P vs NP

Miguel Raggi

Escuela Nacional de Estudios Superiores
UNAM

7 de mayo de 2018

Índice:

Índice:

Equivalencia polinomial

- Resulta que hay una gran cantidad de problemas (computacionales) para los cuales no se conoce ningún algoritmo eficiente (o polinomial).

Equivalencia polinomial

- Resulta que hay una gran cantidad de problemas (computacionales) para los cuales no se conoce ningún algoritmo eficiente (o polinomial).
- Y resulta que se puede probar que muchos problemas muy conocidos son todos en cierto sentido equivalentes:

Equivalencia polinomial

- Resulta que hay una gran cantidad de problemas (computacionales) para los cuales no se conoce ningún algoritmo eficiente (o polinomial).
- Y resulta que se puede probar que muchos problemas muy conocidos son todos en cierto sentido equivalentes:
- Equivalente en el sentido de que si pudiéramos encontrar un algoritmo polinomial para resolver uno de ellos, tendríamos automáticamente un algoritmo polinomial para resolver cualquiera de ellos.

Equivalencia polinomial

- Resulta que hay una gran cantidad de problemas (computacionales) para los cuales no se conoce ningún algoritmo eficiente (o polinomial).
- Y resulta que se puede probar que muchos problemas muy conocidos son todos en cierto sentido equivalentes:
- Equivalente en el sentido de que si pudiéramos encontrar un algoritmo polinomial para resolver uno de ellos, tendríamos automáticamente un algoritmo polinomial para resolver cualquiera de ellos.
- Nadie ha probado que no exista algoritmo polinomial para uno de ellos, pero muchísimos han intentado encontrar algoritmos.

Equivalencia polinomial

- Resulta que hay una gran cantidad de problemas (computacionales) para los cuales no se conoce ningún algoritmo eficiente (o polinomial).
- Y resulta que se puede probar que muchos problemas muy conocidos son todos en cierto sentido equivalentes:
- Equivalente en el sentido de que si pudiéramos encontrar un algoritmo polinomial para resolver uno de ellos, tendríamos automáticamente un algoritmo polinomial para resolver cualquiera de ellos.
- Nadie ha probado que no exista algoritmo polinomial para uno de ellos, pero muchísimos han intentado encontrar algoritmos.
- A veces cuando estás buscando un algoritmo para algún problema, si puedes probar que es equivalente a cualquiera de los problemas anteriores, pues ya mejor debes dejar tu búsqueda.

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?
- En realidad, un matemático súper-genio (o un oráculo), podría simplemente resolver el problema para todos los casos y luego decirle a la computadora que escupa la solución así nada más.

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?
- En realidad, un matemático súper-genio (o un oráculo), podría simplemente resolver el problema para todos los casos y luego decirle a la computadora que escupa la solución así nada más.
- Por ejemplo, búsqueda local muchas veces encuentra la solución óptima en muy poco tiempo.

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?
- En realidad, un matemático súper-genio (o un oráculo), podría simplemente resolver el problema para todos los casos y luego decirle a la computadora que escupa la solución así nada más.
- Por ejemplo, búsqueda local muchas veces encuentra la solución óptima en muy poco tiempo.
- El chiste es que puedas asegurar por completo, de alguna manera, que la respuesta que el algoritmo da en verdad sea correcta.

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?
- En realidad, un matemático súper-genio (o un oráculo), podría simplemente resolver el problema para todos los casos y luego decirle a la computadora que escupa la solución así nada más.
- Por ejemplo, búsqueda local muchas veces encuentra la solución óptima en muy poco tiempo.
- El chiste es que puedas asegurar por completo, de alguna manera, que la respuesta que el algoritmo da en verdad sea correcta.
- A veces el mismo algoritmo te asegura que lo que produce es la respuesta es correcta. Quizás buscó todas las posibilidades, o hizo algo de manera que se puede probar que no hay una mejor.

Garantías

- ¿Qué significa que no exista algoritmo polinomial para un problema?
- En realidad, un matemático súper-genio (o un oráculo), podría simplemente resolver el problema para todos los casos y luego decirle a la computadora que escupa la solución así nada más.
- Por ejemplo, búsqueda local muchas veces encuentra la solución óptima en muy poco tiempo.
- El chiste es que puedas asegurar por completo, de alguna manera, que la respuesta que el algoritmo da en verdad sea correcta.
- A veces el mismo algoritmo te asegura que lo que produce es la respuesta es correcta. Quizás buscó todas las posibilidades, o hizo algo de manera que se puede probar que no hay una mejor.
- Pero a veces podemos asegurar que algo es correcto de alguna manera que involucre adivinanza: certificados.

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?
- No hay un algoritmo polinomial conocido que responda esta pregunta.

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?
- No hay un algoritmo polinomial conocido que responda esta pregunta.
- Sin embargo, si ya supiéramos que la respuesta es sí, basta con dar ese subconjunto de números para probar que la respuesta es sí.

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?
- No hay un algoritmo polinomial conocido que responda esta pregunta.
- Sin embargo, si ya supiéramos que la respuesta es sí, basta con dar ese subconjunto de números para probar que la respuesta es sí.
- Entonces podría ser que tuviéramos un programa "adivinador" que muy rápidamente nos da un subconjunto de los números que sumen 0.

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?
- No hay un algoritmo polinomial conocido que responda esta pregunta.
- Sin embargo, si ya supiéramos que la respuesta es sí, basta con dar ese subconjunto de números para probar que la respuesta es sí.
- Entonces podría ser que tuviéramos un programa "adivinador" que muy rápidamente nos da un subconjunto de los números que sumen 0.
- Ese subconjunto en un certificado de que la respuesta es sí.

Intuición del certificado

- Ejemplo: Consideremos el siguiente problema: Dado un conjunto de números enteros, ¿existe un subconjunto de ellos cuya suma sea 0?
- No hay un algoritmo polinomial conocido que responda esta pregunta.
- Sin embargo, si ya supiéramos que la respuesta es sí, basta con dar ese subconjunto de números para probar que la respuesta es sí.
- Entonces podría ser que tuviéramos un programa "adivinador" que muy rápidamente nos da un subconjunto de los números que sumen 0.
- Ese subconjunto en un certificado de que la respuesta es sí.
- Para la respuesta no no tenemos un certificado tan fácil (de hecho, no se sabe si existe)

P vs NP: Intuición

Vamos a pensar en 4 clases de problemas de decisión (es decir, problemas en los que se hace una pregunta de sí y no y hay que decidir cuál es)

P vs NP: Intuición

Vamos a pensar en 4 clases de problemas de decisión (es decir, problemas en los que se hace una pregunta de sí y no y hay que decidir cuál es)

- P: Son los problemas que se pueden resolver en tiempo polinomial.

P vs NP: Intuición

Vamos a pensar en 4 clases de problemas de decisión (es decir, problemas en los que se hace una pregunta de sí y no y hay que decidir cuál es)

- P: Son los problemas que se pueden resolver en tiempo polinomial.
- NP: Son aquellos que existe un certificado para la respuesta sí que se puede verificar en tiempo polinomial.

P vs NP: Intuición

Vamos a pensar en 4 clases de problemas de decisión (es decir, problemas en los que se hace una pregunta de sí y no y hay que decidir cuál es)

- P: Son los problemas que se pueden resolver en tiempo polinomial.
- NP: Son aquellos que existe un certificado para la respuesta sí que se puede verificar en tiempo polinomial.
- co-NP: Son aquellos que existe un certificado para la respuesta no que se puede verificar en tiempo polinomial.

P vs NP: Intuición

Vamos a pensar en 4 clases de problemas de decisión (es decir, problemas en los que se hace una pregunta de sí y no y hay que decidir cuál es)

- P: Son los problemas que se pueden resolver en tiempo polinomial.
- NP: Son aquellos que existe un certificado para la respuesta sí que se puede verificar en tiempo polinomial.
- co-NP: Son aquellos que existe un certificado para la respuesta no que se puede verificar en tiempo polinomial.

En realidad co-NP no se estudia mucho, pues siempre puede uno preguntar la pregunta al revés y entrar en NP.

NP-completo y NP-difícil

- La clase de problemas NP-completo es un subconjunto de los problemas en NP que, intuitivamente, es tan difícil como cualquier problema en NP. Es decir:

NP-completo y NP-difícil

- La clase de problemas NP-completo es un subconjunto de los problemas en NP que, intuitivamente, es tan difícil como cualquier problema en NP. Es decir:
- Si X es un problema en NP, entonces puede ser convertido a cualquier problema en NP-completo en tiempo polinomial.

NP-completo y NP-difícil

- La clase de problemas NP-completo es un subconjunto de los problemas en NP que, intuitivamente, es tan difícil como cualquier problema en NP. Es decir:
- Si X es un problema en NP, entonces puede ser convertido a cualquier problema en NP-completo en tiempo polinomial.
- Esta clase contiene problemas muy conocidos, todos “equivalentes” entre sí.

NP-completo y NP-difícil

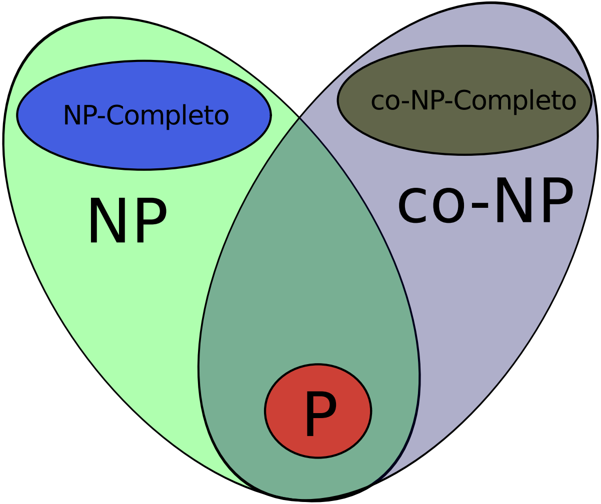
- La clase de problemas NP-completo es un subconjunto de los problemas en NP que, intuitivamente, es tan difícil como cualquier problema en NP. Es decir:
- Si X es un problema en NP, entonces puede ser convertido a cualquier problema en NP-completo en tiempo polinomial.
- Esta clase contiene problemas muy conocidos, todos "equivalentes" entre sí.
- Hay problemas aún más difíciles: Algunos problemas que no se sabe si están en NP, pero que su solución resolvería todos los problemas en NP.

NP-completo y NP-difícil

- La clase de problemas NP-completo es un subconjunto de los problemas en NP que, intuitivamente, es tan difícil como cualquier problema en NP. Es decir:
- Si X es un problema en NP, entonces puede ser convertido a cualquier problema en NP-completo en tiempo polinomial.
- Esta clase contiene problemas muy conocidos, todos "equivalentes" entre sí.
- Hay problemas aún más difíciles: Algunos problemas que no se sabe si están en NP, pero que su solución resolvería todos los problemas en NP.
- A estos les llamamos NP-difícil.

El diagrama más importante de computación

En caso de que P $\neq$ NP (que es lo que la mayoría cree), el diagrama se ve así:



Ejemplos

En:

- P: Decidir si un número es par.

Ejemplos

En:

- P: Decidir si un número es par.
- NP: Isomorfismo de gráficas: Dadas dos gráficas, decidir si son isomorfas (no se sabe si es NP-completo).

Ejemplos

En:

- P: Decidir si un número es par.
- NP: Isomorfismo de gráficas: Dadas dos gráficas, decidir si son isomorfas (no se sabe si es NP-completo).
- NP-completo: Encontrar subconjunto con suma 0; decidir si una gráfica se puede bien colorear con k colores, etc.

Ejemplos

En:

- P: Decidir si un número es par.
- NP: Isomorfismo de gráficas: Dadas dos gráficas, decidir si son isomorfas (no se sabe si es NP-completo).
- NP-completo: Encontrar subconjunto con suma 0; decidir si una gráfica se puede bien colorear con k colores, etc.
- co-NP $\cap$ NP: Dados n, m, decidir si $\exists x \leq n$ (y $x > 1$ con $x|m$) (resulta que existe un certificado de primalidad polinomial).

Ejemplos

En:

- P: Decidir si un número es par.
- NP: Isomorfismo de gráficas: Dadas dos gráficas, decidir si son isomorfas (no se sabe si es NP-completo).
- NP-completo: Encontrar subconjunto con suma 0; decidir si una gráfica se puede bien colorear con k colores, etc.
- co-NP $\cap$ NP: Dados n, m, decidir si $\exists x \leq n$ (y $x > 1$ con $x|m$) (resulta que existe un certificado de primalidad polinomial).
- NP-difícil: Optimización Entera (no está en NP)

Índice:

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es "sí" o "no".

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es “sí” o “no”.
- Tenemos tres tipos de problemas:

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es "sí" o "no".
- Tenemos tres tipos de problemas:
    - Decisión: La respuesta es sí o no

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es “sí” o “no”.
- Tenemos tres tipos de problemas:
  - Decisión: La respuesta es sí o no
  - Optimización: En un problema (X, f) donde X es un conjunto y f una función, queremos encontrar x tal que $f(x)$ es mínimo (o máximo).

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es "sí" o "no".
- Tenemos tres tipos de problemas:
  - Decisión: La respuesta es sí o no
  - Optimización: En un problema (X, f) donde X es un conjunto y f una función, queremos encontrar x tal que $f(x)$ es mínimo (o máximo).
  - Evaluación: Una relajación del problema de optimización: En un problema (X, f) donde X es un conjunto y f una función, queremos encontrar valor de f en el punto en donde alcance su mínimo (o máximo).

Decisión vs Optimización vs Evaluación

- Todo lo anterior es para problemas de decisión: Problemas en que la respuesta es "sí" o "no".
- Tenemos tres tipos de problemas:
  - Decisión: La respuesta es sí o no
  - Optimización: En un problema (X, f) donde X es un conjunto y f una función, queremos encontrar x tal que $f(x)$ es mínimo (o máximo).
  - Evaluación: Una relajación del problema de optimización: En un problema (X, f) donde X es un conjunto y f una función, queremos encontrar valor de f en el punto en donde alcance su mínimo (o máximo).
- Recordemos que tanto X como f están dados por algoritmos: Para X el algoritmo debe decidir si x está o no en X para cualquier objeto posible x.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?
- Ejemplo 2: Número de clan:

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?
- Ejemplo 2: Número de clan:
  - Optimización: Encontrar el máximo clan de una gráfica.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?
- Ejemplo 2: Número de clan:
  - Optimización: Encontrar el máximo clan de una gráfica.
  - Evaluación: Encontrar el número de clan de una gráfica.

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?
- Ejemplo 2: Número de clan:
  - Optimización: Encontrar el máximo clan de una gráfica.
  - Evaluación: Encontrar el número de clan de una gráfica.
  - Decisión: Dado k, ¿existe un clan de G de tamaño k o más?

Problemas asociados

- Dado un problema de optimización, tiene asociados un problema de evaluación y un problema de decisión.
- Ejemplo 1: Número cromático:
  - Optimización: Encontrar una coloración de la gráfica que utilice la menor cantidad posible de colores.
  - Evaluación: Encontrar el número cromático de una gráfica.
  - Decisión: Dado k, ¿existe una coloración de G con k colores?
- Ejemplo 2: Número de clan:
  - Optimización: Encontrar el máximo clan de una gráfica.
  - Evaluación: Encontrar el número de clan de una gráfica.
  - Decisión: Dado k, ¿existe un clan de G de tamaño k o más?
- El problema de decisión usualmente toma la siguiente forma: Dado c, ¿existe $x \in X$ con $f(x) \leq c$?

Convertir de uno al otro

- Dado el problema de decisión, casi siempre podemos resolver el de evaluación. Por ejemplo, para el problema de bien-colorear una gráfica:

Convertir de uno al otro

- Dado el problema de decisión, casi siempre podemos resolver el de evaluación. Por ejemplo, para el problema de bien-colorear una gráfica:
- Decisión $\rightarrow$ Evaluación: para cada k entre 1 y $|V(G)|$ nos preguntamos si se puede colorear con k colores. El primero en el que se pueda colorear es el número cromático.

Convertir de uno al otro

- Dado el problema de decisión, casi siempre podemos resolver el de evaluación. Por ejemplo, para el problema de bien-colorear una gráfica:
- Decisión $\to$ Evaluación: para cada k entre 1 y $|V(G)|$ nos preguntamos si se puede colorear con k colores. El primero en el que se pueda colorear es el número cromático.
- No hay ninguna manera consistente de pasar del problema de evaluación al de optimización.

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:
- Evaluación $\rightarrow$ Optimización: Recursivamente:

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:
- Evaluación $\rightarrow$ Optimización: Recursivamente:
  - Supongamos que tenemos la función χ que para cualquier gráfica encuentra el número cromático.

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:
- Evaluación $\to$ Optimización: Recursivamente:
  - Supongamos que tenemos la función χ que para cualquier gráfica encuentra el número cromático.
  - Para cada pareja de vértices u, v que no haya arista de u a v, consideramos la contracción $G/(u, v)$. Si $\chi(G/(u, v)) = \chi(G)$, entonces coloreamos la gráfica contraída y luego des-contraemos.

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:
- Evaluación $\rightarrow$ Optimización: Recursivamente:
  - Supongamos que tenemos la función χ que para cualquier gráfica encuentra el número cromático.
  - Para cada pareja de vértices u, v que no haya arista de u a v, consideramos la contracción $G/(u, v)$. Si $\chi(G/(u, v)) = \chi(G)$, entonces coloreamos la gráfica contraída y luego des-contraemos.
  - Si no existen esos u, v es porque la única coloración es la que usa todos los colores, pues en la coloración óptima de G usualmente habrá dos vértices que tengan el mismo color.

Ejemplo donde si se puede regresar

- A veces sí se puede. Por ejemplo, para número cromático:
- Evaluación $\rightarrow$ Optimización: Recursivamente:
  - Supongamos que tenemos la función χ que para cualquier gráfica encuentra el número cromático.
  - Para cada pareja de vértices u, v que no haya arista de u a v, consideramos la contracción $G/(u, v)$. Si $\chi(G/(u, v)) = \chi(G)$, entonces coloreamos la gráfica contraída y luego des-contraemos.
  - Si no existen esos u, v es porque la única coloración es la que usa todos los colores, pues en la coloración óptima de G usualmente habrá dos vértices que tengan el mismo color.
- Tarea: Demuestra que en el problema de clanes también se puede regresar uno.

Índice:

Máquina de Turing

- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.

Máquina de Turing

- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.
- La Máquina de Turing determinista es el modelo matemático más común para una computadora.

Máquina de Turing

- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.
- La Máquina de Turing determinista es el modelo matemático más común para una computadora.
- Nota: Los números al azar de la computadora en realidad no son al azar realmente, sólo muy difíciles de predecir dados algunos de ellos (aunque conociendo la formulita, puede uno generarlos... la computadora lo hace).

Máquina de Turing

- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.
- La Máquina de Turing determinista es el modelo matemático más común para una computadora.
- Nota: Los números al azar de la computadora en realidad no son al azar realmente, sólo muy difíciles de predecir dados algunos de ellos (aunque conociendo la formulita, puede uno generarlos... la computadora lo hace).
- La formalización es un poco rara, pero se puede probar que todo lo que puede hacer la computadora lo puede hacer la máquina de Turing y viceversa.

Máquina de Turing

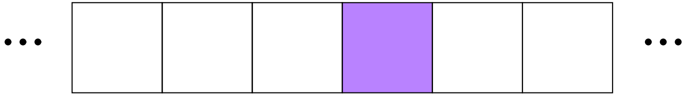
- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.
- La Máquina de Turing determinista es el modelo matemático más común para una computadora.
- Nota: Los números al azar de la computadora en realidad no son al azar realmente, sólo muy difíciles de predecir dados algunos de ellos (aunque conociendo la formulita, puede uno generarlos... la computadora lo hace).
- La formalización es un poco rara, pero se puede probar que todo lo que puede hacer la computadora lo puede hacer la máquina de Turing y viceversa.
- Bueno... no todo. La máquina de Turing tiene memoria ilimitada y la computadora no.

Máquina de Turing

- Antes de poder formalizar bien qué es eso de NP y etc, tenemos que definir bien qué es lo que puede y qué es lo que no puede hacer una computadora.
- La Máquina de Turing determinista es el modelo matemático más común para una computadora.
- Nota: Los números al azar de la computadora en realidad no son al azar realmente, sólo muy difíciles de predecir dados algunos de ellos (aunque conociendo la formulita, puede uno generarlos... la computadora lo hace).
- La formalización es un poco rara, pero se puede probar que todo lo que puede hacer la computadora lo puede hacer la máquina de Turing y viceversa.
- Bueno... no todo. La máquina de Turing tiene memoria ilimitada y la computadora no.
- Obviamente las computadoras no están construidas en la realidad como una máquina de Turing: eso sería extremadamente lento.

Máquina de Turing: Intuición

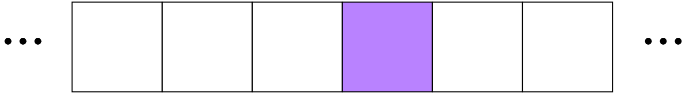
- Uno se imagina a una máquina de Turing como una banda infinita de cuadraditos:



en donde hay un símbolo en cada cuadrito.

Máquina de Turing: Intuición

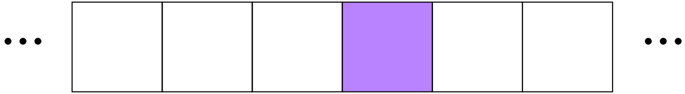
- Uno se imagina a una máquina de Turing como una banda infinita de cuadraditos:

  en donde hay un símbolo en cada cuadrito.
- Hay un cuadrito indicado (el cuadrito “actual”)

Máquina de Turing: Intuición

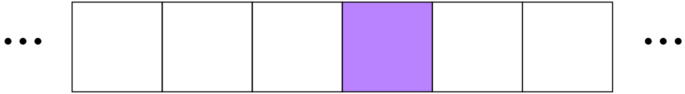
- Uno se imagina a una máquina de Turing como una banda infinita de cuadraditos:



  en donde hay un símbolo en cada cuadrito.
- Hay un cuadrito indicado (el cuadrito "actual")
- Hay estados que tiene el programa.

Máquina de Turing: Intuición

- Uno se imagina a una máquina de Turing como una banda infinita de cuadraditos:



  en donde hay un símbolo en cada cuadrito.
- Hay un cuadrito indicado (el cuadrito "actual")
- Hay estados que tiene el programa.
- La máquina tiene además una serie de instrucciones del tipo: "Si estás en el estado 47 y ves el simbolito 8, escribe 3, muévete a la derecha y pasa al estado 21".

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.
- Γ es un alfabeto finito de los símbolos que pueden estar en la banda.

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.
- Γ es un alfabeto finito de los símbolos que pueden estar en la banda.
- $b \in \Gamma$ es el símbolo “vacío” (es el único símbolo que puede ocurrir una infinidad de veces)

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.
- Γ es un alfabeto finito de los símbolos que pueden estar en la banda.
- $b \in \Gamma$ es el símbolo "vacío" (es el único símbolo que puede ocurrir una infinidad de veces)
- $q_0 \in Q$ es el estado inicial.

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.
- Γ es un alfabeto finito de los símbolos que pueden estar en la banda.
- $b \in \Gamma$ es el símbolo "vacío" (es el único símbolo que puede ocurrir una infinidad de veces)
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Máquina de Turing

Formalmente, una máquina de Turing es una sextupla $M = \langle Q, \Gamma, b, \delta, q_0, F \rangle$, donde:

- Q es un conjunto finito y no vacío de estados.
- Γ es un alfabeto finito de los símbolos que pueden estar en la banda.
- $b \in \Gamma$ es el símbolo "vacío" (es el único símbolo que puede ocurrir una infinidad de veces)
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.
- $\delta : Q \setminus F \times \Gamma \to Q \setminus F \times \Gamma \times \{L, R, N\}$ es la función de transición.

Información en la banda

- Una máquina de Turing actúa en un conjunto inicial de símbolos escritos en la banda, donde todos menos una cantidad finita de ellos son b.

Información en la banda

- Una máquina de Turing actúa en un conjunto inicial de símbolos escritos en la banda, donde todos menos una cantidad finita de ellos son b.
- Formalmente, una máquina de Turing actúa en una función $\Sigma : \mathbb{Z} \to \Gamma$ donde $\Sigma(n) = b$ para todos los $n \in \mathbb{Z}$ salvo una cantidad finita.

Información en la banda

- Una máquina de Turing actúa en un conjunto inicial de símbolos escritos en la banda, donde todos menos una cantidad finita de ellos son b.
- Formalmente, una máquina de Turing actúa en una función $\Sigma : \mathbb{Z} \to \Gamma$ donde $\Sigma(n) = b$ para todos los $n \in \mathbb{Z}$ salvo una cantidad finita.
- El número de cuadritos desde el primero hasta el último que no son b es el tamaño de la instancia

Información en la banda

- Una máquina de Turing actúa en un conjunto inicial de símbolos escritos en la banda, donde todos menos una cantidad finita de ellos son b.
- Formalmente, una máquina de Turing actúa en una función $\Sigma : \mathbb{Z} \to \Gamma$ donde $\Sigma(n) = b$ para todos los $n \in \mathbb{Z}$ salvo una cantidad finita.
- El número de cuadritos desde el primero hasta el último que no son b es el tamaño de la instancia
- Se puede probar que todas las cosas que hace la computadora, como sumar, restar, etc. las puede hacer una máquina de Turing. Veremos algunos ejemplos, pero probarlo es muy largo.

Información en la banda

- Una máquina de Turing actúa en un conjunto inicial de símbolos escritos en la banda, donde todos menos una cantidad finita de ellos son b.
- Formalmente, una máquina de Turing actúa en una función $\Sigma : \mathbb{Z} \to \Gamma$ donde $\Sigma(n) = b$ para todos los $n \in \mathbb{Z}$ salvo una cantidad finita.
- El número de cuadritos desde el primero hasta el último que no son b es el tamaño de la instancia
- Se puede probar que todas las cosas que hace la computadora, como sumar, restar, etc. las puede hacer una máquina de Turing. Veremos algunos ejemplos, pero probarlo es muy largo.
- El poder se encuentra en la función de transición.

Índice:

Definiciones de Problema y P

- Sea
$\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \ : \ x(n) = b$ para todo n salvo una cantidad finita.$\}$

Definiciones de Problema y P

- Sea
  $\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \; : \; x(n) = b$ para todo n salvo una cantidad finita.$\}$
- Un problema de decisión es una función $\mathcal{P} : \mathcal{I} \to \{$SÍ, NO$\}$ donde $\mathcal{I} \subset \mathcal{X}$.

Definiciones de Problema y P

- Sea
  $\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \ : \ x(n) = b$ para todo n salvo una cantidad finita.$\}$
- Un problema de decisión es una función $\mathcal{P} : \mathcal{I} \to \{$SÍ, NO$\}$ donde $\mathcal{I} \subset \mathcal{X}$.
- Podemos escribir $\mathcal{I} = \mathcal{I}_S \cup \mathcal{I}_N$, donde $\mathcal{I}_S$ es el conjunto de instancias en donde la respuesta es sí (resp. $\mathcal{I}_N$).

Definiciones de Problema y P

- Sea
  $\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \ : \ x(n) = b$ para todo n salvo una cantidad finita.$\}$
- Un problema de decisión es una función $\mathcal{P} : \mathcal{I} \to \{$SÍ, NO$\}$ donde $\mathcal{I} \subset \mathcal{X}$.
- Podemos escribir $\mathcal{I} = \mathcal{I}_S \cup \mathcal{I}_N$, donde $\mathcal{I}_S$ es el conjunto de instancias en donde la respuesta es sí (resp. $\mathcal{I}_N$).
- Dado $\mathcal{P}$ un problema, un algoritmo $\mathcal{A}$ que resuelve $\mathcal{P}$ es una Máquina de Turing con *dos estados finales*, SÍ y NO, tal que para todo $x \in \mathcal{I}$, la máquina termina en el estado $\mathcal{P}(x)$.

Definiciones de Problema y P

- Sea
  $\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \ : \ x(n) = b$ para todo n salvo una cantidad finita.$\}$
- Un problema de decisión es una función $\mathcal{P} : \mathcal{I} \to \{\text{SÍ}, \text{NO}\}$ donde $\mathcal{I} \subset \mathcal{X}$.
- Podemos escribir $\mathcal{I} = \mathcal{I}_S \cup \mathcal{I}_N$, donde $\mathcal{I}_S$ es el conjunto de instancias en donde la respuesta es sí (resp. $\mathcal{I}_N$).
- Dado $\mathcal{P}$ un problema, un algoritmo $\mathcal{A}$ que resuelve $\mathcal{P}$ es una Máquina de Turing con *dos estados finales*, SÍ y NO, tal que para todo $x \in \mathcal{I}$, la máquina termina en el estado $\mathcal{P}(x)$.
- Decimos que un problema de decisión $\mathcal{P}$ está en P si existe un algoritmo $\mathcal{A}$ que resuelve a $\mathcal{P}$, y un polinomio p, tal que para todo $x \in \mathcal{I}$, el algoritmo $\mathcal{A}$ termina en una cantidad de pasos $\leq p(|x|)$.

Definiciones de Problema y P

- Sea
  $\mathcal{X} := \{x \in \Gamma^{\mathbb{Z}} \ : \ x(n) = b$ para todo n salvo una cantidad finita.$\}$
- Un problema de decisión es una función $\mathcal{P} : \mathcal{I} \to \{\text{SÍ}, \text{NO}\}$ donde $\mathcal{I} \subset \mathcal{X}$.
- Podemos escribir $\mathcal{I} = \mathcal{I}_S \cup \mathcal{I}_N$, donde $\mathcal{I}_S$ es el conjunto de instancias en donde la respuesta es sí (resp. $\mathcal{I}_N$).
- Dado $\mathcal{P}$ un problema, un algoritmo $\mathcal{A}$ que resuelve $\mathcal{P}$ es una Máquina de Turing con *dos estados finales*, SÍ y NO, tal que para todo $x \in \mathcal{I}$, la máquina termina en el estado $\mathcal{P}(x)$.
- Decimos que un problema de decisión $\mathcal{P}$ está en P si existe un algoritmo $\mathcal{A}$ que resuelve a $\mathcal{P}$, y un polinomio p, tal que para todo $x \in \mathcal{I}$, el algoritmo $\mathcal{A}$ termina en una cantidad de pasos $\leq p(|x|)$.
- Agregaremos a Γ un símbolo especial \$ (ahorita veremos para qué).

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

- *Termina en $\mathcal{P}(x)$.*

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

- *Termina en $\mathcal{P}(x)$.*
- *Cuando $x \in \mathcal{I}_S$ y $w = c(x)$, $\mathcal{A}$ termina en tiempo polinomial.*

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

- *Termina en $\mathcal{P}(x)$.*
- *Cuando $x \in \mathcal{I}_S$ y $w = c(x)$, $\mathcal{A}$ termina en tiempo polinomial.*

- Definimos co-NP de la misma manera, cambiando "SI" por "NO".

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

- *Termina en $\mathcal{P}(x)$.*
- *Cuando $x \in \mathcal{I}_S$ y $w = c(x)$, $\mathcal{A}$ termina en tiempo polinomial.*

- Definimos co-NP de la misma manera, cambiando “SI” por “NO”.
- Notemos que uno no necesariamente debe poder encontrar el certificado $c(x)$ en tiempo polinomial, sólo debemos ver que existe.

NP: Certificados

Definición

Decimos que un problema de decisión $\mathcal{P}$ está en NP si existe:

- *Un polinomio p.*
- *Un certificado $c : \mathcal{I}_S \to \mathcal{X}$ tal que $|c(x)| \leq p(|x|)$ para toda $x \in \mathcal{I}_S$.*
- *Un algoritmo $\mathcal{A}$.*

de modo que para toda instancia $x \in \mathcal{I}_S$ y para todo $w \in \mathcal{X}$, $\mathcal{A}$ evaluado en $(x\$w)$ satisface:

- *Termina en $\mathcal{P}(x)$.*
- *Cuando $x \in \mathcal{I}_S$ y $w = c(x)$, $\mathcal{A}$ termina en tiempo polinomial.*

- Definimos co-NP de la misma manera, cambiando "SI" por "NO".
- Notemos que uno no necesariamente debe poder encontrar el certificado $c(x)$ en tiempo polinomial, sólo debemos ver que existe.
- Obviamente P $\subset$ NP.

Ejemplos de problemas en NP

- Coloración de Gráficas.

Ejemplos de problemas en NP

- Coloración de Gráficas.
- Clan

Ejemplos de problemas en NP

- Coloración de Gráficas.
- Clan
- Circuito Hamiltoniano

Ejemplos de problemas en NP

- Coloración de Gráficas.
- Clan
- Circuito Hamiltoniano
- Agente Viajero

Observación de NP

- De hecho, podemos decir que todo problema *razonable* de optimización combinatoria está en NP:

Observación de NP

- De hecho, podemos decir que todo problema *razonable* de optimización combinatoria está en NP:
- Si queremos construir el *mejor* objeto tal que no-se-qué, sería muy feo que semejante objeto no pudiera ser revisado en tiempo polinomial, así que el mejor objeto siempre es un certificado de la respuesta SI.

Reducción Polinomial

- Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ problemas de decisión.

Reducción Polinomial

- Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ problemas de decisión.
- Decimos que $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$ si existen algoritmos $\mathcal{A}_1$ y $\mathcal{A}_2$ de modo que $\mathcal{A}_1$ toma tiempo polinomial suponiendo que puede utilizar $\mathcal{A}_2$ en tiempo unitario.

Reducción Polinomial

- Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ problemas de decisión.
- Decimos que $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$ si existen algoritmos $\mathcal{A}_1$ y $\mathcal{A}_2$ de modo que $\mathcal{A}_1$ toma tiempo polinomial suponiendo que puede utilizar $\mathcal{A}_2$ en tiempo unitario.
- Obviamente si $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$, y existe un algoritmo polinomial para $\mathcal{P}_2$, entonces existe un algoritmo polinomial para $\mathcal{P}_1$.

Reducción Polinomial

- Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ problemas de decisión.
- Decimos que $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$ si existen algoritmos $\mathcal{A}_1$ y $\mathcal{A}_2$ de modo que $\mathcal{A}_1$ toma tiempo polinomial suponiendo que puede utilizar $\mathcal{A}_2$ en tiempo unitario.
- Obviamente si $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$, y existe un algoritmo polinomial para $\mathcal{P}_2$, entonces existe un algoritmo polinomial para $\mathcal{P}_1$.
- Decimos que dos problemas son equivalentes si ambos se reducen en tiempo polinomial al otro.

Reducción Polinomial

- Sean $\mathcal{P}_1$ y $\mathcal{P}_2$ problemas de decisión.
- Decimos que $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$ si existen algoritmos $\mathcal{A}_1$ y $\mathcal{A}_2$ de modo que $\mathcal{A}_1$ toma tiempo polinomial suponiendo que puede utilizar $\mathcal{A}_2$ en tiempo unitario.
- Obviamente si $\mathcal{P}_1$ se reduce en tiempo polinomial a $\mathcal{P}_2$, y existe un algoritmo polinomial para $\mathcal{P}_2$, entonces existe un algoritmo polinomial para $\mathcal{P}_1$.
- Decimos que dos problemas son equivalentes si ambos se reducen en tiempo polinomial al otro.
- Nos interesa principalmente un caso particular de esto: Si para cada $x \in \mathcal{I}(\mathcal{P}_1)$ podemos construir $y \in \mathcal{I}(\mathcal{P}_2)$ en tiempo polinomial tal que $\mathcal{P}_1(x) = \mathcal{P}_2(y)$, entonces $\mathcal{P}_1$ se transforma en $\mathcal{P}_2$.

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:
    - $\mathcal{P} \in$ NP.

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:
  - $\mathcal{P} \in$ NP.
  - Todo problema $\mathcal{Q} \in$ NP se puede transformar en tiempo polinomial a $\mathcal{P}$.

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:
  - $\mathcal{P} \in$ NP.
  - Todo problema $\mathcal{Q} \in$ NP se puede transformar en tiempo polinomial a $\mathcal{P}$.
- Para ver que un problema $\mathcal{P}$ es NP-completo, para probar la segunda afirmación, usualmente uno ve que un problema que ya es conocido que es NP-completo se puede reducir a $\mathcal{P}$ en tiempo polinomial.

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:
  - $\mathcal{P} \in$ NP.
  - Todo problema $\mathcal{Q} \in$ NP se puede transformar en tiempo polinomial a $\mathcal{P}$.
- Para ver que un problema $\mathcal{P}$ es NP-completo, para probar la segunda afirmación, usualmente uno ve que un problema que ya es conocido que es NP-completo se puede reducir a $\mathcal{P}$ en tiempo polinomial.
- Pero el primer problema que probaremos que es NP-completo debemos hacerlo directamente.

NP-Completo

- Decimos que un problema $\mathcal{P}$ es NP-completo si:
  - $\mathcal{P} \in$ NP.
  - Todo problema $\mathcal{Q} \in$ NP se puede transformar en tiempo polinomial a $\mathcal{P}$.
- Para ver que un problema $\mathcal{P}$ es NP-completo, para probar la segunda afirmación, usualmente uno ve que un problema que ya es conocido que es NP-completo se puede reducir a $\mathcal{P}$ en tiempo polinomial.
- Pero el primer problema que probaremos que es NP-completo debemos hacerlo directamente.
- Probaremos la existencia de problemas NP-completos en la siguiente sección.

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

$$\delta \subset (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

$$\delta \subset (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

- ¿Qué acción tomará dicha máquina?

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

$$\delta \subset (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

- ¿Qué acción tomará dicha máquina?
- Hay dos maneras de verlo:

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

$$\delta \subset (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

- ¿Qué acción tomará dicha máquina?
- Hay dos maneras de verlo:
  - Una es que siempre tomará la "mejor acción posible" (adivinará de alguna manera).

Máquina de Turing no-determinista

- Una máquina de Turing no-determinista es lo mismo que una Máquina de Turing, pero ahora en vez de que en cada estado y cada símbolo que uno lee haya una única instrucción, puede haber varias.
- Es decir, en vez de que δ sea una función, simplemente será un subconjunto del producto cartesiano:

$$\delta \subset (Q \setminus F \times \Gamma) \times (Q \times \Gamma \times \{L, R, N\})$$

- ¿Qué acción tomará dicha máquina?
- Hay dos maneras de verlo:
  - Una es que siempre tomará la "mejor acción posible" (adivinará de alguna manera).
  - Otra es que tomará todas las posibles acciones y se "dividirá" en varios "universos paralelos" (o bueno, se crearán varias Máquinas de Turing que correrán simultaneamente).

NP

Teorema

Un problema $\mathcal{P}$ está en NP sí y sólo sí es soluble por una máquina de Turing no-determinista, y el tiempo para resolverlo es polinomial para la respuesta SI (y quizás infinito para la respuesta "no")

NP

Teorema

Un problema $\mathcal{P}$ está en NP sí y sólo sí es soluble por una máquina de Turing no-determinista, y el tiempo para resolverlo es polinomial para la respuesta SI (y quizás infinito para la respuesta "no")

Prueba (idea):

- ($\Longrightarrow$) Si $\mathcal{P} \in$ NP entonces existe un certificado c que puede ser verificado en tiempo polinomial.

NP

Teorema

Un problema $\mathcal{P}$ está en NP sí y sólo sí es soluble por una máquina de Turing no-determinista, y el tiempo para resolverlo es polinomial para la respuesta SI (y quizás infinito para la respuesta "no")

Prueba (idea):

- ($\Longrightarrow$) Si $\mathcal{P} \in$ NP entonces existe un certificado c que puede ser verificado en tiempo polinomial.
- Para cada $x \in \mathcal{I}$ podemos crear una máquina de Turing no determinista que evalúe todos los posibles certificados, de uno por uno. pizarrón

NP

Teorema

Un problema $\mathcal{P}$ está en NP sí y sólo sí es soluble por una máquina de Turing no-determinista, y el tiempo para resolverlo es polinomial para la respuesta SI (y quizás infinito para la respuesta "no")

Prueba (idea):

- ($\Longrightarrow$) Si $\mathcal{P} \in$ NP entonces existe un certificado c que puede ser verificado en tiempo polinomial.
- Para cada $x \in \mathcal{I}$ podemos crear una máquina de Turing no determinista que evalúe todos los posibles certificados, de uno por uno. pizarrón
- ($\Leftarrow$) Si la respuesta fue SI en una máquina de Turing no-determinista, quiere decir que existe un camino que pudo tomar la máquina para llegar a la respuesta. Ese camino puede ser codificado como un certificado para la máquina de Turing determinista.

Índice:

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.
- Una fórmula es algo de la forma:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_4 \vee x_3) \wedge \ldots$$

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.
- Una fórmula es algo de la forma:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_4 \vee x_3) \wedge \dots$$

- Donde $\neg$ es la negación, $\vee$ es *o* y $\wedge$ es *y*.

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.
- Una fórmula es algo de la forma:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_4 \vee x_3) \wedge \ldots$$

- Donde $\neg$ es la negación, $\vee$ es *o* y $\wedge$ es *y*.
- Un problema central de lógica es: dada una fórmula $F(\vec{x})$, ¿existe una asignación de las variables x_1, x_2, ... , x_n de modo que $F(x)$ sea verdadero?

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.
- Una fórmula es algo de la forma:

$$(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_4 \vee x_3) \wedge \ldots$$

- Donde $\neg$ es la negación, $\vee$ es *o* y $\wedge$ es *y*.
- Un problema central de lógica es: dada una fórmula $F(\vec{x})$, ¿existe una asignación de las variables $x_1, x_2, \ldots, x_n$ de modo que $F(x)$ sea verdadero?
- Observemos que podemos modelar $\implies$, $\iff$ y cualquier otra cosa lógica que se nos ocurra con $\neg, \vee, \wedge$.

SAT

- Una variable booleana es una variable que puede tomar valores verdadero y falso.
- Una fórmula es algo de la forma:

$$(x_1 \lor x_2 \lor \neg x_3) \land (\neg x_4 \lor x_3) \land \ldots$$

- Donde $\neg$ es la negación, $\lor$ es *o* y $\land$ es *y*.
- Un problema central de lógica es: dada una fórmula $F(\vec{x})$, ¿existe una asignación de las variables $x_1, x_2, \ldots, x_n$ de modo que $F(x)$ sea verdadero?
- Observemos que podemos modelar $\implies$, $\iff$ y cualquier otra cosa lógica que se nos ocurra con $\neg, \lor, \land$.
- Un algoritmo que funciona es revisar todas las posibles asignaciones a x_i, pero hay 2^n posibilidades.

SAT es NP-completo

Teorema (Cook)

SAT es NP-completo.

SAT es NP-completo

Teorema (Cook)

SAT es NP-completo.

Prueba:

- Claramente SAT $\in$ NP, pues una asignación de las variables es un certificado.

SAT es NP-completo

Teorema (Cook)

SAT es NP-completo.

Prueba:

- Claramente SAT $\in$ NP, pues una asignación de las variables es un certificado.
- Sea $\mathcal{P}$ un problema en NP. Utilizando sólo la definición de NP, vamos a reducir a $\mathcal{P}$ en tiempo polinomial a SAT.

SAT es NP-completo

Teorema (Cook)

SAT es NP-completo.

Prueba:

- Claramente SAT $\in$ NP, pues una asignación de las variables es un certificado.
- Sea $\mathcal{P}$ un problema en NP. Utilizando sólo la definición de NP, vamos a reducir a $\mathcal{P}$ en tiempo polinomial a SAT.
- Es decir, para cada $x \in \mathcal{I}(\mathcal{P})$ debemos construir $y \in \mathcal{I}(\mathrm{SAT})$ tal que $\mathcal{P}(x) = \mathrm{SAT}(y)$ (i.e. que la respuesta sea la misma en ambas).

SAT es NP-completo

Teorema (Cook)

SAT es NP-completo.

Prueba:

- Claramente SAT $\in$ NP, pues una asignación de las variables es un certificado.
- Sea $\mathcal{P}$ un problema en NP. Utilizando sólo la definición de NP, vamos a reducir a $\mathcal{P}$ en tiempo polinomial a SAT.
- Es decir, para cada $x \in \mathcal{I}(\mathcal{P})$ debemos construir $y \in \mathcal{I}(\mathrm{SAT})$ tal que $\mathcal{P}(x) = \mathrm{SAT}(y)$ (i.e. que la respuesta sea la misma en ambas).
- Tenemos entonces un algoritmo $\mathcal{A}$, un certificado c, y $\mathcal{A}$ corre en tiempo polinomial p para revisar el certificado.

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.
- El significado intuitivo de $T[i, \gamma, t]$ será: En el tiempo t, en la posición i, el simbolito en la cinta será γ.

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.
- El significado intuitivo de $T[i, \gamma, t]$ será: En el tiempo t, en la posición i, el simbolito en la cinta será γ.
- Para cada $0 \leq t \leq p(|x|)$ y cada $0 \leq i \leq p(|x|) + 1$, creamos una variable $H[i, t]$

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.
- El significado intuitivo de $T[i, \gamma, t]$ será: En el tiempo t, en la posición i, el simbolito en la cinta será γ.
- Para cada $0 \leq t \leq p(|x|)$ y cada $0 \leq i \leq p(|x|) + 1$, creamos una variable $H[i, t]$
- El significado intuitivo de $H[i, t]$ será: En el tiempo t, en la posición i, el algoritmo estará haciendo la operación ℓ.

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.
- El significado intuitivo de $T[i, \gamma, t]$ será: En el tiempo t, en la posición i, el simbolito en la cinta será γ.
- Para cada $0 \leq t \leq p(|x|)$ y cada $0 \leq i \leq p(|x|) + 1$, creamos una variable $H[i, t]$
- El significado intuitivo de $H[i, t]$ será: En el tiempo t, en la posición i, el algoritmo estará haciendo la operación ℓ.
- Para cada estado q y cada tiempo t creamos una variable $Q[q, t]$ si en el tiempo t la máquina está en estado q.

SAT es NP-completo

Vamos a crear un montón de variables:

- Para cada $0 \leq t, i \leq p(|x|)$ y para cada $\gamma \in \Gamma$, creamos una variable booleana $T[i, \gamma, t]$.
- El significado intuitivo de $T[i, \gamma, t]$ será: En el tiempo t, en la posición i, el simbolito en la cinta será γ.
- Para cada $0 \leq t \leq p(|x|)$ y cada $0 \leq i \leq p(|x|) + 1$, creamos una variable $H[i, t]$
- El significado intuitivo de $H[i, t]$ será: En el tiempo t, en la posición i, el algoritmo estará haciendo la operación ℓ.
- Para cada estado q y cada tiempo t creamos una variable $Q[q, t]$ si en el tiempo t la máquina está en estado q.
- La fórmula $F(x)$ constará de varias partes, todas unidas con $\wedge$.

SAT es NP-completo: Condiciones Iniciales

- Condiciones iniciales:

SAT es NP-completo: Condiciones Iniciales

- Condiciones iniciales:
  - Al principio la banda tiene escrito a x: Para cada posición i, si en x el simbolito es γ, entonces

$$T[i, \gamma, 0]$$

SAT es NP-completo: Condiciones Iniciales

- Condiciones iniciales:
  - Al principio la banda tiene escrito a x: Para cada posición i, si en x el simbolito es γ, entonces

$$T[i, \gamma, 0]$$

  - El cuadrito marcado es el 0 al principio:

$$H[0, 0]$$

SAT es NP-completo: Condiciones Iniciales

- Condiciones iniciales:
  - Al principio la banda tiene escrito a x: Para cada posición i, si en x el simbolito es γ, entonces

$$T[i, \gamma, 0]$$

  - El cuadrito marcado es el 0 al principio:

$$H[0, 0]$$

  - Estado Inicial: Si q_0 es el estado inicial,

$$Q[q_0, 0]$$

SAT es NP-completo: Condiciones Iniciales

- Condiciones iniciales:
  - Al principio la banda tiene escrito a x: Para cada posición i, si en x el simbolito es γ, entonces
  $$T[i, \gamma, 0]$$
  - El cuadrito marcado es el 0 al principio:
  $$H[0, 0]$$
  - Estado Inicial: Si q_0 es el estado inicial,
  $$Q[q_0, 0]$$
- En cada momento hay exactamente un símbolo en cada posición de la banda: Para cada t,i y $\gamma \neq \gamma'$,
$$(\neg T[i, \gamma, t] \vee \neg T[i, \gamma', t])$$

SAT es NP-completo: Más Condiciones

- Sólo se puede estar en un estado a la vez: Si $q \neq q'$,

$$\neg Q[q, t] \vee \neg Q[q', t]$$

SAT es NP-completo: Más Condiciones

- Sólo se puede estar en un estado a la vez: Si $q \neq q'$,

$$\neg Q[q,t] \vee \neg Q[q',t]$$

- Sólo se puede tener un cuadrito marcado a la vez: Si $i \neq i'$:

$$\neg H[i,t] \vee \neg H[i',t]$$

SAT es NP-completo: Más Condiciones

- Sólo se puede estar en un estado a la vez: Si $q \neq q'$,

$$\neg Q[q,t] \vee \neg Q[q',t]$$

- Sólo se puede tener un cuadrito marcado a la vez: Si $i \neq i'$:

$$\neg H[i,t] \vee \neg H[i',t]$$

- Para que cambie la banda, debes escribir en ella: Para todos i, t y $\gamma \neq \gamma'$, tenemos que

$$\neg T[i,\gamma,t] \vee \neg T[i,\gamma',t+1] \vee H[i,t]$$

o equivalentemente,

$$(T[i,\gamma,t] \wedge T[i,\gamma',t+1]) \implies H[i,t]$$

SAT es NP-completo: Más Condiciones

- Sólo se puede estar en un estado a la vez: Si $q \neq q'$,

$$\neg Q[q,t] \vee \neg Q[q',t]$$

- Sólo se puede tener un cuadrito marcado a la vez: Si $i \neq i'$:

$$\neg H[i,t] \vee \neg H[i',t]$$

- Para que cambie la banda, debes escribir en ella: Para todos i, t y $\gamma \neq \gamma'$, tenemos que

$$\neg T[i,\gamma,t] \vee \neg T[i,\gamma',t+1] \vee H[i,t]$$

  o equivalentemente,

$$(T[i,\gamma,t] \wedge T[i,\gamma',t+1]) \implies H[i,t]$$

- Debemos terminar en un estado final: Si $F = \{f_0, f_1, ..., f_s\}$

$$Q[f_0, p(|x|)] \vee Q[f_1, p(|x|)] \vee ... \vee Q[f_s, p(|x|)]$$

SAT es NP-completo: Condiciones de Transición

- Condiciones de transición: Supongamos que $T[i, \gamma, t]$, $H[i, t]$ y $Q[q, t]$ son verdaderos: En el tiempo t, estamos en el estado q, en la posición i y vemos escrito γ en la banda.

SAT es NP-completo: Condiciones de Transición

- Condiciones de transición: Supongamos que $T[i, \gamma, t]$, $H[i, t]$ y $Q[q, t]$ son verdaderos: En el tiempo t, estamos en el estado q, en la posición i y vemos escrito γ en la banda.
- Es decir, la fórmula será $(T[i, \gamma, t] \wedge H[i, t] \wedge Q[q, t]) \implies$ algo.

SAT es NP-completo: Condiciones de Transición

- Condiciones de transición: Supongamos que $T[i, \gamma, t]$, $H[i, t]$ y $Q[q, t]$ son verdaderos: En el tiempo t, estamos en el estado q, en la posición i y vemos escrito γ en la banda.
- Es decir, la fórmula será $(T[i, \gamma, t] \wedge H[i, t] \wedge Q[q, t]) \implies$ algo.
- Ese algo debe representar las posibilidades para el siguiente paso de la máquina:

SAT es NP-completo: Condiciones de Transición

- Condiciones de transición: Supongamos que $T[i,\gamma,t]$, $H[i,t]$ y $Q[q,t]$ son verdaderos: En el tiempo t, estamos en el estado q, en la posición i y vemos escrito γ en la banda.
- Es decir, la fórmula será $(T[i,\gamma,t] \wedge H[i,t] \wedge Q[q,t]) \implies$ algo.
- Ese algo debe representar las posibilidades para el siguiente paso de la máquina:
- Recordemos que

$$\delta \subset (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$$

SAT es NP-completo: Condiciones de Transición

- Condiciones de transición: Supongamos que $T[i, \gamma, t]$, $H[i, t]$ y $Q[q, t]$ son verdaderos: En el tiempo t, estamos en el estado q, en la posición i y vemos escrito γ en la banda.
- Es decir, la fórmula será $(T[i, \gamma, t] \wedge H[i, t] \wedge Q[q, t]) \implies$ algo.
- Ese algo debe representar las posibilidades para el siguiente paso de la máquina:
- Recordemos que

$$\delta \subset (Q \setminus F) \times \Gamma \times Q \times \Gamma \times \{-1, 0, 1\}$$

- Dado lo anterior, deben existir q', γ' y d tal que $(q, \gamma, q', \gamma', d) \in \delta$ y lo siguiente es verdadero:

$$H[i + d, t + 1] \wedge Q[q', t + 1] \wedge T[i, \gamma', t + 1]$$

SAT es NP-completo: Conclusión

- Ya es todo.

SAT es NP-completo: Conclusión

- Ya es todo.
- Podemos crear el problema en SAT dado anteriormente en tiempo polinomial (no es inmediato, pero podemos revisar cada una de las condiciones).

SAT es NP-completo: Conclusión

- Ya es todo.
- Podemos crear el problema en SAT dado anteriormente en tiempo polinomial (no es inmediato, pero podemos revisar cada una de las condiciones).
- Claramente $\mathcal{P}(x) = \mathrm{SAT}(y)$ cuando y es la fórmula que se obtuvo a partir de x en la manera descrita (creando la máquina de Turing). $\blacksquare$

SAT es NP-completo: Conclusión

- Ya es todo.
- Podemos crear el problema en SAT dado anteriormente en tiempo polinomial (no es inmediato, pero podemos revisar cada una de las condiciones).
- Claramente $\mathcal{P}(x) = \mathrm{SAT}(y)$ cuando y es la fórmula que se obtuvo a partir de x en la manera descrita (creando la máquina de Turing). $\blacksquare$
- Entonces, si pudiéramos resolver SAT en tiempo polinomial, podríamos resolver $\mathcal{P}$.

Índice:

Otros NP-completos

- Ahora que probamos que SAT es NP-completo, ya es mucho más fácil ver que otros problemas lo son.

Otros NP-completos

- Ahora que probamos que SAT es NP-completo, ya es mucho más fácil ver que otros problemas lo son.
- Para ver $\mathcal{P}$ es NP-completo, tenemos que ver que:

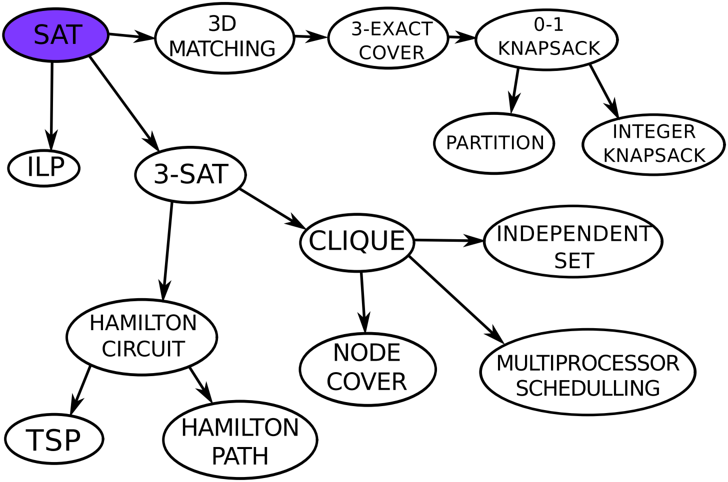
Otros NP-completos

- Ahora que probamos que SAT es NP-completo, ya es mucho más fácil ver que otros problemas lo son.
- Para ver $\mathcal{P}$ es NP-completo, tenemos que ver que:
  - $\mathcal{P}$ está en NP.

Otros NP-completos

- Ahora que probamos que SAT es NP-completo, ya es mucho más fácil ver que otros problemas lo son.
- Para ver $\mathcal{P}$ es NP-completo, tenemos que ver que:
  - $\mathcal{P}$ está en NP.
  - Dada una fórmula de SAT (o una instancia de algún otro problema NP-completo) podemos transformarla en tiempo polinomial a una instancia de $\mathcal{P}$)

Diagrama de problemas NP-completos



3-SAT

3-SAT

- Restringimos SAT al caso en donde todas las cláusulas tienen 3 variables (i.e. es decir, todas son del tipo:

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

3-SAT

- Restringimos SAT al caso en donde todas las cláusulas tienen 3 variables (i.e. es decir, todas son del tipo:

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

- Claramente está en NP.

3-SAT

- Restringimos SAT al caso en donde todas las cláusulas tienen 3 variables (i.e. es decir, todas son del tipo:

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

- Claramente está en NP.
- Supongamos que tenemos una fórmula en SAT con cláusulas $C_1, C_2, \ldots, C_n$ y vamos a transformar el problema a uno en donde todas las cláusulas tienen tamaño 3.

3-SAT

- Restringimos SAT al caso en donde todas las cláusulas tienen 3 variables (i.e. es decir, todas son del tipo:

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

- Claramente está en NP.
- Supongamos que tenemos una fórmula en SAT con cláusulas $C_1, C_2, ..., C_n$ y vamos a transformar el problema a uno en donde todas las cláusulas tienen tamaño 3.
- Con probar que podemos transformar una claúsula, ya es suficiente, pero se les queda de ejercicio.

CLIQUE es NP-completo

CLIQUE es NP-completo

- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.

CLIQUE es NP-completo

- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.
- Dada una fórmula F en 3-SAT, vamos a construir una gráfica G y un número k tal que G tiene un K_k sí y sólo sí la fórmula tiene solución.

CLIQUE es NP-completo

- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.
- Dada una fórmula F en 3-SAT, vamos a construir una gráfica G y un número k tal que G tiene un K_k sí y sólo sí la fórmula tiene solución.
- Supongamos que F está compuesto por las cláusulas C_1, C_2, ... , C_k (es decir, k es el número de cláusulas)

CLIQUE es NP-completo

- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.
- Dada una fórmula F en 3-SAT, vamos a construir una gráfica G y un número k tal que G tiene un K_k sí y sólo sí la fórmula tiene solución.
- Supongamos que F está compuesto por las cláusulas C_1, C_2, ... , C_k (es decir, k es el número de cláusulas)
- Habrá 7 vértices de G por cada cláusula: Uno por cada asignación de las variables involucradas en C_i, salvo la asignación en la que los 3 son falsos.

CLIQUE es NP-completo

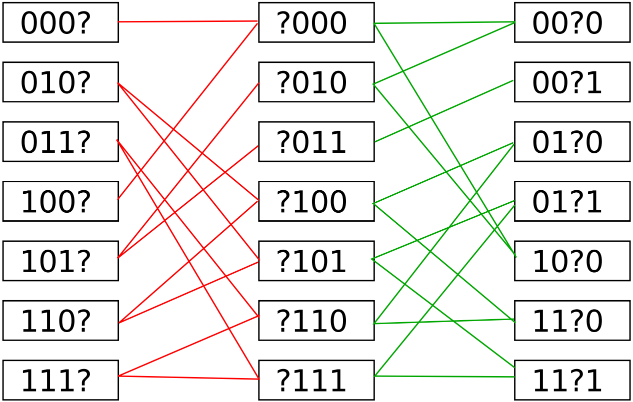
- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.
- Dada una fórmula F en 3-SAT, vamos a construir una gráfica G y un número k tal que G tiene un K_k sí y sólo sí la fórmula tiene solución.
- Supongamos que F está compuesto por las cláusulas C_1, C_2, ... , C_k (es decir, k es el número de cláusulas)
- Habrá 7 vértices de G por cada cláusula: Uno por cada asignación de las variables involucradas en C_i, salvo la asignación en la que los 3 son falsos.
- Vamos a poner aristas entre vértices de cláusulas distintas cuando las asignaciones parciales son "congruentes entre sí".

CLIQUE es NP-completo

- Obviamente está en NP. Vamos a reducir 3-SAT a CLIQUE.
- Dada una fórmula F en 3-SAT, vamos a construir una gráfica G y un número k tal que G tiene un K_k sí y sólo sí la fórmula tiene solución.
- Supongamos que F está compuesto por las cláusulas C_1, C_2, ... , C_k (es decir, k es el número de cláusulas)
- Habrá 7 vértices de G por cada cláusula: Uno por cada asignación de las variables involucradas en C_i, salvo la asignación en la que los 3 son falsos.
- Vamos a poner aristas entre vértices de cláusulas distintas cuando las asignaciones parciales son "congruentes entre sí".
- Se entiende mucho más con un ejemplo.

CLIQUE es NP-Completo: Ejemplo

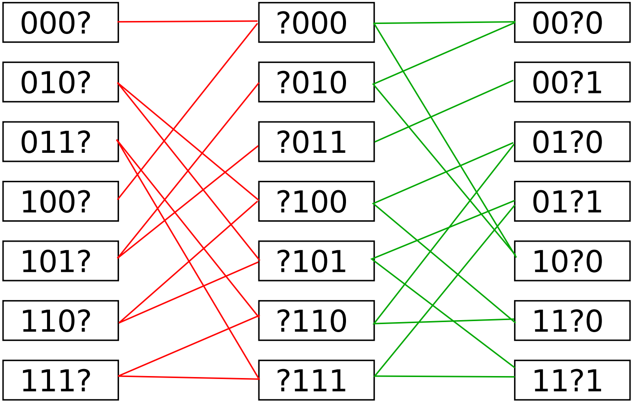
$$(x \vee y \vee \neg z) \wedge (y \vee z \vee \neg w) \wedge (\neg x \vee y \vee \neg w)$$



Falta: 001? ?001 10?1

CLIQUE es NP-Completo: Ejemplo

$$(x \vee y \vee \neg z) \wedge (y \vee z \vee \neg w) \wedge (\neg x \vee y \vee \neg w)$$



Falta: 001? ?001 10?1

Nota: No puse las aristas del tercero al primero

CLIQUE es NP-completo

- Claramente, una gráfica completa de k vértices corresponde a una asignación de las variables en que cada cláusula es verdadera.

CLIQUE es NP-completo

- Claramente, una gráfica completa de k vértices corresponde a una asignación de las variables en que cada cláusula es verdadera.
- ¡Y ya! $\blacksquare$.

VERTEX COVER y INDEPENDENT SET

- INDEPENDENT SET: Dada una gráfica, decidir si tiene un conjunto independiente de tamaño k.

VERTEX COVER y INDEPENDENT SET

- INDEPENDENT SET: Dada una gráfica, decidir si tiene un conjunto independiente de tamaño k.
- Encontrar el conjunto independiente máximo es lo mismo que encontrar el número de clan en el complemento de la gráfica, así que INDEPENDENT SET es NP-completo.

VERTEX COVER y INDEPENDENT SET

- INDEPENDENT SET: Dada una gráfica, decidir si tiene un conjunto independiente de tamaño k.
- Encontrar el conjunto independiente máximo es lo mismo que encontrar el número de clan en el complemento de la gráfica, así que INDEPENDENT SET es NP-completo.
- VERTEX COVER: Dada una gráfica, decidir si existe un subconjunto de k vértices donde cada arista es incidente al menos un vértice del subconjunto.

VERTEX COVER y INDEPENDENT SET

- INDEPENDENT SET: Dada una gráfica, decidir si tiene un conjunto independiente de tamaño k.
- Encontrar el conjunto independiente máximo es lo mismo que encontrar el número de clan en el complemento de la gráfica, así que INDEPENDENT SET es NP-completo.
- VERTEX COVER: Dada una gráfica, decidir si existe un subconjunto de k vértices donde cada arista es incidente al menos un vértice del subconjunto.
- El complemento de un conjunto independiente es una cubierta de vértices, así que VERTEX COVER es NP-completo.

GRAPH COLORING es NP-completo

- Veamos que 3-SAT se reduce a 4-GRAPH-COLORING, que es un caso particular de GRAPH COLORING.

GRAPH COLORING es NP-completo

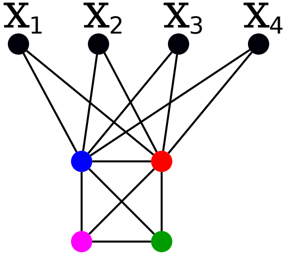
- Veamos que 3-SAT se reduce a 4-GRAPH-COLORING, que es un caso particular de GRAPH COLORING.
- Pon un K_4. Para colorearlo con 4 colores, deberá haber uno de cada color. Digamos que los colores son Rojo, Azul, Verde y Fucsia.

GRAPH COLORING es NP-completo

- Veamos que 3-SAT se reduce a 4-GRAPH-COLORING, que es un caso particular de GRAPH COLORING.
- Pon un K_4. Para colorearlo con 4 colores, deberá haber uno de cada color. Digamos que los colores son Rojo, Azul, Verde y Fucsia.
- Pon un vértice por cada variable $x_1, ..., x_n$ y fuerza a que esos vértices sean Verdes o Fucsias (que correspondan a Verdadero o Falso),

GRAPH COLORING es NP-completo

- Veamos que 3-SAT se reduce a 4-GRAPH-COLORING, que es un caso particular de GRAPH COLORING.
- Pon un K_4. Para colorearlo con 4 colores, deberá haber uno de cada color. Digamos que los colores son Rojo, Azul, Verde y Fucsia.
- Pon un vértice por cada variable $x_1, ..., x_n$ y fuerza a que esos vértices sean Verdes o Fucsias (que correspondan a Verdadero o Falso), Ejemplo:



4-GRAPH COLORING

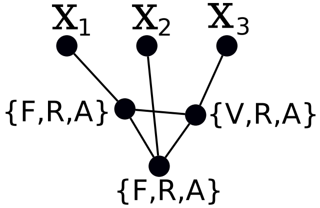
- Por cada cláusula tendrás que poner alguna construcción que tenga 8 opciones, y que una de ellas (exactamente) no se pueda, pero las otras 7 sí.

4-GRAPH COLORING

- Por cada cláusula tendrás que poner alguna construcción que tenga 8 opciones, y que una de ellas (exactamente) no se pueda, pero las otras 7 sí.
- Ejemplo: $(x_1 \vee x_2 \vee \neg x_3)$

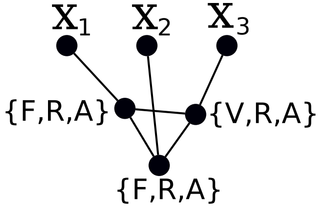
4-GRAPH COLORING

- Por cada cláusula tendrás que poner alguna construcción que tenga 8 opciones, y que una de ellas (exactamente) no se pueda, pero las otras 7 sí.
- Ejemplo: $(x_1 \vee x_2 \vee \neg x_3)$



4-GRAPH COLORING

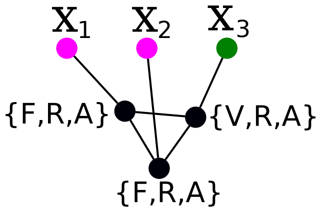
- Por cada cláusula tendrás que poner alguna construcción que tenga 8 opciones, y que una de ellas (exactamente) no se pueda, pero las otras 7 sí.
- Ejemplo: $(x_1 \vee x_2 \vee \neg x_3)$



- ¿Qué pasaría si no se cumpliera la cláusula?

4-GRAPH COLORING

- Por cada cláusula tendrás que poner alguna construcción que tenga 8 opciones, y que una de ellas (exactamente) no se pueda, pero las otras 7 sí.
- Ejemplo: $(x_1 \vee x_2 \vee \neg x_3)$



- ¿Qué pasaría si no se cumpliera la cláusula?

MULTIPROCESSOR SCHEDULING

- La intuición de este problema es que si tienes una bola de tareas en donde algunas deben ser realizadas antes que otras, y una bola de personas (o computadoras), encontrar la mínima cantidad de tiempo en que se pueden realizar las tareas.

MULTIPROCESSOR SCHEDULING

- La intuición de este problema es que si tienes una bola de tareas en donde algunas deben ser realizadas antes que otras, y una bola de personas (o computadoras), encontrar la mínima cantidad de tiempo en que se pueden realizar las tareas.
- Formalmente: Dado un conjunto parcialmente ordenado $(\mathcal{J}, \leq)$ (las tareas), un entero m (el número de máquinas), y un entero T (el tiempo que tienes para resolver el problema), ¿existe un horario $S : \mathcal{J} \to [T]$ de modo que:

MULTIPROCESSOR SCHEDULING

- La intuición de este problema es que si tienes una bola de tareas en donde algunas deben ser realizadas antes que otras, y una bola de personas (o computadoras), encontrar la mínima cantidad de tiempo en que se pueden realizar las tareas.
- Formalmente: Dado un conjunto parcialmente ordenado $(\mathcal{J}, \leq)$ (las tareas), un entero m (el número de máquinas), y un entero T (el tiempo que tienes para resolver el problema), ¿existe un horario $S : \mathcal{J} \to [T]$ de modo que:
  - Si $J_1 < J_2$ entonces $S(J_1) < S(J_2)$.

MULTIPROCESSOR SCHEDULING

- La intuición de este problema es que si tienes una bola de tareas en donde algunas deben ser realizadas antes que otras, y una bola de personas (o computadoras), encontrar la mínima cantidad de tiempo en que se pueden realizar las tareas.
- Formalmente: Dado un conjunto parcialmente ordenado $(\mathcal{J}, \leq)$ (las tareas), un entero m (el número de máquinas), y un entero T (el tiempo que tienes para resolver el problema), ¿existe un horario $S : \mathcal{J} \to [T]$ de modo que:
  - Si $J_1 < J_2$ entonces $S(J_1) < S(J_2)$.
  - Para todo $i \in [T]$, $|\{J \in \mathcal{J} : S(J) = i\}| \leq m$.

MULTIPROCESSOR SCHEDULING

- La intuición de este problema es que si tienes una bola de tareas en donde algunas deben ser realizadas antes que otras, y una bola de personas (o computadoras), encontrar la mínima cantidad de tiempo en que se pueden realizar las tareas.
- Formalmente: Dado un conjunto parcialmente ordenado $(\mathcal{J}, \leq)$ (las tareas), un entero m (el número de máquinas), y un entero T (el tiempo que tienes para resolver el problema), ¿existe un horario $S : \mathcal{J} \to [T]$ de modo que:
    - Si $J_1 < J_2$ entonces $S(J_1) < S(J_2)$.
    - Para todo $i \in [T]$, $|\{J \in \mathcal{J} : S(J) = i\}| \leq m$.
- Obviamente MULTIPROCESSOR SCHEDULING está en NP, pues la función S es un certificado.

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a
- Además, vamos a tomar trabajos B, C, y D (en 3 niveles) de manera que intentemos forzar lo siguiente: En el tiempo

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a
- Además, vamos a tomar trabajos B, C, y D (en 3 niveles) de manera que intentemos forzar lo siguiente: En el tiempo
  - 1: Haremos B, y k de V (que formarán el K_k)

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a
- Además, vamos a tomar trabajos B, C, y D (en 3 niveles) de manera que intentemos forzar lo siguiente: En el tiempo
  - 1: Haremos B, y k de V (que formarán el K_k)
  - 2: Haremos C, y $\binom{k}{2}$ de A y el resto de V.

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a
- Además, vamos a tomar trabajos B, C, y D (en 3 niveles) de manera que intentemos forzar lo siguiente: En el tiempo
  - 1: Haremos B, y k de V (que formarán el K_k)
  - 2: Haremos C, y $\binom{k}{2}$ de A y el resto de V.
  - 3: Haremos D, y los que sobran de A.

MULTIPROCESSOR SCHEDULING es NP-completo

- Transformaremos CLIQUE en MULTIPROCESSOR SCHEDULING.
- Dada una gráfica $G = (V, A)$ y un número k, queremos ver si G tiene un clan de tamaño k o más.
- Podemos suponer que G es conexa, pues podemos hacer el problema en cada componente conexa (y encontrar componentes conexas es polinomial).
- Vamos a tomar $T = 3$ y vamos a poner trabajos V y A (vértices y aristas), donde $v \leq a \iff v$ es un vértice de la arista a
- Además, vamos a tomar trabajos B, C, y D (en 3 niveles) de manera que intentemos forzar lo siguiente: En el tiempo
  - 1: Haremos B, y k de V (que formarán el K_k)
  - 2: Haremos C, y $\binom{k}{2}$ de A y el resto de V.
  - 3: Haremos D, y los que sobran de A.
- Si logramos forzar eso, ya tendremos un clan de tamaño k. Veamos que hay que pedirle a B, C y D.

MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:

MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:
  - $|C| = 1$

MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:
  - $|C| = 1$
  - $m = \binom{k}{2} + (|V| - k) + |C| = k + |B| = |A| - \binom{k}{2} + |D|$

MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:
  - $|C| = 1$
  - $m = \binom{k}{2} + (|V| - k) + |C| = k + |B| = |A| - \binom{k}{2} + |D|$
- Estas ecuaciones nos definen m (el número de máquinas).

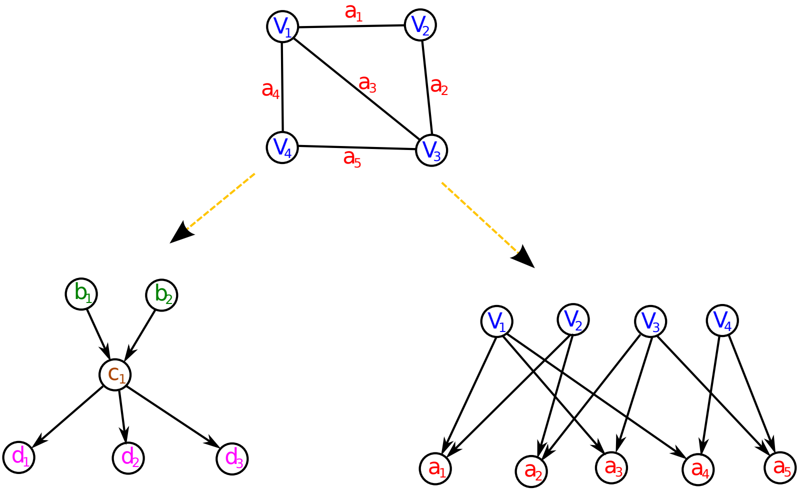
MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:
  - $|C| = 1$
  - $m = \binom{k}{2} + (|V| - k) + |C| = k + |B| = |A| - \binom{k}{2} + |D|$
- Estas ecuaciones nos definen m (el número de máquinas).
- Como hay exactamente $3m$ trabajos, todas las máquinas deben estar trabajando siempre.

MULTIPROCESSOR SCHEDULING es NP-completo

- Definimos $\mathcal{J} = V \cup A \cup B \cup C \cup D$, donde B, C, y D son conjuntos finitos no vacíos disjuntos que satisfacen:
  - $|C| = 1$
  - $m = \binom{k}{2} + (|V| - k) + |C| = k + |B| = |A| - \binom{k}{2} + |D|$
- Estas ecuaciones nos definen m (el número de máquinas).
- Como hay exactamente $3m$ trabajos, todas las máquinas deben estar trabajando siempre.
- No puede haber vértices en el tiempo 3, ni aristas en el tiempo 1.

Ejemplo con $k = 3$, $m = 5$



Ejemplo

En el ejemplo, sí hay un horario que funciona:

Máquina	**Tiempo 1**	**Tiempo 2**	**Tiempo 3**
Máquina 1	b_1	c_1	d_1
Máquina 2	b_2	a_1	d_2
Máquina 3	v_1	a_2	d_3
Máquina 4	v_2	a_3	a_4
Máquina 5	v_3	v_4	a_5

MULTIPROCESSOR SCHEDULING es NP-completo

- Como no puede haber aristas en el tiempo 1, deberá haber k vértices.

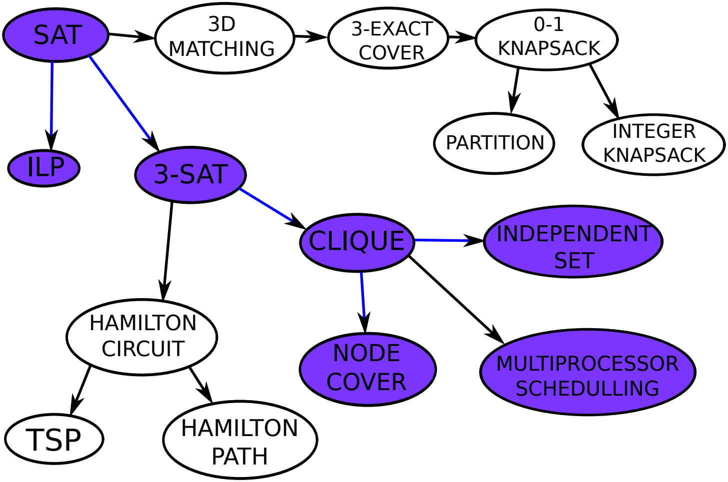
MULTIPROCESSOR SCHEDULING es NP-completo

- Como no puede haber aristas en el tiempo 1, deberá haber k vértices.
- En el tiempo 2, entonces, tendrá que haber al menos $\binom{k}{2}$ aristas, pero esas tienen que ser incidentes a los vértices del tiempo 1.

MULTIPROCESSOR SCHEDULING es NP-completo

- Como no puede haber aristas en el tiempo 1, deberá haber k vértices.
- En el tiempo 2, entonces, tendrá que haber al menos $\binom{k}{2}$ aristas, pero esas tienen que ser incidentes a los vértices del tiempo 1.
- La única manera es que los vértices del tiempo 1 formen un K_k. $\blacksquare$

Cómo vamos en el diagrama



HAMILTON CIRCUIT es NP-completo

- HAMILTON CIRCUIT: Dada una gráfica G, ¿tiene un circuito Hamiltoniano?

HAMILTON CIRCUIT es NP-completo

- HAMILTON CIRCUIT: Dada una gráfica G, ¿tiene un circuito Hamiltoniano?
- Veremos que 3-SAT se transforma polinomialmente en HAMILTON CIRCUIT.

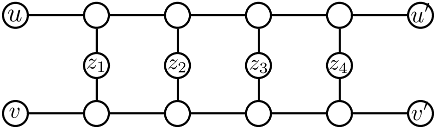
HAMILTON CIRCUIT es NP-completo

- HAMILTON CIRCUIT: Dada una gráfica G, ¿tiene un circuito Hamiltoniano?
- Veremos que 3-SAT se transforma polinomialmente en HAMILTON CIRCUIT.
- Sea F una fórmula con cláusulas $C_1, ..., C_m$ con variables $x_1, ..., x_n$.

HAMILTON CIRCUIT es NP-completo

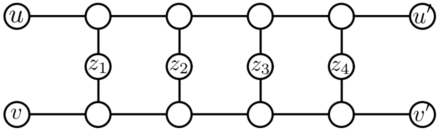
- HAMILTON CIRCUIT: Dada una gráfica G, ¿tiene un circuito Hamiltoniano?
- Veremos que 3-SAT se transforma polinomialmente en HAMILTON CIRCUIT.
- Sea F una fórmula con cláusulas $C_1, ..., C_m$ con variables $x_1, ..., x_n$.
- La idea será crear una gráfica con varias "partes", que en caso de tener circuito hamiltoniano, dicho circuito deberá darnos una asignación de verdadero y falso para las variables.

HAMILTON CIRCUIT es NP-completo



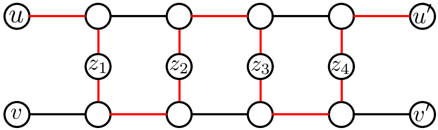
- Supongamos que una parte de la gráfica G se ve así, donde u, u', v y v' pueden tener más aristas pero los demás no tienen más.

HAMILTON CIRCUIT es NP-completo



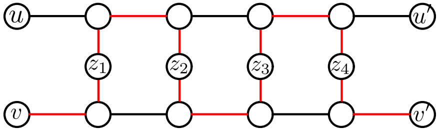
- Supongamos que una parte de la gráfica G se ve así, donde u, u', v y v' pueden tener más aristas pero los demás no tienen más.
- Si G tuviera circuito Hamiltoniano, tendría que ser uno de dos.

HAMILTON CIRCUIT es NP-completo



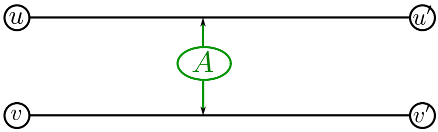
- Supongamos que una parte de la gráfica G se ve así, donde u, u', v y v' pueden tener más aristas pero los demás no tienen más.
- Si G tuviera circuito Hamiltoniano, tendría que ser uno de dos.

HAMILTON CIRCUIT es NP-completo



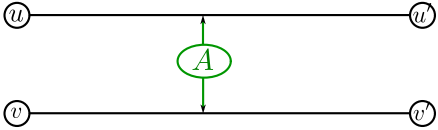
- Supongamos que una parte de la gráfica G se ve así, donde u, u', v y v' pueden tener más aristas pero los demás no tienen más.
- Si G tuviera circuito Hamiltoniano, tendría que ser uno de dos.

HAMILTON CIRCUIT es NP-completo



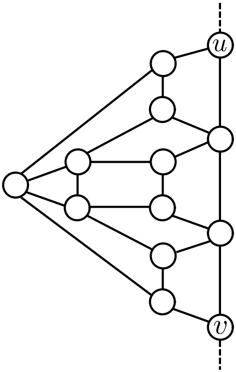
- Podemos verlo como que hay una arista de u a u' y una de v a v', pero que cualquier circuito Hamiltoniano debe usar sólo uno de los dos.

HAMILTON CIRCUIT es NP-completo



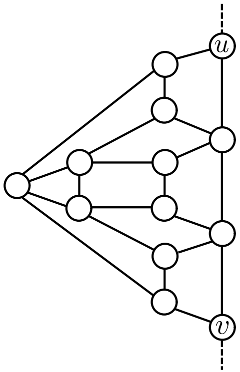
- Podemos verlo como que hay una arista de u a u' y una de v a v', pero que cualquier circuito Hamiltoniano debe usar sólo uno de los dos.
- Denotamos esta situación poniendo una A y las dos aristas.

HAMILTON CIRCUIT es NP-completo



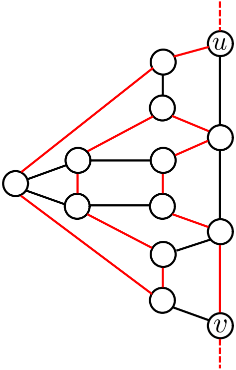
- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.

HAMILTON CIRCUIT es NP-completo



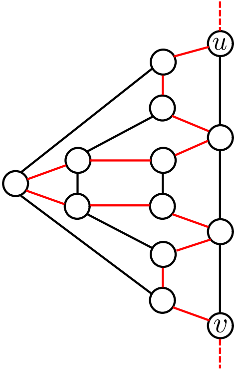
- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

HAMILTON CIRCUIT es NP-completo



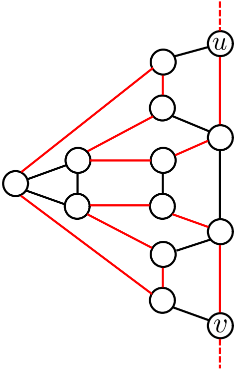
- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

HAMILTON CIRCUIT es NP-completo



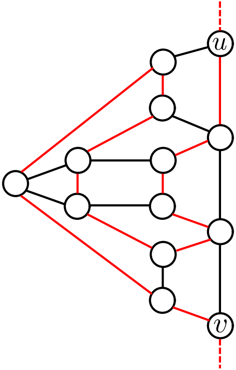
- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

HAMILTON CIRCUIT es NP-completo



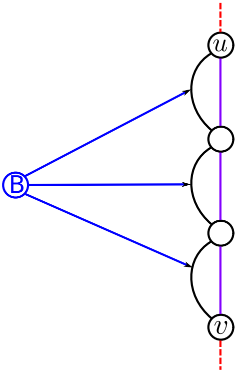
- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

HAMILTON CIRCUIT es NP-completo



- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

HAMILTON CIRCUIT es NP-completo



- De la misma manera, veamos que en una gráfica como ésta hay varias opciones para un circuito.
- No podemos utilizar todas las aristas de la derecha, pero podemos utilizar cualquier subconjunto de ellas.

A y B

- Vamos a utilizar B para asegurar que las cláusulas sean todas verdaderas: Es decir, que utilizar las 3 aristas de la derecha "corresponda" a cuando la cláusula es falsa.

A y B

- Vamos a utilizar B para asegurar que las cláusulas sean todas verdaderas: Es decir, que utilizar las 3 aristas de la derecha "corresponda" a cuando la cláusula es falsa.
- Vamos a utilizar A para "pegar" las cláusulas: que si una variable es "verdadera" en una cláusula, deba serlo en las otras.

A y B

- Vamos a utilizar B para asegurar que las cláusulas sean todas verdaderas: Es decir, que utilizar las 3 aristas de la derecha "corresponda" a cuando la cláusula es falsa.
- Vamos a utilizar A para "pegar" las cláusulas: que si una variable es "verdadera" en una cláusula, deba serlo en las otras.
- Ponemos entonces una copia de B por cada cláusula en serie y un ciclo de tamaño 2 por cada variable (también en serie).

A y B

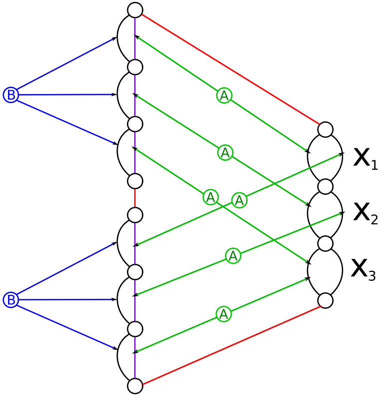
- Vamos a utilizar B para asegurar que las cláusulas sean todas verdaderas: Es decir, que utilizar las 3 aristas de la derecha "corresponda" a cuando la cláusula es falsa.
- Vamos a utilizar A para "pegar" las cláusulas: que si una variable es "verdadera" en una cláusula, deba serlo en las otras.
- Ponemos entonces una copia de B por cada cláusula en serie y un ciclo de tamaño 2 por cada variable (también en serie).
- Después "pegamos" las cláusulas al correspondiente de las variables que tienen con A's.

A y B

- Vamos a utilizar B para asegurar que las cláusulas sean todas verdaderas: Es decir, que utilizar las 3 aristas de la derecha "corresponda" a cuando la cláusula es falsa.
- Vamos a utilizar A para "pegar" las cláusulas: que si una variable es "verdadera" en una cláusula, deba serlo en las otras.
- Ponemos entonces una copia de B por cada cláusula en serie y un ciclo de tamaño 2 por cada variable (también en serie).
- Después "pegamos" las cláusulas al correspondiente de las variables que tienen con A's.
- Con un ejemplo se entiende mejor, obviamente.

Ejemplo

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



HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?

HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?
- Vamos a transformar HAMILTON CIRCUIT a HAMILTON PATH.

HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?
- Vamos a transformar HAMILTON CIRCUIT a HAMILTON PATH.
- Sea G una gráfica en la cual nos preguntamos si existe un circuito Hamiltoniano.

HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?
- Vamos a transformar HAMILTON CIRCUIT a HAMILTON PATH.
- Sea G una gráfica en la cual nos preguntamos si existe un circuito Hamiltoniano.
- Podemos suponer que no hay vértices aislados.

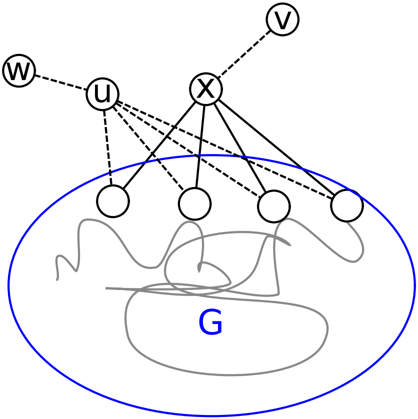
HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?
- Vamos a transformar HAMILTON CIRCUIT a HAMILTON PATH.
- Sea G una gráfica en la cual nos preguntamos si existe un circuito Hamiltoniano.
- Podemos suponer que no hay vértices aislados.
- Sea x un vértice cualquiera específico y consideramos sus vecinos.

HAMILTON PATH es NP-completo

- HAMILTON PATH: Dada una gráfica G, ¿existe una trayectoria que visite todos los vértices exactamente una vez?
- Vamos a transformar HAMILTON CIRCUIT a HAMILTON PATH.
- Sea G una gráfica en la cual nos preguntamos si existe un circuito Hamiltoniano.
- Podemos suponer que no hay vértices aislados.
- Sea x un vértice cualquiera específico y consideramos sus vecinos.
- Agregaremos 3 vértices nuevos, u, v, w y unimos a u con todos los vecinos de x, unimos a v con x y a w con u.

HAMILTON PATH es NP-completo



HAMILTON PATH es NP-completo

- Claramente esta nueva gráfica tiene camino Hamiltoniano sí y sólo si G tenía ciclo Hamiltoniano:

HAMILTON PATH es NP-completo

- Claramente esta nueva gráfica tiene camino Hamiltoniano sí y sólo si G tenía ciclo Hamiltoniano:
- Si existe un camino Hamiltoniano, debe empezar en x y terminar en w.

HAMILTON PATH es NP-completo

- Claramente esta nueva gráfica tiene camino Hamiltoniano sí y sólo si G tenía ciclo Hamiltoniano:
- Si existe un camino Hamiltoniano, debe empezar en x y terminar en w.
- Es decir, su segundo y penúltimo vértice deberán ser x y u.

HAMILTON PATH es NP-completo

- Claramente esta nueva gráfica tiene camino Hamiltoniano sí y sólo si G tenía ciclo Hamiltoniano:
- Si existe un camino Hamiltoniano, debe empezar en x y terminar en w.
- Es decir, su segundo y penúltimo vértice deberán ser x y u.
- Pero esto quiere decir que podemos "pegar" a x con u y crear un ciclo Hamiltoniano.

HAMILTON PATH es NP-completo

- Claramente esta nueva gráfica tiene camino Hamiltoniano sí y sólo si G tenía ciclo Hamiltoniano:
- Si existe un camino Hamiltoniano, debe empezar en x y terminar en w.
- Es decir, su segundo y penúltimo vértice deberán ser x y u.
- Pero esto quiere decir que podemos "pegar" a x con u y crear un ciclo Hamiltoniano.
- Si existe un ciclo Hamiltoniano en G, podemos, al "regresar" a x mejor ir a u y obtener un camino Hamiltoniano en la nueva gráfica.

TSP es NP-completo

- El problema del Agente Viajero es más general que HAMILTON CIRCUIT, de la siguiente manera:

TSP es NP-completo

- El problema del Agente Viajero es más general que HAMILTON CIRCUIT, de la siguiente manera:
- Ponemos las aristas de G con costo 1 y las de $\bar{G}$ con costo 2.

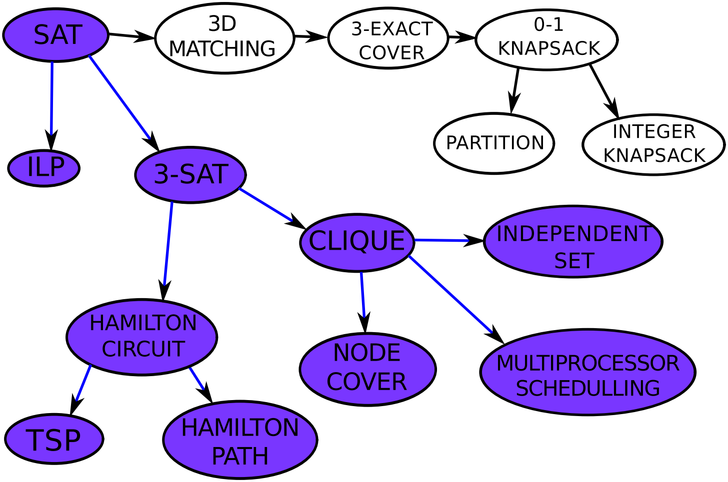
TSP es NP-completo

- El problema del Agente Viajero es más general que HAMILTON CIRCUIT, de la siguiente manera:
- Ponemos las aristas de G con costo 1 y las de $\bar{G}$ con costo 2.
- Nos preguntamos que si existe un circuito que pase por todos los vértices con costo menor o igual a $|V(G)|$ (es la versión de decisión de TSP).

TSP es NP-completo

- El problema del Agente Viajero es más general que HAMILTON CIRCUIT, de la siguiente manera:
- Ponemos las aristas de G con costo 1 y las de $\bar{G}$ con costo 2.
- Nos preguntamos que si existe un circuito que pase por todos los vértices con costo menor o igual a $|V(G)|$ (es la versión de decisión de TSP).
- Es equivalente a que G tenga un circuito Hamiltoniano.

Diagrama


SAT
3D MATCHING
3-EXACT COVER
0-1 KNAPSACK
PARTITION
INTEGER KNAPSACK
ILP
3-SAT
CLIQUE
INDEPENDENT SET
HAMILTON CIRCUIT
NODE COVER
MULTIPROCESSOR SCHEDULLING
TSP
HAMILTON PATH