Análisis y Diseño de Algoritmos

Análisis de Complejidad

Rensso Víctor Hugo Mora Colque[1]

[1]Universidad Católica San Pablo
Departamento de Ciencia de la Computación

Table of Contents

Table of Contents

Algoritmos

- Los algoritmos forman parte del día a día de las personas. Ejemplos de algoritmos:
  - instrucciones para el uso de medicamentos,
  - indicaciones para montar un aparato,
  - una receta culinaria
- Secuencia de acciones ejecutables para la obtención de una solución para un determinado problema
- Según Dijkstra, un algoritmo corresponde a una descripción de un patrón de comportamiento, expresado en términos de un conjunto finito de acciones.

El papel de los algoritmos en computación

Definición

Un **algoritmo** es un conjunto finito de instrucciones precisas para ejecutar una computación.

- Un algoritmo puede ser visto como una herramienta para resolver un problema computacional bien especificado.
- Un algoritmo puede recibir como entrada un conjunto y puede producir como salida otro conjunto.

Origen de la palabra algoritmo

Figure: Portada del libro Álgebra de Baldor[1]

Al-Juarismi (780-850), el gran matemático que le dio a Occidente los números y el sistema decimal, era además astrónomo, cortesano y favorito del Califa al-Mam'un. Era un emigrante de Persia oriental a Bagdad y producto de su época, la Edad de Oro del islam. Fue autor del tratado Kitab al-jabr w'al muquabala, libro que fue traducido al latín y usado extensamente en Europa. La traducción latina del nombre fue relacionada con la aritmética hindú y su relación con los pasos a seguir para resolver problemas determinados.

[1] Baldor, A. (2008). Álgebra de Baldor (2 ed.). México: Patría.

- Introduction to algorithms, Cormen[2]:
  Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input onto the output.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd. ed.). The MIT Press.

Observaciones

- Las reglas son precisas
- Conjunto de reglas finito
- Tiempo finito de ejecución
- Reglas ejecutadas **por computador**

Consecuencias

- Definir las herramientas adecuadas (lenguajes de programación)
- La mayor parte de los algoritmos utiliza métodos de organización (estructura de datos)
- Tiempo finito, no hay mucho interés en soluciones que duren siglos en dar una respuesta
- Distintos tipos de computadores
- Estudio de algoritmos para computadoras (en adelante solo serán algoritmos)

Para qué ADA?

- Los algoritmos son el objeto de la carrera
- La ciencia[3] esta la creación, estudio, análisis de los algoritmos (Paradigma científico - F. Bacon)
- Si los algoritmos resuelven problemas, entonces cuál es la mejor solución?
- Comparar, mejorar, crear

[3] Peter J. Denning. 2005. Is computer science science? Commun. ACM 48, 4 (April 2005), 27–31.

Walktrough del curso

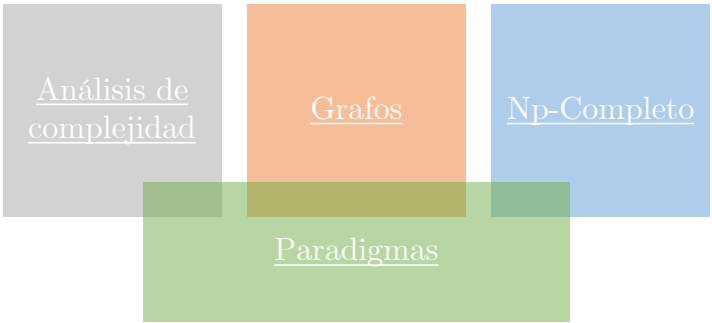


Figure: Módulos de la materia.

Table of Contents

$$\begin{array}{rccl} f: & A & \to & B \\ & a & \to & b = f(a) \end{array}$$

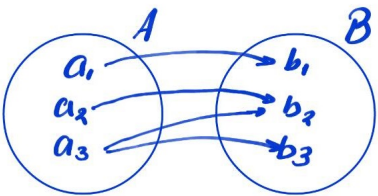


Figure: Noción de conjuntos relaciones y funciones.

Metodología

Conjunto de conceptos de traen cohesión a principios y técnicas mostrando cuando, como y porque usarlos en distintas situaciones.

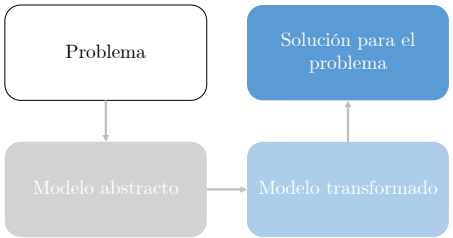


Figure: Proceso del modelado.

Línea de ferrocarril - problema

Suponga una línea de ferrocarril que une a seis ciudades.

Problema

Encontrar un subconjunto de las líneas del ferrocarril (representado por la matriz de adyacencia), que una todas las ciudades y que al mismo tiempo tenga una cantidad de línea mínimo.

	B	C	D	E	F
A	5	-	10	-	-
B		5	10	20	-
C			20	-	30
D				20	-
E					10

Table: Matriz de adyacencia.

Línea de ferrocarril - modelado

Encontrar una representación para el problema

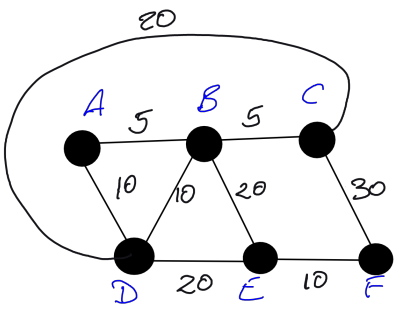


Figure: Malla del ferrocarril representado por un grafo.

Línea de ferrocarril - algoritmo

Algorithm 1: Ejemplo de un algoritmo **greedy**:

Data: Grafo que representa las líneas del tren.
Result: Subgrafo con la distribución mínima.

1. Seleccione arbitrariamente cualquier vértice y coloque en el conjunto de vértices conectados.
2. Escoja entre el conjunto de vértices no conectados el que este más próximo de un vértice ya conectado. Si existe más de uno escoja cualquiera.
3. Repita los pasos 1 y 2 hasta que todos los vértices sean conectados.

Muestre las cuatro soluciones.

Línea de ferrocarril - sumario

- Qué fue realizado?
  1. Obtención del modelo matemático para resolver el problema.
  2. Formulación del algoritmo en términos del modelo. **Osea, esa es la técnica de resolución de problemas en Ciencia de la Computación**.

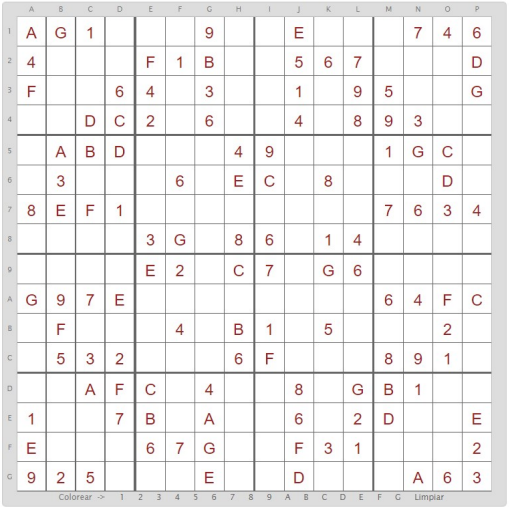


Figure: Supersudoku ejemplo de tablero.

Deep Blue vs Kasparov

El 10 de febrero de 1996, Deep Blue (IBM) ganó, por primera vez en la historia a un campeón en un ritmo normal de torneo. Sin embargo, supo sobreponerse a esta primera derrota. Así, tras un match que duró una semana, Kasparov ganó a Deep Blue 4 a 2.

Figure: Humano vs máquina.

Cuestiones sobre el modelamiento

- El objetivo es proyectar un algoritmo para resolver el problema
  Note que el sudoku y ajedrez tienen características distintas
- El proyecto tiene dos aspectos:
  1. El algoritmo propiamente dicho,
  2. La estructura de datos a ser usada en el algoritmo.
- En general el algoritmo define la estructura de datos a ser utilizada

Cuestiones sobre el modelamiento

- Un posible algoritmo para resolver el juego del sudoku puede estar pasado en el paradigma de **fuerza bruta**. Intentando escrutar todas las posibilidades hasta encontrar una solución.
- Utilizando la estraetgia propuesta, ¿Cuántas posibilidades para la configuración de la figura?

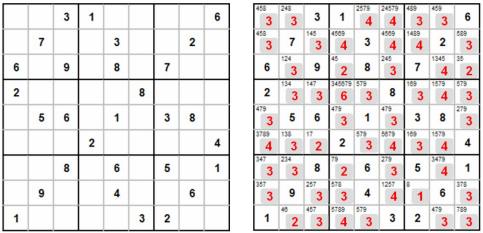


Figure: Los super-índices indican el número de opciones por esa posición.

Cuestiones sobre el modelamiento

- Un posible algoritmo para resolver el juego del sudoku puede estar pasado en el paradigma de **fuerza bruta**. Intentando escrutar todas las posibilidades hasta encontrar una solución.
- Utilizando la estraetgia propuesta, ¿Cuántas posibilidades para la configuración de la figura?

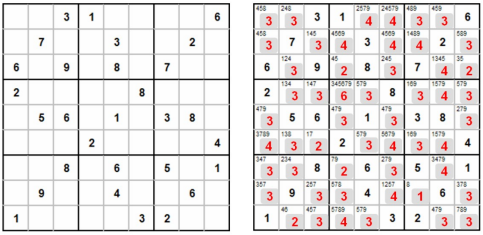


Figure: Los super-índices indican el número de opciones por esa posición.

Existen $1 \times 2^5 \times 3^{32} \times 4^{13} \times 6 = 2387598332983920265317580 8 \approx 23.8 \times 10^{24}$ posibilidades!

Estructura de datos

- Estructura de datos e algoritmos están fuertemente relacionados

  No se puede estudiar estructura de datos sin considerar los algoritmos asociados a ellos

  Escojer un algoritmo determinado también depende la estructura de datos
- Para resolver un problema es necesario escoger una abstracción de la realidad, en general mediante la definición de un conjunto de datos que representa la situación real

Selección de los datos

- La selección de los datos es determinada generalmente por las operaciones que se realizarán sobre ellos
- Considere la operación multiplicación:
  - Números pequeños son fáciles de almacenar en la mayoría de tipos de los lenguajes de programación
  - Números “grandes” necesitan de una estructura que permita su almacenamiento en memoria principal

- Programar básicamente es estructurar los datos e **instanciar** los algoritmos
- Los programas representan una clase especial de algoritmos capaces de ser **ejecutados por un computador**
- Un computador solo ejecuta lenguaje máquina (secuencias complejas y des-confortables)
- Utilizamos medios más cómodos para comandar la pc
- Los **lenguajes de programación son una herramienta** y no son el fin

Análisis de un algoritmo en particular

- Cual es el costo de el algoritmo?
- Características que deben ser investigadas
  - análisis del número de veces que cada parte del algoritmo debe ser ejecutada
  - estudio de la cantidad de memoria analizada

Análisis de un algoritmo en particular

- Cuál es el mejor algoritmo para ese problema?
- Qué es “mejor”?
- Distintas métricas de comparación

Costo de un algoritmo

- Depende del contexto del algoritmo
- Primera linea de comparación análisis de complejidad
- Plataforma real vs modelo matemático

Medida de costo usando modelo matemático

- Computador idealizado
- Se especifica el costo de cada linea de código
- Es usual ignorar el costo de algunas operaciones menos significativas
- Por ejemplo: en algoritmos de ordenación el número de comparaciones entre elementos del conjunto y se ignoran las operaciones aritméticas, de atribución, de manipulación de índices, etc.

Función de complejidad

- Para medir el costo de ejecución de un algoritmo es común definir una función de costo o **función de complejidad** f
- $f(n)$ es el costo que tomará el algoritmo dado una variable aleatoria n
- $f(n)$ puede medir tanto tiempo o espacio
- En el caso que mida tiempo no es necesariamente tiempo real o como una unidad, si no el número de veces que determinada operaciones que una determinada operación es ejecutada

Menor elemento

Considere el algoritmo para encontrar el menor elemento de un vector de enteros $A_{[1..n]}, n \geq 1$

Algorithm 2: Min(A)

Data: n es el tamaño del vector A

Result: min_value

```
1 min_value = A[1]
2 for i=2 to n do
3     if min_value > A[i] then
4         min_value = A[i]
5 return min_value
```

- Sea $f(n)$ el número de comparaciones entre los elementos de A
- Entonces la función de complejidad $f(n) = n - 1$

Observaciones

- La medida de costo de un algoritmo depende principalmente de variables aleatorias que influyen directamente en el conteo de las instrucciones o bloques esenciales del algoritmo, ejemplo: en el caso del algoritmos de ordenación la variable es el tamaño de la entrada
- En el caso de la función Min la función de costo no varia sea cual sea el tamaño de la entrada o distribución de los datos
- En general la función de costo esta relacionada con alguna característica de los datos de entrada

Mejor caso, peor caso y caso esperado

- Dependiendo del algoritmo, éste puede comportarse de diversa forma frente a diversas entradas, pero con alguna característica fija. Ejemplo: vectores del mismo tamaño.
- Funciones de complejidad distintas
- El caso medio supone una distribución de probabilidad de la cual se puede encontrar una media de ejecuciones

Mayor y menor elemento

- Considere el algoritmo para encontrar el menor y mayor elemento de un vector de enteros $A_{[1..n]}, n \geq 1$
- La función de complejidad $f(n)$ esta en relación al número de comparaciones entre los elementos del vector A
- El tamaño del vector n será mayor que 1

Mayor y menor elemento: algoritmo 1

Algorithm 3: MaxMin1(A)

Data: n es el tamaño del vector A

Result: max, min

```
1 min_value = A[1], max_value = A[1]
2 for i=2 to n do
3     if min > A[i] then
4         min = A[i]
5     if max < A[i] then
6         max = A[i]
```

Se cual sea la distribución de los datos para un vector de tamaño n la función $f(n) = 2(n-1)$

Mayor y menor elemento: algoritmo 2

Algorithm 4: MaxMin2(A)

Data: n es el tamaño del vector A

Result: max, min

```
1 min_value = A[1], max_value = A[1]
2 for i=2 to n do
3     if min > A[i] then
4         min = A[i]
5     else if max < A[i] then
6         max = A[i]
```

- Mejor caso $f(n) = n - 1$
- Peor caso $f(n) = 2(n - 1)$
- Caso medio, sucede cuando $A[i]$ es menor la mitad de veces $f(n) = n - 1 + \frac{n-1}{2} = \frac{3}{2}(n - 1)$

Mayor y menor elemento: algoritmo 3

Algorithm 5: MaxMin3(A)

Data: n es el tamaño del vector A

Result: max, min

```
1  if n mod2 > 0 then
2      A[n + 1] = A[n]
3      end = n
4  else
5      end = n − 1
6  if A[1] > A[2] then
7      max = A[1]
8      min = A[2]
9  else
10     max = A[2]
11     min = A[1]
```

Mayor y menor elemento: algoritmo 3, continuación

```
1  i = 3 while i < end do
2      if A[i] > A[i + 1] then
3          if A[i] > max then
4              max = A[i]
5          if A[i + 1] < min then
6              min = A[i + 1]
7      else
8          if A[i] < min then
9              min = A[i]
10         if A[i + 1] > max then
11             max = A[i + 1]
12     i = i + 2
```

Mayor y menor elemento

- Los elementos en este caso son comparados en pares
- Cuando n es impar el elemento en la posición n es duplicado
- Para esta implementación, $f(n) = \frac{n}{2} + \frac{n-2}{2} + \frac{n-2}{2} = \frac{3n}{2} - 2$ para cualquier caso!

Mayor y menor elemento

algoritmos	$f(n)$		
	Mejor caso	Peor Caso	Caso medio
MaxMin1	2(n-1)	2(n-1)	2(n-1)
MaxMin2	n-1	2(n-1)	$\frac{3}{2}(n-1)$
MaxMin3	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$	$\frac{3n}{2}-2$

Comportamiento de funciones

- El parámetro n brinda una medida de dificultad para resolver el problema
- Para valores "suficientemente pequeños", en general la mayoría de algoritmos suelen costar poco, inclusive los ineficientes
- La selección del algoritmo no suele representar un problema crítico para situaciones con conjuntos de datos pequeños
- Luego el análisis de algoritmos es realizado sobre valores muy grandes
- Estudio del comportamiento de las **funciones de costo** para grandes valores
- El comportamiento asintótico de $f(n)$ representa el límite de comportamiento cuando el valor de n crece

Dominación asintótica

- El análisis de un algoritmo generalmente cuenta con apenas algunas operaciones elementares
- La medida de costo o medida de complejidad relata el crecimiento asintótico de la operación considerada

Dominación asintótica

Definición

Una función $f(n)$ **domina asintóticamente** a otra función $g(n)$ si existen dos constantes positivas c y n_0 tal que, para $n \geq n_0$, tenemos $|g(n)| \leq c \times |f(n)|$

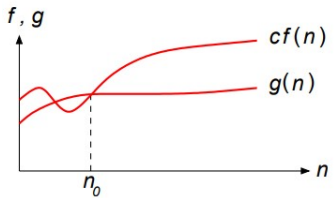


Figure: Función asintótica.

Dominación asintótica-ejemplo

- Sean $g(n) = (n+1)^2$ y $f(n) = n^2$
- Las funciones $f(n)$ y $g(n)$ dominan asintóticamente una a la otra ya que :

$$|(n+1)^2| \leq 4|n^2|$$

para $n \geq 1, y$

$$|n^2| \leq |(n+1)^2|$$

para $n \geq 0$

Dominación asintótica

- **Función de costo o función de complejidad**
  - $f(n)$ es la medida de costo necesaria para ejecutar un algoritmo, n es la variable aleatoria relacionada con el conjunto de datos de entrada
  - En el caso $f(n)$ represente el "tiempo", entonces f es llamada función de complejidad de tiempo del algoritmo, en el caso qeu se refiera a uso de memoria, la función f es llamada función de complejidad de espacio del algoritmo
- Observación: tiempo no se refiere a la unidad cronológica
  - Es importante resaltar que la complejidad de tiempo en la realidad no es una representación cronológica directamente. Se refiere al número de veces que una determinada operación considerada relevante es ejecutada

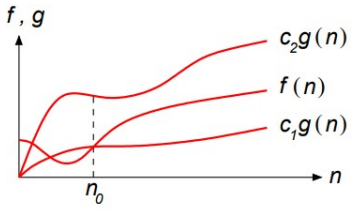
Costo asintótico de funciones

- Nos interesa comparar algoritmos para valores grandes de n
- El costo de una función $f(n)$ representa el límite de comportamiento de costo cuando n crece
- En general, **la variable n esta relacionada directamente con el tamaño de los datos de entrada**
- Para valores pequeños de n, aun algoritmos ineficientes no tienen un costo elevado para ser ejecutado

Notación asintótica

- Notación Θ
- Notación O (big O, o grande O)
- Notación Ω

Notación Θ



$$f(n) = \Theta(g(n))$$

Notación Θ

- La notación Θ limita la función por valores constantes
- $f(n) = \Theta(g(n))$, si existen constantes positivas c_1, c_2 y n_0 tal que, para $n \geq n_0$, el valor de $f(n)$ esta siempre entre $c_1 g(n)$ y $c_2 g(n)$
- Se puede decir que $g(n)$ es un límite asintótico firme (*asymptotically tight bound*) para $f(n)$.

Definición

$f(n) = \Theta(g(n)), \exists\ c_1 > 0,\ c_2 > 0,\ n_0 \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall\ n \geq n_0$

Observe que al notación Θ define un conjunto de funciones

Conjuntos

$\Theta(g(n)) = \{f : N \to R^+ \mid c_1 > 0,\ c_2 > 0,\ n_0 \mid 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall\ n \geq n_0\}$

Notación Θ - ejemplo

- Muestre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Notación Θ - ejemplo

- Muestre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- Para probar entonces encontramos las constantes c_1, c_2, n_0 tal que:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

para todo $n \geq n_0$.

Notación Θ - ejemplo

- Muestre que $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- Para probar entonces encontramos las constantes c_1, c_2, n_0 tal que:

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

para todo $n \geq n_0$.
Si dividimos la expresión por n^2 tenemos

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

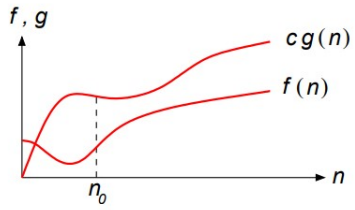
Notación Θ - ejemplo

- La inecuación más a la derecha es válida para cualquier valor $n_0 \geq 1$ si tomamos $c_2 \geq \frac{1}{2}$
- De la misma forma la inecuación de la izquierda es válida para $n_0 \geq 7$ si tomamos $c_2 \geq \frac{1}{14}$
- La selección de éstas variables depende de la función $\frac{1}{2}n^2 - 3n$
- Una función diferente que pertenezca a $\Theta(n^2)$ probablemente requerirá de otras constantes
- Existen otras posibles constantes, **pero lo importante es que existan**

Notación Θ - ejemplo

- Usando la definición formal de Θ pruebe que $6n^3 \neq \Theta(n^2)$

Notación O



$$f(n) = O(g(n))$$

Notación O

- La notación O define un limite superior
- $f(n) = O(g(n))$, si existen constantes positivas c y n_0 tal que, para $n \geq n_0$, el valor de $f(n)$ es menor o igual a $cg(n)$
- Se puede decir que $g(n)$ es un límite asintótico superior (*asymptotically upper bound*) para $f(n)$.

Definición

$f(n) = \Theta(g(n)), \exists\ c,\ n_0 \mid 0 \leq f(n) \leq cg(n),\ \forall\ n \geq n_0$

Observe que al notación Θ define un conjunto de funciones

Conjuntos

$\Theta(g(n)) = \{f : N \to R^+ \mid c,\ n_0 \mid 0 \leq f(n) \leq cg(n),\ \forall\ n \geq n_0\}$

Notación O - ejemplos

- Sea $f(n) = (n+1)^2$

Notación O - ejemplos

- Sea $f(n) = (n+1)^2$
- Entonces $f(n) = O(n^2)$, cuando $n_0 = 1$ y $c = 4$ ya que
$$(n+1)^2 \leq 4n^2 \text{ y } n_0 = 1$$
para todo $n \geq n_0$.

Notación O - ejemplos

- Sea $f(n) = (n+1)^2$
- Entonces $f(n) = O(n^2)$, cuando $n_0 = 1$ y $c = 4$ ya que
$$(n+1)^2 \leq 4n^2 \text{ y } n_0 = 1$$
para todo $n \geq n_0$.
- Sea $f(n) = n$ y $g(n) = n^2$. Muestre que $g(n) \neq O(n)$

Notación O - ejemplos

- Sea $f(n) = (n+1)^2$
- Entonces $f(n) = O(n^2)$, cuando $n_0 = 1$ y $c = 4$ ya que
$$(n+1)^2 \leq 4n^2 \text{ y } n_0 = 1$$
para todo $n \geq n_0$.
- Sea $f(n) = n$ y $g(n) = n^2$. Muestre que $g(n) \neq O(n)$
No existe constante que sea mayor que la función lineal

Notación O - ejemplos

- Muestre que $g(n) = 3n^3 + 2n^2 + n$ es $O(n^2)$

Notación O - ejemplos

- Muestre que $g(n) = 3n^3 + 2n^2 + n$ es $O(n^2)$
- Muestre que $log_5 n = O(logn)$

Notación O - ejemplos

- Muestre que $g(n) = 3n^3 + 2n^2 + n$ es $O(n^2)$
- Muestre que $log_5 n = O(logn)$ necesitamos encontrar una constante, en este caso $log_b c$

Notación O

- Cuando la notación O es utilizada para expresar el tiempo de ejecución de un algoritmo en el peor caso, también se esta definiendo el límite superior del tiempo de ejecución de ese algoritmo para todas las entradas
- Por ejemplo en el algoritmo de ordenación por inserción el peor caso es $O(n^2)$, para cualquier entrada posible necesitamos encontrar

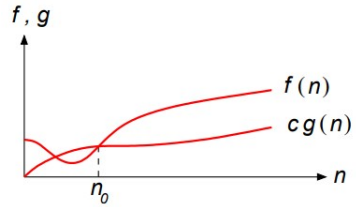
Notación O

- Técnicamente es un abuso decir que el tiempo de ejecución del algoritmo de ordenación por inserción es $O(n^2)$, sin especificar que es para el peor caso
- Siempre es necesario mencionar que esta notación se refiere al peor caso y no generalizar al algoritmo con esta notación

Notación O

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times O(f(n)) &= O(f(n)) \quad c = constante \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n)) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Notación Ω



$$f(n) = \Omega(g(n))$$

Notación Ω

- La notación Ω define un limite inferior
- $f(n) = \Omega(g(n))$, si existen constantes positivas c y n_0 tal que, para $n \geq n_0$, el valor de $f(n)$ es mayor o igual a $cg(n)$
- Se puede decir que $g(n)$ es un límite asintótico inferior (*asymptotically lower bound*) para $f(n)$.

Definición

$f(n) = \Theta(g(n)), \exists\ c,\ n_0 \mid 0 \leq cg(n) \leq f(n),\ \forall\ n \geq n_0$

Observe que al notación Θ define un conjunto de funciones

Conjuntos

$\Theta(g(n)) = \{f : N \to R^+ \mid c,\ n_0 \mid 0 \leq cg(n) \leq f(n),\ \forall\ n \geq n_0\}$

Notación Ω

- La notación Ω es utilizada para expresa el tiempo de ejecución de un algoritmo en el mejor caso, definiendo un límite inferior del tiempo de ejecución de ese algoritmo para todas las entradas
- Por ejemplo el algoritmo de ordenación por inserción es $\Omega(n)$ en el mejor caso

Notación Ω

- Mostrar que $f(n) = 3n^3 + 2n^2$ es $\Omega(n^3)$

Teorema

- Sea $f(n)$ y $g(n)$,

  $$f(n) = \Theta(n)$$

  si y solamente si,

  $$f(n) = O(n)$$

  , y

  $$f(n) = \Omega(n)$$

Notaciones

- Existen otras notaciones
  - Notación o (o pequeño)
  - Notación ω
- Éstas dos notaciones no son utilizadas normalmente, presentan notaciones laxas sobre las funciones de complejidad

Comparación de programas

- Se puede validar programas comparando las funciones de complejidad
- Un programa con tiempo de ejecución $O(n)$ es mejor que otro con tiempo $O(n^2)$, sin embargo, las constantes de proporcionalidad pueden alterar esta consideración
- Ejemplo: un un programa lleva $100n$ unidades de tiempo para ser ejecutado y otro toma $2n^2$. Cuál de los programas es mejor?
  - Depende del tamaño del problema
  - Para $n < 50$, el programa con tiempo $2n^2$ de que el posee tiempo $100n$
  - Para problemas con entrada de datos pequeña es preferible utilizar el programa con tiempo de ejecución $O(n^2)$
  - Entretanto, cuando n crece, el programa con tiempo de ejecución $O(n^2)$ lleva mucho mas tiempo que el programa n
- Vamos a enfocarnos en valores grande de n

Complejidad constante

- $f(n) = O(1)$
  - El uso del algoritmo es independiente del valor de n
  - Las instrucciones del algoritmo son ejecutados un número fijo de veces
- Qué significa un algoritmo sea $O(2)$ o $O(5)$??

Complejidad logarítmica

- $f(n) = O(logn)$
  - Ocurre típicamente en algoritmos que resuelven un problema transformando en problemas menores
  - Puede ser considerado casi como una constante
- Suponiendo que la base del logaritmo sea 2:
  - Para $n = 1000,\ log_2 \approx 10$
  - Para $n = 1000000,\ log_2 \approx 20$
- Algoritmo de búsqueda binaria

Complejidad lineal

- $f(n) = O(n)$
  - En general, un pequeño trabajo es realizado sobre la entrada, ejemplo en un array sería por cada elemento
  - Es la mejor situación posible para un algoritmo que tiene que procesar toda la data de entrada
  - Cada vez que n se duplica, el tiempo de ejecución también se duplica
- Ejemplos:
  - Buscar el menor elemento de un array
  - Algoritmo para determinar la planaridad de un grafo

Complejidad lineal logarítmica

- $f(n) = O(nlogn)$
  - Este tiempo de ejecución ocurre típicamente en algoritmos que resuelven un problema quebrando el problema en problemas menores, resolviendo cada uno independientemente y después agrupando las soluciones
  - Caso típico de los algoritmos basados en el paradigma divide y vencerás
- Suponiendo que la base del algoritmo sea 2:
  - Para $n = 1000000,\ log_2 \approx 20000000$
  - Para $n = 2000000,\ log_2 \approx 42000000$
- Algoritmo Mergesort

Complejidad cuadrática

- $f(n) = O(n^2)$
  - Algoritmos de este orden de complejidad ocurren cuando los items de datos son procesados en pares, muchas veces en un bucle dentro de otro
  - Para $n = 1000$, el número de operaciones es del orden de 1000000
  - Siempre que n se duplica el tiempo de ejecución es multiplicado por 4
  - Algoritmos de este tipo son útiles para resolver problemas de tamaño *relativamente pequeño*
- Ejemplo: algoritmos de ordenación simple como el insertion o selection

Complejidad cúbica

- $f(n) = O(n^3)$
  - Algoritmos de este orden de complejidad son útiles apenas para problemas *relativamente pequeños*
  - Para $n = 100$, el número de operaciones es del orden de 1000000
  - Siempre que n se duplica el tiempo de ejecución es multiplicado por 8
- Ejemplo: algoritmo para multiplicar matrices

Complejidad exponencial

- $f(n) = O(2^n)$
  - Algoritmos de este orden de complejidad son útiles sólo desde el vista práctico
  - Usualmente ocurren en ka solución de problemas cuando se utiliza la fuerza bruta para resolverlos
  - Para $n = 20$, el número de operaciones es cerca de 1000000
  - Siempre que n se duplica el tiempo de ejecución queda elevado al cuadrado
- Ejemplo: el agente viajero

Complejidad factorial

- $f(n) = O(n!)$
  - La complejidad es mucho peor que la función exponencial
  - Usualmente en algoritmos de fuerza bruta
  - Para $n = 20$, el número de operaciones es 2432902008176640000, 19 dígitos
  - Para $n = 40$, el número de operaciones es un número de 48 dígitos
- Ejemplo: problemas de combinatorias

Comparación

Função de custo	Tamanho n					
	10	20	30	40	50	60
n	0,00001 s	0,00002 s	0,00003 s	0,00004 s	0,00005 s	0,00006 s
n^2	0,0001 s	0,0004 s	0,0009 s	0,0016 s	0,0.35 s	0,0036 s
n^3	0,001 s	0,008 s	0,027 s	0,64 s	0,125 s	0.316 s
n^5	0,1 s	3,2 s	24,3 s	1,7 min	5,2 min	13 min
2^n	0,001 s	1 s	17,9 min	12,7 dias	35,7 anos	366 seg
3^n	0,059 s	58 min	6,5 anos	3855 sec	10^8 sec	10^{13} sec

Jerarquía de funciones

La siguiente jerarquía de funciones puede ser definida desde el punto de vista asintótico

$$1 \prec loglogn \prec logn \prec n^{\epsilon} \prec n^{c} \prec n^{logn} \prec c^{n} \prec n^{n} \prec c^{c^{n}}$$

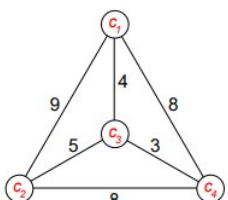
donde ϵ y c son constantes arbitrarias $0 < \epsilon < 1 < c$
<Grafique las funciones>

Algoritmo exponencial vs polinomial

- La distinción entre estos dos tipos de algoritmos se torna significativa cuando el "tamaño" del problema a ser resuelto crece
- Los algoritmos polinomiales son muchos más útiles en el práctica que los algoritmos exponenciales
- Generalmente, los algoritmos exponenciales son simples variaciones de una búsqueda exhaustiva
- Los algoritmos polinomiales son generalmente obtenidos a través de un entendimiento mas profundo de la estructura del problema
- Son muy pocos algoritmos exponenciales que en la práctica pueden ser útiles. Un ejemplo para mencionar es el algoritmo *simplex* para programación lineal cuya complejidad es exponencial en el peor caso, sin embargo, ejecuta muy rápido en la práctica

Problema del agente viajero

- Un agente viajero desea visitar n ciudades de forma tal que su viaje inicie y termine en la misma ciudad, y cada ciudad debe ser visitada una única vez
- Dada la siguiente figura, una representación de grafos para una instancia de este problema, podemos encontrar una solución en el circuito c_1, c_3, c_4, c_2, c_1



Problema del agente viajero

- Un algoritmo simple sería verificar todas las rutas e escoger la menor de ellas que cumple con las condiciones
- Hay $(n-1)!$ rutas posibles e la distancia total recorrida en cada ruta envuelve n adiciones, luego el número total de adiciones es $n!$
- En el ejemplo anterior tendríamos 24 adiciones
- Suponga ahora 50 ciudades el número de adiciones sería $50! \approx 10^{64}$
- En un computador que ejecuta 10^9 adiciones por segundo, el tiempo total para resolver el problema con 50 ciudades sería mayor que 10^{45} siglos sólo para ejecutar las adiciones
- Este problema aparece con frecuencia en problemas de optimización de recorrido de caminos, especialmente en transporte

Tiempo de ejecución

- Procedimientos no recursivos
- Procedimientos recursivos

Procedimiento no recursivo

Algoritmo para ordenar (ascendente) n elementos de un array A

Algorithm 6: Ordena(A)

Data: i, j, min, x
Result: A ordenado

```
1 for i=1 to n-1 do
2     min = i
3     for j = i + 1 to n do
4         if A[j] < A[min] then
5             min = j
6     x = A[min]
7     A[min] = A[i]
8     A[i] = x
```

Procedimiento no recursivo

Algoritmo para ordenar (ascendente) n elementos de un array A

Algorithm 7: Ordena(A)

Data: i, j, min, x
Result: A ordenado

```
1 for i=1 to n-1 do
2     min = i
3     for j = i + 1 to n do
4         if A[j] < A[min] then
5             min = j
6     x = A[min]
7     A[min] = A[i]
8     A[i] = x
```

Desarrollo: $O(n^2)$

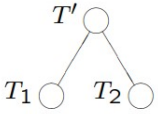
Insertion sort

- Escribir algoritmo
- Complejidad

Algoritmos recursivos

- Un objeto es recursivo cuando es definido parcialmente en términos de sí mismo
- Ejemplo 1: Números naturales
  - 1 es un numero natural
  - el sucesor de un número natural es también un número natural
- Ejemplo 2: Función factorial
  - $0! = 1$
  - si $n > 0$ entonces $n! = n(n-1)!$

- Ejemplo 3: árboles
  - el árbol vacío es un árbol
  - si T_1 y T_2 son árboles entonces T' es un árbol



Algoritmos recursivos: ejemplos

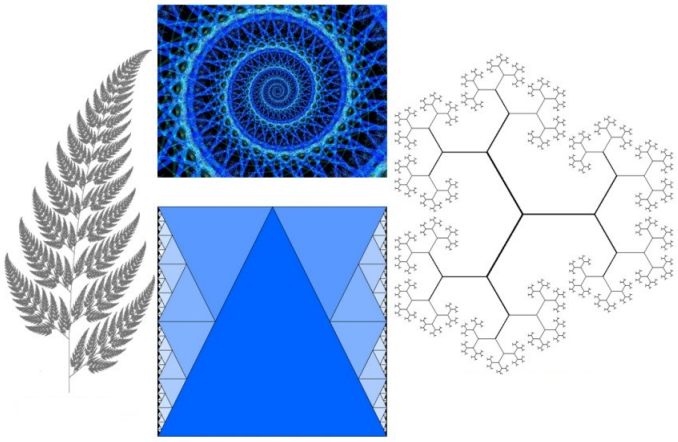


Figure: Fractales.

Algoritmos recursivos: ejemplos

Figure: Figuras recursivas.