Problemas intratables: los NP-difíciles

Elvira Mayordomo

Universidad de Zaragoza

15 de mayo de 2013

Contenido de este tema

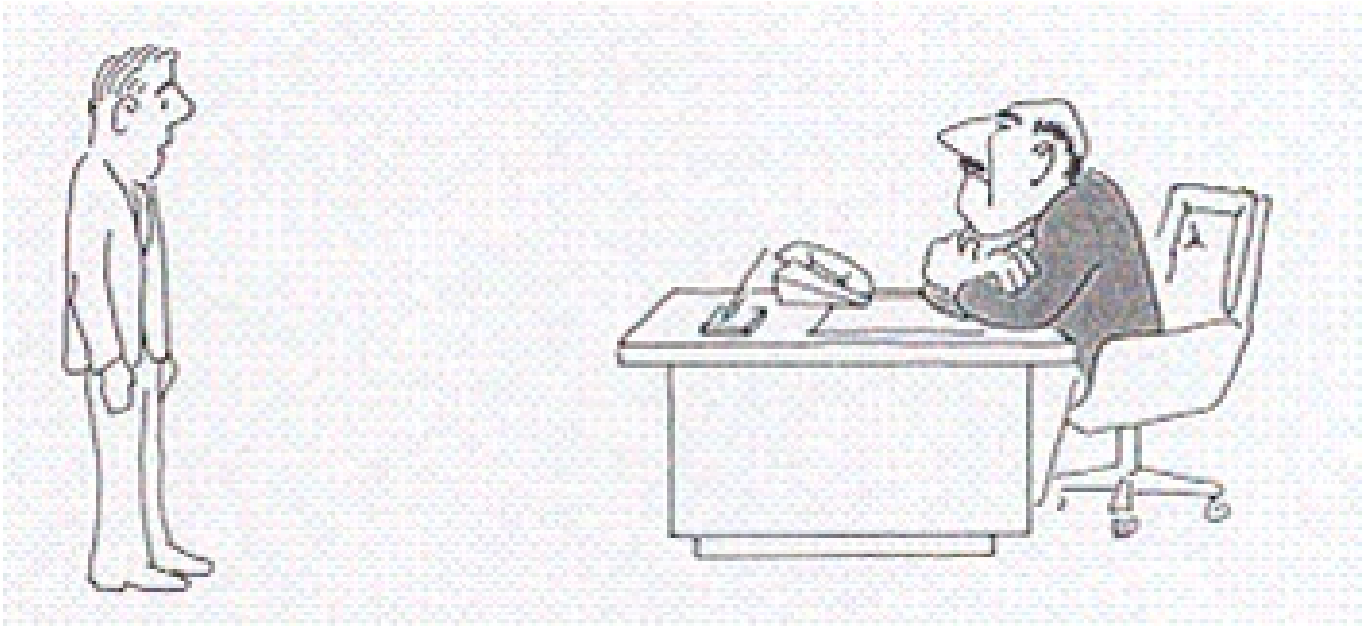
1. **Introducción**
2. Reducciones para construir algoritmos
3. Reducciones y problemas decisionales
4. Reducciones sencillas para demostrar intratabilidad
5. SAT y 3SAT
6. P vs NP
7. Esta asignatura

Este tema está basado en el capítulo 9 de
Steven S. Skiena. The Algorithm Design Manual. Springer 2008.

Una aplicación práctica

- 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.

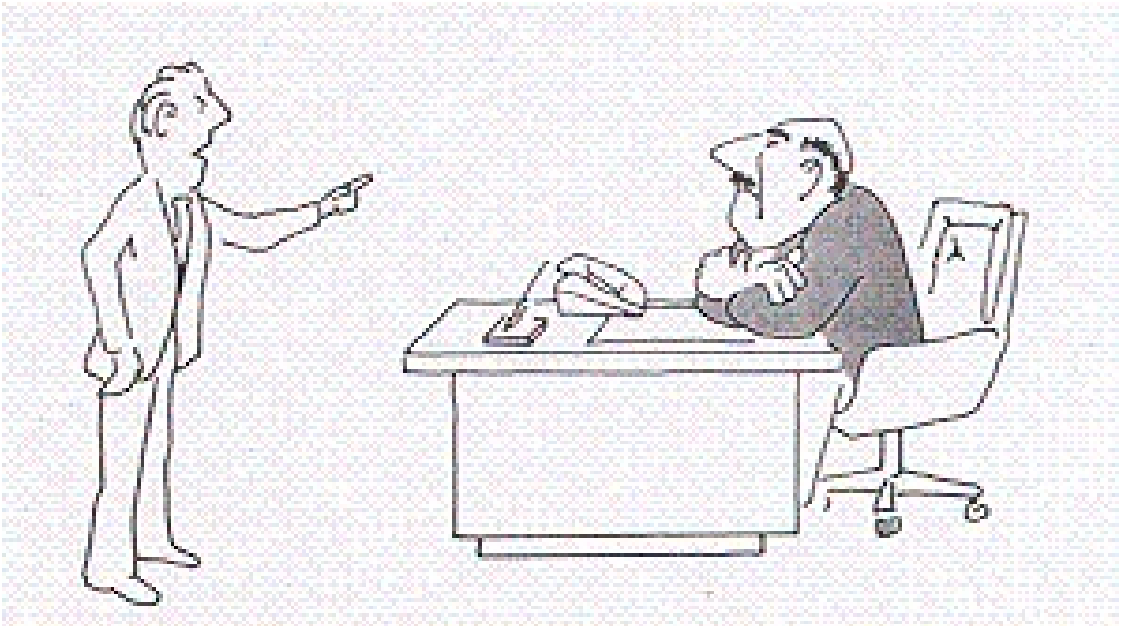
Una aplicación práctica



Te encuentras en una situación muy embarazosa:

"No puedo encontrar un algoritmo eficiente, me temo que no estoy a la altura"

Te gustaría poder decir ...

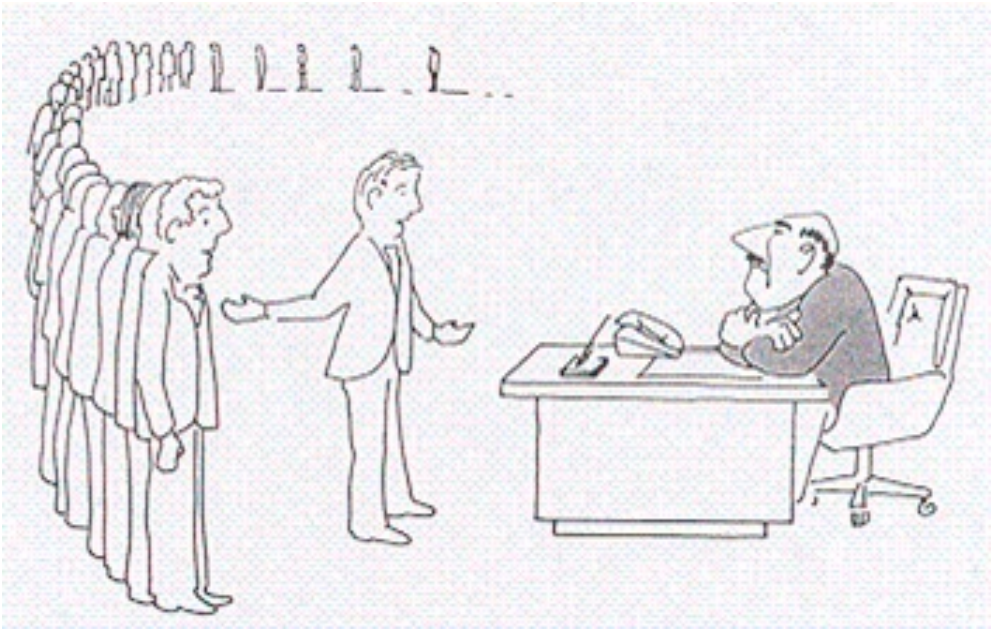


"No puedo encontrar un algoritmo eficiente porque no existe"

"No puedo encontrar un algoritmo eficiente porque no existe"
Eso es decir que el problema es intratable

- En realidad es muy poco frecuente poder decir algo tan tajante, para la mayoría de los problemas, es **muy difícil demostrar que son intratables**
- Pero la teoría de los NP-completos te puede ayudar a no perder tu trabajo diciendo ...

Usando la teoría de los NP-completos ...



"No puedo encontrar un algoritmo eficiente pero tampoco pueden ninguno de estos informáticos famosos"

Intro. del libro sobre NP-completos "Garey, Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman. 1978."

Introducción: Demostrando intratabilidad

- En este tema introducimos técnicas para ver que para un problema no existe ningún algoritmo eficiente.
- ¿En qué nos ayuda esto?
- La teoría de los NP-completos es muy útil para diseñar algoritmos, aunque sólo dé resultados negativos.
  - Permite al diseñador centrar sus esfuerzos más productivamente, no darse de cabezazos contra la pared.
  - Si no conseguimos demostrar que un problema es difícil o intratable, entonces hay que centrarse en encontrar un algoritmo eficiente.
- La teoría de los NP-completos nos permite identificar qué propiedades hacen un problema difícil.
  - Tener una intuición de qué problemas van a ser intratables es importante para un diseñador, y sólo se consigue con experiencia demostrando intratabilidad.

Introducción: Reducciones

- El concepto fundamental es el de reducción entre un par de problemas, que permite compararlos.
- Si A es reducible a B y tenemos un algoritmo eficiente para B entonces tenemos un algoritmo eficiente para A.
- Si A es reducible a B y no existe un algoritmo eficiente para A entonces no existe un algoritmo eficiente para B

$$A \leq B$$

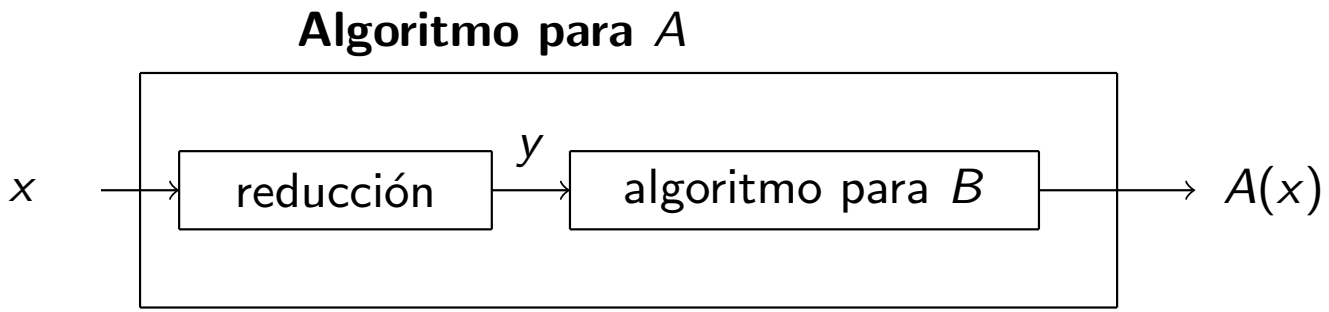
Visto en el tema *7-Programación lineal y reducciones* de la asignatura de *Algoritmia Básica*

Contenido de este tema

1. Introducción
2. **Reducciones para construir algoritmos**
3. Reducciones y problemas decisionales
4. Reducciones sencillas para demostrar intratabilidad
5. SAT y 3SAT
6. P vs NP
7. Esta asignatura

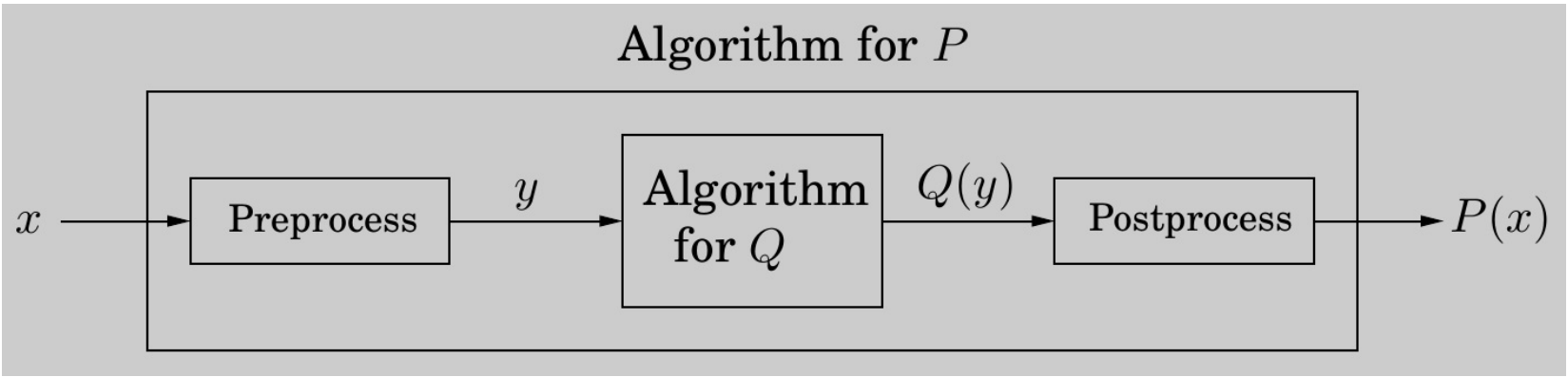
Reducciones para construir algoritmos

- ¿Cómo usar reducciones para hacer algoritmos?
- Usamos algoritmos conocidos para construir otros nuevos.
- Si podemos traducir un problema nuevo que queremos resolver a uno que ya sabemos resolver tenemos un algoritmo para resolver el problema nuevo.
- ¿Cómo? Usamos primero la reducción del nuevo al viejo y después el algoritmo para el viejo.
- El coste en tiempo será la suma del tiempo de la reducción más el tiempo del algoritmo para el viejo.



Reducciones para construir algoritmos

En realidad una reducción puede ser más complicada, veamos lo visto en Algoritmia Básica



Reducciones para construir algoritmos

Todavía puede ser más

$\mathrm{A}(x)$

1 Calcula $y_1, \ldots, y_k$
2 **for** $i = 1$ **to** k
3 Calcula $z_i = B(y_i)$
4 Cálculos finales a partir de $z_1, \ldots, z_k$
5 Resultado $\mathrm{A}(x)$

- El tiempo es la suma de los tiempos de $B(y_i)$ más los cálculos 1 y 4

Par más cercano

Entrada: Un conjunto S de n números, y una cota t
Salida: ¿Existe un par $s_i, s_j \in S$ que cumple $|s_i - s_j| \leq t$?

Para resolver *Par más cercano* basta reducirlo a *Ordenación*, ya que el par más cercano deben de ser dos vecinos tras la ordenación

PAR MÁS CERCANO(S, t)

1 Ordena S
2 ¿Es mín$_{1 \leq i < n} |s_i - s_{i+1}| \leq t$?

Par más cercano

1. La complejidad del algoritmo anterior depende de la complejidad de *Ordenación*. Si ordenamos en $O(n \log n)$ podemos resolver *Par más cercano* en $O(n \log n + n) = O(n \log n)$.
2. Esta reducción y la cota superior $\Omega(n \log n)$ para ordenación no dice nada de cuánto debe tardar un algoritmo para *Par más cercano*. ¿Y si hay un algoritmo más rápido que no pase por ordenar?
3. Si supiéramos que *Par más cercano* necesita al menos tiempo $\Omega(n \log n)$ con esta reducción podríamos demostrar que *Ordenación* tarda $\Omega(n \log n)$

Subsecuencia creciente más larga

En AB visteis técnicas de programación dinámica para calcular la distancia de edición (comparaciones de secuencias). También se pueden usar para encontrar la subsecuencia creciente más larga ...

Problema: Distancia de edición
Entrada: Cadenas de caracteres o enteros S y T; coste de cada inserción (c_{ins}), borrado (c_{del}), y sustitución (c_{sub})
Salida: ¿Cuál es el coste mínimo de las operaciones para transformar S en T?

Problema: Longitud de subsecuencia creciente más larga (LSCML)
Entrada: Cadena de caracteres o enteros S
Salida: ¿Cuál es la longitud de la subsecuencia más larga de posiciones $\{p_1, \ldots, p_m\}$ tal que $p_i < p_{i+1}$ y $S_{p_i} < S_{p_{i+1}}$?

Subsecuencia creciente más larga

De hecho *Longitud de subsecuencia creciente más larga* se puede reducir a *Distancia de edición*...

LSCML(S)

1 T =Ordena S
2 $c_{ins} = c_{del} = 1$
3 $c_{sub} = \infty$
4 Resultado $|S|$−distanciaDeEdición$(S, T, c_{ins}, c_{del}, c_{sub})/2$

¿Por qué funciona?

- T es la ordenación de S, cualquier subsecuencia común es creciente
- No permitimos sustituciones ($c_{sub} = \infty$) luego la edición óptima deja la subsecuencia creciente más larga y mueve el resto de los elementos (cada uno con un borrado y una inserción, coste 2)
- El coste dividido por 2 es el número de elementos fuera de la SCML

Subsecuencia creciente más larga

El coste es $O(n \log n)$ más el coste de distancia de edición ($O(|S| \cdot |T|) = O(n^2)$)

Hay algoritmos mejores para LSCML ($O(n \log n)$) que no usan programación dinámica

Mínimo común múltiplo

Decimos que a divide a b ($b|a$) si existe un entero d tal que $a = bd$

Problema: m.c.m.
Entrada: Enteros x, y
Salida: ¿Cuál es el menor entero m tal que m es múltiplo de x y m es múltiplo de y ?

Problema: m.c.d.
Entrada: Enteros x, y
Salida: ¿Cuál es el mayor entero d tal que d divide a x y d divide a y ?

m.c.m.(24,36)=72 y m.c.d.(24,36)=12

Mínimo común múltiplo

- Se pueden resolver hallando los factores primos de x, y pero no se conocen algoritmos eficientes para factorizar
- El algoritmo de Euclides es una forma eficiente de hallar el m.c.d. ($O(n)$) (donde n es la suma del número de bits)
- El algoritmo eficiente para m.c.m. pasa por una reducción a m.c.d.

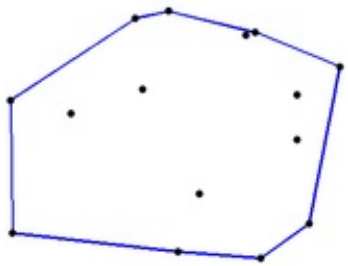
$\text{M.C.M.}(x, y)$

1 Resultado $xy/m.c.d.(x, y)$

Ordenar a partir de Envoltura convexa

Un polígono P es *convexo* si el segmento entre dos puntos dentro de P cae completamente dentro de P

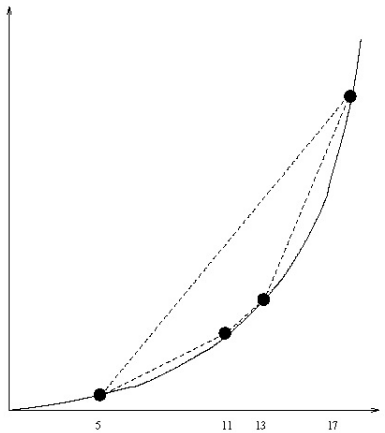
Problema: Envoltura convexa
Entrada: Un conjunto S de n puntos en el plano.
Salida: ¿Cuál es polígono convexo más pequeño que contiene todos los puntos de P?



Ordenar a partir de Envoltura convexa

Queremos reducir *Ordenar* a *Envoltura convexa*. Sea A un conjunto de números a ordenar

- Traducimos cada número x a un punto del plano (x, x^2)
- De esa manera tenemos puntos en la parábola $y = x^2$
- Como la parábola es convexa, cada punto es un vértice de la envoltura convexa
- Los vértices vecinos de la envolvente convexa tienen valores de x vecinos, luego están ordenados por x



Ordenar a partir de Envoltura convexa

ORDENAR(A)

1 Para cada $x \in A$, crear el punto (x, x^2) y añadirlo a S
2 U =EnvolturaConvexa(S)
3 B =Leer la coordenada x de los puntos de U de Izda a Dcha
4 Resultado B

- Crear los n puntos tarda $O(n)$, luego el algortimo anterior tarda el tiempo de *Envoltura convexa*+n
- Si pudiéramos resolver *Envoltura convexa* en menos de $n \log n$ podríamos ordenar en menos de $n \log n$
- Pero eso violaría la cota $\Omega(n \log n)$ de *Ordenar*
- Luego *Envoltura convexa* requiere tiempo $\Omega(n \log n)$

Contenido de este tema

1. Introducción
2. Reducciones para construir algoritmos
3. **Reducciones y problemas decisionales**
4. Reducciones sencillas para demostrar intratabilidad
5. SAT y 3SAT
6. P vs NP
7. Esta asignatura

Reducciones y problemas decisionales

Hemos visto muchos problemas en Algoritmia Básica para los que no hemos encontrado algoritmos eficientes:

- mochila 0-1
- viajante de comercio
- juego del 15
- planificación de tareas con plazo fijo
- graph coloring
- hamiltonian cycles
- ajedrez
- ...

¿Cómo nos ayudan las reducciones en estos casos?

Reducciones y problemas decisionales



Una pelea de niños:

- Adam vence a Bill, y Bill vence a Chuck
- ¿Quién es duro?
- ¿Y si Chuck es Chuck Norris? Entonces Bill y Adam son tan duros como Chuck Norris.
- ¿Y si es el patio de la guardería?

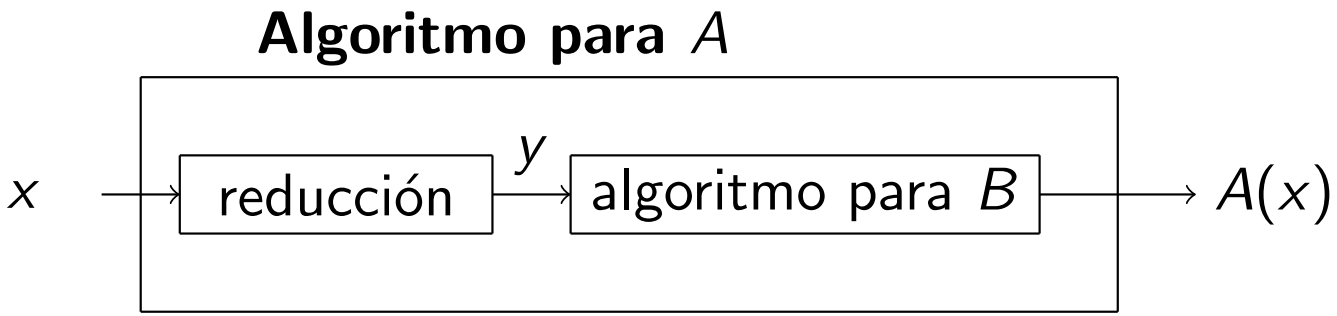
Reducciones: idea principal

- Tenemos dos problemas algorítmicos, A y B
- Tenemos la siguiente reducción + algoritmo para resolver A:

  Algoritmo $A(G)$:

  Traducir la entrada G a una entrada Y de B
  Llamar al método B con Y para resolver esta entrada
  Devolver la solución $B(Y)$ como la solución de $A(G)$

- Este algoritmo resuelve A si la traducción usada es correcta, es decir, si G es traducido a Y entonces $A(G) = B(Y)$.
- Una traducción de entradas de un problema a entradas de otro que preserva las soluciones es una reducción.



Reducciones: coste temporal

- Supongamos que la reducción anterior traduce G a Y en tiempo $O(P(n))$.
- Si mi algoritmo B tarda $O(P'(n))$, puedo resolver A en $O(P(n)+P'(n))$ (tiempo para traducir de G a Y y tiempo de resolver $B(Y)$).
- Si sé que $\Omega(Q(n))$ es una cota inferior para resolver A, entonces $\Omega(Q(n)-P(n))$ es una cota inferior para B
  - ¿Por qué?
  - No puedo resolver A en menos de $Q(n)$ ...
  - Si pudiera resolver B en menos de $Q(n)-P(n)$ entonces usando la reducción tendría un algoritmo para A en menos de $Q(n)$ **Contradicción**.
- (Adam vence a Bill, y Bill vence a Chuck) El primer argumento es Steve demostrando la debilidad de todos con un derechazo a Adam.
- El segundo es la aproximación Chuck Norris para ver que todos son duros.

Reducciones: coste temporal

- Una reducción de A a B demuestra que B no es más fácil que A.
- **Un algoritmo (o la falta de uno) para uno de los problemas implica un algoritmo (o la falta de uno) para el otro.**

Problemas decisionales

- Si una reducción de A a B tiene que preservar las soluciones es que las soluciones para A y para B son del mismo tipo.
- Problemas distintos tienen soluciones de tipo distinto, por ejemplo uno números y el otro listas de vértices o caminos.
- La clase más simple de problemas por tipo de soluciones son los **problemas decisionales** que sólo tienen respuestas posibles *cierto* y *falso*.
- Para hablar de reducciones se suele pensar sólo en problemas decisionales.

Problemas decisionales

- Afortunadamente muchos problemas se pueden reformular esencialmente como decisionales, por ejemplo los de optimización.
- El problema del viajante (decisional)

  Conocida la distancia existente entre cada dos ciudades y una cota k

  ¿Existe un recorrido de longitud $\leq k$ para un viajante que tiene que visitar todas las ciudades y volver al punto de partida?

- A partir de una solución rápida para el decisional, puedes encontrar la longitud mínima de un recorrido por búsqueda binaria.
- Encontrar el recorrido de longitud mínima cuesta un poco más.
- **En este tema nos centraremos en problemas decisionales.**

Problemas decisionales

- Si un problema decisional es intratable entonces el general también lo es
- Por ejemplo si es intratable el problema del viajante decisional (averiguar si existe un recorrido de longitud menor que un k dado) entonces el problema del viajante (encontrar el recorrido más corto) es intratable ¿Por qué?
- O si es intratable el problema del viajante decisional (averiguar si existe un recorrido de longitud menor que un k dado) entonces el problema de cuánto mide el recorrido más corto es intratable ¿Por qué?

Reducciones:recordamos

- Si A es reducible a B podemos resolver A en **el tiempo de la reducción más el tiempo de resolver B**.
- Si A es reducible a B y tenemos un algoritmo eficiente para B entonces tenemos un algoritmo eficiente para A.
- Si A es reducible a B y no existe un algoritmo eficiente para A entonces no existe un algoritmo eficiente para B

$$A \leq B$$

Nemotécnico: A es tan fácil o más fácil que B

Contenido de este tema

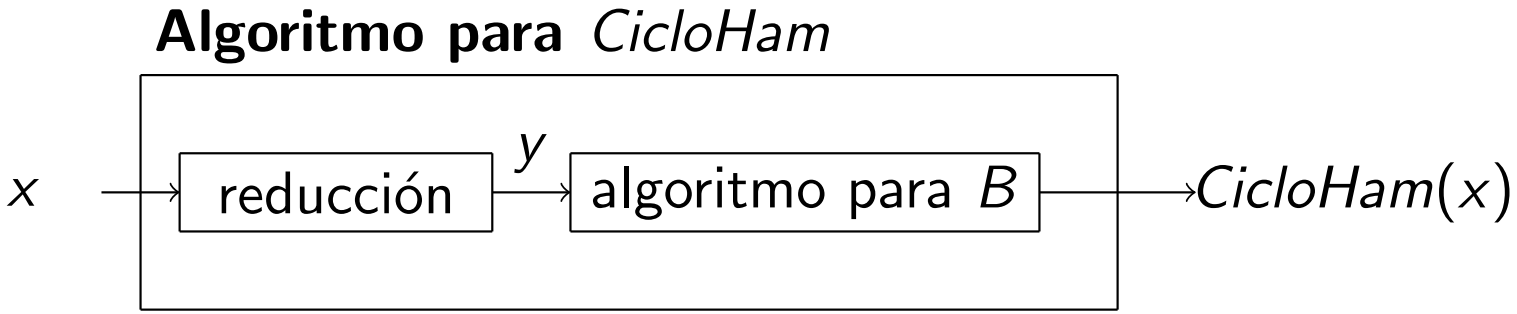
1. Introducción
2. Reducciones para construir algoritmos
3. Reducciones y problemas decisionales
4. **Reducciones sencillas para demostrar intratabilidad**
5. SAT y 3SAT
6. P vs NP
7. Esta asignatura

Reducciones sencillas para intratabilidad

- Queremos demostrar que algunos problemas son intratables, es decir, que no tienen algoritmos eficientes
- Partimos de que *Ciclo Hamiltoniano* y *Cobertura de Vértices* son intratables. Más adelante justificamos por qué
- Si *Ciclo Hamiltoniano* es reducible a B entonces B es intratable
- Si *Cobertura de Vértices* es reducible a B entonces B es intratable

Reducción de *Ciclo Hamiltoniano* a B

$$CicloHam \leq B$$



- Transformamos la entrada de *Ciclo Hamiltoniano* en entrada de B
- La solución de B nos da la solución para *Ciclo Hamiltoniano*
- Como *Ciclo Hamiltoniano* es intratable entonces B es intratable

Las reducciones son transitivas

- Si A es reducible a B y B es reducible a C entonces A es reducible a C
- Si A es intratable podemos demostrar que C es intratable demostrando primero que B es intratable (con $A \leq B$) y después que C es intratable (con $B \leq C$)

Veremos los siguientes ejemplos

- TSP es intratable
- Conjunto Independiente es intratable
- Problema general de planificación de películas es intratable
- Clique es intratable

TSP es intratable

- Vamos a utilizar una reducción de *Ciclo Hamiltoniano* a *TSP*
- Empezamos con la definición de los dos problemas (versión decisional de los vistos en AB)

Problema: Ciclo Hamiltoniano
Entrada: Un grafo G.
Salida: ¿Existe un ciclo hamiltoniano en G, es decir, un camino que visita una vez cada vértice y vuelve al vértice inicial?

Problema: TSP
Entrada: n el número de ciudades, la matriz de distancias $n \times n$ y cota superior k.
Salida: ¿Existe un recorrido por las n ciudades y volviendo al punto de partida con distancia total $\leq k$?

TSP es intratable: reducir Ciclo Ham. a TSP

- Los dos problemas tratan de encontrar ciclos cortos, TSP en un grafo con pesos
- Para reducir *Ciclo Hamiltoniano* a *TSP* tenemos que convertir cada entrada de *Ciclo Hamiltoniano* (un grafo) en una entrada de *TSP* (ciudades y distancias)

CicloHamiltoniano(G)

1 $n =$ número de vértices de G
2 **for** $i = 1$ **to** n
3 **for** $j = 1$ **to** n
4 **if** (i, j) es arista de G
5 $d(i, j) = 1$
6 **else** $d(i, j) = 2$
7 Resultado TSP(n, d, n).

TSP es intratable: reducir Ciclo Ham. a TSP

- La reducción anterior tarda tiempo $O(n^2)$ (los dos for anidados), podemos resolver *Ciclo Hamiltoniano* en $O(n^2)+$ el tiempo de TSP
- Es correcta, la respuesta de *Ciclo Hamiltoniano* con entrada G es la misma que *TSP* con entrada (n, d, n):
  - Si G tiene un circuito hamiltoniano $(v_i, \ldots, v_n)$ entonces el mismo recorrido para (n, d, n) tiene distancia total $\sum_{i=1}^{n-1} d(v_i, v_{i+1}) + d(v_n, v_1) = n$
  - Si G no tiene un circuito hamiltoniano entonces (n, d, n) no tiene recorrido con distancia total $\leq n$ porque un recorrido así sólo puede pasar por n distancias 1, luego pasa por n aristas del grafo y sería un ciclo hamiltoniano de G.
- Tenemos una reducción eficiente de *Ciclo Hamiltoniano* a *TSP*. Como *Ciclo Hamiltoniano* es intratable entonces *TSP* **es intratable**.

Conjunto Independiente es intratable

- Vamos a utilizar una reducción de *Cobertura de Vértices* (VC) a *Conjunto Independiente*
- Empezamos con la definición de los dos problemas

Problema: Cobertura de Vértices
Entrada: Un grafo G (con vértices V) y $k \in \mathbb{N}$.
Salida: ¿Existe un conjunto U de k vértices de G tal que cada arista (i,j) de G cumple que $i \in U$ ó $j \in U$?

- Cuanto más pequeño k más difícil es la Cobertura de Vértices
- Un conjunto de vértices S de G es *independiente* si no hay ninguna arista (i,j) de G con los dos vértices en S.
- Cuanto más grande S más difícil es que sea independiente

Problema: Conjunto Independiente
Entrada: Un grafo G (con vértices V) y $k \in \mathbb{N}$.
Salida: ¿Existe un conjunto independiente de k vértices de G?

Conjunto Independiente es intratable: reducir Cobertura de Vértices a Conjunto Independiente

- Los dos problemas tratan de encontrar conjuntos de vértices, el primero con un vértice de cada arista, el segundo que no contenga aristas
- Si U es una Cobertura de los vértices de G entonces $V - U$ es un conjunto independiente (ya que si los dos vértices de una arista (i, j) de G están en $V - U$ entonces U no cubre esa arista)
- Para reducir *Cobertura de Vértices* a *Conjunto Independiente* usamos la idea anterior

CoberturaVértices(G, k)

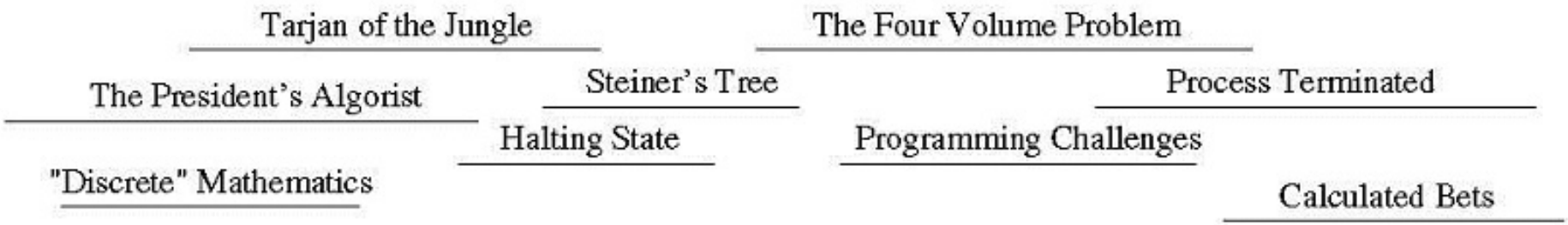
1 $k' = |V| - k$
2 Resultado ConjuntoIndependiente$(G,\ k')$.

Conjunto Independiente es intratable: reducir Cobertura de Vértices a Conjunto Independiente

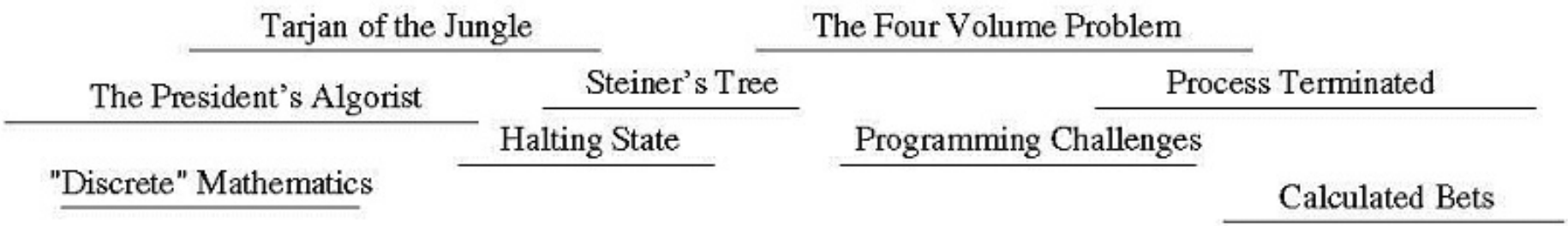
- La reducción anterior tarda tiempo $O(1)$, podemos resolver *Cobertura de Vértices* en el tiempo de *Conjunto Independiente*
- Es correcta, la respuesta de *Cobertura de Vértices* con entrada G, k es la misma que *Conjunto Independiente* con entrada $(G, |V| - k)$
- Tenemos una reducción eficiente de *Cobertura de Vértices* a *Conjunto Independiente*. Como *Cobertura de Vértices* es intratable entonces *Conjunto Independiente* **es intratable**.

El problema de planificación de películas

- Imagínate que eres es un **actor muy cotizado**, que tienes ofertas para protagonizar n películas
- En cada oferta viene especificado en el primer y último día de rodaje
- Para aceptar el trabajo, debes comprometerte a estar disponible durante todo este período entero
- Por lo tanto no puedes aceptar simultáneamente dos papeles cuyos intervalos se superpongan



El problema de planificación de películas



- Para un artista como tú, los criterios para aceptar un trabajo son claros: quieres ganar tanto **dinero** como sea posible
- Debido a que cada una de estas películas paga la misma tarifa por película, esto implica que buscas la mayor cantidad posible de puestos de trabajo (intervalos) de tal manera que dos cualesquiera de ellos no estén en conflicto entre sí

El problema de planificación de películas

- Tú (o tu agente) debes resolver el siguiente problema de algoritmia:

  Problema: Planificación de películas
  Entrada: Un conjunto I de n intervalos, $k \in \mathbb{N}$
  Salida: ¿Cuál es el mayor conjunto de intervalos de I que dos a dos sean disjuntos? o ¿qué películas puedo hacer ganando la mayor cantidad de dinero?
- No es muy difícil encontrar un algoritmo eficiente (¿ideas?)
- Pero vamos a considerar otro problema más difícil ...

El problema GENERAL de planificación de películas

- Más difícil: un proyecto de película no tiene porqué rodarse en un sólo intervalo, puede tener un calendario discontinuo
- Por ejemplo "Tarzán de la jungla" se rodará en Enero-Marzo y Mayo-Junio, "Terminator" se rodará en Abril y en Agosto, "Babe" se rodará en Junio-Julio
- El mismo actor puede rodar "Tarzán de la jungla" y "Terminator" pero no "Tarzán de la jungla" y "Babe"
- Vamos a ver que la versión decisional es **intratable**
  Problema: Planificación general de películas
  Entrada: Un conjunto I de n *conjuntos de intervalos*, $k \in \mathbb{N}$
  Salida: ¿Existen k *conjuntos de intervalos* de I que dos a dos sean disjuntos?

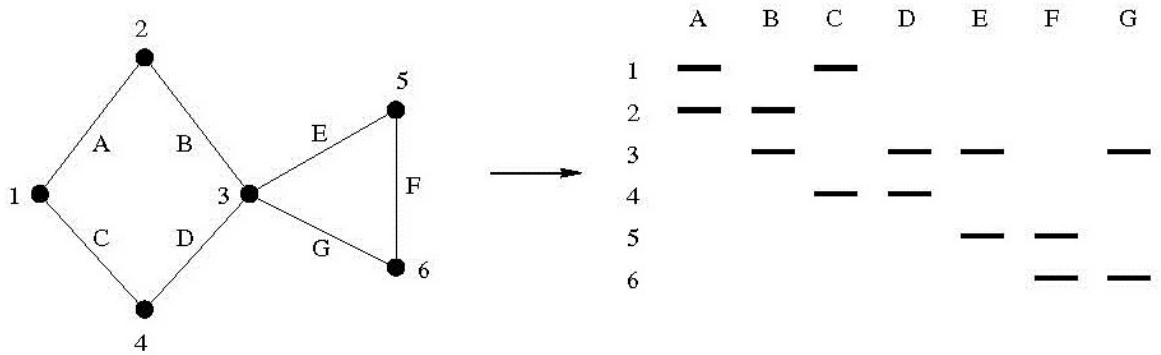
Prob. GENERAL de planif. de películas es intratable

- Necesitamos hacer una reducción desde otro problema que sepamos intratable
- Vamos a intentarlo desde *Conjunto Independiente*
- ¿En qué se parecen los dos problemas? ...
  - Los dos pretenden seleccionar el subconjunto más grande posible – de vértices y de películas resp.
  - Intentemos traducir los vértices en películas
  - Intentemos traducir vértices con una arista en películas incompatibles

Prob. GENERAL de planif. de películas es intratable

ConjuntoIndependiente(G, k)

1 m =número de aristas de G
2 n =número de vértices de G
3 $I = \emptyset$
4 **for** $j = 1$ **to** n
5 $\quad$ $pelicula(j) = \emptyset$
6 $\quad$ $I = anadir(I, pelicula(j))$
7 **for** la $i$ésima arista de G (x, y), $1 \leq i \leq m$
8 $\quad$ $pelicula(x) = anadir(pelicula(x), [i, i + 0{,}5])$
9 $\quad$ $pelicula(y) = anadir(pelicula(y), [i, i + 0{,}5])$
10 Resultado GralPaInifPeliculas(I, k).

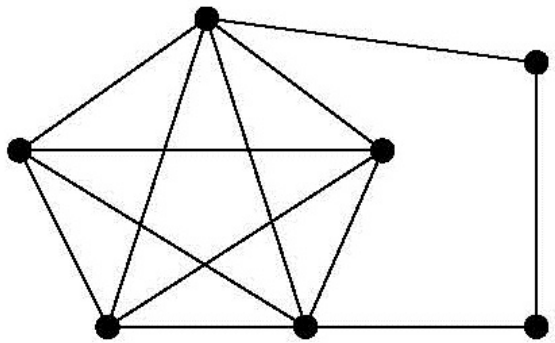


Prob. GENERAL de planif. de películas es intratable

- Cada arista se traduce en un intervalo
- Cada vértice es una película que contiene los intervalos de las aristas desde ese vértice
- Dos vértices unidos por una arista (prohibido en el *Conjunto Independiente*) definen un par de películas que comparten un intervalo (prohibido en el calendario del actor) y viceversa
- Los mayores subconjuntos que satisfacen los dos problemas son los mismos
- Tenemos una reducción, un algoritmo eficiente para el *problema gral. de planificación de películas* nos da un algoritmo eficiente para *Conjunto Independiente*
- Como *Conjunto Independiente* es intratable, *problema gral. de planificación de películas* es intratable

Clique es intratable

- Vamos a utilizar una reducción de *Conjunto Independiente* a *Clique*
- Empezamos con la definición de Clique
- Un **clique social** es un conjunto de personas que todos conocen a todos. Un **clique** en un grafo es un conjunto de vértices que todos están unidos a todos



Problema: Clique
Entrada: Un grafo G (con vértices V) y $k \in \mathbb{N}$.
Salida: ¿Tiene G un clique de k vértices, es decir, existe un conjunto U de k vértices de G tal que cada par $x, y \in U$, existe la arista (x, y) en G?

Clique es intratable

- Cuanto más grande k más difícil es Clique
- En un grafo social los cliques corresponden a lugares de trabajo, vecindarios, parroquias, escuelas ...
- ¿Y en el grafo de internet?

Clique es intratable: reducir Conj. Independiente a Clique

- Los dos problemas tratan de encontrar conjuntos de vértices, el primero que no contenga aristas, el segundo que contenga todas las aristas
- ¿Cómo podemos relacionarlos?
- Para reducir *Conjunto Independiente* a *Clique* cambiamos aristas por no aristas y no aristas por aristas, es decir, **complementamos el grafo**

ConjuntoIndependiente(G, k)

1 Construir un grafo G' con los mismos vértices de G
2 aristas de $G' = \emptyset$
3 **for** $x \in V$
4 **for** $y \in V$
5 Si (x, y) no es arista de G la añadimos a las aristas de G'
6 Resultado Clique(G', k).

Clique es intratable: reducir Conj. Independiente a Clique

- La reducción anterior tarda tiempo $O(n^2)$, podemos resolver *Conjunto Independiente* en el tiempo de *Clique* $+O(n^2)$
- Tenemos una reducción eficiente de *Conjunto Independiente* a *Clique*. Como *Conjunto Independiente* es intratable entonces *Clique* **es intratable**
- Hemos hecho una cadena transitiva:

$$\text{Cobertura de Vértices} \leq \text{Conjunto Independiente}$$
$$\text{Conjunto Independiente} \leq \text{Clique}$$

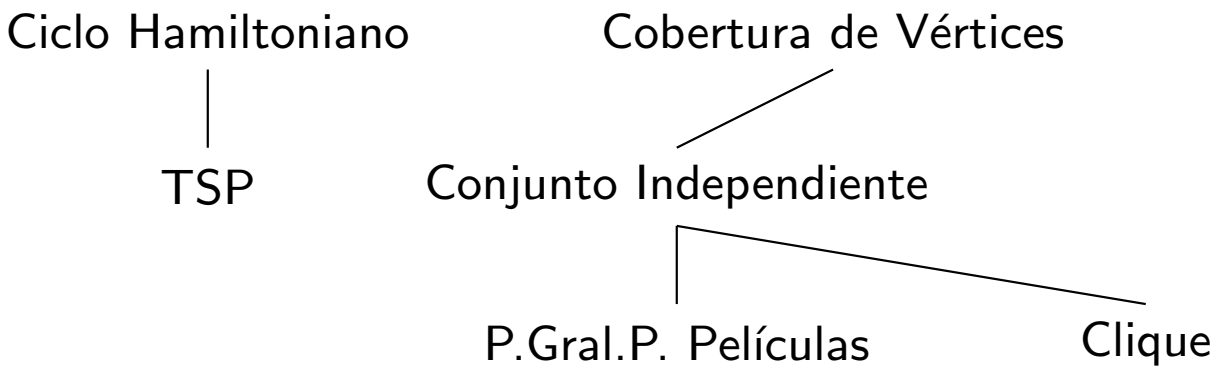
- Nota: La misma reducción (complementar el grafo) sirve para reducir *Clique* a *Conjunto Independiente*, luego los dos problemas son equivalentes

Contenido de este tema

1. Introducción
2. Reducciones para construir algoritmos
3. Reducciones y problemas decisionales
4. Reducciones sencillas para demostrar intratabilidad
5. **SAT y 3SAT**
6. P vs NP
7. Esta asignatura

Problemas vistos

- Hemos visto que los siguientes 4 problemas son intratables: *TSP*, *Conjunto Independiente*, *Problema GENERAL de planificación de películas*, *Clique*
- Para ello hemos usado (sin justificar) que *Ciclo Hamiltoniano* y *Cobertura de Vértices* son intratables



- ¿Por qué son intratables *Ciclo Hamiltoniano* y *Cobertura de Vértices*?

SAT

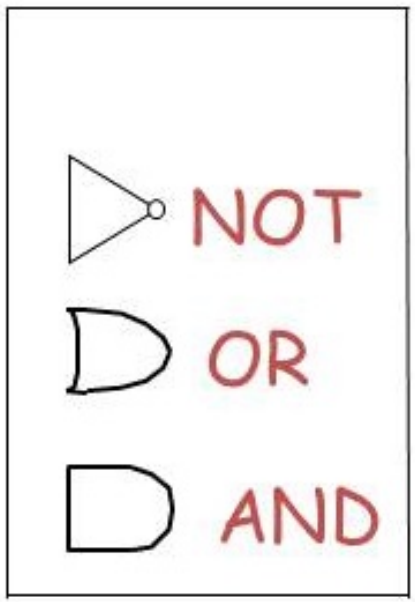
- Para demostrar la intratabilidad de otros problemas debemos empezar con un único problema que sea **EL** candidato a intratable
- La **madre de todos los intratables** en un problema de lógica llamado Satisfacibilidad (SAT)

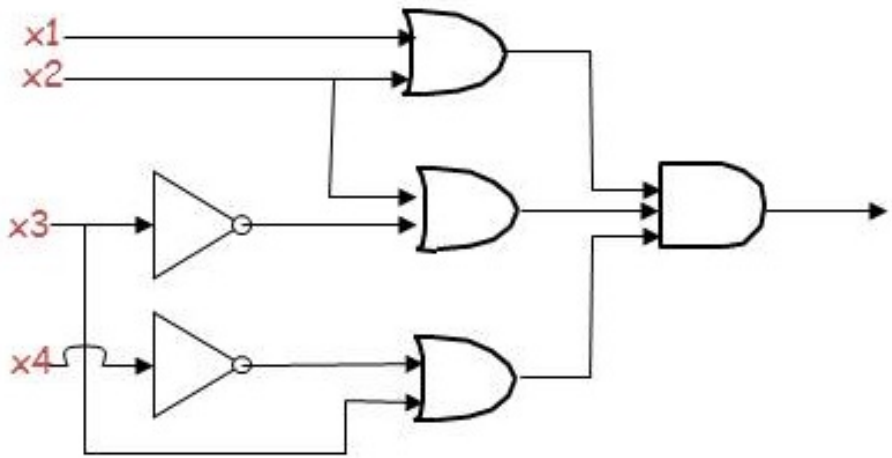
SAT

Problema: SAT

Entrada: Un circuito booleano en CNF C con una única salida.

Salida: ¿Existe una asignación de las entradas de C que da salida Cierto?





SAT: ejemplos

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2)$$

$$(x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1)$$

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1) \wedge (\neg x_3)$$

SAT

- Existen múltiples concursos para programas que resuelvan SAT (SAT solvers)
- Es el paradigma de problema intratable, es ampliamente aceptado que es intratable (aunque no se ha demostrado)
- Daremos alguna razón que hace sospechar que SAT es intratable

Variantes de SAT

- Si restringimos a una sola entrada por puerta OR es fácil encontrar un algoritmo eficiente
- Si restringimos a dos entradas por puerta OR se puede encontrar un algoritmo eficiente, este problema se llama 2-SAT
- ¿Y si restringimos a tres entradas por puerta OR?
- Vamos a ver que este último caso es intratable y nos es útil para demostrar que otros problemas son intratables

3-SAT

Problema: 3-SAT
Entrada: Un circuito booleano en CNF C en el que **cada puerta OR contiene exactamente 3 entradas**, con una única salida.
Salida: ¿Existe una asignación de las entradas de C que da salida Cierto?

- Ejemplo de entrada: $(x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee x_2 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee x_3)$
- Podemos demostrar que 3-SAT es intratable haciendo una reducción de SAT a 3-SAT

Reducción de SAT a 3-SAT

¿Cómo transformamos un circuito de SAT en uno de 3-SAT?

- Si tenemos un OR con una sola entrada x creamos dos variables nuevas v_1, v_2 y sustituimos el OR por

$$(x \vee v_1 \vee v_2) \wedge (x \vee \neg v_1 \vee v_2) \wedge (x \vee v_1 \vee \neg v_2) \wedge (x \vee \neg v_1 \vee \neg v_2)$$

- Si tenemos un OR con dos entradas x, y creamos una variable nueva v y sustituimos el OR por

$$(x \vee y \vee v) \wedge (x \vee y \vee \neg v)$$

- Si tenemos un OR con **más de tres** entradas $y_1, \ldots, y_n$
  - Creamos $n-3$ variables nuevas $v_1, \ldots, v_{n-3}$
  - Sustituimos $(y_1 \vee \ldots \vee y_n)$ por $F_1 \wedge \ldots \wedge F_{n-2}$ con

$$F_1 = (y_1 \vee y_2 \vee \neg v_1)$$

$$F_j = (v_{j-1} \vee y_{j+1} \vee \neg v_j),\ 2 \leq j \leq n-3$$

$$F_{n-2} = (v_{n-3} \vee y_{n-1} \vee y_n)$$

Reducción de SAT a 3-SAT

- Si tenemos un OR con **más de tres** entradas $y_1, \ldots, y_n$
  - Creamos $n-3$ variables nuevas $v_1, \ldots, v_{n-3}$
  - Sustituimos $(y_1 \vee \ldots \vee y_n)$ por $F_1 \wedge \ldots \wedge F_{n-2}$ con

$$F_1 = (y_1 \vee y_2 \vee \neg v_1)$$

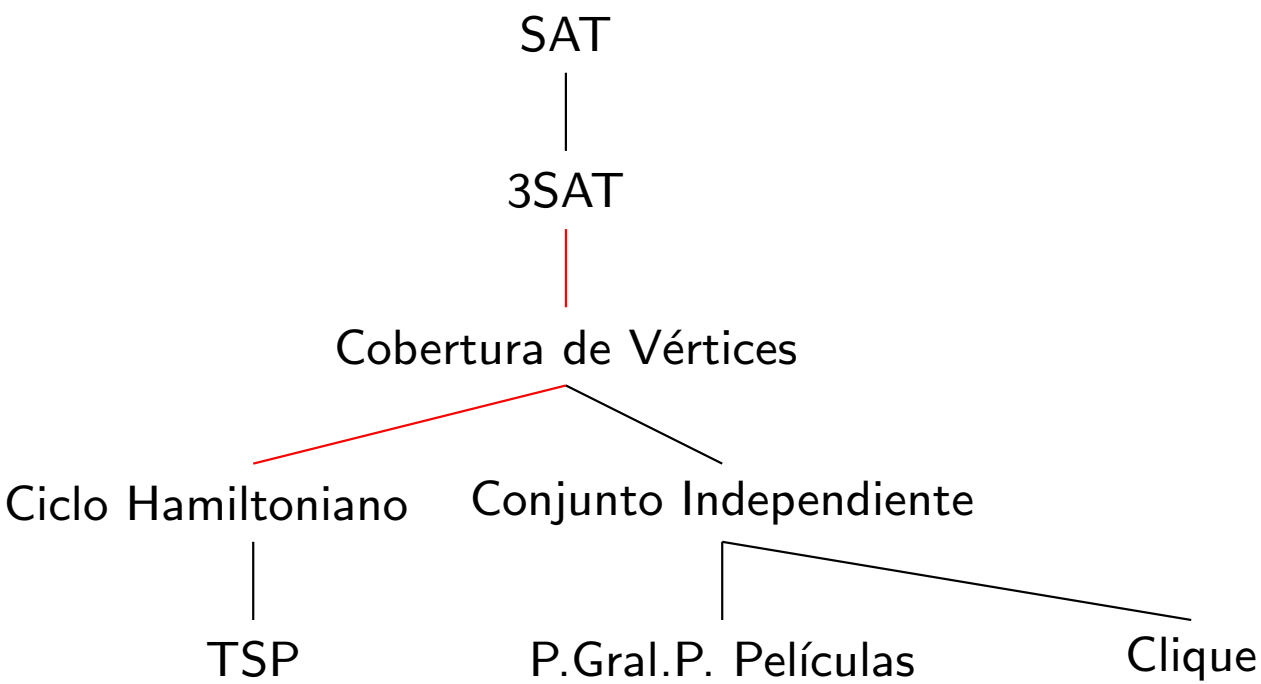
$$F_j = (v_{j-1} \vee y_{j+1} \vee \neg v_j),\ 2 \leq j \leq n-3$$

$$F_{n-2} = (v_{n-3} \vee y_{n-1} \vee y_n)$$

- El caso más complicado es este último ...
  - Si hay una asignación que hace cierta $(y_1 \vee \ldots \vee y_n)$ entonces hace cierto por lo menos un y_i, podemos hacer cierto $F_1 \wedge \ldots \wedge F_{n-2}$ con $v_1 = v_2 = \ldots = v_{i-2}$ =falso y $v_{i-1} = v_i = \ldots = v_{n-3}$ =cierto
  - Probemos todos los casos, si una asignación hace cierta y_1 ó y_2
  - si hace cierta y_i con $3 \leq i \leq n-2$
  - si hace cierta y_{n-1} ó y_n ...
- También es cierto que si una asignación hace cierta $F_1 \wedge \ldots \wedge F_{n-2}$ entonces la misma asignación restringida a las variables originales hace cierta $(y_1 \vee \ldots \vee y_n)$ ¿Por qué?
- Así que tenemos una reducción de SAT a 3-SAT

Problemas intratables vistos

- Hemos visto las reducciones que aparecen en negro
- Las que aparecen en rojo son más complicadas y no las veremos (algunas están en las secciones 9.5-9.8 del Skiena)



El arte de demostrar intratabilidad

- En general es más fácil pensar una reducción (o una demostración de intratabilidad) que explicarla/entenderla
- Una sutil diferencia puede convertir un problema intratable en tratable o viceversa Camino más corto/camino más largo. Pasar por todos los vértices/aristas una sola vez
- Lo primero que hay que hacer si sospechamos que un problema es intratable es mirar el libro de Garey Johnson
- Si esto no funciona, para reducir un intratable A a un problema nuevo N
  - Haz A lo más simple posible
  - Haz N lo más difícil posible
  - Seleccionar el A adecuado por las razones adecuadas
  - Si te quedas atascado, alterna entre intentar ver que N es intratable y encontrar un algoritmo eficiente para N

Recomendable

- Leer las secciones 9.5-9.8 del Skiena
- Incluyen demostraciones de intratabilidad más complicadas y en general filosofía de estas demostraciones

Contenido de este tema

1. Introducción
2. Reducciones para construir algoritmos
3. Reducciones y problemas decisionales
4. Reducciones sencillas para demostrar intratabilidad
5. SAT y 3SAT
6. **P vs NP**
7. Esta asignatura

Definición de NP-completo y NP-difícil

- A es **NP-difícil** si SAT es reducible a A (SAT $\leq A$). También llamado problema difícil, intratable, NP-hard
- A es **NP-completo** si SAT es reducible a A y A es reducible a SAT (SAT $\leq A$ y $A \leq$ SAT), es decir, si A es equivalente a SAT
- En esta asignatura nos interesan principalmente los problemas difíciles, es decir, tan difíciles como SAT (o más)
  - **Nota:** Los NP-difíciles vistos hasta ahora (3SAT, Cobertura de Vértices, Ciclo Hamiltoniano, TSP, Conjunto Independiente, Clique) son todos NP-completos, es decir, equivalentes a SAT
  - Seguramente algunos NP-difíciles no son NP-completos (y son más difíciles que SAT), por ejemplo el juego del ajedrez

El problema abierto P vs NP

- Antes de cerrar este tema vamos a mencionar uno de los problemas más famosos de la Informática

$$\text{¿Es P} = \text{ NP?}$$

- Existe un premio de un millón de dólares para el que consiga resolverlo

El problema abierto P vs NP: descubrir vs. verificar

- Intuitivamente P vs NP consiste en saber si verificar es más fácil que encontrar
- Por ejemplo, si estás haciendo un examen y "accidentalmente" ves la respuesta de un compañero, ¿has ganado algo? ¿es más fácil verificar si la respuesta es correcta que encontrarla tú solo?
- Pensemos en TSP ¿es más fácil verificar si un camino mide como mucho k que encontrar un camino de longitud $\leq k$?
- En SAT ¿es más fácil verificar si una asignación hace cierto C que encontrar una asignación que lo haga cierto?
- En *Cobertura de Vértices*, es más fácil comprobar si un conjunto de k vértices cubre G que encontrar un conjunto de k vértices que cubra G?
- La respuesta parece obvia pero no lo es, no tenemos ninguna prueba de que verificar sea más fácil que encontrar ...

La clase P

- Un problema A está en P ($A \in$ P) si existe un algoritmo eficiente que resuelva A
- Un algoritmo eficiente es un algoritmo que tarda tiempo como mucho polinómico
- Polinómico es $O(n)$ (lineal) o $O(n^2)$ (cuadrático) o en general $O(n^k)$ para un k fijo
- En la práctica polinómico suele ser como mucho $O(n^3)$
- En AB vimos muchos ejemplos (por ej. camino más corto), hace nada vimos el Problema de planificación de películas (la primera versión)

La clase NP

- Un problema A está en NP ($A \in$ NP) si sus soluciones pueden ser verificadas de forma eficiente (en tiempo polinómico)
- Por ejemplo TSP, SAT, *Cobertura de Vértices*
- Es un club menos exclusivo que P
- Todos los miembros de P tienen pase libre en NP: si un problema se puede resolver completamente de forma eficiente entonces también puedes comprobar una solución de forma eficiente

  Resuélvelo desde cero y compara la solución con la candidata que te dan

- La pregunta del millón de dólares es si hay problemas en NP que no pueden ser miembros de P. Si no es así entonces P=NP, pero si existe al menos un problema en NP que no está en P, P$\neq$NP
- La mayoría de los algoritmistas opinan que P$\neq$NP

SAT y NP

- Si $A \in$ NP entonces $A \leq SAT$ (Teorema de Cook)
- Es decir, todos los problemas de NP son reducibles a SAT
- Esta es la principal razón por la que se cree que SAT es intratable (la madre de todos los intratables)
- SAT es tan difícil (o mas) como cualquier problema en NP

Los NP-completos

- Si B es NP-completo (equivalente a SAT) para cada $A \in$ NP, $A \leq B$
- Es decir, todos los problemas de NP son reducibles a cualquier NP-completo
- Los NP-completos son equivalentes a SAT, son igual de intratables
- Para demostrar que P$\neq$NP bastaría probar que un NP-completo (el que sea) no tiene algoritmos eficientes que lo resuelvan
- Para demostrar que P=NP bastaría probar que un NP-completo (el que sea) tiene un algoritmo eficiente que lo resuelva (poco esperable)

Preguntas equivalentes

- ¿P= NP?
- ¿Existe un algoritmo eficiente que resuelva *SAT*?
- ¿Existe un algoritmo eficiente que resuelva *tu NP-completo favorito*?

Cualquiera de esas preguntas está abierta, la respuesta vale un millón de dólares, y se cree que la respuesta es no

Contenido de este tema

1. Introducción
2. Reducciones para construir algoritmos
3. Reducciones y problemas decisionales
4. Reducciones sencillas para demostrar intratabilidad
5. SAT y 3SAT
6. P vs NP
7. **Esta asignatura**

¿Y ahora qué?

- ¿Qué hacemos con los problemas intratables?
- Si hemos demostrado que es intratable, es porque para empezar necesitábamos resolverlo
- No nos conformamos con saber que no hay algoritmos eficientes que los resuelvan completamente, nos sigue interesando resolverlos de alguna manera
- En este curso veremos varias aproximaciones
  1. **Algoritmos aproximados**: en algunos casos podemos encontrar eficientemente respuestas *cercanas* a la óptima
  2. **Heurísticas** (por ejemplo Simulated annealing o Algoritmos genéticos): encuentran rápidamente soluciones aunque sin garantía ninguna
  3. **Algoritmos probabilistas**: En algunos casos utilizar el azar nos da algoritmos eficientes
  4. **Análisis amortizado, complejidad en media**: A veces un algoritmo puede ser eficiente cuando se trata de usarlo para un número masivo de entradas