe.

Ejemplo. Veamos un Ãºltimo ejemplo, que viene de la lÃ³gica. Se llama el **problema 3-SAT**. Tomemos \(x_1,\ldots,x_n\) variables binarias (pueden ser falsas o verdaderas). Tomemos una expresiÃ³n del siguiente estilo: \(\square\)

\[\begin{align*} E=(a_{11}\lor a_{12} \lor a_{13}) \land (a_{21}\lor a_{22} \lor a_{23}) \land \ldots \land (a_{m1}\lor a_{m2}\lor a_{m3}) \end{align*}\]

AquÃ cada \(a_{ij}\) es alguna variable \(x_k\) o su negaciÃ³n. A cada \((a_{j1}\lor a_{j2} \lor a_{j3})\) le llamamos una **clÃ¡usula**.

Estamos pensando que la expresiÃ³n ya estÃ¡ simplificada, es decir, que no hay clÃ¡usulas redundantes. Notemos que entonces \(m\) es como mucho \(8\binom{n}{3}\): como mucho podemos tener tantas clÃ¡usular como formas de elegir \(3\) variables, y para cada una de ellas hay \(8\) formas de elegir cuÃ¡les estÃ¡n negadas y cuÃ¡les no.

Â¿Podremos dar una asignaciÃ³n de verdad a las variables \(x_1,\ldots,x_n\) de modo que la expresiÃ³n en total sea verdadera?

Este es un problema en NP pues un testigo serÃa la asignaciÃ³n de verdad que funciona. Como hay a lo mucho \(8\binom{n}{3}\) clÃ¡usulas, podemos verificar si en efecto la expresiÃ³n evalÃºa a verdadero en tiempo \(O(n^3)\).

Pero, suena muy difÃcil encontrar dicha asignaciÃ³n. Hay \(2^n\) formas de hacer asignaciones para las \(n\) variables y pasar por todas tomarÃa tiempo \(O(2^n)\). Â¿Se puede encontrar una asginaciÃ³n en tiempo polinomial? Nadie sabe.

Reducciones y la clases NP-completo y NP-difÃcil#

\(\square\)

Queremos hablar de otras dos clases computacionales mÃ¡s, pero para ello necesitamos unas pocas definiciones. La siguiente definiciÃ³n refleja la idea de cuÃ¡ndo podemos usar un problema para resolver otro.

DefiniciÃ³n. Dados problemas algorÃtmicos \(A\) y \(B\), vamos a decir que existe una **reducciÃ³n del problema \(A\) al problema \(B\)** si podemos pasar (algoritmicamente) cualquier instancia \(I\) del problema \(A\), a una instancia \(\varphi(I)\) del problema \(B\), con la propiedad de que la respuesta para \(I\) en \(A\) es Â«sÃÂ» si y sÃ³lo si la respuesta para \(\varphi(I)\) en \(B\) es Â«sÃÂ».

Cuando hay una reducciÃ³n de \(A\) a \(B\) podemos usar a \(B\) como subrutina para resolver \(A\). Supongamos que una instancia de tamaÃ±o \(n\) de \(A\) pasa a una de tamaÃ±o a lo mÃ¡s \(T(n)\) de \(B\) en tiempo \(g(n)\). Supongamos que las instancias de \(B\) de tamaÃ±o \(n\) se pueden resolver en tiempo \(f(n)\). Entonces podemos resolver una instancia de \(A\) de tamaÃ±o \(n\) en \(g(n)+f(T(n))\) pasos.

Supongamos por el momento que el tamaÃ±o de intancia de la reducciÃ³n no crece mucho y sigue siendo \(O(n)\). Supongamos tambiÃ©n que hacer el cambio de instancia es muy rÃ¡pido, menos de lo que tarda resolverla en \(B\). Entonces \(g(n)+f(T(n))=O(f(n))\). Tenemos entonces intuitivamente que Â«si \(B\) se puede resolver rÃ¡pido, entonces \(A\) se puede resolver rÃ¡pidoÂ». La contrapositiva es muy interesante tambiÃ©n: Â«si tenemos la garantÃa de que \(A\) no se puede resolver rÃ¡pido, entonces \(B\) tampoco se puede resolver rÃ¡pidoÂ». Esto da un mÃ©todo para encontrar cotas inferiores para el tiempo de resoluciÃ³n de ciertos problemas algorÃtmicos.

Para definir la clase \(NP\), nos interesa una definiciÃ³n un poco mÃ¡s restrictiva de reducciÃ³n.

DefiniciÃ³n. Diremos que hay una **reducciÃ³n polinomial de** \(A\) **a** \(B\) si hay una reducciÃ³n del problema \(A\) al problema \(B\) en donde ademÃ¡s se cumple que hay enteros positivos \(k\) y \(\ell\) tales que para cualquier instancia \(I\) de \(A\):

- Obtener \(\varphi(I)\) toma tiempo \(O(|I|^k)\).
- La instancia \(\varphi(I)\) tiene tamaÃ±o \(O(|I|^\ell)\).

La discusiÃ³n anterior nos lleva a lo siguiente.

ObservaciÃ³n. Si hay una reducciÃ³n polinomial de \(A\) a \(B\), y \(B\) estÃ¡ en la clase de complejidad \(P\), entonces \(A\) tambiÃ©n estÃ¡ en la clase de complejidad \(P\).

Estamos listos para definir las dos Ãºltimas clases que nos faltan.

La **clase de complejidad** \(NP\)-difÃcil consiste de aquellos problemas \(B\) tales que para cualquier problema \(A\) en \(NP\) se cumple que hay una reducciÃ³n polinomial de \(A\) a \(B\). La **clase de complejidad** \(NP\)-completo consiste de aquellos problemas en \(NP\)-difÃcil, que ademÃ¡s estÃ¡n en \(NP\).

Â¿QuÃ© sucederÃa si algÃºn problema en \(NP\)-difÃcil lo pudiÃ©ramos resolver en tiempo polinomial? Por la observaciÃ³n anterior tendrÃamos que *cualquier* problema en \(NP\) podrÃamos resolverlo en tiempo polinomial. PasarÃa lo mismo si algÃºn problema en \(NP\)-completo lo pudiÃ©ramos resolver en tiempo polinomial.

El problema P vs. NP#

El primer problema que se demostrÃ³ que era \(NP\)-completo es el problema SAT, una versiÃ³n mÃ¡s general del problema lÃ³gico que discutimos arriba. Esto lo hicieron Stephen Cook y Leonid Levin. Luego, Richard Karp encontrÃ³ muchos otros mÃ¡s. De hecho, se dio cuenta de que 21 problemas de fuerte interÃ©s para la comunidad computacional eran \(NP\)-completos.

De ahÃ en adelante se ha demostrado que muchÃsimos problemas son \(NP\)-completos. Algunos de ellos son muy tÃ©cnicos, pero tambiÃ©n hay otros muy naturales tanto en el contexto matemÃ¡tico, como en el computacional, como en el aplicado. En la siguiente lista se mencionan algunos, comenzando con los tres que vimos anteriormente que estÃ¡n en \(NP\).

- Suma de subconjuntos.
- Trayectoria hamiltoniana.
- 3-SAT
- NÃºmero de clique de una grÃ¡fica.
- NÃºmero cromÃ¡tico de una grÃ¡fica.
- Problema del agente viajero: el que hemos discutido como el del brazo del robot.
- NÃºmero de empacamiento.
- Encontrar cortes mÃ¡ximos.

En esta lista hay algunos problemas que definimos inicialmente como optimizaciÃ³n. Para transformarlos en problema de decisiÃ³n, agregamos, por ejemplo, un valor \(k\) a la entrada y los transformamos al problema de optimizaciÃ³n de saber si hay un objeto cuya funciÃ³n a optimizar es mayor (o menor) que \(k\).

Muchos de los problemas \(NP\)-completos son fundamentales en varias Ã¡reas. Miles de personas, incluidos los expertos de dichas Ã¡reas, se los han encontrado y han intentado resolverlos en tiempo polinomial. Hasta ahora, nadie ha logrado resolver ninguno de ellos en tiempo polinomial. Ni nadie ha demostrado que para alguno de ellos esto sea imposible. Como son problemas \(NP\)-completos, basta con que uno de ellos estÃ© en \

(P\) para que todos lo estÃ©n. Si esto llegara a suceder, las clases de complejidad \(P\) y \(NP\) serÃan la misma y por lo tanto en un sentido muy amplio serÃa igual responder problemas que verificar su soluciÃ³n.

Por todas estas razones, la siguiente es una fundamental en matemÃ¡ticas y ciencias de la computaciÃ³n.

Problema. Â¿SerÃ¡ que \(P=NP\)?

Tarea moral#

Los siguientes problemas te ayudarÃ¡n a practicar lo visto en esta entrada. Para resolverlos, necesitarÃ¡s usar herramientas matemÃ¡ticas, computacionales o ambas.

1. Considera los siguientes problemas. Explica, por lo menos de manera informal, cÃ³mo constuirÃas testigos y verificadores para ver que estÃ¡n en NP:
    - Dada una grÃ¡fica \(G\) y un entero \(k\), decidir si \(G\) se puede colorear propiamente con \(k\) colores.
    - Dada una grÃ¡fica \(G\) con aristas ponderadas y un real \(r\), dedicir si existe una trayectoria de peso menor que \(k\) que pase por todas las aristas.
    - Dado un conjunto \(X\) de enteros positivos, saber si se puede hacer una particiÃ³n de \(X\) en conjuntos \(X_1\) y \(X_2\) tal que la suma de los elementos en \(X_1\) sea igual a la suma de los elementos en \(X_2\).
2. Justifica, por lo menos de manera un poco informal, por quÃ© todos los problemas de la clase \(P\) estÃ¡n en la clase \(NP\).
3. Para saber mÃ¡s del problema de \(P\) vs. \(NP\), y de otros Problemas del Milenio del Instituto Clay, revisa la lista reproducciÃ³n relacionada con ello en el canal de Arilinâs Math World.
4. En la definiciÃ³n de reducciÃ³n polinomial pedimos que \(\varphi(I)\) sea de tamaÃ±o polinomial en \(I\). Explica por lo menos intuitivamente por quÃ© esto es redundante, es decir, por quÃ© el hecho de que la reducciÃ³n toma tiempo polinomial ya implica esto.
5. Muestra con cuidado que el problema de encontrar un emparejamiento mÃ¡ximo en una grÃ¡fica bipartita tiene una reducciÃ³n polinomial el problema de encontrar un flujo mÃ¡ximo en una red.

anterior

Algoritmos de Ford-Fulkerson y Edmonds-Karp

siguiente

Problemas de grÃ¡ficas NP-completos

Contenido

Por Leonardo Ignacio MartÃnez Sandoval