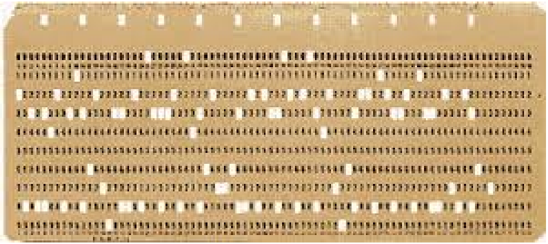
Análisis de Algoritmos

Juan Manuel Rabasedas



Análisis de Algoritmos

El "Análisis de Algoritmos" determinar el consumo de recursos: energía, ancho de banda, tiempo de computo, espacio de memoria o almacenamiento.

- Preciso para poder comparar.
- Seleccionar el más eficiente.
- Abstracto para no depender de detalles que cambian todo el tiempo. compiladores, arquitecturas.
- Análisis Asintótica y modelo de costo.
- Corrección de los algoritmos

Análisis Asintótico

- Nos interesa saber el orden de crecimiento de los recursos que consumen un algoritmo.
- Necesitamos analizarlos instancias grandes de su entrada para evidenciarlo.
- Análisis Asintótico de su eficiencia.
- Obtenemos una ecuación en términos de la entrada.

Analicemos una algoritmo A y otro B donde ambos resuelven el problema P en términos de su entrada n

- $W_A(n) = 2\ n\ log(n) + 3\ n + 4\ log(n) + 5$
- $W_B(n) = 6\ n + 7\ log^2(n) + 8\ log(n) + 9$
- ¿Qué algoritmo preferimos?

La eficiencia de un algoritmo depende principalmente del término dominante en la expresión de su complejidad

Análisis Asintótico

- Nos interesa saber el orden de crecimiento de los recursos que consumen un algoritmo.
- Necesitamos analizarlos instancias grandes de su entrada para evidenciarlo.
- Análisis Asintótico de su eficiencia.
- Obtenemos una ecuación en términos de la entrada.

Analicemos una algoritmo A y otro B donde ambos resuelven el problema P en términos de su entrada n

- $W_A(n) = \Theta(n\ log(n))$
- $W_B(n) = \Theta(n)$
- ¿Qué algoritmo preferimos?

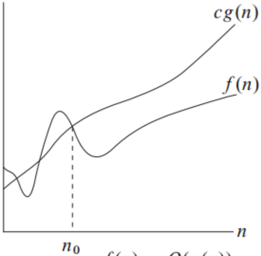
La eficiencia de un algoritmo depende principalmente del término dominante en la expresión de su complejidad

Hay varias clases de funciones que capturan distintas propiedades:

Definición (O)

Sean $f, g : \mathbb{N} \to \mathbb{R}$. *Decimos que* f *tiene orden de crecimiento* $O(g)$ *(y escribimos* $f \in O(g)$*), si existen constantes* $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$, *tales que:*

$$\forall\, n \geqslant n_0 \quad \cdot \quad 0 \leqslant f(n) \leqslant c\, g(n)$$



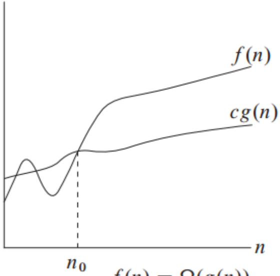
$f(n) = O(g(n))$

Notación Asintótica

Definición (Ω)

Sean $f, g : \mathbb{N} \to \mathbb{R}$. *Decimos que* f *tiene orden de crecimiento* $\Omega(g)$ *(y escribimos* $f \in \Omega(g)$*), si existen constantes* $c \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$, *tales que:*

$$\forall\, n \geqslant n_0 \quad \cdot \quad 0 \leqslant c\, g(n) \leqslant f(n)$$

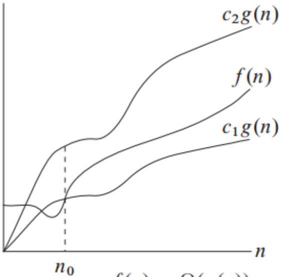


$f(n) = \Omega(g(n))$

Definición (Θ)

Sean $f, g : \mathbb{N} \to \mathbb{R}$. *Decimos que* f *tiene orden de crecimiento* $\Theta(g)$ *(y escribimos* $f \in \Theta(g)$*), si existen constantes* $c_1 c_2 \in \mathbb{R}^+$, $n_0 \in \mathbb{N}$, *tales que:*

$$\forall\, n \geqslant n_0 \quad \cdot \quad 0 \leqslant c_1\, g(n) \leqslant f(n) \leqslant c_2\, g(n)$$



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

Modelo de Costo

Modelos basados en:

Máquinas: considera el costo de las instrucciones ejecutadas por el algoritmo en la máquina subyacente.

Lenguajes: considera una función de costo sobre el lenguaje de implementación del algoritmo.

- Ambos ignoran factores constantes que dependan del hardware.
- Los modelos basados en máquinas son más precisos,
- pero su análisis es complejo y poco expresivo.
- Los modelos basados en lenguajes son más simples,
- pero menos precisos en algunos casos.
- Permite analizar algoritmos paralelos más fácilmente.

Modelo Basado en Lenguajes

Vamos a usar un modelo de costo basado en 2 métricas:

- **Work** número total de operaciones realizadas.
- Costo secuencial.
- **Span** la cadena más larga de dependencias en el computo.
- Costo paralelo.
- Notaremos con $\mathsf{W}\,(e)$ para el Work de una expresion e y $\mathsf{S}\,(e)$ para su Span.

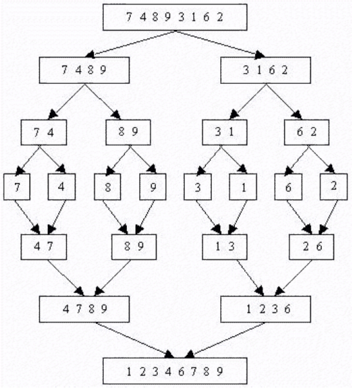
Work

$$
\begin{aligned}
&\mathsf{W}\ (c) && = k_c \\
&\mathsf{W}\ (e_1\ op\ e_2) && = k_{op} + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_2) \\
&\mathsf{W}\ (e_1, e_2) && = k \quad + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_2) \\
&\mathsf{W}\ (e_1 || e_2) && = k \quad + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_2) \\
&\mathsf{W}\ (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) && = k \quad + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_2)\ \mathsf{Eval}\ (e_1) = \mathsf{True} \\
&\mathsf{W}\ (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) && = k \quad + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_3)\ \mathsf{Eval}\ (e_1) = \mathsf{False} \\
&\mathsf{W}\ (\textbf{let}\ x = e_1\ \textbf{in}\ e_2) && = k \quad + \mathsf{W}\ (e_1) + \mathsf{W}\ (e_2\ [\mathsf{Eval}\ (e_1)\ /\ x]) \\
&\mathsf{W}\ (\{f\ (x) : x\ \in\ \mathsf{A}\}) && = k \quad + \Sigma_{x\,\in\,\mathsf{A}}\ \mathsf{W}\ (f\ (x))
\end{aligned}
$$

Span

$$
\begin{aligned}
&\mathsf{S}\ (c) &&= k_c \\
&\mathsf{S}\ (e_1\ op\ e_2) &&= k_{op} + \mathsf{S}\ (e_1) + \mathsf{S}\ (e_2) \\
&\mathsf{S}\ (e_1, e_2) &&= k \quad + \mathsf{S}\ (e_1) + \mathsf{S}\ (e_2) \\
&\mathsf{S}\ (e_1 || e_2) &&= k \quad + max\ \mathsf{S}\ (e_1)\ \mathsf{S}\ (e_2) \\
&\mathsf{S}\ (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) &&= k \quad + \mathsf{S}\ (e_1) + \mathsf{S}\ (e_2)\ \mathsf{Eval}\ (e_1) = \mathsf{True} \\
&\mathsf{S}\ (\textbf{if}\ e_1\ \textbf{then}\ e_2\ \textbf{else}\ e_3) &&= k \quad + \mathsf{S}\ (e_1) + \mathsf{S}\ (e_3)\ \mathsf{Eval}\ (e_1) = \mathsf{False} \\
&\mathsf{S}\ (\textbf{let}\ x = e_1\ \textbf{in}\ e_2) &&= k \quad + \mathsf{S}\ (e_1) + \mathsf{S}\ (e_2\ [\mathsf{Eval}\ (e_1)\ /\ x]) \\
&\mathsf{S}\ (\{f\ (x) : x\ \in\ \mathsf{A}\}) &&= k \quad + max_{x\ \in\ \mathsf{A}}\ \mathsf{S}\ (f\ (x))
\end{aligned}
$$

Ejemplo Merge Sort



- Dividimos el problema hasta que sea trivial resolverlo.
- Resolvemos el problema recursivamente.
- Combinamos las soluciones en una solución del problema inicial.

Ejemplo Merge Sort

$$
\begin{aligned}
&msort\ [\,] &&= [\,] \\
&msort\ [x] &&= [x] \\
&msort\ xs &&= \textbf{let}\ (ls, rs) \;\;= split\ xs \\
& && \qquad\;\; (ls', rs') = (msort\ ls || msort\ rs) \\
& && \quad\;\, \textbf{in}\ merge\ (ls', rs')
\end{aligned}
$$

$$
\begin{aligned}
&split\ [\,] &&= ([\,], [\,]) \\
&split\ [x] &&= ([x], [\,]) \\
&split\ (x : y : zs) &&= \textbf{let}\ (xs, ys) = split\ zs \\
& && \quad\;\, \textbf{in}\ (x : xs, y : ys)
\end{aligned}
$$

$$
\begin{aligned}
&merge\ ([\,], ys) = ys \\
&merge\ (xs, [\,]) = xs \\
&merge\ (x : xs, y : ys) = \textbf{if}\ x \leqslant y\ \textbf{then}\ x : merge\ (xs, y : ys) \\
&\qquad\qquad\qquad\qquad\quad\;\; \textbf{else}\ y : merge\ (x : xs, ys)
\end{aligned}
$$

Merge Sort **Work**

$$
\begin{aligned}
&msort\ [\,] \quad = [\,] \\
&msort\ [x] = [x] \\
&msort\ xs \ \ = \textbf{let}\ (ls, rs) \ \ = split\ xs \\
&\qquad\qquad\qquad\quad (ls', rs') = (msort\ ls || msort\ rs) \\
&\qquad\qquad\quad\ \textbf{in}\ merge\ (ls', rs')
\end{aligned}
$$

$$
\begin{aligned}
&W_{msort}(0) = k_0 \\
&W_{msort}(1) = k_1 \\
&W_{msort}(n) = W_{split}(n) + 2\ W_{msort}(\tfrac{n}{2}) + W_{merge}(n) + k_2 \text{ si } n > 1
\end{aligned}
$$

Merge Sort **Work**

$$
\begin{aligned}
&split\ [\,] &&= ([\,],[\,])\\
&split\ [x] &&= ([x],[\,])\\
&split\ (x:y:zs) &&= \textbf{let}\ (xs,ys) = split\ zs\\
& && \quad\ \textbf{in}\ (x:xs, y:ys)
\end{aligned}
$$

$W_{split}(0) = k_3$
$W_{split}(1) = k_4$
$W_{split}(n) = W_{split}(n-2) + k_5$ si $n > 1$

$W_{split} \in O(n)$

Merge Sort **Work**

$$
\begin{aligned}
&merge\ ([\,], ys) &&= ys\\
&merge\ (xs, [\,]) &&= xs\\
&merge\ (x : xs, y : ys) &&= \textbf{if } x \leqslant y \textbf{ then } x : merge\ (xs, y : ys)\\
& && \quad \textbf{else } y : merge\ (x : xs, ys)
\end{aligned}
$$

$W_{merge}(0) = k_6$
$W_{merge}(n) = W_{merge}(n-1) + k_7$

n es la suma de las longitudes de las listas $(x : xs)$ y $(y : ys)$

$W_{merge} \in O(n)$

Merge Sort **Work**

$W_{msort}(0) = k_0$ $W_{msort}(1) = k_1$
$W_{msort}(n) = W_{split}(n) + 2\ W_{msort}(\frac{n}{2}) + W_{merge}(n) + k_2$ si $n > 1$

$W_{msort}(n) = 2\ W_{msort}(\frac{n}{2}) + k'\ n$ si $n > 1$

$W_{msort} \in O(n\ log(n))$

Merge Sort **Span**

$$
\begin{aligned}
&msort\ [\,] &&= [\,]\\
&msort\ [x] &&= [x]\\
&msort\ xs &&= \textbf{let}\ (ls, rs) \ \ = split\ xs\\
& && \qquad\quad (ls', rs') = (msort\ ls || msort\ rs)\\
& && \quad\ \textbf{in}\ merge\ (ls', rs')
\end{aligned}
$$

$S_{msort}(0) = k_0$
$S_{msort}(1) = k_1$
$S_{msort}(n) = S_{split}(n) + S_{msort}(\frac{n}{2}) + S_{merge}(n) + k_2$ si $n > 1$

Merge Sort **Span**

$$
\begin{aligned}
&split\ [\,] &&= ([\,],[\,])\\
&split\ [x] &&= ([x],[\,])\\
&split\ (x:y:zs) &&= \textsf{\textbf{let}}\ (xs,ys) = split\ zs\\
& && \quad\ \textsf{\textbf{in}}\ (x:xs, y:ys)
\end{aligned}
$$

$$
\begin{aligned}
&S_{split}(0) = k_3\\
&S_{split}(1) = k_4\\
&S_{split}(n) = S_{split}(n-2) + k_5 \text{ si } n > 1
\end{aligned}
$$

$S_{split} \in S(n)$

Merge Sort **Span**

$$
\begin{aligned}
&merge\ ([\,], ys) = ys \\
&merge\ (xs, [\,]) = xs \\
&merge\ (x : xs, y : ys) = \textbf{if}\ x \leqslant y\ \textbf{then}\ x : merge\ (xs, y : ys) \\
&\qquad\qquad\qquad\qquad\qquad\ \textbf{else}\ y : merge\ (x : xs, ys)
\end{aligned}
$$

$$
\begin{aligned}
&S_{merge}(0) = k_6 \\
&S_{merge}(n) = S_{merge}(n-1) + k_7
\end{aligned}
$$

$$S_{merge} \in O(n)$$

Merge Sort **Span**

$S_{msort}(0) = k_0$
$S_{msort}(1) = k_1$
$S_{msort}(n) = S_{split}(n) + S_{msort}(\frac{n}{2}) + S_{merge}(n) + k_2$ si $n > 1$

$S_{msort}(n) = S_{msort}(\frac{n}{2}) + k'\ n$ si $n > 1$

$S_{msort} \in S(n)$

Referencias

- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliford Stein
- "Algorithms: Parallel and Sequential" by Umut A. Acar and Guy E. Blelloch.
- Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. Communications of the ACM, $31(9) : 1116 - 1127$, September 1988.
- Alfred V. Aho, J. E. Hopcroft, and Jeffrey D. Ullman. The Design and Analysis of Computer Algorithms. Addison-Wesley, 1974