Complejidad y problemas $\mathcal{NP}$

Programación de Estructuras de Datos y Algoritmos Fundamentales (TC1031)

M.C. Xavier Sánchez Díaz
mail@tec.mx

Outline

Órdenes más comunes

Recapitulación

- $\mathcal{O}(1)$ que es constante
- $\mathcal{O}(\log n)$ que es logarítmico
- $\mathcal{O}(n)$ que es *lineal*
- $\mathcal{O}(n \log n)$
- $\mathcal{O}(n^2)$ que es *cuadrático*
- $\mathcal{O}(n^2 logn)$
- $\mathcal{O}(n^m)$ que es *polinomial*
- $\mathcal{O}(m^n)$ que es *exponencial*
- $\mathcal{O}(n!)$ que es *factorial*

El cómputo

Introducción

Las **ciencias computacionales** se encargan de estudiar el *cómo* resolver los problemas (no las computadoras). A esto se le conoce como cómputo y se relaciona directamente con todo lo que tiene que ver con *cómo* generar cálculos.

- ¿Podemos calcularlo todo?
- ¿Qué información necesitamos para poder calcular la respuesta a alguna pregunta?

El cómputo
Introducción

Las **ciencias computacionales** se encargan de estudiar el *cómo* resolver los problemas (no las computadoras). A esto se le conoce como cómputo y se relaciona directamente con todo lo que tiene que ver con *cómo* generar cálculos.

- ¿Podemos calcularlo todo?
- ¿Qué información necesitamos para poder calcular la respuesta a alguna pregunta?

El cómputo

Introducción

Las **ciencias computacionales** se encargan de estudiar el *cómo* resolver los problemas (no las computadoras). A esto se le conoce como cómputo y se relaciona directamente con todo lo que tiene que ver con *cómo* generar cálculos.

- ¿Podemos calcularlo todo?
- ¿Qué información necesitamos para poder calcular la respuesta a alguna pregunta?

Breve historia I

Introducción

- En 1928, David Hilbert y Wilhelm Ackermann (GER) se cuestionan si es posible contestar a la pregunta:

Si te doy un enunciado lógico, ¿hay alguna manera consistente de asegurar si el enunciado es universalmente válido?

$$MagicBox(\Phi) = \begin{cases} 0 & \text{si } \Phi \text{ no es válida} \\ 1 & \text{si } \Phi \text{ es válida} \end{cases}$$

- En 1935, Alonzo Church (US) genera un concepto abstracto de cómputo usando funciones (λ-calculus) que demuestra que no existe manera de saberlo.

Breve historia II

Introducción

- En 1936, Alan Turing (UK) genera un concepto abstracto de cómputo usando máquinas de estados (*Turing Machines*) que demuestra que no existe manera de saberlo. Unos meses después, Turing demuestra que su sistema es equivalente al de Church.

- El método de Turing es más fácil de entender y explicar, y se acepta como estándar.

Ejemplo: Stanford's CS103 de Keith Schwarz

Determinismo vs No-determinismo

Introducción

La palabra **determinismo** hace referencia a que la solución a un problema se puede obtener después de un número **determinado** de pasos que depende del **estado** en el que se encuentra el problema. Algunas operaciones deterministas son:

- Sumar dos números
- Multiplicar dos matrices
- Ordenar una lista
- Encontrar el número más pequeño en un arreglo

La palabra **determinismo** hace referencia a que la solución a un problema se puede obtener después de un número **determinado** de pasos que depende del **estado** en el que se encuentra el problema. Algunas operaciones deterministas son:

- Sumar dos números
- Multiplicar dos matrices
- Ordenar una lista
- Encontrar el número más pequeño en un arreglo

Determinismo vs No-determinismo

Introducción

La palabra **determinismo** hace referencia a que la solución a un problema se puede obtener después de un número **determinado** de pasos que depende del **estado** en el que se encuentra el problema. Algunas operaciones deterministas son:

- Sumar dos números
- Multiplicar dos matrices
- Ordenar una lista
- Encontrar el número más pequeño en un arreglo

Determinismo vs No-determinismo

Introducción

La palabra **determinismo** hace referencia a que la solución a un problema se puede obtener después de un número **determinado** de pasos que depende del **estado** en el que se encuentra el problema. Algunas operaciones deterministas son:

- Sumar dos números
- Multiplicar dos matrices
- Ordenar una lista
- Encontrar el número más pequeño en un arreglo

Determinismo vs No-determinismo

Introducción

La palabra **determinismo** hace referencia a que la solución a un problema se puede obtener después de un número **determinado** de pasos que depende del **estado** en el que se encuentra el problema. Algunas operaciones deterministas son:

- Sumar dos números
- Multiplicar dos matrices
- Ordenar una lista
- Encontrar el número más pequeño en un arreglo

Problemas $\mathcal{P}$

Si en su lugar, nos limitamos tener una máquina determinista (es decir que sólo puede estar en un estado), entonces sólo algunos de los problemas $\mathcal{NP}$ se pueden resolver en un tiempo *decente* ($\mathcal{O}(n^m)$)

A lo mucho, en tiempo polinomial.

- Son problemas de **decisión**: ¿Puedes? *Yes or no*.
- Se verifican **fácilmente**: si ya tengo una solución para comparar, puedo ver si voy por buen camino o si el problema está bien hecho.
- Se resuelven **siempre** con el mismo número de pasos (en el peor de los casos)

Si en su lugar, nos limitamos tener una máquina determinista (es decir que sólo puede estar en un estado), entonces sólo algunos de los problemas $\mathcal{NP}$ se pueden resolver en un tiempo *decente* ($\mathcal{O}(n^m)$)

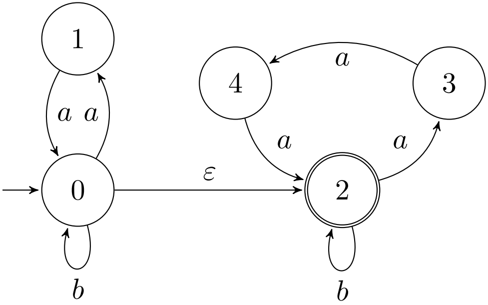
A lo mucho, en tiempo polinomial.

- Son problemas de **decisión**: ¿Puedes? *Yes or no*.
- Se verifican **fácilmente**: si ya tengo una solución para comparar, puedo ver si voy por buen camino o si el problema está bien hecho.
- Se resuelven **siempre** con el mismo número de pasos (en el peor de los casos)

Determinismo vs No-determinismo

Introducción

El **no-determinismo** es lo contrario. Considera el siguiente *autómata no determinista* donde ε es la palabra vacía (o sea ”):



La máquina está en **dos estados distintos a la vez** y las acciones siguientes dependen del estado en el que estás... que es en ambos...

Si tuviéramos una máquina de ésas no deterministas podríamos **decidir o verificar** si algunos problemas se pueden resolver en un tiempo *decente* ($\mathcal{O}(n^m)$):

A lo mucho, en tiempo en polinomial.

- Son también problemas de **decisión**
- Son también fácilmente **verificables**

Se llaman $\mathcal{NP}$ por *Non-deterministic Polynomial*, pueden ser resueltos y verificados por una máquina de Turing no determinista en un tiempo Polinomial ($\mathcal{O}(n^m)$).

Problemas $\mathcal{NP}$

Clases de complejidad

Si tuviéramos una máquina de ésas no deterministas podríamos **decidir o verificar** si algunos problemas se pueden resolver en un tiempo *decente* ($\mathcal{O}(n^m)$):

A lo mucho, en tiempo en polinomial.

- Son también problemas de **decisión**
- Son también fácilmente **verificables**

Se llaman $\mathcal{NP}$ por *Non-deterministic Polynomial*, pueden ser resueltos y verificados por una máquina de Turing no determinista en un tiempo Polinomial ($\mathcal{O}(n^m)$).

Si tuviéramos una máquina de ésas no deterministas podríamos **decidir o verificar** si algunos problemas se pueden resolver en un tiempo *decente* ($\mathcal{O}(n^m)$):

A lo mucho, en tiempo en polinomial.

- Son también problemas de **decisión**
- Son también fácilmente **verificables**

Se llaman $\mathcal{NP}$ por *Non-deterministic Polynomial*, pueden ser resueltos y verificados por una máquina de Turing no determinista en un tiempo Polinomial ($\mathcal{O}(n^m)$).

. . .

¿Qué?

Otra manera de ver a $\mathcal{NP}$

Clases de complejidad

Si ambos son de decisión y fácilmente verificables, ¿cómo sé si algún problema es $\mathcal{P}$ o es $\mathcal{NP}$?.

- Los problemas en $\mathcal{P}$ los **resolvemos** con una máquina determinista y los **verificamos** con la misma máquina en tiempo **polinomial**.
- Los problemas en $\mathcal{NP}$ los "'**resolvemos**"' con una máquina no determinista pero los **verificamos** con una máquina determinista en tiempo polinomial.

Otra manera de ver a $\mathcal{NP}$

Clases de complejidad

Si ambos son de decisión y fácilmente verificables, ¿cómo sé si algún problema es $\mathcal{P}$ o es $\mathcal{NP}$?.

- Los problemas en $\mathcal{P}$ los **resolvemos** con una máquina determinista y los **verificamos** con la misma máquina en tiempo **polinomial**.
- Los problemas en $\mathcal{NP}$ los "'**resolvemos**"' con una máquina no determinista pero los **verificamos** con una máquina determinista en tiempo polinomial.

Otra manera de ver a $\mathcal{NP}$

Clases de complejidad

Si ambos son de decisión y fácilmente verificables, ¿cómo sé si algún problema es $\mathcal{P}$ o es $\mathcal{NP}$?.

- Los problemas en $\mathcal{P}$ los **resolvemos** con una máquina determinista y los **verificamos** con la misma máquina en tiempo **polinomial**.
- Los problemas en $\mathcal{NP}$ los “‘**resolvemos**”’ con una máquina no determinista pero los **verificamos** con una máquina determinista en tiempo polinomial.

Para cada uno de estos problemas pregúntate lo siguiente:

¿Es un problema de decisión?

¿Puedo verificarlo rápidamente?

¿Qué complejidad me toma?

¿Si uso otro tipo de máquina reduzco el tiempo de solución?

Armar un rompecabezas. Si comparo cada una de las n piezas contra todas las demás $(n-1)$ entonces sé qué pieza va en cada lugar $\rightarrow \mathcal{O}(n^2)$ ✓

Sentar invitados en una mesa sin conflictos. Por cada silla n checo que no tenga conflicto con las $n-1$ sillas restantes. Luego, por la siguiente silla, checo con las $n-2$ restantes. Y así, o sea $n \times (n-1) \times (n-2) \cdots = \mathcal{O}(n!)$ ✗

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor. No es de decisión siquiera ✗

Ordenar una lista de números. El peor de los casos es que vengan en el orden contrario así que a lo mucho comparo todos contra todos los demás $\rightarrow \mathcal{O}(n^2)$ ✓

Saber si un programa finalizará dada su instrucción inicial. Es de decisión pero es imposible de resolver ✗

Ejemplos

Clases de complejidad

Armar un rompecabezas. Si comparo cada una de las n piezas contra todas las demás $(n-1)$ entonces sé qué pieza va en cada lugar $\rightarrow \mathcal{O}(n^2)$ ✓

Sentar invitados en una mesa sin conflictos. Por cada silla n checo que no tenga conflicto con las $n-1$ sillas restantes. Luego, por la siguiente silla, checo con las $n-2$ restantes. Y así, o sea $n \times (n-1) \times (n-2) \cdots = \mathcal{O}(n!)$ ✗

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor. No es de decisión siquiera ✗

Ordenar una lista de números. El peor de los casos es que vengan en el orden contrario así que a lo mucho comparo todos contra todos los demás $\rightarrow \mathcal{O}(n^2)$ ✓

Saber si un programa finalizará dada su instrucción inicial. Es de decisión pero es imposible de resolver ✗

Armar un rompecabezas. Si comparo cada una de las n piezas contra todas las demás $(n-1)$ entonces sé qué pieza va en cada lugar $\to \mathcal{O}(n^2)$ ✓

Sentar invitados en una mesa sin conflictos. Por cada silla n checo que no tenga conflicto con las $n-1$ sillas restantes. Luego, por la siguiente silla, checo con las $n-2$ restantes. Y así, o sea $n \times (n-1) \times (n-2) \cdots = \mathcal{O}(n!)$ ✗

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor. No es de decisión siquiera ✗

Ordenar una lista de números. El peor de los casos es que vengan en el orden contrario así que a lo mucho comparo todos contra todos los demás $\to \mathcal{O}(n^2)$ ✓

Saber si un programa finalizará dada su instrucción inicial. Es de decisión pero es imposible de resolver ✗

Ejemplos

Clases de complejidad

Armar un rompecabezas. Si comparo cada una de las n piezas contra todas las demás $(n-1)$ entonces sé qué pieza va en cada lugar $\rightarrow \mathcal{O}(n^2)$ ✓

Sentar invitados en una mesa sin conflictos. Por cada silla n checo que no tenga conflicto con las $n-1$ sillas restantes. Luego, por la siguiente silla, checo con las $n-2$ restantes. Y así, o sea $n \times (n-1) \times (n-2) \cdots = \mathcal{O}(n!)$ ✗

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor. No es de decisión siquiera ✗

Ordenar una lista de números. El peor de los casos es que vengan en el orden contrario así que a lo mucho comparo todos contra todos los demás $\rightarrow \mathcal{O}(n^2)$ ✓

Saber si un programa finalizará dada su instrucción inicial. Es de decisión pero es imposible de resolver ✗

Ejemplos

Clases de complejidad

Armar un rompecabezas. Si comparo cada una de las n piezas contra todas las demás $(n-1)$ entonces sé qué pieza va en cada lugar $\rightarrow \mathcal{O}(n^2)$ ✓

Sentar invitados en una mesa sin conflictos. Por cada silla n checo que no tenga conflicto con las $n-1$ sillas restantes. Luego, por la siguiente silla, checo con las $n-2$ restantes. Y así, o sea $n \times (n-1) \times (n-2) \cdots = \mathcal{O}(n!)$ ✗

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor. No es de decisión siquiera ✗

Ordenar una lista de números. El peor de los casos es que vengan en el orden contrario así que a lo mucho comparo todos contra todos los demás $\rightarrow \mathcal{O}(n^2)$ ✓

Saber si un programa finalizará dada su instrucción inicial. Es de decisión pero es imposible de resolver ✗

¿Puedes visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible?. Empiezo en uno, y me voy a alguno, y luego a otro y luego a otro. . . y anoto su distancia. Después empiezo en otro, y reviso las otras $n-1$ posibilidades, otras $n-2$ veces. . . $= \mathcal{O}(n!)$ ✗

Asignar salones de clase a profesores. Pruebo para uno de los n profesores alguno de los m salones disponibles y veo si no tiene problema a esa hora en ese salón. Luego reviso que no haya conflictos con los demás $n-1$ profesores en sus $m-1$ salones. Y asigno otro. . . $= \mathcal{O}(n!)$ ✗

¿Notas algún patrón?

Ejemplos

Clases de complejidad

¿Puedes visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible?. Empiezo en uno, y me voy a alguno, y luego a otro y luego a otro... y anoto su distancia. Después empiezo en otro, y reviso las otras $n-1$ posibilidades, otras $n-2$ veces... $= \mathcal{O}(n!)$ ✗

Asignar salones de clase a profesores. Pruebo para uno de los n profesores alguno de los m salones disponibles y veo si no tiene problema a esa hora en ese salón. Luego reviso que no haya conflictos con los demás $n-1$ profesores en sus $m-1$ salones. Y asigno otro... $= \mathcal{O}(n!)$ ✗

¿Notas algún patrón?

¿Puedes visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible?. Empiezo en uno, y me voy a alguno, y luego a otro y luego a otro. . . y anoto su distancia. Después empiezo en otro, y reviso las otras $n-1$ posibilidades, otras $n-2$ veces. . . $= \mathcal{O}(n!)$ ✗

Asignar salones de clase a profesores. Pruebo para uno de los n profesores alguno de los m salones disponibles y veo si no tiene problema a esa hora en ese salón. Luego reviso que no haya conflictos con los demás $n-1$ profesores en sus $m-1$ salones. Y asigno otro. . . $= \mathcal{O}(n!)$ ✗

¿Notas algún patrón?

Algunos de los que no se pueden resolver en tiempo polinomial podrían resolverse fácilmente si de manera no determinista **generamos** una posible solución (*good guess*) y la **verificamos** de manera determinista.

Esos son los $\mathcal{NP}$.

¿Puedes usar el mismo método para los $\mathcal{P}$?
Por supuesto, porque $\mathcal{P} \subseteq \mathcal{NP}$

El no-determinismo de $\mathcal{NP}$

Clases de Complejidad

Algunos de los que no se pueden resolver en tiempo polinomial podrían resolverse fácilmente si de manera no determinista **generamos** una posible solución (*good guess*) y la **verificamos** de manera determinista.

Esos son los $\mathcal{NP}$.

¿Puedes usar el mismo método para los $\mathcal{P}$?
Por supuesto, porque $\mathcal{P} \subseteq \mathcal{NP}$

Resolver vs Verificar

Reducciones

Ya vimos que **resolver** es más *difícil* que **verificar**:

Resolver un sudoku. Tendrías que ir de cuadro en cuadro, probar con un número, y luego checar que dé la suma; y cambiar de uno por uno, para evitar conflictos con los $n-1$ restantes, que no tenga conflictos con los $n-2$ restantes… $= \mathcal{O}(n!)$

Verificar la solución de un sudoku. Recibo una solución propuesta y me voy de una por una en las n casillas para revisar que todo cuadre. Si cuadra, perfecto. Si no, reporto que está mal. Esto toma $\mathcal{O}(n)$ operaciones.

Ya vimos que **resolver** es más *difícil* que **verificar**:

Resolver un sudoku. Tendrías que ir de cuadro en cuadro, probar con un número, y luego checar que dé la suma; y cambiar de uno por uno, para evitar conflictos con los $n-1$ restantes, que no tenga conflictos con los $n-2$ restantes… $= \mathcal{O}(n!)$

Verificar la solución de un sudoku. Recibo una solución propuesta y me voy de una por una en las n casillas para revisar que todo cuadre. Si cuadra, perfecto. Si no, reporto que está mal. Esto toma $\mathcal{O}(n)$ operaciones.

Una reducción absurda

Reducción

Una reducción es una transformación de un problema $easy$ a uno $harder$. Por ejemplo:

- $easy =$ "No puedo levantar este auto porque pesa demasiado"
- $harder =$ "Me pregunto si podré levantar este barco de carga"

Por medio de una reducción, podemos convertir el problema $easy$ a un problema $harder$, por ejemplo pensando que metemos el auto *dentro* del barco de carga. Si podemos levantar el barco de carga, entonces podemos levantar el auto.

De este modo, resolver el problema $harder$ ayudó a resolver el problema $easy$.

Una reducción absurda

Reducción

Una reducción es una transformación de un problema $easy$ a uno $harder$. Por ejemplo:

- $easy =$ "No puedo levantar este auto porque pesa demasiado"
- $harder =$ "Me pregunto si podré levantar este barco de carga"

Por medio de una reducción, podemos convertir el problema $easy$ a un problema $harder$, por ejemplo pensando que metemos el auto *dentro* del barco de carga. Si podemos levantar el barco de carga, entonces podemos levantar el auto.

De este modo, resolver el problema $harder$ ayudó a resolver el problema $easy$.

Una reducción absurda

Reducción

Una reducción es una transformación de un problema $easy$ a uno $harder$. Por ejemplo:

- $easy =$ "No puedo levantar este auto porque pesa demasiado"
- $harder =$ "Me pregunto si podré levantar este barco de carga"

Por medio de una reducción, podemos convertir el problema $easy$ a un problema $harder$, por ejemplo pensando que metemos el auto *dentro* del barco de carga. Si podemos levantar el barco de carga, entonces podemos levantar el auto.

De este modo, resolver el problema $harder$ ayudó a resolver el problema $easy$.

Una reducción absurda

Reducción

Una reducción es una transformación de un problema $easy$ a uno $harder$. Por ejemplo:

- $easy =$ "No puedo levantar este auto porque pesa demasiado"
- $harder =$ "Me pregunto si podré levantar este barco de carga"

Por medio de una reducción, podemos convertir el problema $easy$ a un problema $harder$, por ejemplo pensando que metemos el auto *dentro* del barco de carga. Si podemos levantar el barco de carga, entonces podemos levantar el auto.

De este modo, resolver el problema $harder$ ayudó a resolver el problema $easy$.

Reducción en tiempo polinomial

Reducción

Haciendo una serie de *transformaciones* a cualquiera de *nuestros* problemas $\mathcal{NP}$ podemos hacer la siguiente *máquina* (algoritmo):

1. Probemos todas las posibles soluciones para nuestro problema.
2. Si una de ellas termina, entonces detente. Si no, entra en un bucle infinito.

Claramente, si el programa de 2 líneas se detiene significa que nuestro problema $\mathcal{NP}$ tenía una solución óptima.

Reducción en tiempo polinomial

Reducción

Haciendo una serie de *transformaciones* a cualquiera de *nuestros* problemas $\mathcal{NP}$ podemos hacer la siguiente *máquina* (algoritmo):

1. Probemos todas las posibles soluciones para nuestro problema.
2. Si una de ellas termina, entonces detente. Si no, entra en un bucle infinito.

Claramente, si el programa de 2 líneas se detiene significa que nuestro problema $\mathcal{NP}$ tenía una solución óptima.

Reducción en tiempo polinomial

Reducción

Haciendo una serie de *transformaciones* a cualquiera de *nuestros* problemas $\mathcal{NP}$ podemos hacer la siguiente *máquina* (algoritmo):

1. Probemos todas las posibles soluciones para nuestro problema.
2. Si una de ellas termina, entonces detente. Si no, entra en un bucle infinito.

Claramente, si el programa de 2 líneas se detiene significa que nuestro problema $\mathcal{NP}$ tenía una solución óptima.

Haciendo una serie de *transformaciones* a cualquiera de *nuestros* problemas $\mathcal{NP}$ podemos hacer la siguiente *máquina* (algoritmo):

1. Probemos todas las posibles soluciones para nuestro problema.
2. Si una de ellas termina, entonces detente. Si no, entra en un bucle infinito.

Claramente, si el programa de 2 líneas se detiene significa que nuestro problema $\mathcal{NP}$ tenía una solución óptima.

Esta reducción de nuestro problema $\mathcal{NP}$ a este otro problema de *detener la máquina* nos dice entonces que **decidir si la máquina se detendrá** es más difícil que resolver el problema NP (o sea, es el barco de carga de nuestro auto).

También podemos asegurar que resolver el problema $\mathcal{NP}$ (cargar nuestro auto) es *al menos* tan difícil como resolver el problema de **decidir si la máquina se detendrá**.

Esta reducción de nuestro problema $\mathcal{NP}$ a este otro problema de *detener la máquina* nos dice entonces que **decidir si la máquina se detendrá** es más difícil que resolver el problema NP (o sea, es el barco de carga de nuestro auto).

También podemos asegurar que resolver el problema $\mathcal{NP}$ (cargar nuestro auto) es *al menos* tan difícil como resolver el problema de **decidir si la máquina se detendrá**.

El problema de decidir si la máquina se va a detener o no dependiendo del programa (algoritmo) que recibe pertenece a la clase $\mathcal{NP}$-*hard* o $\mathcal{NP}$-difícil.

$\mathcal{NP}$-*hardness*

Un problema X es $\mathcal{NP}$-*hard* si todo problema $L \in \mathcal{NP}$ se puede reducir a X en tiempo polinomial.

Esto nos da una cadenita de complejidad interesante...

$\mathcal{NP}$-hard

Clases de complejidad

El problema de decidir si la máquina se va a detener o no dependiendo del programa (algoritmo) que recibe pertenece a la clase $\mathcal{NP}$-*hard* o $\mathcal{NP}$-difícil.

$\mathcal{NP}$-hardness

Un problema X es $\mathcal{NP}$-*hard* si todo problema $L \in \mathcal{NP}$ se puede reducir a X en tiempo polinomial.

Esto nos da una cadenita de complejidad interesante. . .

Completez de NP

Clases de complejidad

Si un problema L puede reducirse a otro que está en P, entonces
$L \in \mathcal{P}$ que es la clase de los problemas eficientemente resolubles en tiempo polinomial.

Si un problema L puede reducirse a otro que está en $\mathcal{NP}$, entonces
$L \in \mathcal{NP}$ que es la clase de los problemas eficientemente verificables en tiempo polinomial.

Si un problema L puede reducirse a otro que está en $\mathcal{NP}$-*hard*, entonces $L \in \mathcal{NP}$-*hard* que es la clase de los problemas difíciles de resolver para **cualquier** *computadora* (incluso aquellos que no puede resolver o son *indecidibles*).

$\mathcal{NP}$-*completeness*

Un problema L que puede reducirse a un $\mathcal{NP}$, y que también puede reducirse a un $\mathcal{NP}$-*hard* es un problema $\mathcal{NP}$-*Complete* o $\mathcal{NP}$-*Completo*

$\mathcal{NP}$-Complete

Los problemas $\mathcal{NP}$-*Complete*, como pueden ser reducidos a $\mathcal{NP}$-*hard*, también pueden servir para otras reducciones:

- Todos los problemas $\mathcal{NP}$-*Complete* son **reducibles entre sí**—encontrar una manera eficiente de resolver óptimamente uno de ellos resuelve todos los demás.
- Todos los algoritmos para "‘resolver"' los problemas $\mathcal{NP}$-*Complete* corren en tiempo exponencial (o más).
- Todos los algoritmos para resolver los problemas $\mathcal{NP}$-*Complete* no son factibles para una cantidad razonable de tamaño de problema n.

Volviendo a nuestros problemas propuestos

Clases de complejidad

Armar un rompecabezas es un problema $\mathcal{P}$ porque es fácilmente resoluble fácilmente verificable

Sentar invitados en una mesa sin conflictos es un problema $\mathcal{NP}$ porque es fácilmente verificable, pero no es fácilmente resoluble. De hecho, es una instancia de la asignación de salones y es reducible al de saber si la máquina se detuvo por lo que es $\mathcal{NP}$-*complete*

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor no es de decisión así que no está en $\mathcal{NP}$. Sin embargo, la versión de decisión (puedo o no empacar esto...) es $\mathcal{NP}$-*complete*, por lo que se considera $\mathcal{NP}$-*hard*

Ordenar una lista de números es fácilmente resoluble y verificable...es $\mathcal{P}$

Saber si un programa finalizará dada su instrucción inicial es $\mathcal{NP}$-*hard* y es indecidible (o sea que no es posible resolverlo)

Volviendo a nuestros problemas propuestos

Clases de complejidad

Armar un rompecabezas es un problema $\mathcal{P}$ porque es fácilmente resoluble fácilmente verificable

Sentar invitados en una mesa sin conflictos es un problema $\mathcal{NP}$ porque es fácilmente verificable, pero no es fácilmente resoluble. De hecho, es una instancia de la asignación de salones y es reducible al de saber si la máquina se detuvo por lo que es $\mathcal{NP}$-*complete*

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor no es de decisión así que no está en $\mathcal{NP}$. Sin embargo, la versión de decisión (puedo o no empacar esto. . .) es $\mathcal{NP}$-*complete*, por lo que se considera $\mathcal{NP}$-*hard*

Ordenar una lista de números es fácilmente resoluble y verificable. . . es $\mathcal{P}$

Saber si un programa finalizará dada su instrucción inicial es $\mathcal{NP}$-*hard* y es indecidible (o sea que no es posible resolverlo)

Armar un rompecabezas es un problema $\mathcal{P}$ porque es fácilmente resoluble fácilmente verificable

Sentar invitados en una mesa sin conflictos es un problema $\mathcal{NP}$ porque es fácilmente verificable, pero no es fácilmente resoluble. De hecho, es una instancia de la asignación de salones y es reducible al de saber si la máquina se detuvo por lo que es $\mathcal{NP}$-*complete*

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor no es de decisión así que no está en $\mathcal{NP}$. Sin embargo, la versión de decisión (puedo o no empacar esto...) es $\mathcal{NP}$-*complete*, por lo que se considera $\mathcal{NP}$-*hard*

Ordenar una lista de números es fácilmente resoluble y verificable...es $\mathcal{P}$

Saber si un programa finalizará dada su instrucción inicial es $\mathcal{NP}$-*hard* y es indecidible (o sea que no es posible resolverlo)

Volviendo a nuestros problemas propuestos

Clases de complejidad

Armar un rompecabezas es un problema $\mathcal{P}$ porque es fácilmente resoluble fácilmente verificable

Sentar invitados en una mesa sin conflictos es un problema $\mathcal{NP}$ porque es fácilmente verificable, pero no es fácilmente resoluble. De hecho, es una instancia de la asignación de salones y es reducible al de saber si la máquina se detuvo por lo que es $\mathcal{NP}$-*complete*

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor no es de decisión así que no está en $\mathcal{NP}$. Sin embargo, la versión de decisión (puedo o no empacar esto. . .) es $\mathcal{NP}$-*complete*, por lo que se considera $\mathcal{NP}$-*hard*

Ordenar una lista de números es fácilmente resoluble y verificable. . . es $\mathcal{P}$

Saber si un programa finalizará dada su instrucción inicial es $\mathcal{NP}$-*hard* y es indecidible (o sea que no es posible resolverlo)

Volviendo a nuestros problemas propuestos

Clases de complejidad

Armar un rompecabezas es un problema $\mathcal{P}$ porque es fácilmente resoluble fácilmente verificable

Sentar invitados en una mesa sin conflictos es un problema $\mathcal{NP}$ porque es fácilmente verificable, pero no es fácilmente resoluble. De hecho, es una instancia de la asignación de salones y es reducible al de saber si la máquina se detuvo por lo que es $\mathcal{NP}$-*complete*

Empacar la mayor cantidad de valores sin exceder la capacidad de una bolsa dados ciertos objetos de valor no es de decisión así que no está en $\mathcal{NP}$. Sin embargo, la versión de decisión (puedo o no empacar esto. . .) es $\mathcal{NP}$-*complete*, por lo que se considera $\mathcal{NP}$-*hard*

Ordenar una lista de números es fácilmente resoluble y verificable. . . es $\mathcal{P}$

Saber si un programa finalizará dada su instrucción inicial es $\mathcal{NP}$-*hard* y es indecidible (o sea que no es posible resolverlo)

Volviendo a nuestros problemas propuestos

Clases de complejidad

Visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible es un problema $\mathcal{NP}$-*complete* ya que es reducible a otros $\mathcal{NP}$-*complete*

Asignar salones de clase a profesores es también $\mathcal{NP}$-*complete* ya que se puede reducir a otro $\mathcal{NP}$-*complete* como la asignación de asientos sin conflictos o la selección de objetos para empacado

Visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible es un problema $\mathcal{NP}$-*complete* ya que es reducible a otros $\mathcal{NP}$-*complete*

Asignar salones de clase a profesores es también $\mathcal{NP}$-*complete* ya que se puede reducir a otro $\mathcal{NP}$-*complete* como la asignación de asientos sin conflictos o la selección de objetos para empacado

Visitar todos los Starbucks de Monterrey de tal modo que la distancia que recorras sea la menor posible es un problema $\mathcal{NP}$-*complete* ya que es reducible a otros $\mathcal{NP}$-*complete*

Asignar salones de clase a profesores es también $\mathcal{NP}$-*complete* ya que se puede reducir a otro $\mathcal{NP}$-*complete* como la asignación de asientos sin conflictos o la selección de objetos para empacado

Más ejemplos de NP-*complete*

Clases de complejidad

Hay muchísimos más $\mathcal{NP}$-*complete* en la vida diaria:

- ¿Si compro 5 torres de comunicaciones me garantiza que tengo cobertura suficiente para toda la ciudad?
- ¿Hay alguna secuencia de 100 genes que sea común entre individuos de un grupo de tiras de ADN?
- Dada una lista de precios y un presupuesto, ¿puedo comprar más de 15 objetos con ello?
- Dada la distribución demográfica de una ciudad, ¿puedo usar 330,000 metros de tuberías para brindar agua a toda la población?

Más ejemplos de NP-*complete*

Clases de complejidad

Hay muchísimos más $\mathcal{NP}$-*complete* en la vida diaria:

- ¿Si compro 5 torres de comunicaciones me garantiza que tengo cobertura suficiente para toda la ciudad?
- ¿Hay alguna secuencia de 100 genes que sea común entre individuos de un grupo de tiras de ADN?
- Dada una lista de precios y un presupuesto, ¿puedo comprar más de 15 objetos con ello?
- Dada la distribución demográfica de una ciudad, ¿puedo usar 330,000 metros de tuberías para brindar agua a toda la población?

Más ejemplos de NP-*complete*

Hay muchísimos más $\mathcal{NP}$-*complete* en la vida diaria:

- ¿Si compro 5 torres de comunicaciones me garantiza que tengo cobertura suficiente para toda la ciudad?
- ¿Hay alguna secuencia de 100 genes que sea común entre individuos de un grupo de tiras de ADN?
- Dada una lista de precios y un presupuesto, ¿puedo comprar más de 15 objetos con ello?
- Dada la distribución demográfica de una ciudad, ¿puedo usar 330,000 metros de tuberías para brindar agua a toda la población?

Más ejemplos de NP-*complete*

Clases de complejidad

Hay muchísimos más $\mathcal{NP}$-*complete* en la vida diaria:

- ¿Si compro 5 torres de comunicaciones me garantiza que tengo cobertura suficiente para toda la ciudad?
- ¿Hay alguna secuencia de 100 genes que sea común entre individuos de un grupo de tiras de ADN?
- Dada una lista de precios y un presupuesto, ¿puedo comprar más de 15 objetos con ello?
- Dada la distribución demográfica de una ciudad, ¿puedo usar 330,000 metros de tuberías para brindar agua a toda la población?

¿Existe alguna manera eficiente de resolver algún problema fácilmente verificable?
Es decir, alguno de los $\mathcal{NP}$-*complete* puede ser resuelto en tiempo polinomial?

- Es un problema abierto hasta el momento—no existe respuesta alguna a esta pregunta
- Se cree que no es el caso (y que por tanto $\mathcal{P} \neq \mathcal{NP}$) pero nadie ha logrado demostrarlo
- El Clay Institute of Mathematics ofrece $ 1,000,000.00 USD a quien pueda resolverlo, y es uno de los problemas del milenio (quizá el más importante por todo lo que implica)
- Ha habido más de 100 intentos de pruebas para demostrar que $\mathcal{P}$ y $\mathcal{NP}$ son clases distintas; todas han fallado en ser correctas