Análisis de Algoritmos

Introducción

Contexto

- Ya podemos hacer la asociación
  - Problema computacional <==> MTD
  - Entrada <==> Instancia del problema comput.
  - Problemas computables <==> R

Contexto

- Ya podemos hacer la asociación
  - Problema computacional <==> MTD
  - Entrada <==> Instancia del problema comput.
  - Problemas computables <==> R
  - Complejidad <==> "Análisis" de R
    - Computable $\neq$ "Tratable"
  - MTD <==> Algoritmo

Contexto

- Ya podemos hacer la asociación
  - Problema computacional <==> MTD
  - Entrada <==> Instancia del problema comput.
  - Problemas computables <==> R
  - Complejidad <==> "Análisis" de R
    - Computable $\neq$ "Tratable"
  - MTD <==> Algoritmo
    - Problema computable <==> Infinitas MTD (alg.)
    - MTD <==> Cantidad de pasos (δ de transición)

Contexto

- De la clase inicial

  Temas:

  Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

  Computabilidad

  Complejidad

  ==> Problemas (general)

Contexto

- De la clase inicial

  Temas:

  Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

  Computabilidad

  Complejidad

  ==> Problemas (general)

  Algebraico/Matemático (cardinalidad)

  Algorítmico/MT

Contexto

- De la clase inicial

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

Computabilidad

Complejidad

==> Problemas (general)

Algebraico/Matemático (cardinalidad)

Algorítmico/MT

Clases de complejidad/MT

Algoritmia

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis,
Notación Asintótica

Algoritmia

Temas:

Algoritmos (específico) ==> Algoritmia. Análisis, Notación Asintótica

Algoritmia

Diseño (creación-propuesta)

Análisis (comportamiento)

$\left.\begin{array}{l}\text{Diseño (creación-propuesta)}\\ \\ \text{Análisis (comportamiento)}\end{array}\right\}$ ==> De algoritmos

Algoritmia

- Diseño: lo han visto hasta este punto en múltiples asignaturas
- Análisis de Algoritmos (Leiserson): "The theoretical study of computer-program performance and resource usage"
- Análisis de Eficiencia: Cantidad de recursos que se utilizan (tiempo, espacio)

Algoritmia

- ¿Por qué hacer análisis de algoritmos?
  - Comparación teórica
  - Escalabilidad (cuánto más grande se puede, hasta dónde llego)
  - Matemática algorítmica provee un lenguaje de comportamiento de programas
  - Análisis de t y e se pueden relacionar en algunos casos

Algoritmia

- El análisis teórico está apoyado por el principio de invarianza:
  - "Según el principio de invarianza, dos implementaciones diferentes del mismo algoritmo no difieren en tiempo de ejecución más que en una constante multiplicativa"
- Relación directa con
  - Sea M una MTD con k cintas, decimos que M es de complejidad t(n), si para toda entrada de longitud n, M hace a lo sumo t(n) mov.

Tiempo y Espacio

- Análisis de t y e
  - Dado un algoritmo A, el tiempo de ejecución $t_A(n)$ de A es la cantidad de pasos, operaciones o acciones elementales que debe realizar el algoritmo al ser ejecutado en una instancia de tamaño n.
  - El espacio $e_A(n)$ de A es la cantidad de datos elementales que el algoritmo necesita al ser ejecutado en una instancia de tamaño n, sin contar la representación de la entrada

Análisis de Algoritmos

- Definiciones de $t_A(n)$ y $e_A(n)$ son ambiguas en dos sentidos:
  - No especifica claramente cuáles son las operaciones o datos elementales.
  - Dado que existe más de una instancia de tamaño n no está claro cuál de ellas es la que se tiene en cuenta para el análisis (o cuáles, si son relevantes).

Análisis de Algoritmos

- Operaciones Elementales:
  - Tiempo de ejecución está acotado por una constante que depende sólo de la implementación usada
  - No depende del tamaño de la entrada
  - Sólo interesa el número de operaciones elementales
  - En general, las sumas, multiplicaciones y asignaciones son operaciones elementales

Análisis de Algoritmos

- Tipos de instancias (¿Por qué) - Tipos de análisis (si hay dependencia de datos): peor-mejor-promedio-probabilístico
- Relación con
  - Sea M una MTD con k cintas, decimos que M es de complejidad t(n), si para toda entrada de longitud n, M hace a lo sumo t(n) mov.

Análisis de Algoritmos

- Tipos de instancias (¿Por qué) - Tipos de análisis (si hay dependencia de datos): peor-mejor-promedio-probabilístico
- Relación con
  - Sea M una MTD con k cintas, decimos que M es de complejidad t(n), si para toda entrada de longitud n, M hace **a lo sumo t(n)** mov.

Análisis de Algoritmos

- Notación Asintótica
  - Utilizaremos la noción de tiempo ignorando las constantes multiplicativas que lo afectan
  - Caracteriza en forma simple la eficiencia (o uso de recursos) de los algoritmos
  - Se independiza de las características específicas de la implementación
  - Análisis del comportamiento de las funciones en el límite, considerando sólo su tasa de crecimiento (notación asintótica)

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

Análisis de Algoritmos

1. Estructuras de Control

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

1. Estructuras de Control

- Secuencias:

P1; P2 $\qquad\qquad$ t1(n) y t2(n)

$t1;2(n) = t1(n) + t2(n)$

Regla del máximo

$t1;2(n) = t1(n) + t2(n) \in O(\max(t1(n), t2(n)))$

$t1;2(n) = t1(n) + t2(n) \in \Theta(\max(t1(n), t2(n)))$

1. Estructuras de Control

- Condicional (dep. de n asumido):

If (cond)	t1
Then (cuerpo then)	t2
Else (cuerpo else)	t3

Se considera directamente el peor caso:

t1 + max(t2, t3) o directamente

max(t1, t2, t3)

1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For i ← 1 to m
  Do P(i)   Puede ser P(i, n), t(i)

  – Si t(i) no depende de i ==> t(i) = t
    - tfor = m t
    - En cualquier caso, identificar #iter
    - tfor = #iter t

1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For i ← 1 to m

Do P(i) Puede ser P(i, n), t(i)

- Si t(i) depende de i
  - tfor = $\sum_{i=1}^{m} t(i)$
- Sumas útiles:
  - $\sum_{i=1}^{m} i = \frac{m\,(m+1)}{2}$
  - $\sum_{i=1}^{m} i^2 = \frac{m\,(m+1)\,(2m+1)}{6}$

1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For i $\leftarrow$ 1 to m

Do P(i) Puede ser P(i, n), t(i)

- – No confundir "peor caso" con el limite
  - For j $\leftarrow$ i to m o For j $\leftarrow$ 1 to i
  - Cantidad de iteraciones: m-i+1 o i
  - No hay "peor caso" (i=1, i=m)

1. Estructuras de Control

- Iteraciones for o ciclos uniformes:

For i $\leftarrow$ 1 to m
    Do P(i)   Puede ser P(i, n), t(i)

  - Todo lo anterior vale solamente si $1 \leq m$
  - En general: inicio $\leq$ fin

1. Estructuras de Control

- Iteraciones no uniformes
  - while y repeat
  - Cantidad de iteraciones desconocida a priori
  - Dos formas de análisis
    - Funciones de variables que decrece
    - Recurrencias
    - Veremos ambas formas con un ejemplo

1. Estructuras de Control

- Iteraciones no uniformes

```
function bin_search(T[1...n], x)  // x está en T
    i ← 1; j ← n
    While i < j Do          ==> Cantidad determinada por la dif.
        k ← (i + j) % 2
        Case x < T[k]: j ← k-1
             x = T[k]: i, j ← k // Return k
             x > T[k]: i  ← k+1
    Return i
```

(cont.)

1. Estructuras de Control

- Iteraciones no uniformes: función
  - Variables de la iteración
  - El valor de la función decrece a medida que se llevan a cabo más iteraciones
  - El valor de la función debe ser un entero positivo ==> el algoritmo termina
  - ¿Cuándo? Entender la forma en que decrece la función. Ej: bin_search

(cont.)

1. Estructuras de Control

- Iteraciones no uniformes: función
  - bin_search
    - Mejor caso: se encuentra en la 1ra evaluación
    - Peor caso: se encuentra cuando $i=j$
    - Función $d = j - i + 1$ (cantidad de elems.)
    - $d \geq 2$ (análisis de “mitad”)
    - $d = 1$ (índice del elem.)

(cont.)

1. Estructuras de Control

- Iteraciones no uniformes: función
  – bin_search: $d = j – i + 1$

  ¿Decrece? Veamos los valores de i, j y d al principio y al final de una iteración cualquiera, i', j', y d':

  1) $x < T[k]$:

  $j' = (i + j) \div 2 – 1;\ \ i' = i;$

  $d' = (i + j) \div 2 – 1 – i + 1 \leq (i + j) / 2 – i$

  $= - i/2 + j/2 < - i/2 + j/2 + 1/2 = (j – i + 1) / 2 = d/2 < d$

(cont.)

1. Estructuras de Control

- Iteraciones no uniformes: función
  - bin_search: $d = j - i + 1$

¿Decrece? Veamos los valores de i, j y d al principio y al final de una iteración cualquiera, i', j', y d‘:

2) $x > T[k]$:

$j' = j$; $i' = (i + j) \div 2 + 1$;

$d' = j - (i + j) \div 2 - 1 + 1 \leq j - (i + j) / 2 =$

$= j/2 - i/2 < j/2 - i/2 + 1/2 = (j - i + 1) / 2 = d/2 < d$

(cont.)

1. Estructuras de Control

- Iteraciones no uniformes: función
  - bin_search: $d = j - i + 1$

  ¿Decrece? Veamos los valores de i, j y d al principio y al final de una iteración cualquiera, i', j', y d‘:

  3) $x = T[k]$:

  $d' = 1 \leq d/2$

(cont.)

==> $d = j-i+1$decrece en cada iteración

1. Estructuras de Control

- Iteraciones no uniformes: función
  - bin_search: $d = j - i + 1$

  ¿Cuántas iteraciones? d_k: valor de $j_k - i_k + 1$ al final de la iteración k

  $d_0 = n$

  $d_1 = n/2$

  ...

  $d_i = n/2^i$

  Peor caso, “todas” las iteraciones hasta que $d_k = 1$

  $d_k = n/2^k = 1$; ¿k? $n = 2^k \Longrightarrow \log_2 n = \log_2 2^k \Longrightarrow$

  $k = \log_2 2^n \Longrightarrow \lceil k = \log_2 n \rceil$

1. Estructuras de Control

- Iteraciones no uniformes
  - Dos formas de análisis
    - Funciones de variables que decrece
      - Explicación y ejemplo bin_search
    - Recurrencias

1. Estructuras de Control

- Iteraciones no uniformes: recurrencias
  - $t(n)$: t para resolver el problema con n elems.
  - $t(n) \begin{cases} 1 & n = 1 \\ t(n/2) + c & n > 1 \end{cases}$
  - Veremos las recurrencias más adelante
- En general, las iteraciones no uniformes son complicadas de analizar (función y/o recurrencia)

1. Estructuras de Control

- Resumen:
  - Paso a paso, muy detallado
  - Iteraciones no uniformes complicadas
  - t(n) relativamente detallado y “combinado”
  - Puede implicar mucho tiempo por el detalle de cada estructura de control

Análisis de Algoritmos

2. Barómetro
3. Promedio
4. Amortizado

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

2. Instrucción Barómetro

- Una instrucción barómetro es la que se ejecuta al menos tantas veces como cualquier otra excepto quizás una cantidad constante (acotada) de veces.
- Si t(n) es el tiempo del algoritmo a analizar, t(n) es Θ(f(n)) donde f(n) es la cantidad de veces que se ejecuta la instrucción barómetro

2. Instrucción Barómetro

- A tener en cuenta/conocer:
  - Instrucción barómetro
  - Encontrar f(n)
  - Se usan los principios de las estructuras de control para determinar la cantidad de veces que se ejecuta la instrucción barómetro

2. Instrucción Barómetro

- Ej:

```
Function select_sort(T[1...n])  // n pasadas
    For i ← 1 to n-1 Do
        minj ← i; minx ← T[i]
        For j ← i+1 to n Do
            If T[j] < minx Then
                minj ← j
                minx ← T[j]
        T[minj] ← T[i]
        T[i] ← minx
```

2. Instrucción Barómetro

- Ej:

```
Function select_sort(T[1...n])  // n pasadas
    For i ← 1 to n-1 Do
        minj ← i; minx ← T[i]
        For j ← i+1 to n Do
            If T[j] < minx Then
                minj ← j
                minx ← T[j]
        T[minj] ← T[i]
        T[i] ← minx
```

2. Instrucción Barómetro

- Resumen
  - Identificación de barómetro
  - Cantidad de veces <==> Estructuras de contr.
  - $t(n) \in \Theta(f(n))$
    - No se tiene $t(n)$ …

3. Caso Promedio

- Uso implícito/explícito de distribución de probabilidades de las instancias de tamaño n
- En principio, para ordenar vectores, el tiempo de las n! posibles dividido n!
- Asumir que son todas igualmente probables y usar esto en el análisis

3. Caso Promedio

- Ej. de insertion sort, Brassard/Bratley (p. 62, 111)
- En este caso específico, se tiene en cuenta la distribución de probabilidad de los números del arreglo en partes cada vez menores del mismo.
- Es complementaria al análisis de estructuras de control o barómetro (casos)

4. Análisis Amortizado

- Para los casos en los que
  - Es muy poco probable que en todas las llamadas se tenga siempre el peor caso
  - La cantidad de operaciones está relacionada con una secuencia de uso de un algoritmo
  - Ej: estructuras de datos, inserción en un grafo
  - Ej: IncBin a continuación

4. Análisis Amortizado

- IncBin

```
Procedure IncBin(Cntr[1...m]) // Cntr[i] ∈ {0, 1}
    j ← m+1
    Repeat
        j ← j – 1
        C[j] ← 1 – C[j]
    Until (C[j] = 1) Or (j = 1)
```

- Se tiene el peor caso solamente 1 vez cada …

4. Análisis Amortizado

- IncBin

Errores en la explicación de la diapositiva anterior:

- Los bits no se incrementan, se invierten (es lo necesario para incrementar el contador binario representado con un array de bits)
- Los bits se invierten de derecha a izquierda, no de izquierda a derecha, es correcta la expresión "menos a más significativos", es incorrecto "de izquierda a derecha"

4. Análisis Amortizado

- Promedio de t(n) en llamadas sucesivas, no independientes
- Tres métodos $\left\{ \begin{array}{l} \text{Agregado} \\ \text{Contable} \\ \text{Potencial} \end{array} \right.$
- Sería
  - Solo para algunos casos/algoritmos
  - Alternativo a casos mejor/peor/probabilístico

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) = 2 h + 1
    - Dem) ¿t(h+1) = 2(h+1) + 1?

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

= 2 h + 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) $t(h) = 2 h + 1$
    - 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.

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) ¿$t(h+1) = 2(h+1) + 1$?

$$
\begin{aligned}
t(h+1) &= t(h) + 2 && \text{Def. Recurrencia} \\
&= 2\ h + 1 + 2 && \text{Hi} \\
&= 2\ (h+1) + 1 && \text{Lo que se quiere dem.}
\end{aligned}
$$

  - 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

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

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

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

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

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

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

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

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

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

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
```

Análisis de Algoritmos

6. Estructuras de Datos
7. Diseño + Análisis

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

6. Estructuras de Datos

- Al menos una asignatura específica
- La mayoría/varios son recursivos
- En algunos casos: amortizado
  - Al construir un índice en una base de datos

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
  - Algoritmos Greedy
  - Programación dinámica
  - Algoritmos probabilísticos
- Muy específico de cada algoritmo más que del propio diseño

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
  - Algoritmos Greedy
  - Programación dinámica
  - Algoritmos probabilísticos
- Muy específico de cada algoritmo más que del propio diseño

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Divide and Conquer
    - En realidad: recursivos ==> recurrencias

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - Problemas de optimización, construyendo la solución paso a paso de manera iterativa
  - Decisiones en cada paso, dependiendo del estado de avance/solución
  - Se elije entre un conjunto de alternativas que se evalúan ==> lo "mejor"
  - Ejemplo del viajante de comercio

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio

7. Diseño + Análisis

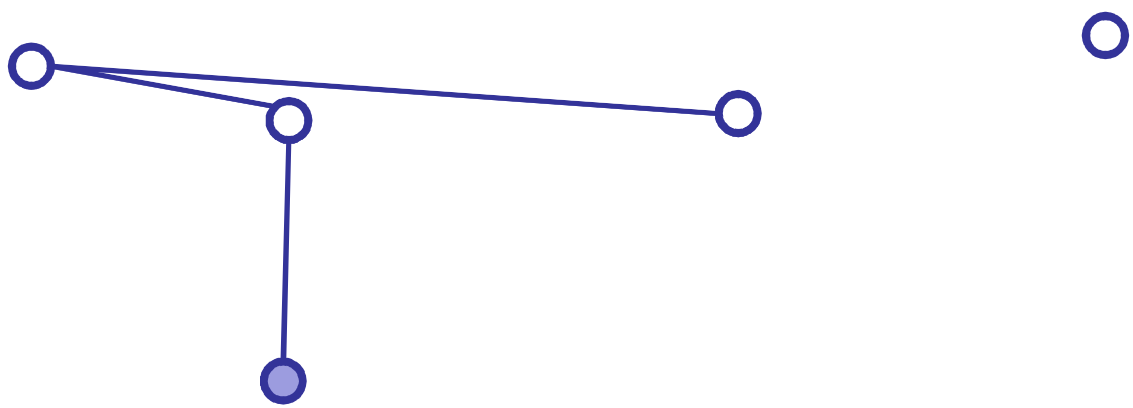
- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio

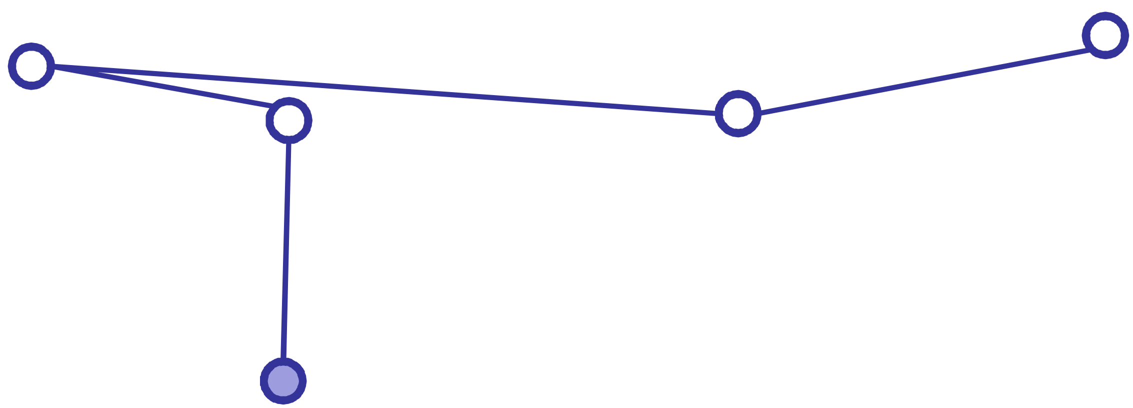
7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio



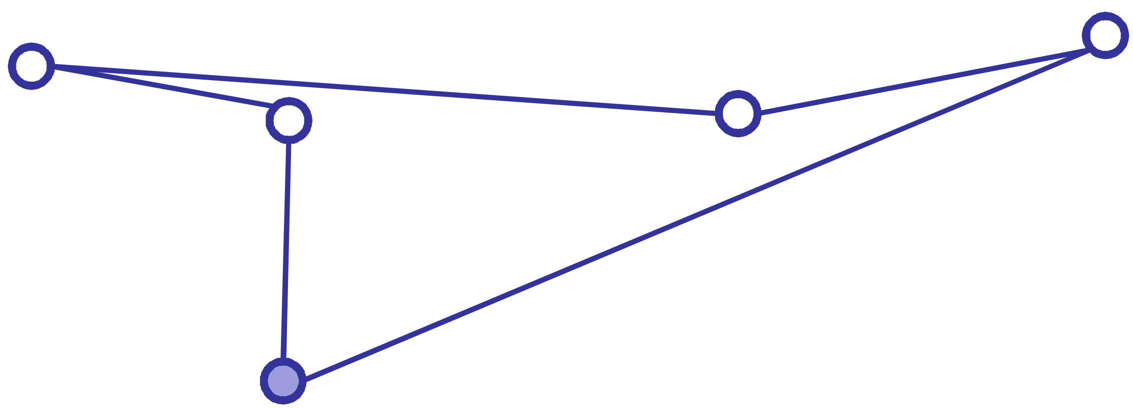
7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio



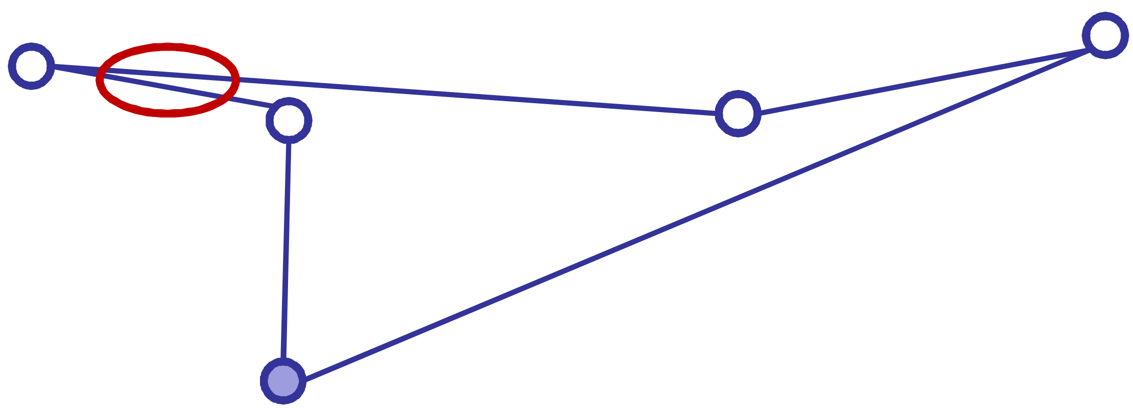
7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio



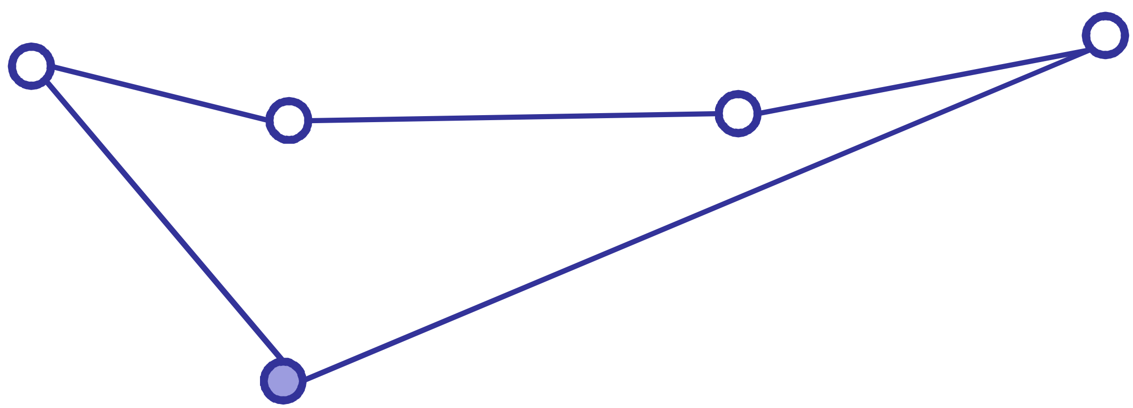
7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio



7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos Greedy
  - …
  - Ejemplo del viajante de comercio



7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Programación dinámica
    - Estrategia bottom up (asociado a top-down, rec.)
    - Se comienza resolviendo las partes o problemas más sencillos posibles
    - Se reutilizan resultados intermedios (tablas)
    - Ej. fibonacci: f(n) = f(n-1) + f(n-2)

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
    - Cuando se debe tomar una decisión, se toma al azar, no se computan costos p/ evaluar.
  - a) Numéricos: intervalo de confianza sobre la respuesta, ej: 90% de acierto para x ± y
  - b) Monte Carlo: respuesta exacta con alta probabilidad, pero puede ser errónea a veces
  - c) Las Vegas: respuesta exacta o sin resp.

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular

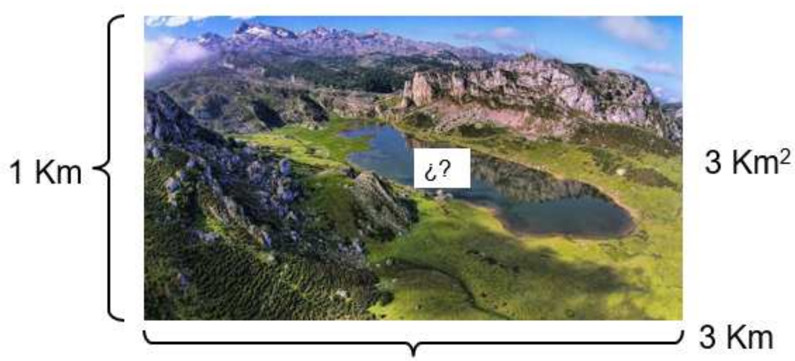
1 Km

3 Km^2

3 Km

7. Diseño + Análisis

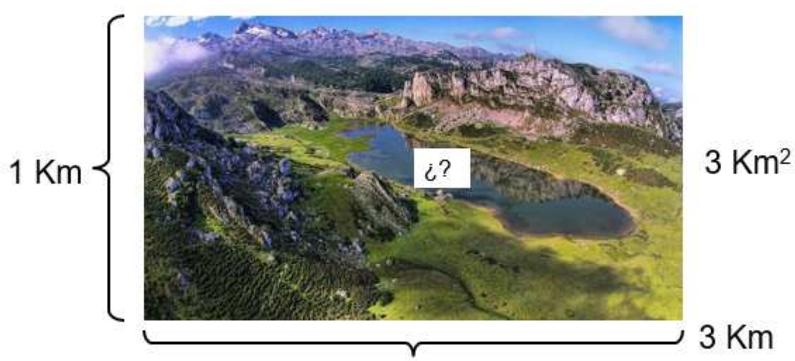
- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



Posiciones aleatorias en el rectángulo
- Dentro del lago
- Fuera del lago
- #in vs. #out
- #in / #out ==> proporción del total ocupada por el lago
- AreaLago = (#in / #out) x 3 km^2

7. Diseño + Análisis

- Más asociados al diseño que al análisis
  - Algoritmos probabilísticos
  - Ej: área de una superficie irregular



Posiciones aleatorias en el rectángulo
- Dentro del lago
- Fuera del lago
- #in vs. #out
- #in / #out ==> proporción del total ocupada por el lago
- AreaLago = (#in / #out) x 3 km^2
- ¿Error? ==> A mayor generación de aleatorios, menor error