Análisis de Algoritmos

5. Recurrencias

Análisis de Algoritmos

(1) Estructuras de control
(2) Barómetro

} Usualmente aplicado a peor caso, son "en detalle"

(3) Análisis del caso promedio
(4) Análisis amortizado

} Usualmente dependiente del "tipo de entrada"

(5) Recurrencias } ¿? Un tipo específico de algoritmos

(6) No veremos análisis asociados a algoritmos específicos *de* estructuras de datos

(7) Diseño + análisis {
- Greedy
- Divide-and-Conquer
- Dynamic programming
- Algoritmos probabilísticos

5. Recurrencias

- Funciones de t dadas en función de sí mismas: "An equation that defines a function in terms of its own value on smaller inputs"
- Una recurrencia describe una secuencia de números, donde el/los término/s inicial/es (cantidad finita) son dados explícitamente y los siguientes términos se definen como una función de uno o más anteriores

5. Recurrencias

- Se los suele asociar en la bibliografía a
  - Estrategia D&C (top-down de diseño)
  - D-S/C-C (Divide-Solve/Conquer-Combine)
  - Cierto si es “igual problema-instancia menor”
- En todos los casos, se asocia a algoritmos básicamente recursivos
  - “Hacer lo mismo pero con menos cantidad”
- Ej: Factorial(n)

5. Recurrencias

- Ej: factorial

```
Factorial(n)
    if (n=0)
        return 1
    else
        return n * Factorial(n-1)
```

  - $t_{fac}(n) \begin{cases} 1 & n = 0 \\ t_{fac}(n-1) + 2 & n > 0 \end{cases}$

5. Recurrencias

- Ej: factorial

```
Factorial(n)
    if (n=0)
        return 1
    else
        return n * Factorial(n-1)
```

– $t_{fac}(n) = \begin{cases} 1 & n = 0 \\ t_{fac}(n-1) + 2 & n > 0 \end{cases}$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  - Evolución de llamadas…
    - $t(n) = t(n-1) + 2$ (1)
    - $t(n) = t(n-2) + 2 + 2$ (2)
    - ... ...
    - $t(n) = t(0) + 2 + ... + 2$ (n veces "+ 2") (n)

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  - Evolución de llamadas…
    - t(n) = t(n-1) + 2 (1)
    - t(n) = t(n-2) + 2 + 2 (2)
    - ... ...
    - t(n) = t(0) + 2 + ... + 2 (n veces “+ 2”) (n)
    - t(n) = t(0) + 2 n

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  - Evolución de llamadas…
    - t(n) = t(n-1) + 2 (1)
    - t(n) = t(n-2) + 2 + 2 (2)
    - ... ...
    - t(n) = t(0) + 2 + ... + 2 (n veces "+ 2") (n)
    - t(n) = t(0) + 2n
    - t(n) = 2n + 1

$$t_{fac}(n) = \begin{cases} 1 & n = 0 \\ t_{fac}(n-1) + 2 & n > 0 \end{cases}$$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  - Evolución de llamadas…
    - $t(n) = t(n-1) + 2$ (1)
    - $t(n) = t(n-2) + 2 + 2$ (2)
    - ... ...
    - $t(n) = t(0) + 2 + ... + 2$ (n veces “+ 2”) (n)
    - $t(n) = t(0) + 2n$
    - $t(n) = 2n + 1$
  - Demostrar (usualmente por inducción)

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  – Demostrar por inducción, $t_{fac}(n) = 2n+1$
    - ¿n = 0? $t(0) = 2x0 + 1 = 1$ que es la rec. $t(0) = 1$
    - Hi) $t(h) = 2h + 1$
    - Dem) ¿$t(h+1) = 2(h+1) + 1$?

$t(h+1) = t(h) + 2$ Def. Recurrencia

$$t_{fac}(n) \begin{cases} 1 & n = 0 \\ t_{fac}(n-1) + 2 & n > 0 \end{cases}$$

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  - Demostrar por inducción, $t_{fac}(n) = 2n+1$
    - ¿n = 0? $t(0) = 2x0 + 1 = 1$ que es la rec. $t(0) = 1$
    - Hi) $t(h) = 2h + 1$
    - Dem) ¿$t(h+1) = 2(h+1) + 1$?

$t(h+1) = t(h) + 2$ Def. Recurrencia

$= 2h + 1 + 2$ Hi

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  – Demostrar por inducción, $t_{fac}(n) = 2n+1$
    - ¿n = 0? $t(0) = 2x0 + 1 = 1$ que es la rec. $t(0) = 1$
    - Hi) $t(h) = 2h + 1$
    - Dem) ¿$t(h+1) = 2(h+1) + 1$?

$t(h+1) = t(h) + 2$ Def. Recurrencia

$= 2h + 1 + 2$ Hi

$= 2(h+1) + 1$ Lo que se quiere dem.

5. Recurrencias

- Ej: factorial, ¿cómo analizarlo?
  – Demostrar por inducción, $t_{fac}(n) = 2n+1$
    - ¿n = 0? $t(0) = 2x0 + 1 = 1$ que es la rec. $t(0) = 1$
    - Hi) $t(h) = 2 h + 1$
    - Dem) ¿$t(h+1) = 2(h+1) + 1$?

      $t(h+1) = t(h) + 2$ Def. Recurrencia

      $= 2 h + 1 + 2$ Hi

      $= 2 (h+1) + 1$ Lo que se quiere dem.

  – Brassard-Bratley: "Intelligent guesswork"

5. Recurrencias

- Brassard-Bratley:
  - Intelligent guesswork
  - Cambio de variables
  - Recurrencias homogéneas
    - Ecuación característica
      - Polinomio característico
        - » ... → Solución

5. Recurrencias

- Brassard-Bratley:
  - Intelligent guesswork
  - Cambio de variables
  - Recurrencias homogéneas
    - Ecuación característica
      - Polinomio característico
        - » ... → Solución
  - Recurrencias no homogéneas
    - Recurrencias homogéneas

5. Recurrencias

- Brassard-Bratley:
  - Intelligent guesswork
  - Cambio de variables
  - Recurrencias homogéneas
    - Ecuación característica
      - Polinomio característico
        - » ... → Solución
  - Recurrencias no homogéneas
    - Recurrencias homogéneas
      - Solución → *Backtrack* (relación con las no homogéneas)

5. Recurrencias

- Dos “recetas”
  - $t(n) = a\ t(n-b) + f(n)$ (reduce restando)
  - $t(n) = a\ t(n/b) + f(n)$ (reduce dividiendo)
    - a: cantidad de llamadas recursivas
    - b: constante
    - f(n): operaciones “extra” llamadas recursivas

5. Recurrencias

- Recursión, n ==> n-b
  - $t(n) = \begin{cases} c & n \le b \\ a\,t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$
  - $t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$
  - Vale con todos $\Theta(\ )$

5. Recurrencias

- Recursión, n ==> n-b:

  Procedure Hanoi (n, i, j) // Traslada los n anillos más pequeños de i a j (1, 2, 3)

  If n > 0 Then

  Hanoi (n-1, i, 6-i-j)

  write "i → j"

  Hanoi(n-1, 6-i-j, j)

5. Recurrencias

- Recursión, n ==> n-b:

```
Procedure Hanoi (n, i, j)
  If n > 0 Then
    Hanoi (n-1, i, 6-i-j)
    write "i → j"
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas

5. Recurrencias

- Recursión, n ==> n-b:

```
Procedure Hanoi (n, i, j)
  If n > 0 Then
    Hanoi (n-1, i, 6-i-j)
    write "i → j"
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas
Reduce en 1 el tamaño

5. Recurrencias

- Recursión, n ==> n-b:

```
Procedure Hanoi (n, i, j)
  If n > 0 Then
    Hanoi (n-1, i, 6-i-j)
    write "i → j"
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas

Reduce en 1 el tamaño

Trabajo extra constante

5. Recurrencias

- Recursión, n ==> n-b:

```
Procedure Hanoi (n, i, j)
  If n > 0 Then
    Hanoi (n-1, i, 6-i-j)
    write "i → j"
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas
Reduce en 1 el tamaño
Trabajo extra constante

a = 2
b = 1
k = 0

$$t(n) = \begin{cases} c & n \le b \\ a\,t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

$$t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

5. Recurrencias

- Recursión, n ==> n-b:

```
Procedure Hanoi (n, i, j)
  If n > 0 Then
    Hanoi (n-1, i, 6-i-j)
    write "i → j"
    Hanoi(n-1, 6-i-j, j)
```

2 llamadas recursivas
Reduce en 1 el tamaño
Trabajo extra constante

$a = 2$
$b = 1$
$k = 0$

$$t(n) = \begin{cases} c & n \le b \\ a\,t(n-b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

$$t(n) \in \begin{cases} O(n^{k+1}) & \text{si } a = 1 \\ O(a^{n \text{ div } b}) & \text{si } a > 1 \end{cases}$$

==> $t(n) \in O(2^n)$

5. Recurrencias

- Dos “recetas”
  - $t(n) = a\ t(n-b) + f(n)$ (reduce restando)
  - $t(n) = a\ t(n/b) + f(n)$ (reduce dividiendo)
    - a: cantidad de llamadas recursivas
    - b: constante
    - f(n): operaciones “extra” llamadas recursivas

5. Recurrencias

- Recursión, n ==> n/b
  - $t(n) = \begin{cases} c & n \le b \\ a\, t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$
  - $t(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$
  - Vale con todos Θ( ). Teorema Maestro o Método Maestro

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a\,t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a\,t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a\ t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k

$$t(n) = \begin{cases} c & n \leq b \\ a\, t(n/b) + f(n) & n > b, \text{ con } f(n) \in O(n^k) \end{cases}$$

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k
  - Aplicar la receta

5. Recurrencias

- Recursión, n ==> n/b
  - Nuevamente, es necesario identificar
    - Cantidad de llamadas recursivas, a
    - Const. por la cual se divide el tamaño de la E/, b
    - Grado del polinomio del trabajo extra, k
  - Aplicar la receta teniendo en cuenta a, b y k

$$t(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \log_b n) & \text{si } a = b^k \\ O(n^k) & \text{si } a < b^k \end{cases}$$

5. Recurrencias

- Recursión, n ==> n/b
  - Ej: bin_search
    - a = 1
    - b = 2
    - k = 0
  - Ej: merge_sort
    - a = 2
    - b = 2
    - k = 1

5. Recurrencias

- Dos “recetas”
  - $t(n) = a\, t(n-b) + f(n)$ (reduce restando)
  - $t(n) = a\, t(n/b) + f(n)$ (reduce dividiendo)
  - No todas las recurrencias “cubiertas”
    - Fibonacci recursiva

```
FibRec(n)
  if (n < 2)
     then return n
  else return FibRec(n-1) + FibRec(n-2)
```

5. Recurrencias

- Sin “receta”: Fibonacci recursiva

```
FibRec(n)
  if (n < 2)
     then return n
  else return FibRec(n-1) + FibRec(n-2)
```

$$t_{fib} = \begin{cases} 1 & n < 2 \\ t_{fib}(n-1) + t_{fib}(n-2) + cte & n \geq 2 \end{cases}$$

Ninguna de las dos recetas es aplicable

5. Recurrencias

- Sin “receta”: Fibonacci recursiva
- Fibonacci iterativa

```
FibIter(n)
    i ← i; j ← 0
    for k ← 1 to n do
        j ← i + j
        i ← j – i
    return j
```