Optimización III (525551)

Módulo I: Tiempo de ejecución de algoritmos

Pr. Julio Aracena.

Departamento de Ingeniería Matemática
Facultad de Ciencias Físicas y Matemáticas
Universidad de Concepción

Primer semestre, 2023

Algoritmo

Definición: Intuitivamente un algoritmo es un procedimiento de cómputo que recibe una entrada y que después de un número finito de operaciones elementales, bien definidas, produce una salida. Formalmente un algoritmo puede ser definido como una máquina de Turing decidora (i.e. que siempre se detiene).

Algoritmo

Definición: Intuitivamente un algoritmo es un procedimiento de cómputo que recibe una entrada y que después de un número finito de operaciones elementales, bien definidas, produce una salida. Formalmente un algoritmo puede ser definido como una máquina de Turing decidora (i.e. que siempre se detiene).

Observación: La definición de algoritmo tiene implícito que el procedimiento siempre termina.

Algoritmo

Definición: Intuitivamente un algoritmo es un procedimiento de cómputo que recibe una entrada y que después de un número finito de operaciones elementales, bien definidas, produce una salida. Formalmente un algoritmo puede ser definido como una máquina de Turing decidora (i.e. que siempre se detiene).

Observación: La definición de algoritmo tiene implícito que el procedimiento siempre termina.

Ejemplo: A1 es un algoritmo y A2 no es algoritmo pues no termina.

---

procedure A1
Input: $x \in \mathbb{N}$
$x \leftarrow x + 1$
return x
end procedure

---

procedure A2
Input: $x \in \mathbb{N}$
while $x \in \mathbb{N}$ **do**
$x \leftarrow x + 1$
end while
return x
end procedure

Tamaño de un objeto matemático

Definición: Se el define el tamaño de un objeto matemático x como el número de bits necesario para codificarlo, según una cierta representación, y que denotaremos por $size(x)$.

Tamaño de un objeto matemático

Definición: Se el define el tamaño de un objeto matemático x como el número de bits necesario para codificarlo, según una cierta representación, y que denotaremos por $size(x)$.

Ejemplo:

1. Si $x \in \mathbb{N}$, entonces $size(x) = \log_2(x)$.
2. Si $x = \frac{p}{q} \in \mathbb{Q}$, entonces $size(x) = \log_2(p) + \log_2(q)$.
3. Si $x = A \in M_{m,n}(\mathbb{R})$, entonces $size(A) = m \times n \times C = O(mn)$,
4. Si x es un grafo (dirigido o no dirigido) $G = (V, E)$, entonces $size(G) = |V| + |E| = O(|V|^2)$.

Tamaño de un objeto matemático

Definición: Se el define el tamaño de un objeto matemático x como el número de bits necesario para codificarlo, según una cierta representación, y que denotaremos por $size(x)$.

Ejemplo:

1. Si $x \in \mathbb{N}$, entonces $size(x) = \log_2(x)$.
2. Si $x = \frac{p}{q} \in \mathbb{Q}$, entonces $size(x) = \log_2(p) + \log_2(q)$.
3. Si $x = A \in M_{m,n}(\mathbb{R})$, entonces $size(A) = m \times n \times C = O(mn)$,
4. Si x es un grafo (dirigido o no dirigido) $G = (V, E)$, entonces $size(G) = |V| + |E| = O(|V|^2)$.

Observaciones:

1. El valor C en $size(A)$ es una constante que no depende de las dimensiones de la matriz A y corresponde a un número fijo de bits usado para codificar sus valores.
2. El tamaño de un grafo (dirigido o no dirigido) G depende de cómo se representa (Para mayor información ver ejemplo 8.5 del libro de **Papadimitriou**).

Tiempo de ejecución de un algoritmo

La complejidad de un algoritmo se puede medir de distintas formas: número de operaciones elementales, memoria usada, en el caso promedio, peor caso, etc. En este curso trabajaremos con el **tiempo de ejecución en el peor caso**.

Tiempo de ejecución de un algoritmo

La complejidad de un algoritmo se puede medir de distintas formas: número de operaciones elementales, memoria usada, en el caso promedio, peor caso, etc. En este curso trabajaremos con el **tiempo de ejecución en el peor caso**.

Definición: El tiempo de ejecución de un algoritmo $\mathcal{A}$ con entrada x corresponde al número de operaciones elementales que ejecuta $\mathcal{A}$ para definir su salida. La **función de tiempo de ejecución** $f : \mathbb{N} \to \mathbb{N}$ del algoritmo se define por: $\forall n \in \mathbb{N},\ f(n)$ es el número máximo de operaciones elementales ejecutadas por el algoritmo (peor caso) con cualquier entrada x de $size(x) = n$.

Tiempo de ejecución de un algoritmo

La complejidad de un algoritmo se puede medir de distintas formas: número de operaciones elementales, memoria usada, en el caso promedio, peor caso, etc. En este curso trabajaremos con el **tiempo de ejecución en el peor caso**.

Definición: El tiempo de ejecución de un algoritmo $\mathcal{A}$ con entrada x corresponde al número de operaciones elementales que ejecuta $\mathcal{A}$ para definir su salida. La **función de tiempo de ejecución** $f : \mathbb{N} \to \mathbb{N}$ del algoritmo se define por: $\forall n \in \mathbb{N},\ f(n)$ es el número máximo de operaciones elementales ejecutadas por el algoritmo (peor caso) con cualquier entrada x de $size(x) = n$.

Observaciones:

- Para un mejor análisis de f supondremos que $f : \mathbb{N} \to \mathbb{R}^+$.
- Una operación elemental es una operación simple y bien definida. **Ejemplos**: asignación de valores, saltos condicionales (if then), operaciones aritméticas simples: $+, -, *, /$, comparación de dos números etc.

Notación asintótica

Nos interesa comparar de manera asintótica el crecimiento de dos funciones que representan el tiempo de ejecución de diferentes algoritmos.

Notación asintótica

Nos interesa comparar de manera asintótica el crecimiento de dos funciones que representan el tiempo de ejecución de diferentes algoritmos.

Definición: Dadas $f, g : \mathbb{N} \to \mathbb{R}^+$ dos funciones. Se denota:

Notación asintótica

Nos interesa comparar de manera asintótica el crecimiento de dos funciones que representan el tiempo de ejecución de diferentes algoritmos.

Definición: Dadas $f, g : \mathbb{N} \to \mathbb{R}^+$ dos funciones. Se denota:

1. $f(n) = O(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ f(n) \leq cg(n)$.
   En este caso, f es acotada superiormente de manera asintótica por g y se dice que **"f es o grande de g"** ó **"f es orden de g"** .

Notación asintótica

Nos interesa comparar de manera asintótica el crecimiento de dos funciones que representan el tiempo de ejecución de diferentes algoritmos.

Definición: Dadas $f, g : \mathbb{N} \to \mathbb{R}^+$ dos funciones. Se denota:

1. $f(n) = O(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ f(n) \leq cg(n)$.
   En este caso, f es acotada superiormente de manera asintótica por g y se dice que **"f es o grande de g"** ó **"f es orden de g"** .
2. $f(n) = \Omega(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ cg(n) \leq f(n)$.
   f es acotada inferiormente de manera asintótica por g y se dice que **"f es omega grande de g"**.

Notación asintótica

Nos interesa comparar de manera asintótica el crecimiento de dos funciones que representan el tiempo de ejecución de diferentes algoritmos.

Definición: Dadas $f, g : \mathbb{N} \to \mathbb{R}^+$ dos funciones. Se denota:

1. $f(n) = O(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ f(n) \leq cg(n)$.
   En este caso, f es acotada superiormente de manera asintótica por g y se dice que **"f es o grande de g"** ó **"f es orden de g"** .
2. $f(n) = \Omega(g(n))$ si $\exists c > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ cg(n) \leq f(n)$.
   f es acotada inferiormente de manera asintótica por g y se dice que **"f es omega grande de g"**.
3. $f(n) = \Theta(g(n))$ si
   $\exists c_1, c_2 > 0, n_0 \in \mathbb{N}, \forall n \geq n_0 : \ c_1 g(n) \leq f(n) \leq c_2 g(n)$. En este caso, se dice que **"g es una cota ajustada asintótica de f"** o **" f es Theta de g"**.

Notación asintótica

4. $f(n) = o(g(n))$ si $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$ o equivalentemente si:

$$\forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0,\ f(n) \leq cg(n).$$

En este caso, f crece estrictamente más lento que g y se dice que "f **es o chica de** g".

Notación asintótica

4. $f(n) = o(g(n))$ si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ o equivalentemente si:

$$\forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0,\ f(n) \leq cg(n).$$

   En este caso, f crece estrictamente más lento que g y se dice que "f **es o chica de** g".

5. $f(n) = \omega(g(n))$ si $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$ o equivalentemente si:

$$\forall c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0,\ g(n) \leq cf(n).$$

   En este caso, f crece estrictamente más rápido que g y se dice que "f **es omega chica de** g".

Notación asintótica

Observaciones:

Notación asintótica

Observaciones:

1. La notación $O(g(n))$ también se usa para representar el conjunto de funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que $f(n) = O(g(n))$. Es decir, también podemos escribir $f(n) \in O(g(n))$ en lugar de $f(n) = O(g(n))$. Además, en una expresión matemática $O(g(n))$ representa cualquier función $f(n) \in O(g(n))$. Por ejemplo: $h(n) + O(g(n))$ significa $h(n) + f(n)$ con $f(n) \in O(g(n))$ cualquiera. Lo mismo ocurre con o, ω, Ω y Θ.

Notación asintótica

Observaciones:

1. La notación $O(g(n))$ también se usa para representar el conjunto de funciones $f : \mathbb{N} \to \mathbb{R}^+$ tal que $f(n) = O(g(n))$. Es decir, también podemos escribir $f(n) \in O(g(n))$ en lugar de $f(n) = O(g(n))$. Además, en una expresión matemática $O(g(n))$ representa cualquier función $f(n) \in O(g(n))$. Por ejemplo: $h(n) + O(g(n))$ significa $h(n) + f(n)$ con $f(n) \in O(g(n))$ cualquiera. Lo mismo ocurre con o, ω, Ω y Θ.
2. Algunos conjuntos de la forma $O(f(n))$ tienen nombres usuales como: $O(1)$ (orden constante), $O(\log(n))$ (orden logarítmico), $O(\log\log(n))$ (orden sublogarítmico), $O(n)$ (orden lineal), $O(n\log(n))$ (orden lineal logarítmico), $O(c^n)$ (orden exponencial) $O(n!)$ (orden factorial), $O(n^n)$ (orden potencial exponencial).

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
3. $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$.

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
3. $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$.
4. *Si* $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$, *entonces* $f(n) \notin O(g(n))$ *y* $f(n) \in \Omega(g(n))$.

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
3. $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$.
4. *Si* $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$, *entonces* $f(n) \notin O(g(n))$ *y* $f(n) \in \Omega(g(n))$.
5. *Si* $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$, *entonces* $f(n) \in O(g(n))$ *y* $f(n) \notin \Omega(g(n))$.

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
3. $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$.
4. *Si* $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = \infty$, *entonces* $f(n) \notin O(g(n))$ *y* $f(n) \in \Omega(g(n))$.
5. *Si* $\lim_{n\to\infty} \dfrac{f(n)}{g(n)} = 0$, *entonces* $f(n) \in O(g(n))$ *y* $f(n) \notin \Omega(g(n))$.

Ejemplo:

Notación asintótica

Propiedades: *Sea* $f, g : \mathbb{N} \to \mathbb{R}^+$ *funciones. Entonces, se tiene las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.
2. $f(n) = o(g(n)) \implies f(n) = O(g(n))$.
3. $f(n) = \omega(g(n)) \implies f(n) = \Omega(g(n))$.
4. *Si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, *entonces* $f(n) \notin O(g(n))$ *y* $f(n) \in \Omega(g(n))$.
5. *Si* $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, *entonces* $f(n) \in O(g(n))$ *y* $f(n) \notin \Omega(g(n))$.

Ejemplo: Sea $k, n \in \mathbb{N}$.

1. $n^k = O(n^{k+l}),\ \forall l \in \mathbb{N} \cup \{0\}$.
2. $n = o(n \log(n))$.
3. $2^n = \omega(n^k)$.
4. $2^n = o(n^n)$.
5. $\log_k(n) = \Theta(\log(n)),\ \forall k \geq 2$, con $\log(n) \equiv \log_2(n)$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1, \ldots, n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1, \ldots, n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1, \ldots, n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

1. $size(a[1, \ldots, n]) = O(n)$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1, \ldots, n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

1. $size(a[1, \ldots, n]) = O(n)$.
2. Cada línea: 3,4,5 y 6 ejecuta una ($O(1)$) operación elemental.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1,\ldots,n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

1. $size(a[1,\ldots,n]) = O(n)$.
2. Cada línea: 3,4,5 y 6 ejecuta una ($O(1)$) operación elemental.
3. Los dos ciclos for son:
   $\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) =$
   $(n-1)+(n-2)+\cdots 1 = O(n^2)$
   iteraciones.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1,\ldots,n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

1. $size(a[1,\ldots,n]) = O(n)$.
2. Cada línea: 3,4,5 y 6 ejecuta una ($O(1)$) operación elemental.
3. Los dos ciclos for son: $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) = (n-1)+(n-2)+\cdots 1 = O(n^2)$ iteraciones.
4. Tiempo de ejecución: $f(n) = O(n^2)$ (iteraciones) por $O(1)$ (operaciones elementales en cada iteración)$= O(n^2)$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de ordenamiento de n enteros.

Algorithm Burbuja

Input: $a[1,\ldots,n]$ // arreglo de n enteros

```
1:  for i = 1 to n − 1 do
2:      for j = n to i + 1 do
3:          if a[j] < a[j − 1] then
4:              x ← a[j − 1]
5:              a[j − 1] ← a[j]
6:              a[j] ← x
7:          end if
8:      end for
9:  end for
10: return x
```

Tiempo de ejecución:

1. $size(a[1,\ldots,n]) = O(n)$.
2. Cada línea: 3,4,5 y 6 ejecuta una ($O(1)$) operación elemental.
3. Los dos ciclos for son: $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) = (n-1)+(n-2)+\cdots 1 = O(n^2)$ iteraciones.
4. Tiempo de ejecución: $f(n) = O(n^2)$ (iteraciones) por $O(1)$ (operaciones elementales en cada iteración)$= O(n^2)$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.
3. Si $n > m/2$, $q = 1$ y $r < m/2$. Luego en cualquier caso $r < m/2$. Por lo tanto, en la iteración i, $r < \frac{m}{2^i}$. Así, en $O(\log(m))$ iteraciones del ciclo While se tiene que $r = 0$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.
3. Si $n > m/2$, $q = 1$ y $r < m/2$. Luego en cualquier caso $r < m/2$. Por lo tanto, en la iteración i, $r < \frac{m}{2^i}$. Así, en $O(\log(m))$ iteraciones del ciclo While se tiene que $r = 0$.
4. Si $m < n$ entonces en una iteración de While se tiene el caso anterior. Luego, el no. de operaciones es: $O(\text{máx}\{\log(n), \log(m)\}) = O(\log(n) + \log(m))$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.
3. Si $n > m/2$, $q = 1$ y $r < m/2$. Luego en cualquier caso $r < m/2$. Por lo tanto, en la iteración i, $r < \frac{m}{2^i}$. Así, en $O(\log(m))$ iteraciones del ciclo While se tiene que $r = 0$.
4. Si $m < n$ entonces en una iteración de While se tiene el caso anterior. Luego, el no. de operaciones es: $O(\text{máx}\{\log(n), \log(m)\}) = O(\log(n) + \log(m))$.
5. En cada iteración se ejecuta $O(1)$ operaciones elementales.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.
3. Si $n > m/2$, $q = 1$ y $r < m/2$. Luego en cualquier caso $r < m/2$. Por lo tanto, en la iteración i, $r < \frac{m}{2^i}$. Así, en $O(\log(m))$ iteraciones del ciclo While se tiene que $r = 0$.
4. Si $m < n$ entonces en una iteración de While se tiene el caso anterior. Luego, el no. de operaciones es: $O(\text{máx}\{\log(n), \log(m)\}) = O(\log(n) + \log(m))$.
5. En cada iteración se ejecuta $O(1)$ operaciones elementales.
6. El tiempo de ejecución es: $T(n) = O(\log(n) + \log(m))$.

Ejemplos de tiempos de ejecución de algoritmos

Algoritmo de Euclides

Algorithm $MCD(m, n)$

Input: $m, n \in \mathbb{N}$

1: **while** $n > 0$ **do**
2: $r \leftarrow m$ mód n // ie r es el resto de dividir m por n
3: $m \leftarrow n$
4: $n \leftarrow r$
5: **end while**
6: **return** m

Tiempo de ejecución:

1. Supongamos $m \geq n$ y $m = qn + r$ con $0 \leq r = m$ mód $n < n$.
2. Si $n \leq m/2$, $r < m/2$.
3. Si $n > m/2$, $q = 1$ y $r < m/2$. Luego en cualquier caso $r < m/2$. Por lo tanto, en la iteración i, $r < \frac{m}{2^i}$. Así, en $O(\log(m))$ iteraciones del ciclo While se tiene que $r = 0$.
4. Si $m < n$ entonces en una iteración de While se tiene el caso anterior. Luego, el no. de operaciones es: $O(\text{máx}\{\log(n), \log(m)\}) = O(\log(n) + \log(m))$.
5. En cada iteración se ejecuta $O(1)$ operaciones elementales.
6. El tiempo de ejecución es: $T(n) = O(\log(n) + \log(m))$.