Algoritmos y Complejidad

Algoritmos y Complejidad

Técnicas y Herramientas

Pablo R. Fillottrani

Depto. Ciencias e Ingeniería de la Computación

Universidad Nacional del Sur

Primer Cuatrimestre 2019



Algoritmos y Complejidad

Técnicas y Herramientas

Técnicas de Demostración

Herramientas Matemáticas Básicas

Notación Asintótica

Estructuras de Datos

Análisis de Algoritmos por Estructuras de Control Resolución de Recurrencias



Algoritmos y Complejidad

Técnicas de Demostración

I pruebas por contradicción

I pruebas por inducción

I principio de inducción

I inducción matemática generalizada

I inducción constructiva

Herramientas ya conocidas. Leer apunte disponible en la página web.

Algoritmos y Complejidad

Herramientas Matemáticas Básicas

I Lógica: cálculo proposicional y de predicados.

I Teoría de Conjuntos: operaciones básicas, producto cartesiano, cardinalidad.

I Teoría de Números: módulo, intervalos, techo y piso.

I Elementos básicos de álgebra y análisis: funciones, relaciones, series, sumatorias y productos, límites, módulos, logaritmos.

I Probabilidades: probabilidad condicional, esperanza, varianza. I Combinatoria: permutaciones, combinaciones.

En el final del apunte se presenta un compendio de fórmulas útiles sobre estos temas.



Algoritmos y Complejidad

Notación Asintótica

Objetivos

Objetivos

I no interesa conocer los valores absolutos de las funciones I permitir una caracterización simple de la eficiencia de un

algoritmo y comparar las performances relativas de distintos

algoritmos

I independizar el análisis de los algoritmos de condiciones

específicas de implementación: lenguaje de programación,

compilador, equipo, etc.



Algoritmos y Complejidad

Notación Asintótica

Objetivos

I se aplica a funciones de tiempo de ejecución o de espacio de memoria de algoritmos en base a la longitud de la entrada:

*f*(*n*) : **N** *−→* **R**+

I se denomina asintótica porque analiza el comportamiento de las funciones en el *límite*, es decir su tasa de crecimiento



Algoritmos y Complejidad

Notación Asintótica

Notación O

Notación *O*(*·*)

*O*(*g*(*n*)) = *{f*(*n*) : *∃c ∈* **R**+*,∃n*0 *∈* **N***,*tal que

*f*(*n*) *≤ cg*(*n*) para todo *n ≥ n*0*}*

I determina una cota superior en la tasa de crecimiento de una función, dentro de un factor constante

I ejemplos:

I 6*n*3 *∈ O*(*n*3) ya que se cumple la definición con *c* = 6*,n*0 = 1

I 3log *n ∈ O*(*n*) ya que se cumple la definición con *c* = 1*,n*0 = 4



Algoritmos y Complejidad

Notación Asintótica

Notación O

| notacion-O |
| --- |

Ejemplos:

I 300*n*2 *∈ O*(*n*2)

I 5*n*4 *−*4*n*3 +10*n*2 +39 *∈ O*(*n*4) I log*b n ∈ O*(log*a n*)*,∀a,b*

I 2*n ∈ O*(*n*!)

I 500000*n ∈ O*(0*,*00001*n*2)

I 0*,*000001*n*2 *6∈ O*(500000*n*) I *n*! *6∈ O*(2*n*)



Algoritmos y Complejidad

Notación Asintótica

Notación Omega

Notación Ω(*·*)

Ω(*g*(*n*)) = *{f*(*n*) : *∃c ∈* **R**+*,∃n*0 *∈* **N***,*tal que

*f*(*n*) *≥ cg*(*n*) para todo *n ≥ n*0*}*

I determina una cota inferior en la tasa de crecimiento de una función, dentro de un factor constante

I ejemplos:

I 6*n*3 *∈* Ω(*n*3) ya que se cumple la definición con *c* = 1*,n*0 = 1 I 1*/*3*n ∈* Ω(log *n*) ya que se cumple la definición con

*c* = 1*/*3*,n*0 = 1



Algoritmos y Complejidad

Notación Asintótica

Notación Omega

| notacion-Omega |
| --- |

Ejemplos:

I 3*n*5 +4*n*3 *−*8*n*2 +10*n ∈* Ω(*n*4) I log*b n ∈* Ω(log*a n*)*,∀a,b*

I *n*! *∈* Ω(2*n*)

I 0*,*00001*n*2 *∈* Ω(50000*n*)

I 50000*n 6∈* Ω(0*,*00001*n*2)

I 2*n 6∈* Ω(*n*!)



Algoritmos y Complejidad

Notación Asintótica

Notación Theta

Notación Θ(*·*)

Θ(*g*(*n*)) = *{f*(*n*) : *∃c,d ∈* **R**+*,∃n*0 *∈* **N***,* tal que

*cg*(*n*) *≤ f*(*n*) *≤ dg*(*n*) para todo *n ≥ n*0*}*

I determina una cota superior e inferior en la tasa de crecimiento de una función, dentro de un factor constante

I ejemplos:

I 6*n*3 *∈* Θ(*n*3) ya que se cumple la definición con

*c* = 6*,d* = 6*,n*0 = 1.

I 1*/*3*n ∈* Θ(*n*) ya que se cumple la definición con

*c* = 1*/*5*,d* = 1*,n*0 = 1.

Algoritmos y Complejidad

Notación Asintótica

Notación Theta

| notacion-Theta |
| --- |

Ejemplos:

I 3*n*2 *∈* Θ(*n*2)

I log *n 6∈* Θ(*n*)

I 2*n*+1 *∈* Θ(2*n*)

I 500000*n*2 *∈* Θ(0*,*00001*n*2)

I log*b n ∈* Θ(log*a n*) para todo *a,b >* 0



Algoritmos y Complejidad

Notación Asintótica

Uso en Ecuaciones

Uso en Ecuaciones

I por ejemplo, *f*(*n*) = 2*n*2 + Θ(*n*) significa que *f*(*n*) es igual a 2*n*2 *más alguna función cualquiera perteneciente a* Θ(*n*)

I 2*n*2 +*O*(*n*) = *O*(*n*2) significa que *no importando que función perteneciente a O*(*n*) *se sume a* 2*n*2*, siempre el resultado es una función en O*(*n*2)

I *f*(*n*) = *O*(*g*(*n*)) + *O*(*h*(*n*)) significa que *f*(*n*) es una función que se puede obtener *sumando punto a punto una función de*

*O*(*g*(*n*)) *con una función de O*(*h*(*n*))

I se evita hacer referencia a detalles que no afectan el

comportamiento general de la función



Algoritmos y Complejidad

Notación Asintótica

Algunas Propiedades útiles

Algunas Propiedades útiles

I *O*(*f*1(*n*) +*f*2(*n*)) = *O*(max(*f*1(*n*)*,f*2(*n*)))

I *f*(*n*) *∈* Θ(*g*(*n*)) si y solo si *g*(*n*) *∈* Θ(*f*(*n*))

I *f*(*n*) *∈ O*(*g*(*n*)) si y solo si *g*(*n*) *∈* Ω(*f*(*n*))

I *f*(*n*) *∈* Θ(*g*(*n*)) si y solo si *f*(*n*) *∈ O*(*g*(*n*))*∧f*(*n*) *∈* Ω(*g*(*n*)) I si l´ım*n→*∞*f* (*n*)

*g*(*n*)*∈* **R**+ entonces *f*(*n*) *∈ O*(*g*(*n*)) y *g*(*n*) *∈ O*(*f*(*n*))

I si l´ım*n→*∞*f* (*n*)

*g*(*n*) = 0 entonces *f*(*n*) *∈ O*(*g*(*n*)) pero *g*(*n*) *6∈ O*(*f*(*n*))



Algoritmos y Complejidad

Notación Asintótica

Ejemplos

ORDENAR UN ARREGLO

Algoritmo: Ordenamiento por Inserción I Costo de la ejecución del algoritmo:

*n−*1

*n−*1

*TI* (*n*) = *c*1*n* + *c*2(*n−*1) + *c*3(*n−*1) + *c*4 *n−*1

∑

*j*=1

*tj* + *c*5

∑

*j*=1

(*tj −*1) +

+*c*6

∑

*j*=1

(*tj −*1) + *c*8(*n−*1)

*n−*1

*n−*1

= (*c*1 + *c*2 + *c*3 + *c*8)*n−*(*c*2 + *c*3 + *c*8) + (*c*4 + *c*5 + *c*6)

∑

*j*=1

*tj −*(*c*5 + *c*6)

∑

*j*=1

1



Algoritmos y Complejidad

Notación Asintótica

Ejemplos

I para analizar el *O*(*·*) se tiene:

*TI*(*n*) = *d*1*n−d*2 +*d*3

*n−*1

*n−*1 ∑ *j*=1

*tj −d*4

*n−*1 ∑ *j*=1

1

*≤ d*1*n*+*d*3 *≤ d*1*n*+*d*3

∑ *j*=1

*n−*1 ∑ *j*=1

*tj n*

= *d*1*n*+*d*3*n*(*n−*1)

*≤ d*1*n*+*d*3*n*2

I luego *T*(*n*) *∈ O*(*n*2).

Algoritmos y Complejidad

Notación Asintótica

Ejemplos

I para analizar el Ω(*·*) se tiene: *n−*1

*n−*1

*TI*(*n*) = *d*1*n−d*2 +*d*3

∑ *j*=1

*n−*1

*tj −d*4

∑ *j*=1

1

*≥ d*1*n−d*2 +*d*3

∑ *j*=1

*j −d*4(*n−*1)

= *d*1*n−d*2 +*d*3(*n−*1)*n/*2*−d*4(*n−*1)

*≥d*32*n*2 +*d*1*n−*(*d*32+*d*4)*n−*(*d*2 +*d*4) *∈* Ω(*n*2)

I recordemos que *T*(*n*) es el tiempo de ejecución en el peor caso para instancias de tamaño *n*

I luego *T*(*n*) *∈* Ω(*n*2) y por lo tanto también *T*(*n*) *∈* Θ(*n*2)

Algoritmos y Complejidad

Notación Asintótica

Ejemplos

NÚMERO DE FIBONACCI

Algoritmo: algoritmo iterativo simple I para analizar el *O*(*·*) se tiene:

*n*

*TFIB*2(*n*) = *b* +

I luego *TFIB*2(*n*) *∈ O*(*n*)

∑ *k*=1

(*c*1 +*c*2 +*c*3) +*d ≤ fn*

I para analizar el Ω(*·*) se tiene:

*n*

*n*

*TFIB*2(*n*) = *b* +

∑ *k*=1

(*c*1 +*c*2 +*c*3) +*d ≥*

∑ *k*=1

(*c*1 +*c*2 +*c*3) *≥ n*

I luego *TFIB*2(*n*) *∈* Ω(*n*), y por lo tanto también *TFIB*2(*n*) *∈* Θ(*n*)

Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

Notación Asintótica Condicional

I muchos algoritmos son más fáciles de analizar si se restringe la atención a instancias cuyos tamaños satisfacen determinadas condiciones

*O*(*g*(*n*) *| P*(*n*)) = *{ f*(*n*) : *∃c ∈* **R**+*∃n*0 *∈* **N***,* tal que

*f*(*n*) *≤ cg*(*n*)para todo *n ≥ n*0

siempre que *P*(*n*) *}*

I análogamente, se definen Ω(*g*(*n*) *| P*(*n*)) notación omega condicional, y Θ(*g*(*n*) *| P*(*n*)) notación Θ condicional

Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

I por ejemplo, *t*(*n*) *∈* Θ(*n*2*| n* = 2*k*) significa que si *n* es potencia de 2 entonces *t*(*n*) *∈* Θ(*n*2)

I nada se está afirmando sobre *t*(*n*) si *n* no es potencia de 2

Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

NÚMERO DE FIBONACCI Algoritmo: algoritmo iterativo complejo log *n*

log *n*

*TFIB*3(*n*) = *c*1+

I si *n* = 2*k* entonces

∑ *k*=1

*c*2+

∑ *k*=1

*c*3

*TFIB*3(*n*) = *c*1+

log *n* ∑

*k*=1

*c*3 *≤ d* log *n*

I y entonces *TFIB*3(*n*) *∈ O*(log *n | n* = 2*k*)



Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

I si *n* = 2*k −*1 entonces

*TFIB*3(*n*) = *c*1+

log *n* ∑

*k*=1

(*c*2+*c*3) *≤ e* log *n*

I y entonces *TFIB*3(*n*) *∈ O*(log *n | n* = 2*k −*1)

I analizando que este último caso es el peor de los casos posible se puede concluir que *TFIB*3(*n*) *∈ O*(log *n*)



Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

Regla de las Funciones de Crecimiento Suave

I sirve para extender lo analizado condicionalmete a todos los tamaños de entrada

Teorema 1

*Sea f* : **N** *−→* **R**+ *una función de crecimiento suave, y t* : **N** *−→* **R**+ *una función eventualmente no decreciente. Luego siempre que t*(*n*) *∈* Θ(*f*(*n*) *| n* = *bk*) *para algún entero b ≥* 2*, entonces*

*t*(*n*) *∈* Θ(*f*(*n*))



Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

Definición

*una función f*(*n*) : **N** *−→* **R**+ *es eventualmente no decreciente si existe n*0 *∈* **N** *tal que para todo n ≥ n*0 *vale f*(*n*) *≤ f*(*n*+1)

Definición

*una función f*(*n*) : **N** *−→* **R**+ *es de crecimiento suave si existe b ∈* **N***,b ≥* 2 *tal que f*(*n*) *es eventualmente no decreciente y*

*f*(*bn*) *∈ O*(*f*(*n*))



Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

I la mayoría de las funciones que se encuentran son de

crecimiento suave: log *n*, *n*, *n* log *n*, *n*2, o cualquier polinomio con coeficiente principal positivo

I funciones tales como *n*log *n*, 2*n* o *n*! no son de crecimiento suave I reglas análogas también son válidas para *O*(*·*) y Ω(*·*)



Algoritmos y Complejidad

Notación Asintótica

Notación Asintótica Condicional

Ejemplo

I si *t*(*n*) es

*t*(*n*) =

*a* si *n* = 1 4*t*(*dn/*2*e*) +*bn* sino

I entonces es fácil probar (usando los métodos de resolución de recurrencias que se verán) que *t*(*n*) = (*a*+*b*)*n*2 *−bn* si *n* = 2*k*, ie *t*(*n*) *∈* Θ(*n*2*| n* = 2*k*)

I luego, como *n*2 es una función de crecimiento suave y *t*(*n*) es eventualmente no decreciente (¿porqué?) se puede aplicar la regla de las funciones de crecimiento suave y concluir que *t*(*n*) *∈* Θ(*n*2) para todo *n*

**

Algoritmos y Complejidad

Estructuras de Datos

I es necesario un manejo fluído de las siguientes estructuras de datos:

I Arreglos y Matrices

I Listas simplemente enlazadas, Pilas y Colas

I Grafos, implementados mediante matriz o lista de adyacencia

I árboles

I Tablas Asociativas (*Hash*)

I Colas con Prioridad (*Heaps*), implementados por árboles binarios completos

I Conjuntos Disjuntos



Algoritmos y Complejidad

Estructuras de Datos

ORDENAR UN ARREGLO

Algoritmo: Heapsort (ordenamiento por construcción de un *heap*)

costo veces

FUNCTION Heapsort(A)

Construir Heap(A) FOR i ::= n DOWNTO 2 A[1] <=> A[i]

A.tamaño- -

A.heapify(1)

ENDFOR

Θ(*n*) 1

*c*1 ∑*ni*=2 1

*c*2 ∑*ni*=2 1

*c*3 ∑*ni*=2 1

Θ(log *n*) ∑*ni*=2 1



Algoritmos y Complejidad

Estructuras de Datos

I calculando el tiempo de ejecución se tiene: *n*

*TH* (*n*) = Θ(*n*) +

∑ *i*=2

(*c*1 +*c*2 +*c*3 + Θ(log *n*)) =

= Θ(*n*) + Θ(*n* log *n*) *∈* Θ(*n* log *n*)

I es fundamental en este ejemplo usar la implementación más eficiente para las operaciones de la estructura de datos



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Secuencia

Secuencia

Algoritmo A

P1

P2

I sea *tA*(*n*) la cantidad de recursos a analizar.

I si *P*1 insume Θ(*f*1(*n*)) recursos y *P*2 insume Θ(*f*2(*n*)) recursos, entonces

*tA*(*n*) = Θ(*f*1(*n*)) + Θ(*f*2(*n*)) = Θ(*f*1(*n*) +*f*2(*n*)) =

= Θ(max(*f*1(*n*)*,f*2(*n*)))



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Condicional

Condicional

Algoritmo A

IF (X)

P1

ELSE

P2

ENDIF

I el tiempo en el peor de los casos es

*tA*(*n*) = *tX* (*n*) +max(*tP*1(*n*)*,tP*2(*n*)) =

= *O*(max(*tX* (*n*)*,tP*1(*n*)*,tP*2(*n*)))



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Condicional

I y también

*tA*(*n*) *≥ c* +max(Θ(*f*1(*n*))*,*Θ(*f*2(*n*))) =

= Ω(max(*c,f*1(*n*)*,f*2(*n*)))

I si el *O*(*·*) y el Ω(*·*) coinciden, entonces se puede definir el Θ(*·*)

Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

Ciclo FOR

Algoritmo B

FOR i ::= 1 TO m

P(i)

ENDFOR

I sean *c*1, *c*2, *c*3 los costos de las operaciones elementales

I si *P*(*i*) insume *t* recursos (no depende de *i* ni de *m*) entonces

*tB*(*n*) = *c*1 + (*m* +1)*c*3 + *mt* + *mc*2 =

= (*c*2 +*c*3 +*t*)*m* + (*c*1 +*c*3) *∈* Θ(*mt*)



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

I si *P*(*i*) insume *t*(*i*) recursos (dependiendo de *i*, del tamaño *n* de la instancia, o de cada instancia en particular) entonces

*m*

*tB*(*n*) =

∑ *i*=1

*t*(*i*)

I para obtener el *O*(*·*) o el Ω(*·*) de esta función se pueden usar las distintas propiedades vistas para obtener la notación asintótica



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

ORDENAR UN ARREGLO Algoritmo: Ordenamiento por Selección

FOR i ::= 1 TO n-1

ind ::= i; min ::= A[i]

costo veces *a n*

*b n−*1 *i*=1 ∑*n*+1

FOR j ::= i+1 TO n

*c* ∑*n−*1 *c* ∑*n−*1

*j*=*i*+11

IF (A[j]<min)

min ::= A[j]

ind ::= j

ENDIF

ENDFOR

A[ind] ::= A[i]; A[i] ::= min

*i*=1 ∑*nj*=*i*+1 1

*c* ∑*n−*1

*i*=1 ∑*nj*=*i*+1 1

*c* ∑*n−*1

*i*=1 ∑*nj*=*i*+1 1

*b n−*1

ENDFOR 

Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

I calculando la cantidad de recursos se tiene *n−*1

*TS*(*n*) = *a*+

∑ *i*=1 *n−*1

(*a*+*b* + (*n−i*)*c*) *n−*1

= *a*+

∑ *i*=1

(*a*+*b* +*cn*)*−c*

∑ *i*=1

*i* =

= *a*+ (*n−*1)(*a*+*b* +*cn*)*−cn*(*n−*1)*/*2

= *a*+*cn*2

2+ (*a*+*b −c*2)*n−*(*a*+*b*) *∈* Θ(*n*2)



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

NÚMERO DE FIBONACCI

Algoritmo: primer algoritmo iterativo (sumas y restas son operaciones elementales)

costo veces

Function FIB2(n)

i ::= 1; j ::= 0 FOR k ::= 1 TO n j ::= i+j

i ::= j-i

ENDFOR

RETURN j

*b* 1

*c*1 ∑*n*+1

*k*=11

*c*2 ∑*ni*=1 1

*c*3 ∑*ni*=1 1

*d* 1



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

I calculando el tiempo de ejecución se tiene *n*

*TFIB*2(*n*) = *b* +*c*1 +

∑ *k*=1

(*c*1 +*c*2 +*c*3) +*d* =

= (*b* +*d*) + (*c*1 +*c*2 +*c*3)*n ∈* Θ(*n*)



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclo FOR

I si la suma y la resta no son operaciones elementales (operan sobre números muy grandes) entonces

costo veces

Function FIB2(n)

i ::= 1; j ::= 0 FOR k ::= 1 TO n j ::= i+j

i ::= j-i

ENDFOR

RETURN j

*b* 1

*c*1 ∑*nk*=1 1

*c*2 *∗ tamano*˜ (*j*) ∑*ni*=1 1

*c*3 *∗ tamano*˜ (*j*) ∑*ni*=1 1

*d* 1



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control Ciclo FOR

I resultando

*TFIB*2(*n*) = *b* +

*n*

∑ *k*=1

(*c*1 + (*c*2 +*c*3) *∗ tamano*˜ (*j*)) +*d* = *n*

*≤* (*b* +*d*) +

∑ *k*=1

(*c*1 + (*c*2 +*c*3) *∗ tamano*˜ (*Fk* )) *n*

*≤* (*b* +*d*) +*nc*1 +

∑ *k*=1

*dk* por *tamano*˜ (*Fk* ) *∈* Θ(*k*)

= (*b* +*d*) +*nc*1 +*dn*(*n*+1)

2=

= (*b* +*d*) +*nc*1 +*d*2*n*2 +*d*2*n ∈ O*(*n*2)



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclos WHILE y REPEAT

Ciclos WHILE y REPEAT

I no es tan fácil para los casos de ciclos **repeat** o **while**, no se sabe cuántas veces serán ejecutados.

I algunas de las técnicas a aplicar pueden ser:

1. encontrar una función en las variables involucradas cuyo valor decrezca en cada iteración, y que sea siempre positiva

2. tratar la iteración como si fuese un procedimiento recursivo, y aplicar el método para recursividad

3. elegir como cota del cuerpo del bucle el tiempo de ejecución de una de sus sentencias, la cual se denomina barómetro. Luego se debe contar cuántas veces se ejecuta el barómetro

I ningún método es aplicable para todos los casos, y solo a través

de la experiencia se puede detectar cuál usar



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclos WHILE y REPEAT

MÁXIMO COMÚN DIVISOR

Algoritmo: Algoritmo de Euclides

Function EUCLIDES(m,n)

WHILE m>0

temp ::= m

m ::= n mod m

n ::= temp

ENDWHILE

RETURN n



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclos WHILE y REPEAT

I se puede observar las propiedades:

I *ni* = *mi−*1, *mi* = *ni−*1 mod *mi−*1 siempre que *i ≥* 1

I *ni ≥ mi* siempre que *i >* 1

I para todo *n,m* tal que *n ≥ m* vale *n* mod *m < n/*2

I *ni* = *mi−*1 = *ni−*2 mod *mi−*2 *< ni−*2*/*2 si *i >* 2

I luego, en dos iteraciones *n*0 se reduce a menos de la mitad; en cuatro a menos del cuarto; etc. Como *mi >* 0 entonces no puede haber más de 2log2 *n*0 iteraciones. Y *T*(*n*) *∈ O*(log *n*).



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclos WHILE y REPEAT

ORDENAR UN ARREGLO

Algoritmo: Ordenamiento por Cubículos (enteros hasta *s*)

array U[1..s] ::= 0

FOR i ::= 1 TO n

k ::= T[i]; U[k]++

ENDFOR

i ::= 0

FOR k ::= 1 TO s

WHILE U[k] != 0

T[i++] ::= k; U[k]- - ENDWHILE

ENDFOR

Θ(*n*)

Θ(1)

barómetro

Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Ciclos WHILE y REPEAT

I el barómetro se ejecuta *U*[*k*]0 +1 veces por cada *k* I luego el tiempo total es: ∑*sk*=1(*U*[*k*]0 +1)Θ(1) I y vale

*s*

*T*(*n*) = Θ(*n*) + Θ(1) + *s*

∑ *k*=1

(*U*[*k*]0 +1)Θ(1) = *s*

= Θ(*n*) +

∑ *k*=1

*U*[*k*]0Θ(1) +

∑ *k*=1

Θ(1)

= Θ(*n*) + Θ(*n*) + Θ(*s*) *∈* Θ(max(*n,s*))

I el problema de este algoritmo es el límite máximo de los números a utilizar, y el espacio de memoria auxiliar



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Recursividad

I una simple inspección del algoritmo da origen a una recurrencia, que “simula” el flujo de control del algoritmo

function F(n)

IF (x)

P1(n)

ELSE

P2(n)

F(m); % con m<n

ENDIF

I *t*(*n*) *∈ O*(*max*(*tX* (*n*)*,tP*1(*n*)*,tP*2(*n*) +*t*(*m*)))

I luego se debe aplicar algún método para resolver la recurrencia

Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Recursividad

ELEMENTO MAYOR

Function MAXIMO(T)

IF n=1

RETURN T[1]

ELSE

x ::= MAXIMO(T[1..n-1])

IF (x>T[n])

RETURN x

ELSE

RETURN T[n]

ENDIF

ENDIF



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Recursividad

I genera la siguiente recurrencia:

*TMAX* (*n*) =

*a* si *n* = 1. *b* +*TMAX* (*n−*1) si *n >* 1



Algoritmos y Complejidad

Análisis de Algoritmos por Estructuras de Control

Recursividad

NÚMERO DE FIBONACCI

Algoritmo: algoritmo recursivo

function FIB1(n)

IF n<2

RETURN n

ELSE

RETURN (FIB1(n-1)+FIB1(n-2))

ENDIF

I que genera la recurrencia

*TFIB*1(*n*) =

*a* si *n <* 2 *TFIB*1(*n−*1) +*TFIB*1(*n−*2) +*b* si *n ≥* 2



Algoritmos y Complejidad

Resolución de Recurrencias

I veremos dos técnicas básicas y una auxiliar que se aplican a diferentes clases de recurrencias:

Técnicas de Resolución de Recurrencias



método del teorema maestro

método de la ecuación característica 

cambio de variables

I no analizaremos su demostración formal, sólo consideraremos su aplicación para las recurrencias generadas a partir del análisis

de algoritmos



Algoritmos y Complejidad

Resolución de Recurrencias

Método del Teorema Maestro

Método del Teorema Maestro I se aplica en casos como:

*T*(*n*) =

5 si *n* = 0 9*T*(*n/*3) +*n* si *n 6*= 0

I es importante identificar:

I la cantidad de llamadas recursivas

I el cociente en el que se divide el tamaño de las instancias I la sobrecarga extra a las llamadas recursivas



Algoritmos y Complejidad

Resolución de Recurrencias

Método del Teorema Maestro

Teorema 2

*Sean a ≥* 1*, b >* 1 *constantes, f*(*n*) *una función y T*(*n*) *una*

*recurrencia definida sobre los enteros no negativos de la forma T*(*n*) = *aT*(*n/b*) +*f*(*n*)*, donde n/b puede interpretarse como bn/bc o dn/be Entonces valen:*

1. *si f*(*n*) *∈ O*(*n*log*b a−ε*) *para algún ε >* 0 *entonces*

*T*(*n*) *∈* Θ(*n*log*b a*)*.*

2. *si f*(*n*) *∈* Θ(*n*log*b a*) *entonces T*(*n*) *∈* Θ(*n*log*b a*lg *n*)*.*

3. *si f*(*n*) *∈* Ω(*n*log*b a*+*ε*) *para algún ε >* 0*, y satisface*

*af*(*n/b*) *≤ cf*(*n*) *para alguna constante c <* 1*, entonces*

*T*(*n*) *∈* Θ(*f*(*n*))*.*

**

Algoritmos y Complejidad

Resolución de Recurrencias

Método del Teorema Maestro

Ejemplos:

1. si *T*(*n*) = 9*T*(*n/*3) +*n* entonces *a* = 9, *b* = 3, se aplica el caso 1 con *ε* = 1 y *T*(*n*) *∈* Θ(*n*2)

2. si *T*(*n*) = *T*(2*n/*3) +1 entonces *a* = 1, *b* = 3*/*2, se aplica el caso 2 y *T*(*n*) = Θ(lg *n*)

3. si *T*(*n*) = 3*T*(*n/*4) +*n* lg *n* entonces *a* = 3, *b* = 4,

*f*(*n*) *∈* Ω(*n*log4 3+0*,*2) y 3(*n/*4)lg(*n/*4) *≤* 3*/*4*n* lg *n*, por lo que se aplica el caso 3 y *T*(*n*) *∈* Θ(*n* lg *n*)

4. si *T*(*n*) = 2*T*(*n/*2) +*n* lg *n*, no se puede aplicar el caso 3 porque *f*(*n*) = *n* lg *n 6∈* Ω(*n*1+*ε*) para cualquier *ε >* 0



Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica

Método de la Ecuación Característica

I se aplica a ciertas recurrencias lineales con coeficientes constantes como:

*T*(*n*) =



5 si *n* = 0 10 si *n* = 1 

5*T*(*n−*1) +8*T*(*n−*2) +2*n* si *n >* 1

I en general, para recurrencias de la forma:

*T*(*n*) = *a*1*T*(*n−*1) +*a*2*T*(*n−*2) +*···*+*akT*(*n−k*) +*bnp*(*n*) donde *ai,*1 *≤ i ≤ k,b* son constantes y *p*(*n*) es un polinomio en

*n* de grado *s*

**

Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica

Ejemplos:

I en *t*(*n*) = 2*t*(*n−*1) +3*n*, *a*1 = 2, *b* = 3, *p*(*n*) = 1, *s* = 0

I en *t*(*n*) = *t*(*n−*1) +*t*(*n−*2) +*n*, *a*1 = 1, *a*2 = 1, *b* = 1, *p*(*n*) = *n*, *s* = 1



Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica

I para resolver la recurrencia

*T*(*n*) = *a*1*T*(*n−*1) +*a*2*T*(*n−*2) +*···*+*akT*(*n−k*) +*bnp*(*n*): 1. encontrar las raíces no nulas de la ecuación característica:

(*xk −a*1*xk−*1 *−a*2*xk−*2 *−··· −ak* )(*x −b*)*s*+1 = 0

Raíces: *ri,*1 *≤ i ≤ l ≤ k*, cada una con multiplicidad *mi*.

2. las soluciones son de la forma de combinaciones lineales de estas raíces de acuerdo a su multiplicidad

*T*(*n*) =

*l*∑ *i*=1

*mi*

∑ *j*=1

*cijnj−*1*rni*

3. si se necesita, se encuentran valores para las constantes *cij,*1 *≤ i ≤ l,*0 *≤ j ≤ mi −*1 y *di,*0 *≤ i ≤ s −*1 según la recurrencia original y las condiciones iniciales (valores de la recurencia para *n* = 0*,*1*,...*)

Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica

Ejemplo:

*T*(*n*) =

0 si n=0 2*T*(*n−*1) +1 si *n >* 0

1. si *b* = 1 y *p*(*n*) = 1 de grado 0, la ecuación característica (*x −*2)(*x −*1)0+1 = 0, con *r*1 = 2*,m*1 = 1 y *r*2 = 1*,m*2 = 1 2. la solución general es de la forma *T*(*n*) = *c*112*n* +*c*211*n*. 3. a partir de las condiciones iniciales se encuentra:

*c*11 +*c*21 = 0 de *n* = 0

2*c*11 +*c*21 = 1 de *n* = 1

de donde *c*11 = 1 y *c*21 = *−*1. 

4. la solución es *T*(*n*) = 2*n −*1

Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica Ejemplo:

*T*(*n*) =

*n* si n=0,1,2 5*T*(*n−*1)*−*8*T*(*n−*2) +4*T*(*n−*3) si *n >* 2

1. si *b* = 0 y *p*(*n*) = 1, la ecuación característica es *x*3 *−*5*x*2 +8*x −*4 = 0, con *r*1 = 1*,m*1 = 1 y *r*2 = 2*,m*2 = 2 2. la solución general es de la forma *T*(*n*) = *c*111*n* +*c*212*n* +*c*22*n*2*n*. 3. a partir de las condiciones iniciales se obtienen:

*c*11 +*c*21 = 0 de *n* = 0

*c*11 +2*c*21 +2*c*22 = 1 de *n* = 1

*c*11 +4*c*21 +8*c*22 = 2 de *n* = 2

de donde *c*11 = *−*2, *c*21 = 2 y *c*22 = *−*1*/*2 

4. la solución es entonces la función *T*(*n*) = 2*n*+1 *−n*2*n−*1 *−*2

Algoritmos y Complejidad

Resolución de Recurrencias

Método de la Ecuación Característica

Ejemplo: número de Fibonacci

*F*(*n*) =

*n* si n=0,1 *F*(*n−*1) +*F*(*n−*2) si *n ≥* 2

1. si *b* = 0 y *p*(*n*) = 1, la ecuación característica es *x*2 *−x −*1 = 0, 2y *φ*ˆ =1*−√*5

con raíces *φ* =1+*√*5

2

2. la solución general es *F*(*n*) = *c*11*φn* +*c*21*φ*ˆ*n*. 3. a partir de las condiciones iniciales se obtienen:

*c*11 +*c*21 = 0 de *n* = 0

*c*11*φ* +*c*21*φ*ˆ = 1 de *n* = 1

cuyas soluciones son *c*11 = 1*/√*5 y *c*21 = *−*1*/√*5 4. la solución es *F*(*n*) = *√*15(*φn −φ*ˆ*n*).

Algoritmos y Complejidad

Resolución de Recurrencias

Cambio de Variable

Cambio de Variable

I por ejemplo para la recurrencia

*T*(*n*) =

*a* si n=1 2*T*(*n/*2) +*n* log2 *n* sino

no se puede ninguno de los dos métodos anteriores I se define una nueva recurrencia *S*(*i*) = *T*(2*i*), con el objetivo de llevarla a una forma en la que se pueda resolver siguiendo algún método anterior

I el caso general queda

*S*(*i*) = *T*(2*i*) = 2*T*(2*i/*2) +2*ii* = 2*T*(2*i−*1) +*i*2*i* = 2*S*(*i −*1) +*i*2*i* con *b* = 2 y *p*(*i*) = *i* de grado 1

Algoritmos y Complejidad

Resolución de Recurrencias

Cambio de Variable

I la ecuación característica de esta recurrencia es

(*x −*2)(*x −*2)1+1 = 0 con raíz 2 de grado 3

I la solución es entonces *S*(*i*) = *c*112*i* +*c*12*i*2*i* +*c*13*i*22*i*

I volviendo a la variable original queda

*T*(*n*) = *c*11*n*+*c*12(log2 *n*)*n*+*c*13(log2 *n*)2*n*.

I se pueden obtener los valores de las constantes sustituyendo esta solución en la recurrencia original:

*T*(*n*)*−*2*T*(*n/*2) = *n* log2 *n* = (*c*12 *−c*13)*n*+2*c*13*n*(log2 *n*)

de donde *c*12 = *c*13 y 2*c*12 = 1



Algoritmos y Complejidad

Resolución de Recurrencias

Cambio de Variable

I por lo tanto *T*(*n*) *∈* Θ(*n* log2*n | n* es potencia de 2)

I si se puede probar que *T*(*n*) es eventualmente no decreciente, por la regla de las funciones de crecimiento suave se puede

extender el resultado a todos los *n* (dado que *n* log2*n* es de

crecimiento suave). En este caso *T*(*n*) *∈* Θ(*n* log2*n*)

