COMPLEJIDAD COMPUTACIONAL
Introducción y Problemas computacionales

Carlos Augusto Meneses E. (2018)

0. Introducción

- Objetivo

"Estimar un algoritmo en términos de los recursos físicos que va a utilizar, antes de ser codificado en un lenguaje específico y por lo tanto antes de considerar una máquina para ejecutarlo"

0. Introducción

- Análisis de Algoritmos

"Es una de las herramientas con las que cuenta el ingeniero, para hacer la evaluación de un diseño".

Permite conocer la calidad de un programa y compararlo con otros para resolver un problema (sin necesidad de desarrollarlos).

El análisis se basa en las características estructurales que respaldan al programa (incluyendo el uso de la memoria).

0. Introducción

- Análisis de Algoritmos

¿ Como saber cual solución es mejor para resolver un problema computacional ?

1- Desarrollar los programas y ejecutarlos para medir el tiempo que gastan en resolver el problema.	Pueden existir muchos algoritmos para resolver el problema.	- Hay que variar los datos de entrada y ver comportamiento para promediar. (Solución costosa).
2- Establecer una medida de calidad de los algoritmos sin tener que implementarlos	Asociar a cada algoritmo, una función matemática que mida su eficiencia.	No requiere la implementación de los algoritmos.

0. Introducción

- Tiempo de ejecución de un algoritmo

- Velocidad de procesamiento (hardware)
- Compilador y sistema operativo
- Volúmen de datos a procesar.
- Algoritmo (estructura)

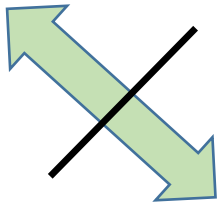
$T_A(n)$: Tiempo empleado por el algoritmo A para procesar una entrada de tamaño n y producir una solución al problema.

0. Introducción

- Complejidad (rendimiento)

Complejidad (rendimiento)

Algoritmo



- *Velocidad del Computador*
- *Facilidades de eficiencia del lenguaje de programación*
- *Estilo hábil del programador*

- *Caracterizar las operaciones básicas del algoritmo.*
- *Considerar los datos (tamaño y configuración)*

0. Introducción

- Problemas computacionales

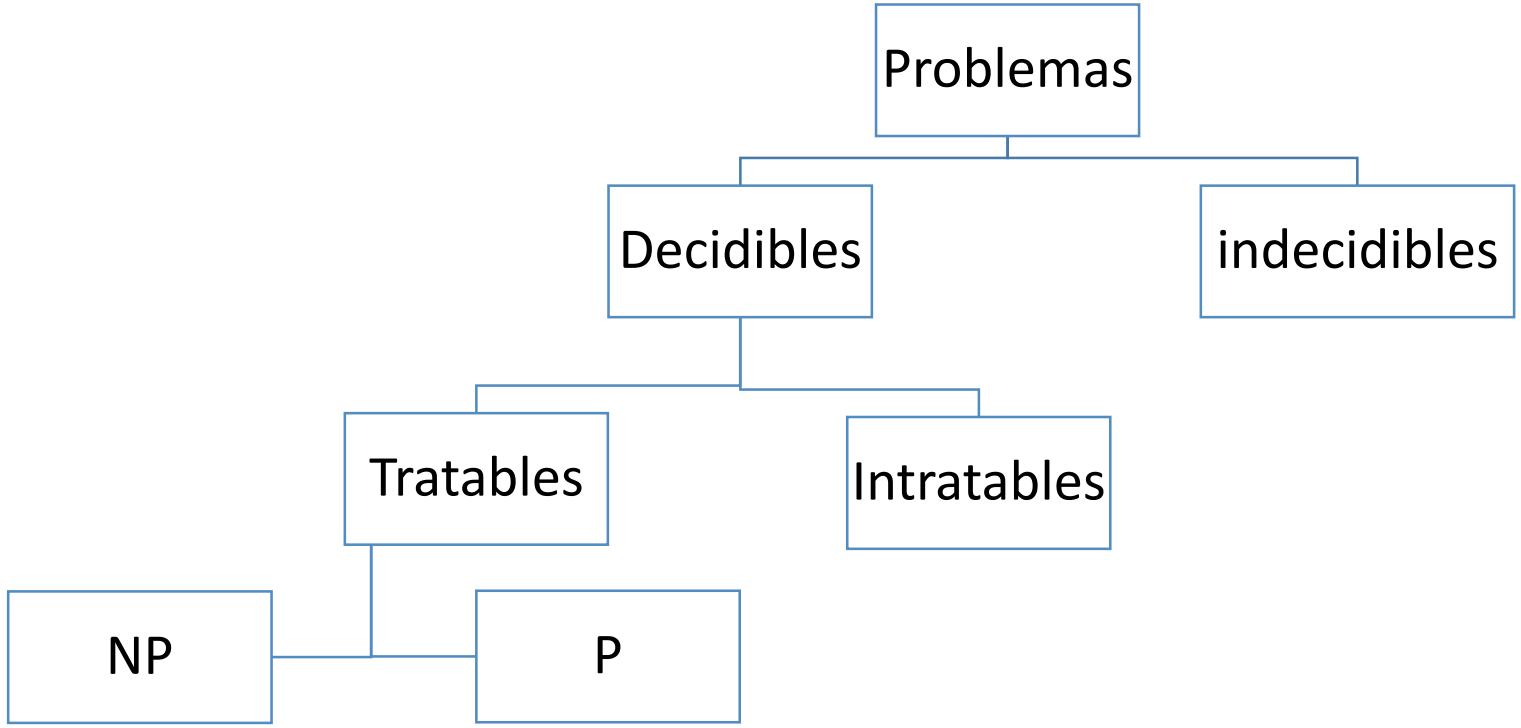
¿Qué es un problema computacional?:
Es una pregunta a ser respondida, tiene parámetros con valores no especificados:

- Descripción de sus parámetros (I/O)
- Descripción de las propiedades que debe cumplir de la solución.

Instanciación de un problema: Se especifican los valores particulares para todos los parámetros del problema.

0. Introducción

- Problemas computacionales



0. Introducción

- Problemas computacionales – De decisión

Problemas de Decisión.

- Respuesta es “si” o “no” (binario).

- Se puede ver como un lenguaje formal:

- Los elementos que pertenecen al Lenguaje, son las instancias con respuesta “si”. Los que no pertenecen → la respuesta es “no”.
- *Objetivo*: Decidir con la ayuda de un algoritmo “Si una entrada pertenece al Lenguaje Formal”
- Casi todo problema se puede convertir en un problema de decisión.

0. Introducción

- Problemas computacionales – Decibilidad

Problemas Indecidibles

- No existe un algoritmo para resolverlo.
- No tienen solución computacional

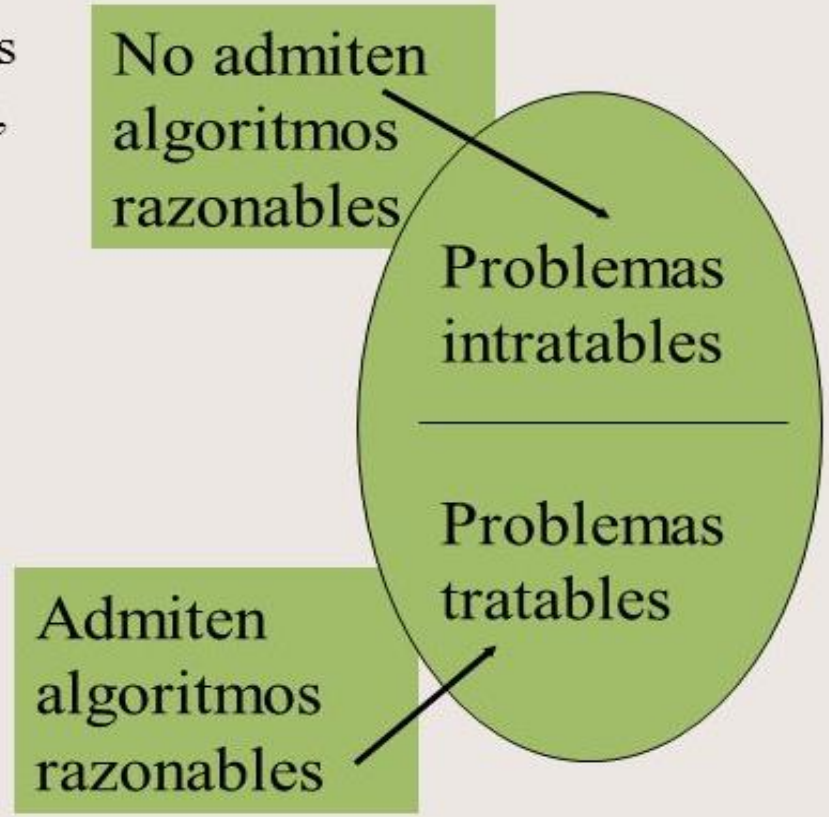
Problemas Decidibles.

- Existe un algoritmo para resolverlo.
- ***Problemas Intratables***. Son irresolubles por la cantidad de tiempo que requieren.
- **Problemas Tratables**. Se clasifican en:
  - Polinomiales (N). Existe un polinomio que acota su tiempo de ejecución.
    - ✓ De tiempo polinómico. Algoritmos "suficientemente eficiente" ***O(p(n))*** con función de complejidad temporal para alguna funión polinómica ***p***.
    - ✓ De tiempo exponencial. Algoritmos "muy ineficientes"
  - No Polinomiales (NP). No existe un polinomio que lo acote.
    ejemplo: "viajero de comercio"

0. Introducción

- Problemas computacionales

- Una **función polinomial** en N es aquella que esta limitada por N^k, para algún k.
- Un algoritmo cuyo tiempo de ejecución esta acotado por una función polinomial lo consideramos **razonable**. En otro caso **irrazonable**.
- En términos de problemas algorítmicos diremos que son problemas **tratables** o **intratables**



0. Introducción

- Problemas computacionales – Clase P

Clase P (*problemas con solución polinómica*):

- Contiene los problemas que se solucionan en tiempo polinómico por una máquina de Turing Determinista.
- P es invariante para todos los modelos de cómputo que son polinómicamente equivalentes a la máquina de *Turing Determinista*.
- P corresponde a la clase de problemas que son solubles en una computadora.

Clase NP (*problemas con solución No Polinómica*).

- Consta de los problemas “verificables” en tiempo polinómico.
- Problemas “No deterministas en tiempo polinómico”.
- Se utilizan Máquinas de Turing No Deterministas.

0. Introducción

- Problemas computacionales – P=NP?

Si P=NP:

- Cualquier problema “polinómicamente verificable” sería “polinómicamente decidible”.
- La mayoría de los investigadores cree que estas clases no son iguales. Se han realizado bastantes esfuerzos, sin éxito, para encontrar algoritmos de tiempo polinomial para varios problemas en NP.

Si P<>NP:

- Eso conllevaría a mostrar que no existe un algoritmo “eficiente” para reemplazar a la búsqueda por fuerza bruta.
- Si se demuestra que un problema NP es intratable, todos lo serían.

0. Introducción

- Un poco de Historia

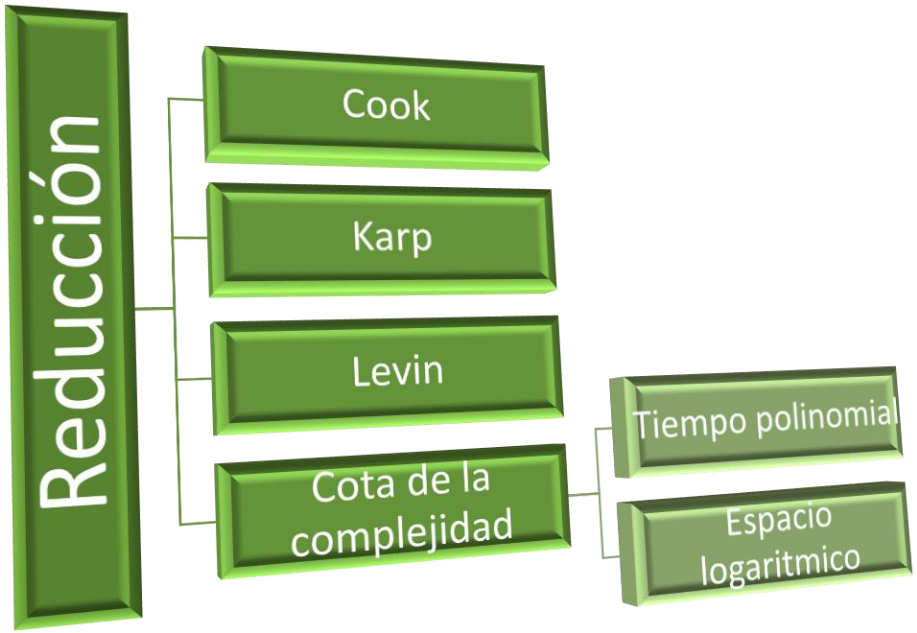
1936	• Máquinas de Turing (noción de computadora robusta y flexible)
50's	• Demostró ser un Modelo Teórico correcto de cómputo (pero fallaba al cuantificar el tiempo y la memoria)
60´s	• Hartmanis & Stearns. Idea: "Medir el tiempo y el espacio como una función de la longitud de la entrada" (nace la Teoría de la Complejidad Computacional)
1965	• Edmons. "Definió un buen algoritmo" como: "aquel cuyo tiempo de ejecución puede ser acotado por un polinomio" → Introduce la NP-Completitud y su pregunta fundamental P=NP?
70´s	• Stephen Cook & Leonid Levin. Probaron que existen problemas relevantes que son NP-Completos
80's	• Se produjo un auge de los modelos finitos. Acercándose a P=NP, introdujo técnicas combinatorias para mejorar el entendimiento de límites de estos modelos.
90´s	• Se empiezan a estudiar nuevos modelos de cómputo como las computadoras cuánticas. Limitante: desarrollar el hardware para este modelo.

0. Introducción

- Problemas computacionales – Reducción polinomial

Reducción polinomial:

- Una reducción es una transformación de un problema en otro problema.
- Un problema Q puede ser reducido a otro problema Q', si cualquier instancia del problema Q puede ser "fácilmente" expresada como una instancia del problema Q', y cuya solución proporcione una solución para la instancia de Q.
- La reducción polinomial → es en tiempo polinomial.



0. Introducción

0.1. Complejidad computacional – Clase de Problemas NP-Completos

Contiene a los problemas más difíciles en NP → son los que estén más lejos de estar en P.

Debido a que el problema P=NP no ha sido resuelto, → reducir un problema B, a otro problema A, indicaría que no se conoce solución en tiempo polinomial para A.

Por lo tanto, una solución en tiempo polinomial para A → Existe una solución polinomial para B. Y además: como todos los problemas NP pueden ser reducidos a este conjunto, → Encontrar un problema NP-completo que pueda ser resuelto en un tiempo polinomial, significaría que P=NP.

Quizás la razón de mayor peso por la cual los científicos de la computación creen que P es distinto de NP, es la existencia de la clase de problemas "NP-completos".

Desde el punto de vista práctico, el fenómeno de la NP-completitud puede prevenir la pérdida de tiempo cuando se busca un algoritmo de tiempo polinomial no existente para resolver un problema determinado

0. Introducción

0.1. Complejidad computacional - Factores

Factores que se tienen en cuenta para establecer complejidad:

- El tipo de problema computacional. Generalmente, de decisión.
- El Modelo de Cómputo. Más común: la Máquina de Turing Determinista (o No determinista, o Cuántica).
- El recurso que se acota. En términos de Tiempo, espacio, ...