

Análisis de la eficiencia de un algoritmo

IIC2283

Decisión versus computación

En el capítulo anterior definimos Máquinas de Turing que aceptan lenguajes.

- ▶ Una MT M con alfabeto de entrada Σ define el lenguaje $L(M) = \{w \in \Sigma^* \mid M \text{ acepta } w\}$

Este tipo de Máquinas de Turing sirven para decidir la pertenencia de un elemento a un lenguaje.

- ▶ Hablamos entonces de **problemas de decisión**

Pero también nos interesa computar valores.

- ▶ Ordenar una lista, multiplicar dos números, ...

Decisión versus computación

En un **problema de computación** queremos un algoritmo para calcular una función $f : \Sigma^* \rightarrow \Sigma^*$

- ▶ ¿Cómo puede ser representado el problema de ordenar una lista de enteros como el problema de calcular una función? ¿Qué alfabeto usaría?

Podemos modificar las Máquinas de Turing para que calculen funciones.

Ejercicio

Indique cómo hacer esta modificación para que una MT M calcule una función $f : \Sigma^* \rightarrow \Sigma^*$

- ▶ Sólo se debe llegar a una convención de como representar la salida, para lo cual puede ser útil tener más de una cinta.
- ▶ ¿Son útiles los estados finales en este caso?

Unificando los modelos

Vamos a pensar en un algoritmo $\mathcal{A}$ como una función $\mathcal{A} : \Sigma^* \rightarrow \Sigma^*$

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

- ▶ ¿Cómo se representa un problema de decisión?

Pero hay que tener cuidado porque el modelo es demasiado general.

- ▶ Hay funciones $\mathcal{A} : \Sigma^* \rightarrow \Sigma^*$ que no pueden ser calculadas por una MT. ¿Por qué?

La noción de algoritmo en este curso

Supuesto

Vamos a considerar funciones $\mathcal{A} : \Sigma^* \rightarrow \Sigma^*$ para las cuales existen Máquinas de Turing que las calculan.

Pero no estamos interesados en construir estas Máquinas de Turing, vamos a calcular estas funciones en otros modelos de computación.

Tiempo de ejecución de un algoritmo

A cada algoritmo $\mathcal{A}$ asociamos una función $\text{tiempo}_{\mathcal{A}} : \Sigma^* \rightarrow \mathbb{N}$ tal que:

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

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

Tiempo de ejecución de un algoritmo en el peor caso

El primer tipo de análisis que vamos a realizar de la complejidad de un algoritmo va a estar basado en el peor caso.

Para cada algoritmo $\mathcal{A}$ asociamos una función $t_{\mathcal{A}} : \mathbb{N} \rightarrow \mathbb{N}$ tal que

$$t_{\mathcal{A}}(n) = \text{máx}\{\text{tiempo}_{\mathcal{A}}(w) \mid w \in \Sigma^* \text{ y } |w| = n\},$$

donde $|w|$ es el largo de w

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 $f : \mathbb{N} \rightarrow \mathbb{R}_0^+$, 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 $f : \mathbb{N} \rightarrow \mathbb{R}_0^+$

Definición

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

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

- También usamos la notación g es $O(f)$, lo cual es formalizado como $g \in O(f)$

Ejercicio

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

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

Definición

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

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

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 $n \geq n_0$: $c \cdot f(n) \leq g(n) \leq d \cdot f(n)$

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 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ 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 $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)}$ no existe

Búsqueda binaria

Suponga que tiene una lista ordenada (de menor a mayor)
 $L[1 \dots n]$ de números enteros con $n \geq 1$

¿Cómo podemos verificar si un número a está en L ?

Búsqueda binaria

```
BúsquedaBinaria( $a, L, i, j$ )  
  if  $i > j$  then return no  
  else if  $i = j$  then  
    if  $L[i] = a$  then return  $i$   
    else return no  
  else  
     $p := \lfloor \frac{i+j}{2} \rfloor$   
    if  $L[p] < a$  then return BúsquedaBinaria( $a, L, p + 1, j$ )  
    else if  $L[p] > a$  then return BúsquedaBinaria( $a, L, i, p - 1$ )  
    else return  $p$ 
```

Llamada inicial al algoritmo: **BúsquedaBinaria**($a, L, 1, n$)

Tiempo de ejecución de búsqueda binaria

¿Cuál es la complejidad del algoritmo?

- ▶ ¿Qué operaciones vamos a considerar?
- ▶ ¿Cuál es el peor caso?

Si contamos sólo las comparaciones, entonces la siguiente expresión define la complejidad del algoritmo:

$$T(n) = \begin{cases} c & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + d & n > 1 \end{cases}$$

donde $c \in \mathbb{N}$ y $d \in \mathbb{N}$ son constantes tales que $c \geq 1$ y $d \geq 1$

Esta es una **ecuación de recurrencia**

Solucionando una ecuación de recurrencia

¿Cómo podemos solucionar una ecuación de recurrencia?

- ▶ Técnica básica: sustitución de variables

Para la ecuación anterior usamos la sustitución $n = 2^k$

- ▶ Vamos a resolver la ecuación suponiendo que n es una potencia de 2
- ▶ Vamos a utilizar inducción para demostrar que la solución obtenida nos da el orden del algoritmo

Ecuaciones de recurrencia: sustitución de variables

Si realizamos la sustitución $n = 2^k$ en la ecuación:

$$T(n) = \begin{cases} c & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + d & n > 1 \end{cases}$$

obtenemos:

$$T(2^k) = \begin{cases} c & k = 0 \\ T(2^{k-1}) + d & k > 0 \end{cases}$$

Ecuaciones de recurrencia: sustitución de variables

Extendiendo la expresión anterior obtenemos:

$$\begin{aligned}T(2^k) &= T(2^{k-1}) + d \\&= (T(2^{k-2}) + d) + d \\&= T(2^{k-2}) + 2d \\&= (T(2^{k-3}) + d) + 2d \\&= T(2^{k-3}) + 3d \\&= \dots\end{aligned}$$

Deducimos la expresión general para $k - i \geq 0$:

$$T(2^k) = T(2^{k-i}) + i \cdot d$$

Ecuaciones de recurrencia: sustitución de variables

Considerando $i = k$ obtenemos:

$$\begin{aligned} T(2^k) &= T(1) + k \cdot d \\ &= c + k \cdot d \end{aligned}$$

Dado que $k = \log_2(n)$, obtenemos que $T(n) = c + d \cdot \log_2(n)$ para n potencia de 2

Usando inducción vamos a extender esta solución y vamos a demostrar que $T(n) \in O(\log_2(n))$

Inducción constructiva

Sea $T(n)$ definida como:

$$T(n) = \begin{cases} c & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + d & n > 1 \end{cases}$$

Queremos demostrar que $T(n) \in O(\log_2(n))$

- Vale decir, queremos demostrar que existen $e \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $T(n) \leq e \cdot \log_2(n)$ para todo $n \geq n_0$

Inducción nos va servir tanto para demostrar la propiedad y como para determinar valores adecuados para e y n_0

- Por esto usamos el término **inducción constructiva**

Inducción constructiva

Dado que $T(1) = c$ y $\log_2(1) = 0$ no es posible encontrar un valor para e tal que $T(1) \leq e \cdot \log_2(1)$

Dado que $T(2) = (c + d)$, si consideramos $e = (c + d)$ tenemos que $T(2) \leq e \cdot \log_2(2)$

► Definimos entonces $e = (c + d)$ y $n_0 = 2$

Tenemos entonces que demostrar lo siguiente:

$$(\forall n \geq 2)(T(n) \leq e \cdot \log_2(n))$$

Inducción constructiva y fuerte

¿Cuál es el principio de inducción adecuado para el problema anterior?

- ▶ Tenemos n_0 como punto de partida
- ▶ n_0 es un caso base, pero podemos tener otros
- ▶ Dado $n > n_0$ tal que n no es un caso base, suponemos que la propiedad se cumple para todo $k \in \{n_0, \dots, n - 1\}$

Inducción constructiva y fuerte

Queremos demostrar que $(\forall n \geq 2) (T(n) \leq e \cdot \log_2(n))$

- ▶ 2 es el punto de partida y el primer caso base
- ▶ También 3 es un caso base ya que $T(3) = T(1) + d$ y para $T(1)$ no se cumple la propiedad
- ▶ Para $n \geq 4$ tenemos que $T(n) = T(\lfloor \frac{n}{2} \rfloor) + d$ y $\lfloor \frac{n}{2} \rfloor \geq 2$, por lo que resolvemos este caso de manera inductiva
 - ▶ Suponemos que la propiedad se cumple para todo $k \in \{2, \dots, n-1\}$

La demostración por inducción fuerte

Casos base:

$$\begin{aligned} T(2) &= c + d = e \cdot \log_2(2) \\ T(3) &= c + d < e \cdot \log_2(3) \end{aligned}$$

Caso inductivo:

Suponemos que $n \geq 4$ y para todo $k \in \{2, \dots, n-1\}$ se tiene que $T(k) \leq e \cdot \log_2(k)$

La demostración por inducción fuerte

Usando la definición de $T(n)$ y la hipótesis de inducción concluimos que:

$$\begin{aligned} T(n) &= T(\lfloor \frac{n}{2} \rfloor) + d \\ &\leq e \cdot \log_2(\lfloor \frac{n}{2} \rfloor) + d \\ &\leq e \cdot \log_2(\frac{n}{2}) + d \\ &= e \cdot \log_2(n) - e \cdot \log_2(2) + d \\ &= e \cdot \log_2(n) - (c + d) + d \\ &= e \cdot \log_2(n) - c \\ &< e \cdot \log_2(n) \end{aligned}$$

Un segundo ejemplo de inducción constructiva

Considere la siguiente ecuación de recurrencia:

$$T(n) = \begin{cases} 0 & n = 0 \\ n^2 + n \cdot T(n-1) & n > 0 \end{cases}$$

Queremos determinar una función $f(n)$ para la cual se tiene que $T(n) \in O(f(n))$

- ¿Alguna conjetura sobre quién podría ser $f(n)$?

Una posible solución para la ecuación de recurrencia

Dada la forma de la ecuación de recurrencia, podríamos intentar primero con $f(n) = n!$

Tenemos entonces que determinar $c \in \mathbb{R}^+$ y $n_0 \in \mathbb{N}$ tales que $T(n) \leq c \cdot n!$ para todo $n \geq n_0$

- Pero nos vamos a encontrar con un problema al tratar de usar la hipótesis de inducción

Una posible solución para la ecuación de recurrencia

Supongamos que la propiedad se cumple para n :

$$T(n) \leq c \cdot n!$$

Tenemos que:

$$\begin{aligned} T(n+1) &= (n+1)^2 + (n+1) \cdot T(n) \\ &\leq (n+1)^2 + (n+1) \cdot (c \cdot n!) \\ &= (n+1)^2 + c \cdot (n+1)! \end{aligned}$$

Pero no existe una constante c para la cual $(n+1)^2 + c \cdot (n+1)! \leq c \cdot (n+1)!$

► Dado que $n \in \mathbb{N}$

¿Cómo solucionamos el problema con la demostración?

Una demostración por inducción puede hacerse más simple considerando una propiedad más fuerte.

- Dado que la hipótesis de inducción se va a volver más fuerte

Vamos a seguir tratando de demostrar que $T(n) \in O(n!)$ pero ahora considerando una propiedad más fuerte.

Vamos a demostrar lo siguiente:

$$(\exists c \in \mathbb{R}^+)(\exists d \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)(T(n) \leq c \cdot n! - d \cdot n)$$

Inducción constructiva sobre una propiedad más fuerte

Para tener una mejor idea de los posibles valores para c , d y n_0 vamos a considerar primero el paso inductivo en la demostración.

Supongamos que la propiedad se cumple para n :

$$T(n) \leq c \cdot n! - d \cdot n$$

Tenemos que:

$$\begin{aligned} T(n+1) &= (n+1)^2 + (n+1) \cdot T(n) \\ &\leq (n+1)^2 + (n+1) \cdot (c \cdot n! - d \cdot n) \\ &= c \cdot (n+1)! + (n+1)^2 - d \cdot n \cdot (n+1) \\ &= c \cdot (n+1)! + ((n+1) - d \cdot n) \cdot (n+1) \end{aligned}$$

Inducción constructiva sobre una propiedad más fuerte

Para poder demostrar que la propiedad se cumple para $n + 1$ necesitamos que lo siguiente sea cierto:

$$(n + 1) - d \cdot n \leq -d$$

De lo cual concluimos la siguiente restricción para d :

$$\frac{(n + 1)}{(n - 1)} \leq d$$

Si consideramos $n \geq 2$ concluimos que $d \geq 3$

► Consideramos entonces $n_0 = 2$ y $d = 3$

Inducción constructiva sobre una propiedad más fuerte

Para concluir la demostración debemos considerar el caso base $n_0 = 2$

Tenemos que:

$$\begin{aligned} T(0) &= 0 \\ T(1) &= 1^2 + 1 \cdot T(0) = 1 \\ T(2) &= 2^2 + 2 \cdot T(1) = 6 \end{aligned}$$

Entonces se debe cumplir que $T(2) \leq c \cdot 2! - 3 \cdot 2$, vale decir,

$$6 \leq c \cdot 2 - 6$$

Concluimos que $c \geq 6$, por lo que consideramos $c = 6$

- Tenemos entonces que $(\forall n \geq 2)(T(n) \leq 6 \cdot n! - 3 \cdot n)$, de lo cual concluimos que $T(n) \in O(n!)$