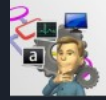
Análisis de algoritmos

Tema 07: Completitud NP

Contenido

- Introducción
- P y NP
- La clase P (Polinómicamente acotado)
- La clase NP (No determinista Polinómicamente acotado)
  - P $\subseteq$ NP
  - P vs NP
- NP-completo
- Problemas NP

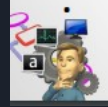


Introducción

- La mayoría de los algoritmos revisados hasta el momento no rebasan un orden de complejidad temporal $O(n^3)$.

- Podemos considerar que todos los algoritmos que son menores ha este orden tienen un tiempo de ejecución relativamente corto.

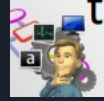
Función de coste	Tamaño n 10	20	30	40	50	60
n	0,00001 seg.	0,00002 seg.	0,00003 seg.	0,00004 seg.	0,00005 seg.	0,00006 seg.
n^2	0,0001 seg.	0,0004 seg.	0,0009 seg.	0,0016 seg.	0,0035 seg.	0,0036 seg.
n^3	0,001 seg.	0,008 seg.	0,027 seg.	0,064 seg.	0,125 seg.	0,316 seg.
n^5	0,1 seg.	3,2 seg.	24,3 seg.	1,7 min.	5,2 min.	13 min.
2^n	0,001 seg.	1 seg.	17,9 min.	12,7 días	35,7 años	366 siglos
3^n	0,059 seg.	58 min.	6,5 años	3855 siglos	10^8 siglos	10^{13} siglos



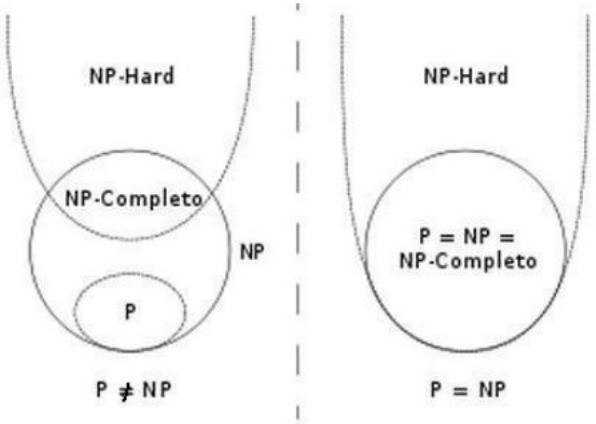
- Es posible observar que los algoritmos cuya complejidad se describe con **funciones polinómicas simples** se pueden ejecutar con entradas relativamente grandes en un tiempo razonable.

- Si la complejidad es 2^n, el algoritmo de nada sirve a menos que las entradas sean muy pequeñas.

- Existen problemas cuya complejidad podría describirse con funciones exponenciales, problemas cuya resolución incluso con los mejores algoritmos conocidos requeriría muchos años o siglos de tiempo de computadora con entradas moderadamente grandes. Por lo que se requiere distinguir de estos problemas a los problemas *dóciles* (es decir, "no tan difíciles") y los *renuentes* (es decir, "difíciles" o muy tardados).



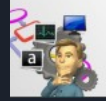
- También existen una clase de problemas que tienen una propiedad irritante: ni siquiera sabemos si se pueden resolver de manera eficiente o no; no se han descubierto algoritmos razonablemente rápidos para estos problemas, pero tampoco se ha podido demostrar que los problemas requieren mucho tiempo. Dado que muchos de estos problemas son problemas de optimización que se presentan con frecuencia en aplicaciones, la falta de algoritmos eficientes tiene importancia real.



- .

P y NP

- Los problemas computacionales los podemos dividir en dos conjuntos
  - **Tratables:** Problemas para los cuales existe un algoritmo de complejidad polinomial.
  - **Intratables:** Problemas para los que no se conoce ningún algoritmo de complejidad polinomial.
- A los problemas tratables se les conoce también como problemas **P** (de orden **polinomial**). Asimismo a los problemas no tratables se les llama también **NP** (de orden **no determinístico polinomial**).
  - Ejemplos de problemas NP
    - Agente Viajero
    - Caminos Hamiltonianos
    - Caminos Eulerianos

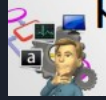


La clase P (Polinómicamente acotado)

- Decimos que un algoritmo está polinómicamente acotado si su complejidad de peor caso está acotada por una función polinómica del tamaño de las entradas (es decir, si existe un polinomio p tal que, para toda entrada de tamaño n, el algoritmo termine después de cuando más p(n) pasos).

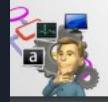
- Decimos que un problema está polinómicamente acotado si existe un algoritmo polinómicamente acotado para resolverlo.

- **P** es la clase de problemas de decisión que están polinómicamente acotados.



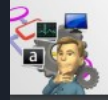
- Podría parecer un tanto extravagante utilizar la existencia de una cota de tiempo polinómica como criterio para definir la clase de problemas más o menos razonables: los polinomios pueden ser muy grandes. No obstante, hay varias razones de peso para hacerlo.

  - Si bien no es verdad que todos los problemas en P tengan un algoritmo de eficiencia aceptable, sí podemos asegurar que, si un problema no está en P, será extremadamente costoso y probablemente imposible de resolver en la práctica.



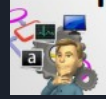
- Un motivo para usar una cota polinómica para definir P es que los polinomios tienen propiedades de “cierre”. Podríamos obtener un algoritmo para un problema complejo combinando varios algoritmos para problemas más sencillos. La complejidad del algoritmo compuesto podría estar acotada por la suma, multiplicación y composición de las complejidades de sus algoritmos componentes.

- Puesto que los polinomios están cerrados bajo estas operaciones, cualquier algoritmo que se construya de diversas formas naturales a partir de varios algoritmos polinómicamente acotados también estará polinómicamente acotado.



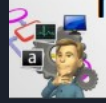
- Un motivo más para emplear una cota polinómica es que hace a P independiente del modelo de cómputo formal específico que se use.
- Los modelos difieren en cuanto al tipo de operaciones permitidas, los recursos de memoria disponibles y los costos asignados a diferentes operaciones.

- Un problema que requiere Θ(f(n)) pasos con un modelo podría requerir más de Θ(f(n)) pasos con otro, pero en el caso de prácticamente todos los modelos realistas, si un problema está acotado polinómicamente con uno, estará acotado polinómicamente con los demás.



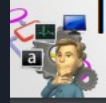
La clase NP (No determinista Polinómicamente acotado)

- Es el conjunto de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.
- NP es la clase de problemas de decisión para la cual **existe un algoritmo no determinista** polinómicamente acotado.
- NP es la clase de problemas de decisión para los que una **solución propuesta** dada con una entrada dada se puede verificar rápidamente (en tiempo polinómico) para determinar si realmente es una solución (es decir, si satisface los requisitos del problema)

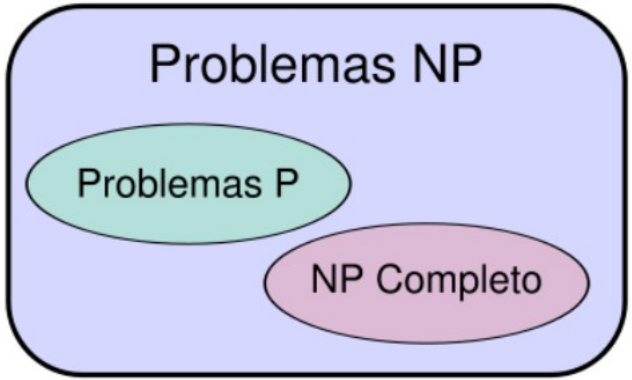


- La importancia de esta clase de problemas de decisión es que contiene muchos problemas de búsqueda y de optimización para los que se desea saber si existe una cierta solución o si existe una mejor solución que las conocidas.

- En esta clase están el problema del agente viajero donde se quiere saber si existe una ruta óptima que pasa por todos los nodos en un cierto grafo y el problema de satisfacibilidad booleana en donde se desea saber si una cierta fórmula de lógica proposicional puede ser cierta para algún conjunto de valores booleanos para las variables.



- Dada su importancia, se han hecho muchos esfuerzos para encontrar algoritmos que decidan algún problema de NP en tiempo polinómico. Sin embargo, pareciera que para algunos problemas de NP (los del conjunto NP-completo) no es posible encontrar un algoritmo mejor que simplemente realizar una búsqueda exhaustiva.

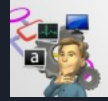


$P \subseteq NP$

- Un algoritmo determinista para resolver un problema de decisión es, con una modificación menor, un caso especial de un algoritmo no determinista. Si A es un algoritmo determinista para resolver un problema de decisión, basta con hacer que A sea un algoritmo no determinista.

- La pregunta importante es, ¿P = NP o es P un subconjunto propio de NP? Dicho de otro modo, ¿es el **no determinismo** más potente que el **determinismo** en el sentido de que algunos problemas se pueden resolver en tiempo polinómico con un "conjeturador" no determinista pero no se pueden resolver en tiempo polinómico con un algoritmo determinista?.

  - Ver en http://www.claymath.org/prizeproblems/index.htm cómo ganar un millón de dólares resolviendo la pregunta ¿P vs NP?



P vs NP

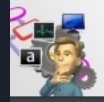
Cómo resolver la pregunta ¿P vs NP?

- Para intentar P ≠ NP:
  - Demostrar que hay un problema que está en NP pero no en P.

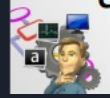
- Para intentar P=NP:
  - Demostrar que todos los problemas de NP se pueden resolver en tiempo polinómico, así que NP ⊆ P.

Con los NP-completos, esto se simplifica.

- Para intentar P=NP:
  - Demostrar que hay un problema NP-completo que se puede resolver en tiempo polinómico.



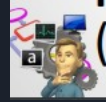
- P ≠ NP es equivalente a “existe un problema NP-completo que no está en P”
- P = NP es equivalente a “existe un problema NP-completo que está en P”
- Existe una larga lista de NP-completos (ver Garey Johnson).
- Añadir un problema a la vista quiere decir que el problema es tan difícil como cualquiera de los que ya están *(puedes decirle al jefe “No puedo encontrar un algoritmo eficiente, pero tampoco pueden un montón de científicos famosos”).*



Convención de nombres que incluyen las siglas NP

Los nombres de familias de problemas con las siglas NP es algo confusa. Los problemas NP-hard no son todos NP, a pesar de que estas siglas aparecen es el nombre de la familia. Sin embargo, los nombres están actualmente muy arraigados y plantear un cambio de nomenclatura resulta poco realista. Por otra parte, las familias de problemas con las siglas NP son todas definidas tomando como referencia la familia NP:

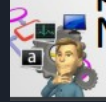
- **NP-completo —** Problemas que son completos en NP, es decir, los más difíciles de resolver en NP;
- **NP-hard** — (NP-difícil) quiere decir al menos tan complejo como NP (pero no necesariamente en NP);
- **NP-easy** — (NP-fácil) quiere decir como mucho tan difícil como NP (pero no necesariamente en NP);
- **NP-equivalente** — significa igualmente difícil que NP, (pero no necesariamente en NP).



NP-completo

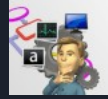
- La clase de complejidad **NP-completo es el subconjunto de los problemas de decisión en NP tal que todo problema en NP se puede reducir en cada uno de los problemas de NP-completo**. Se puede decir que los problemas de NP-completo son los problemas más difíciles de NP y muy probablemente no formen parte de la clase de complejidad P.

- La razón es que de tenerse una solución polinómica para un problema NP-completo, todos los problemas de NP tendrían también una solución en tiempo polinómico. Si se demostrase que un problema NP-completo, llamémoslo A, no se pudiese resolver en tiempo polinómico, el resto de los problemas NP-completos tampoco se podrían resolver en tiempo polinómico. Esto se debe a que si uno de los problemas NP-completos distintos de A, digamos X, se pudiese resolver en tiempo polinómico, entonces A se podría resolver en tiempo polinómico, por definición de NP-completo.

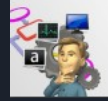


NP-Completo en otras palabras

- Supón que tu jefe te pide que escribas un algoritmo eficiente para un problema extremadamente importante para tu empresa.
  - Después de horas de romperte la cabeza sólo se te ocurre un algoritmo de "fuerza bruta", que analizándolo ves que cuesta tiempo exponencial.
  - Te encuentras en una situación muy embarazosa:
  - "No puedo encontrar un algoritmo eficiente, me temo que no estoy a la altura".
  - Te gustaría poder decirle al jefe: "No puedo encontrar un algoritmo eficiente, ¡porque no existe!".

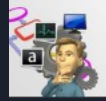


- Para la mayoría de los problemas, es muy difícil demostrar que son intratables, porque la mayoría de los problemas interesantes que no se saben resolver son NP- completos.
  - Los NP-completos parecen intratables.
  - Nadie ha sabido demostrar que los NP-completos son intratables.



Lista de 21 problemas NP-completos de Karp

- SAT (Problema de satisfacibilidad booleana, para fórmulas en forma normal conjuntiva)
  - 0-1 INTEGER PROGRAMMING (Problema de la programación lineal entera)
  - CLIQUE (Problema del clique, véase también Problema del conjunto independiente)
    - SET PACKING (Problema del empaquetamiento de conjuntos)
    - VERTEX COVER (Problema de la cobertura de vértices)
      - SET COVERING (Problema del conjunto de cobertura)
      - FEEDBACK NODE SET
      - FEEDBACK ARC SET
      - DIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano dirigido)
        - UNDIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano no dirigido)
- 3-SAT (Problema de satisfacibilidad booleana de 3 variables por cláusula)
  - CHROMATIC NUMBER (Problema de la coloración de grafos)
    - CLIQUE COVER (Problema de la cobertura de cliques)
    - EXACT COVER (Problema de la cobertura exacta)
      - HITTING SET
      - STEINER TREE
      - 3-DIMENSIONAL MATCHING (Problema del matching tridimensional)
      - KNAPSACK (Problema de la mochila)
        - JOB SEQUENCING (Problema de las secuencias de trabajo)
        - PARTITION (Problema de la partición)
          - MAX-CUT (Problema del corte máximo)



https://es.wikipedia.org/wiki/Lista_de_21_problemas_NP-completos_de_Karp