Complejidad

Dpto. Ingeniería Industrial, Universidad de Chile

IN3701, Optimización

7 de marzo de 2011

Contenidos

¿Qué es un Algoritmo?

Definition (Algoritmo)

- Un algoritmo es un conjunto de pasos bien definido que toma una entrada y produce una salida.
- Un algoritmo es una secuencia de pasos que transforma la entrada en la salida deseada.

Definition (Algoritmos relacionados a problemas computacionales)

Un problema computacional es una deseada relación entre una entrada y una salida.
Un algoritmo resuelve un problema computacional si logra producir la relación deseada.

Problemas, Instancias, Correctitud

Definition (Problema de ordenamiento)

Entrada: una secuencia de números $\langle a_1 \ldots, a_n \rangle$.

Salida: Un reordenamiento $\langle a'_1, \ldots, a'_n \rangle$ tal que $a'_i \leq a'_{i+1}, i = 1, \ldots, n-1$.

Ejemplo:

Buscamos un algoritmo que dada la secuencia $\langle 31, 41, 59, 26, 41, 58 \rangle$ entregue la secuencia $\langle 26, 31, 41, 41, 58, 59 \rangle$
Una entrada particular se llama una instancia del problema.

Definition (Algoritmos Correctos)

Un algoritmo se dice correcto si para todas las instancias termina con una salida correcta. Un algoritmo correcto se dice resolver el problema computacional asociado.

- Algoritmos incorrectos pueden ser de interés (Ej: Algoritmos aleatorios).

Algoritmos y sus límites:

- Un problema / función / pregunta es computable si existe un algoritmo que lo resuelva / compute / responda.
- Computabilidad depende del modelo de máquina que usemos.
- No todos los problemas son iguales...
  - Existen clasificaciones de dificultad para problemas (Complejidad).
  - $\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NP}$-completo, $\mathcal{NP}$-duro, co-$\mathcal{NP}$, $\mathcal{P}$-space, etc.
  - La pregunta más famosa en complejidad: $\mathcal{P} \neq \mathcal{NP}$?
- ¿ ... y si los computadores fueran instantáneos?
  - Aún necesitamos saber si un algoritmo es correcto
  - Cualquier algoritmo correcto bastaría
  - Ejemplo algoritmo universal: (Particionar + Enumerar)
- ¿ ... Y en el mundo real?
  - Tiempo y memoria (espacio) es limitado.
  - Múltiples algoritmos para un problema, distinto uso de recursos.
  - InsertionSort ordena n elem. en approx. $c_1 n^2$ tiempo.
  - MergeSort ordena n elem. en approx. $c_2 n \log_2(n)$ tiempo.
  - Hardware es también un factor en la velocidad.

Eficiencia y Velocidad

- Consideremos máquina A de 1GHz y máquina B de 10MHz (A es 100 veces más rápida que B).
  - En A corremos InsertionSort en 10^6 elementos con $c_1 = 2$.
  - En B corremos MergeSort en 10^6 elementos con $c_2 = 50$.
  - El resultado es $t_A = \frac{2 \cdot (10^6)^2}{10^9} = 2000\text{s}$, $t_B = \frac{50 \cdot 10^6 \cdot \log_2(10^6)}{10^7} = 100\text{s}$
- Supongamos que disponemos de un PC de 1MHz, ¿qué tamaño de problemas podemos resolver en el tiempo?

instr.	sec.	min.	hora	día	mes	año	siglo
$\log_2(n)$	2^{10^6}	$2^{10^{7,77}}$	$2^{10^{9,55}}$	$2^{10^{10,93}}$	$2^{10^{12,42}}$	$2^{10^{13,49}}$	$2^{10^{15,49}}$
$\sqrt{n}$	$2^{10^{1,6}}$	$10^{15,55}$	$10^{19,11}$	$10^{21,87}$	$10^{24,84}$	$10^{26,99}$	$10^{30,99}$
n	10^6	$10^{7,77}$	$10^{9,55}$	$10^{10,93}$	$10^{12,42}$	$10^{13,49}$	$10^{15,49}$
$n \log_2(n)$	$10^{4,79}$	$10^{6,44}$	$10^{8,12}$	$10^{9,44}$	$10^{10,86}$	$10^{11,90}$	$10^{13,83}$
n^2	1000	$10^{3,88}$	$10^{4,77}$	$10^{5,46}$	$10^{6,21}$	$10^{6,74}$	$10^{7,74}$
n^3	100	391	1532	4420	$10^{4,14}$	$10^{4,49}$	$10^{5,16}$
2^n	19	25	31	36	41	44	51
$n!$	9	11	12	13	15	16	17

InsertionSort

Problema:

entrada: secuencia de n números $\langle a_1, \ldots, a_n \rangle$

salida: reordenamiento $\langle a'_1, \ldots, a'_n \rangle$ tal que $a'_i \leq a'_{i+1}$.

InsertionSort

1: $[A]$ secuencia de entrada
2: **for** $j = 2, j \leq length[A]$ **do**
3: $key \leftarrow A[j]$, $i \leftarrow j - 1$.
4: /* *inserta* $A[j]$ *en la secuencia ordenada* $A[1], \ldots, A[j-1]$
5: **while** $i > 0$ and $A[i] > key$ **do**
6: $A[i+1] \leftarrow A[i]$, $i \leftarrow i - 1$.
7: $A[i+1] \leftarrow key$.

Convenciones y Recursos

- Convenciones en pseudo-códigos:
  - Identación identifica bloques de código.
  - Aceptamos controles como **while**, **for**, **if-then-else**
  - Texto que siga /* es considerado comentario.
  - Para asignar valores usamos $a \leftarrow b$.
  - Aceptamos uso de arreglos indexados por números consecutivos, i.e. $[A] = A[1], \ldots, A[n]$.
  - Las expresiones booleanas son evaluadas de izquierda a derecha y cortocircuitadas. Por ejemplo (**True** or **A**) siempre retorna **True** sin evaluar **A**.
  - Funciones reciben parámetros por valor, y guardan una copia local.
- Midiendo Recursos:
  - Analizar un algoritmo es tratar de predecir cuantos recursos usará el algoritmo.
  - Recursos pueden ser memoria, disco duro, ancho de banda de comunicaciones.
  - Típicamente nosotros usaremos tiempo.
  - La idea es proveer de una escala de calidad de algoritmos.

Modelo Computacional

- Necesitamos definir un tipo de máquina a usar, que sea medianamente realista.
- Usaremos máquinas **RAM** (Random Access Machine), i.e. un PC común y corriente ($\longleftrightarrow$ a Máquina de Turing Universal 1936).
  - Instrucciones son ejecutadas secuencialmente (sin paralelismo)
  - Datos nativos son enteros y puntos flotante.
  - Control incluye condicionales, loops, llamada a sub-rutinas y retorno de rutinas.
  - Aritmética incluye $+, -, \cdot, /$ y multiplicación por 2^k.
  - Instrucciones básicas toman una unidad de tiempo.
  - Asumimos codificación binaria.
  - Números nativos usan k bits (típicamente 32 o 64).
  - En general los numeros enteros necesitan $\lceil \log(n) \rceil$ bits para ser representados.
  - ¿Qué pasa con racionales?
  - ¿Qué pasa con los irracionales?

Buscando algo razonable

- Tiempo ejecución depende en el tamaño del problema.
- Instancias del mismo tamaño pueden demorar distinto.
- ¿Cómo definimos tamaño?
  - Depende del problema.... usamos el tamaño natural.
  - Para ordenar, el tamaño es el largo de la entrada.
    - Si los numeros son nativos, basta decir n.
    - Si los numeros son arbitrarios, necesitamos el número de bits para guardarlos a todos, i.e. $\sum_{i=1}^{n} \lceil \log_2(a_i) \rceil$.
  - Para triangularizar $M \in \mathbb{Z}^{n \times m}$, el tamaño es $nm \lceil \log_2(m_{\text{máx}}) \rceil$.
  - Computar $a \cdot b$ tamaño es $\lceil \log_2(a) \rceil + \lceil \log_2(b) \rceil$.
- Tiempo de ejecución es el número de pasos básicos que el algoritmo realiza en una instancia dada.
  - Mismo tamaño puede llevar distinto tiempo.
  - Sort(1,2,3,4,5,6).
  - Sort(6,3,4,2,1,5).
  - ¿Qué tiempo reportamos?

Análisis de InsertionSort

InsertionSort (datos nativos)

1:	$[A]$ secuencia de entrada.	$c_1 n$
2:	**for** $j = 2, j \leq length[A]$ **do**	
3:	$key \leftarrow A[j]$, $i \leftarrow j - 1$.	$c_2(n-1)$
4:	/* *inserta* $A[j]$ *en la secuencia ordenada* $A[1], \ldots, A[j-1]$	
5:	**while** $i > 0$ and $A[i] > key$ **do**	
6:	$A[i+1] \leftarrow A[i]$, $i \leftarrow i - 1$.	$c_3 \sum(t_j : j = 2, \ldots, n)$
7:	$A[i+1] \leftarrow key$.	$c_4(n-1)$

- $T(n) = -(c_2 + c_4) + (c_1 + c_2 + c_4)n + c_3 \sum(t_j : j = 2, \ldots, n)$.
  - Tiempo de ejecución variable: $1 \leq t_j \leq j$.
  - Mejor caso: $T(n) \geq an + b$, i.e. $T(n)$ es al menos lineal.
  - Peor caso: $T(n) \leq an^2 + bn + c$, i.e. $T(n)$ es a lo más cuadrática.
  - ¿Podemos hablar de tiempo promedio?
    - Necesitamos definir distribución en la entrada.
  - Peor caso nos da cota superior del tiempo.
  - En la mayoría de los casos, el peor caso es bastante representativo del promedio.

Órdenes de crecimiento

- Nuestro objetivo es clasificar algoritmos.
- Hemos hecho bastantes simplificaciones en el modelo computacional.
- Haremos una simplificación más, cuando comparamos tiempos de ejecución, eliminamos las constantes
  - Informalmente $\mathcal{O}(an^2 + bn + c) = \mathcal{O}(an^2 + bn) = \mathcal{O}(an^2) = \mathcal{O}(n^2)$.
  - $\mathcal{O}(n^k + \sum(a_i x^i : i = 0, \ldots, k-1)) = \mathcal{O}(n^k)$.
  - $\mathcal{O}(2^n + n^k) = \mathcal{O}(2^n)$.

Notación Θ (asintóticamente equivalente)

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, decimos que $f(n) = \Theta(g(n)) \Leftrightarrow \exists$ $c_1, c_2 \in \mathbb{R}_+, n_o \in \mathbb{N}$ tal que $c_1 g(n) \leq f(n) \leq c_2 g(n)\ \forall n \geq n_o$.

Notación $\mathcal{O}$ (cota superior asintótica)

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, decimos que $f(n) = \mathcal{O}(g(n)) \Leftrightarrow \exists\, c \in \mathbb{R}_+, n_o \in \mathbb{N}$ tal que $f(n) \leq c g(n)\ \forall n \geq n_o$.

Notación Asintótica

Notación Ω (cota inferior asintótica)

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, decimos que $f(n) = \Omega(g(n)) \Leftrightarrow \exists\, c \in \mathbb{R}_+, n_o \in \mathbb{N}$ tal que $cg(n) \leq f(n)\ \forall n \geq n_o$.

Teorema

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = \mathcal{O}(g(n))$ y $f(n) = \Omega(g(n))$.

Notación o (sobre-estimación cota superior asintótica)

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, decimos que $f(n) = o(g(n)) \Leftrightarrow$ $\lim(f(n)/g(n) : n \to \infty) = 0$.

Notación ω (sub-estimación cota inferior asintótica)

Dado $f, g : \mathbb{R}_+ \to \mathbb{R}$, decimos que $f(n) = \omega(g(n)) \Leftrightarrow$ $\lim(g(n)/f(n) : n \to \infty) = 0$.

¿En Español?

- Podemos hacer una analogía entre estas notaciones y comparaciones entre números:
  - $f(n) = \Theta(g(n)) \approx f = g.$
  - $f(n) = \mathcal{O}(g(n)) \approx f \leq g.$
  - $f(n) = o(g(n)) \approx f < g.$
  - $f(n) = \Omega(g(n)) \approx f \geq g.$
  - $f(n) = \omega(g(n)) \approx f > g.$
- Algunas diferencias:
  - No todas las funciones son comparables en términos asintóticos: $n, n^{1+sin(n)}$.

Problema Computacional

Problema Abstracto:

Un problema abstracto Q es una relación binaria entre un conjunto de instancias I y un conjunto de soluciones S.

Ejemplo:

Problema: Camino mas Corto

Instancias: Es una tripleta formada por un grafo $G = (V, A)$ y dos nodos $u, v \in V$.

Soluciones: Secuencia de arcos consecutivos en G, incluyendo la secuencia vacía para representar que no existen caminos.

Problemas de Decisión y Optimización

Problema de Decisión

Un problema se dice de decisión, si el conjunto de soluciones posibles se puede reducir a $\{0, 1\}$ (equivalentemente a $\{si, no\}$).

Ejemplo:

Problema: Camino de largo fijo

Instancias: Una 4-tupla consistente de un Grafo $G = (V, A)$, un par de nodos $s, t \in V$ y un entero k.

Soluciones: 1 si existe un camino de s a t que no use mas de k arcos, 0 si no.

Problemas de Decisión y Optimización

Problema de Optimización

Un problema se dice de optimización si su objetivo es buscar el mínimo o máximo de una función sobre un conjunto definido (posiblemente vacío).

- La teoría de complejidad se centra en torno a problemas de decisión.
- En general, un problema de decisión puede re-escribirse como un problema de optimización y viceversa.
- El último punto permite usar la teoría de complejidad a problemas de optimización también.

Definición

$\mathcal{P}$ es el conjunto de problemas para los cuales existe un algoritmo que lo resuelve (en el peor caso) en tiempo polinomial.

Certificados Polinomiales

- Problemas en $\mathcal{P}$ se consideran fáciles.
- Hay Problemas no polinomiales?
- Consideremos $HAM - CYCLE$ (ciclo hamiltoniano):
  - Dado un grafo G, decidir si existe un ciclo simple en G que visite todos los nodos en G ($G \in HAM - CYCLE$).
  - Si conocemos un ciclo satisfaciendo esto, la respuesta es obvia.
  - Chequear si un conjunto de arcos es un ciclo hamiltoniano es $\mathcal{P}$.
  - Para saber si $G \in HAM - CYCLE$ basta chequear todos los sub-conjuntos de arcos (esto no es $\mathcal{P}$).
  - Consideremos el algoritmo $\mathcal{A}$, tal que $\mathcal{A}(G, C) = 1 \Leftrightarrow C$ ciclo hamiltoniano de G.
  - $G \in HAM - CYCLE \Leftrightarrow \exists C : \mathcal{A}(G, C) = 1$.

Algoritmo de Certificación

Un algoritmo $\mathcal{A}(\cdot, \cdot)$ certifica un problema P si
$p \in P \Leftrightarrow \exists c : \mathcal{A}(p, c) = 1$.
$\mathcal{A}$ se dice polinomial si $T(|p|, |c|) = \mathcal{O}(|p|^k)$ para algún k fijo.

Clase $\mathcal{NP}$

Definición $\mathcal{NP}$

$\mathcal{NP}$ es la clase de problemas para los cuales existe un algoritmo que los certifica en tiempo polinomial.

- Claramente $HAM - CYCLE \in \mathcal{NP}$.
- Decidir si un LP es factible ($LPFEAS$) esta en $\mathcal{NP}$.
- $LPFEAS^c \in \mathcal{NP}$
- No es claro que $HAM - CYCLE^c \in \mathcal{NP}$.

Una Visión Alternativa

Pensemos en un mundo infinitamente paralelo....

- Preguntas abiertas:
  - es $\mathcal{NP}$= $\mathcal{P}$?
  - es $\mathcal{NP}$= co-$\mathcal{NP}$?
  - existe algún problema en $(\mathcal{NP} \cap co - \mathcal{NP}) \setminus \mathcal{P}$?

La idea central

¿Por qué creemos que $\mathcal{P} \neq \mathcal{NP}$?

- El argumento principal es la existencia de problemas $\mathcal{NP}$-completos.
  - Los problemas $\mathcal{NP}$-completos satisfacen lo siguiente: Si alguno de ellos esta en $\mathcal{P}$, entonces todos los problemas están en $\mathcal{P}$; i.e. $\mathcal{P} = \mathcal{NP}$.
  - A pesar de mas de 40 años de trabajo, no se conoce ningún algoritmo polinomial para ningún problema $\mathcal{NP}$-completo.
  - $HAM - CYCLE$ es $\mathcal{NP}$-completo.
  - Si $\mathcal{NP} \setminus \mathcal{P} \neq \emptyset$, $HAM - CYCLE$ sería uno de ellos.
- En algún sentido, los problemas $\mathcal{NP}$-completos, son los más difíciles en la clase $\mathcal{NP}$.

Reducibilidad

Un problema P puede reducirse a otro problema P', si cualquier instancia $p \in P$ puede refrasearse como un problema $p' \in P'$, cuya solución provee la respuesta al problema original.

Reducibilidad

Ejemplo:

El problema de resolver ecuaciones lineales $ax + b = 0$ puede reducirse al problema de resolver ecuaciones cuadráticas $0x^2 + ax + b = 0$.

Algoritmo de Reducción

Dados dos problemas P_1, P_2, el algoritmo $\mathcal{A}$ es una reducción de P_1 en P_2 si $p \in P_1 \Leftrightarrow \mathcal{A}(p) \in P_2$.
Si $\mathcal{A}$ es polinomial, decimos que $P_1 \leq_p P_2$.

- Consecuencias:
  - Consideremos $\mathcal{A}_2$ un algoritmo para P_2, y $\mathcal{A}_r$ un algoritmo de reducción de P_1 a P_2.
  - Si $\mathcal{A}_1(p) = \mathcal{A}_2(\mathcal{A}_r(p))$, entonces $\mathcal{A}_1$ es un algoritmo para P_1.
  - Si $\mathcal{A}_2$ y $\mathcal{A}_r$ son polinomiales, entonces $\mathcal{A}_1$ es polinomial.
  - Si $\mathcal{A}_r$ es polinomial, y $P_2 \in \mathcal{P}$ entonces $P_1 \in \mathcal{P}$.
  - Si $\mathcal{A}_r$ es polinomial, y $P_2 \in \mathcal{NP}$ entonces $P_1 \in \mathcal{NP}$.

$\mathcal{NP}$-completitud

Problemas $\mathcal{NP}$-completos:

Un problema P se dice $\mathcal{NP}$-completo si:

- $P \in \mathcal{NP}$.
- $\forall L \in \mathcal{NP}\ L \leq_p P$.

- Existen problemas $\mathcal{NP}$-completos?
  - Un primer problema es 3-SAT.
  - Otros problemas incluyen:
    - Problema de clique en un grafo.
    - Programación entera.
    - Vertex Cover.
    - $HAM - CYCLE$.
- ¿Cómo podemos demostrar que un problema es $\mathcal{NP}$-completo?
  - El primero es el difícil, la idea, es codificar un algoritmo, con su entrada, como una secuencia de pasos lógicos.
  - Pero después es fácil.... Como demostraríamos que IP es $\mathcal{NP}$-completo?