Análisis de la eficiencia de un algoritmo

IIC2283 – Diseño y Análisis de Algoritmos

Departmento de Ciencia de la Computación
Pontificia Universidad Católica de Chile

2023-2

Una noción general de algoritmo

Sea $\mathcal{P}$ un problema a resolver, $\mathcal{I}_\mathcal{P}$ su conjunto de instancias (casos posibles del problema) y $\mathcal{S}_\mathcal{P}$ el conjunto de soluciones a las instancias

Vamos a pensar en un algoritmo $\mathcal{A}$ como una función $\mathcal{A} : \mathcal{I}_\mathcal{P} \to \mathcal{S}_\mathcal{P}$

Esta es una representación general que incluye tanto a problemas de decisión como a los de computación.

Tiempo de ejecución de un algoritmo

A cada algoritmo $\mathcal{A}$ asociamos una función $\text{tiempo}_{\mathcal{A}} : \mathcal{I}_{\mathcal{P}} \to \mathbb{N}$ tal que:

$\text{tiempo}_{\mathcal{A}}(I)$: número de pasos realizados por $\mathcal{A}$ con entrada $I \in \mathcal{I}_{\mathcal{P}}$

Para definir esta función tenemos que definir qué operaciones vamos a contar, y qué costo les asignamos.

Es importante definir el modelo de computación sobre el que vamos a trabajar

Modelos de Computación

Antes de diseñar y analizar algoritmos, es importante definir el *modelo de computación* subyacente.

Esto significa definir el tipo de computador hipotético sobre el cual se ejecutarán nuestros algoritmos.

Cada modelo de computación especifica las operaciones elementales que el computador hipotético puede ejecutar.

Esto afecta la manera en que uno diseña y analiza algoritmos sobre esos modelos, entonces es importante aclararlo desde un principio.

Dos modelos típicos: las Máquinas de Turing y las Máquinas de Acceso Aleatorio (RAM, por sus siglas en inglés).

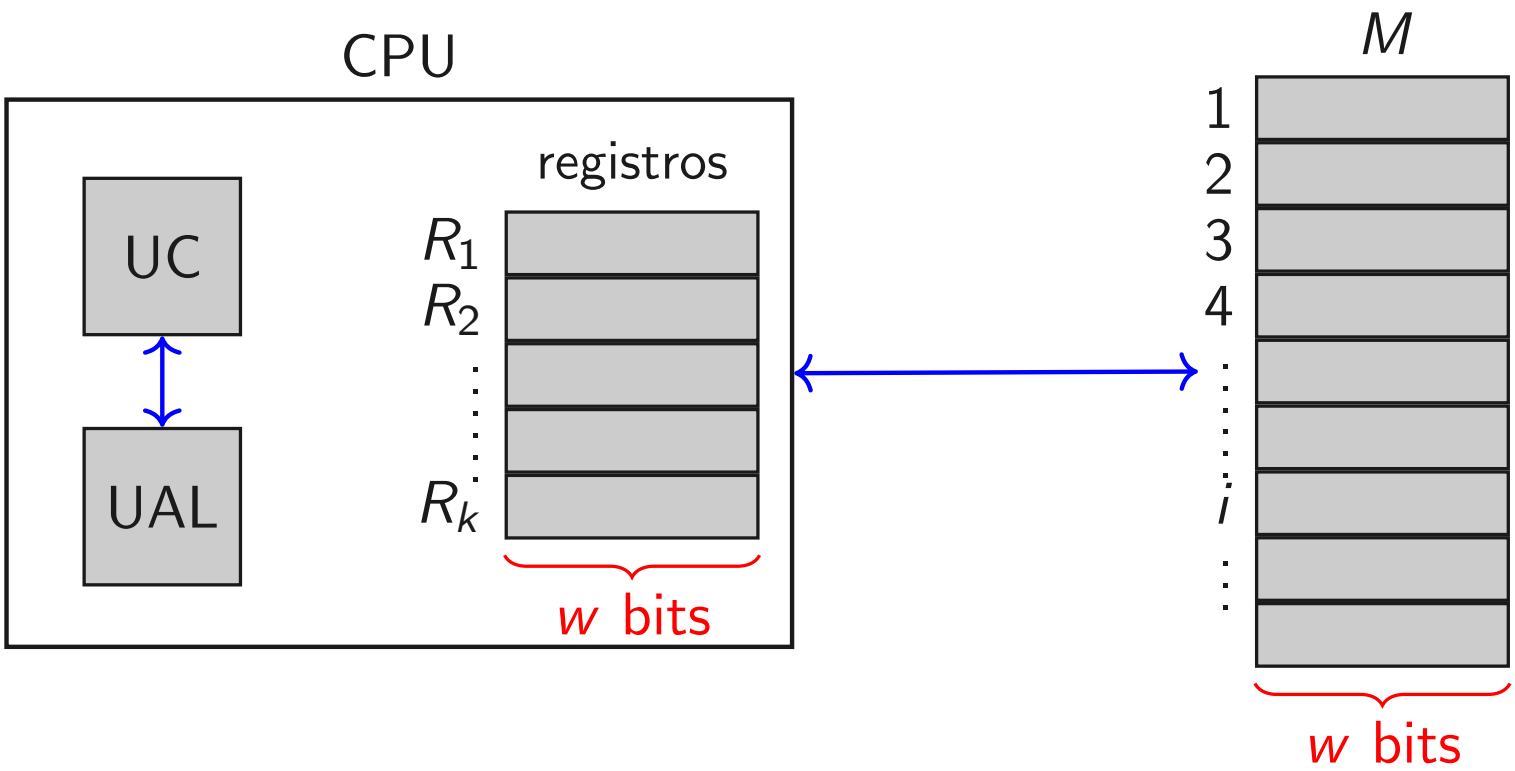
El Modelo de Computación Word RAM

Word RAM (simplemente RAM) es uno de los modelos más usados en la literatura por su similitud con los computadores actuales

Es una máquina hipotética que consiste de:

- **CPU**: la cual está formada por
  - **Registros**: k celdas de w bits cada una (el parámetro más importante del modelo)
  - **Instrucciones**: un conjunto de instrucciones que la CPU es capaz de ejecutar ($\leftarrow$, $+$, $\times$, $-$, $/$, **and**, **or**, **not**, $=$, $\leq$, $\geq$, $<$, $>$, $\neq$, saltos implícitos o explícitos, y ops. a nivel de bits).
  - **Unidad de control**: ejecuta las instrucciones de un algoritmo por *pasos* o *ciclos del procesador* (una instrucción por ciclo)
- **Memoria**: una cantidad infinita de celdas de w bits cada una, $M[1]$, $M[2]$, ..., que pueden ser accedidas de manera aleatoria (i.e., en el orden que un algoritmo necesite) para lectura y escritura y cada acceso toma un ciclo del procesador

El Modelo de Computación Word RAM



Recordatorio:

A cada algoritmo $\mathcal{A}$ asociamos una función $\text{tiempo}_{\mathcal{A}} : \mathcal{I}_{\mathcal{P}} \to \mathbb{N}$ tal que:

$\text{tiempo}_{\mathcal{A}}(w)$: número de pasos realizados por $\mathcal{A}$ con entrada $w \in \mathcal{I}_{\mathcal{P}}$

Tiempo de ejecución de un algoritmo

Tiempo de Ejecución

Sea $\mathcal{A}$ un algoritmo que resuelve un problema abstracto $\mathcal{P}$, y sea $I \in \mathcal{I}_{\mathcal{P}}$ una instancia particular de $\mathcal{P}$. El *tiempo de ejecución* del algoritmo $\mathcal{A}$ para la instancia I, denotado $\text{tiempo}_{\mathcal{A}}(I)$, es la cantidad de instrucciones que ejecuta $\mathcal{A}$ para resolver I.

El tiempo de ejecución es clave para comparar algoritmos que resuelven un mismo problema

Medir el tiempo en base a la cantidad de operaciones elementales permite independizarnos de las implementaciones particulares de un algoritmo

Tiempo de ejecución de un algoritmo

Algoritmo 1: MÁXIMO($S[1..n]$)

```
1 m ← 1
2 for i ← 2 to n do
3 |   if S[m] ≤ S[i] then
4 |   |   m ← i
5 |   end
6 end
7 return m
```

Este algoritmo realiza $n - 1$ comparaciones entre elementos de S para encontrar el máximo

Análisis de Mejor y Peor Caso

Es común que el tiempo de ejecución de un algoritmo dependa de la instancia que se está resolviendo

Por ejemplo:

Algoritmo 2: PERTENENCIA(x, $S[1..n]$)

Entrada: Un conjunto no necesariamente ordenado representado por un arreglo $S[1..n]$, y un elemento x.
Salida: la posición i tal que $S[i] = x$, o $n+1$ si $x \notin S$.

```
1 for i ← 1 to n do
2 |   if x = S[i] then
3 |   |   return i
4 |   end
5 end
6 return n + 1                                  // Indica que x ∉ S
```

Tiempo de ejecución de un algoritmo en el mejor caso

Tiempo de Ejecución de Mejor Caso

Sea $\mathcal{A}$ un algoritmo que resuelve un problema abstracto $\mathcal{P}$. El tiempo de ejecución de mejor caso para $\mathcal{A}$ sobre todas las instancias de tamaño n se define como:

$$t_{\mathcal{A}}(n) \equiv \text{mín}\,\{\text{tiempo}_{\mathcal{A}}(I) \mid I \in \mathcal{I}_{\mathcal{P}},\ |I| = n\}.$$

- El mejor caso del algoritmo anterior consiste en buscar el elemento $x = S[1]$ (ejecuta una cantidad constante de instrucciones)
- Este tipo de análisis es optimista y no suele aportar demasiada información
- Al analizar algoritmos, siempre es conveniente ser "relativamente" pesimista

Tiempo de ejecución de un algoritmo en el peor caso

Tiempo de Ejecución de Peor Caso

Sea $\mathcal{A}$ un algoritmo que resuelve el problema $\mathcal{P}$. El tiempo de ejecución de peor caso para $\mathcal{A}$ sobre todas las instancias de tamaño n se define como:

$$T_{\mathcal{A}}(n) \equiv \text{máx}\{\text{tiempo}_{\mathcal{A}}(I) \mid I \in \mathcal{I}_{\mathcal{P}},\ |I| = n\}.$$

- El peor caso del algoritmo anterior es buscar $x = S[n]$ (o $x \notin S$), ejecutando n ciclos
- Este tipo de análisis es pesimista
  - Previene al usuario sobre el peor escenario y permite tomar recaudos
  - Puede penalizar demasiado a ciertos algoritmos con peores casos cuya probabilidad de ocurrencia es baja

Notaciones asintóticas

Es importante distinguir la complejidad de mejor y peor caso de un algoritmo

Sin embargo, a veces es preferible denotar el tiempo de ejecución de un algoritmo de tal manera que se incluyan todas las instancias del problema

Las notaciones asintóticas permiten **acotar** el tiempo de ejecución de todas las instancias

Permiten clasificar funciones que modelan la complejidad de un algoritmo para compararlas

Estudiamos a continuación el concepto de cota asintótica

Notación asintótica

En muchos casos, nos interesa conocer el *orden* de un algoritmo en lugar de su complejidad exacta.

- Queremos decir que un algoritmo es lineal o cuadrático, en lugar de decir que su complejidad es $3n^2 + 17n + 22$

Vamos a desarrollar notación para hablar del orden de un algoritmo.

Notación asintótica

Supuesto

La complejidad de un algoritmo va a ser medida en términos de funciones de la forma $f : \mathbb{N} \to \mathbb{R}_0^+$, donde $\mathbb{R}^+ = \{r \in \mathbb{R} \mid r > 0\}$ y $\mathbb{R}_0^+ = \mathbb{R}^+ \cup \{0\}$

Estas funciones incluyen a las funciones definidas en las transparencias anteriores, y también sirven para modelar el tiempo de ejecución de un algoritmo

La notación $O(f)$

Sea $f : \mathbb{N} \to \mathbb{R}_0^+$

Definición

$$O(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(g(n) \leq c \cdot f(n))\}$$

Decimos entonces que $g \in O(f)$

- También usamos la notación g es $O(f)$, lo cual es formalizado como $g \in O(f)$
- $O(f)$ son todas las funciones g que crecen más lentamente o igual que f
- Para algoritmo $\mathcal{A}$ se tiene que el tiempo de ejecución de cualquier instancia es $O(T_{\mathcal{A}}(n))$

Ejercicio

Demuestre que $3n^2 + 17n + 22 \in O(n^2)$

Las notaciones $\Omega(f)$ y $\Theta(f)$

Definición

$$\Omega(f) = \{g : \mathbb{N} \to \mathbb{R}_0^+ \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)\,(c \cdot f(n) \leq g(n))\}$$

$$\Theta(f) = O(f) \cap \Omega(f)$$

- Si para un algoritmo $\mathcal{A}$ se tiene que $t_{\mathcal{A}}(n) \in \Theta(T_{\mathcal{A}}(n))$, entonces el tiempo de ejecución de cualquier instancia es $\Theta(T_{\mathcal{A}}(n))$
- Siempre que se pueda, usar Θ en lugar de O para acotar el tiempo de ejecución de un algoritmo

Ejercicios

1. Demuestre que $3n^2 + 17n + 22 \in \Theta(n^2)$
2. Demuestre que $g \in \Theta(f)$ si y sólo si existen $c, d \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tal que para todo $n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$

Las notaciones $O(f)$, $\Omega(f)$ y $\Theta(f)$

Ejemplo:

$$O(n) = \{1, 2, \ldots, \lg\lg n, \ldots, \lg n, \ldots, \sqrt{n}, n^{1/3}, \ldots, \underbrace{n, n+1, 2n, \ldots}_{\Theta(n)}\}$$

$$\Omega(n) = \{\underbrace{n, n+1, 2n, \ldots}_{\Theta(n)}, n\lg n, \ldots, n\sqrt{n}, \ldots, n^2, \ldots, 2^n, \ldots, n!, n^n, \ldots\}$$

Ejercicios

1. Sea $p(n)$ un polinomio de grado $k \geq 0$ con coeficientes en los números enteros. Demuestre que $p(n) \in O(n^k)$.

2. ¿Cuáles de las siguientes afirmaciones son ciertas?
   - $n^2 \in O(n)$
   - Si $f(n) \in O(n)$, entonces $f(n)^2 \in O(n^2)$
   - Si $f(n) \in O(n)$, entonces $2^{f(n)} \in O(2^n)$

3. Suponga que $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ existe y es igual a ℓ. Demuestre lo siguiente:
   - Si $\ell = 0$, entonces $f \in O(g)$ y $g \notin O(f)$
   - Si $\ell = \infty$, entonces $g \in O(f)$ y $f \notin O(g)$
   - Si $\ell \in \mathbb{R}^+$, entonces $f \in \Theta(g)$

4. Encuentre funciones f y g tales que $f \in \Theta(g)$ y $\lim_{n\to\infty} \frac{f(n)}{g(n)}$ no existe

Ejercicios

Para el siguiente algoritmo clásico para ordenar una lista L (de menor a mayor, asumiendo una relación de orden total sobre los elementos de L).

InsertionSort($L[1 \ldots n]$: lista de elementos)
    for $i := 2$ **to** n **do**
        $j := i - 1$
        while $j \geq 1$ **and** $L[j] > L[j+1]$ **do**
            $aux := L[j]$
            $L[j] := L[j+1]$
            $L[j+1] := aux$
            $j := j - 1$
    return L

1. Encuentre el tiempo de ejecución de mejor y peor caso
2. Acote asintóticamente el tiempo de ejecución para todas las posibles entradas de n elementos que puede recibir el algoritmo (use la notación adecuada)