



estudios de postgrado
en computación



Análisis y Diseño de Algoritmos (AyDA)

Isabel Besembel Carrera

TÉCNICAS DE PRUEBA Y CORRECCIÓN DE ALGORITMOS



Unidad 1. Tema 3.

Técnicas de pruebas y corrección de algoritmos

- ⦿ Pruebas de corrección parcial
- ⦿ Pruebas de parada
- ⦿ Pruebas de programas iterativos y recursivos
- ⦿ Eliminación de la recursividad
- ⦿ Objetivo:
 - Desarrollar habilidades en el uso de las técnicas de prueba y corrección de algoritmos

Prueba de algoritmos

- ◉ Un algoritmo A define una secuencia de pasos que se aplica a cualquier instancia del problema definida por los datos L sobre un conjunto de resultados R
 - $\forall d \in L$, A puede definir o bien una secuencia de cálculo finito que da el resultado $A(d) \in R$ o bien una secuencia infinita
- ◉ A se escribe para calcular una función f de $D \subset L$
- ◉ A es correcto $\Leftrightarrow A$ calcula bien f

$$\forall d \in D, A(d) = f(d)$$

- ◉ Método usual de pruebas: enfoque empírico

Se escoge un conjunto finito de datos d_1, d_2, \dots, d_n , se ejecuta A para cada uno y se verifica $A(d_1)=f(d_1), A(d_2)=f(d_2), \dots, A(d_n)=f(d_n)$

Prueba de algoritmos

- ⊙ El método es insuficiente pues no recubre D
- ⊙ El método **no** prueba que A es correcto, sólo prueba que **A es incorrecto**, si para algún d_i , $A(d_i) \neq f(d_i)$

Ejemplo: a y b son iniciados en $A \in \mathbb{N}$ y $B \in \mathbb{N}^+$

Marzo/05		divisionEntera(Entero: a, b, c, r)
{ pre: $a=A \wedge b=B \wedge A \geq 0 \wedge B > 0$ }		{ pos: $A=Bc+r \wedge c \geq 0 \wedge r \geq 0 \wedge r < B$ }
1	r, c = a, 0	a, b, c, r: Entero. Numerador, denominador, cociente y residuo, respectivamente.
2	(r ≥ b) [r, c = r - b, c + 1]	
1	a = 0, b = 1 => c = 0, r = 0	Condición de borde en ambas entradas
2	a = 0, b = 17 => c = 0, r = 0	Condición de borde en a
3	a = 42, b = 17 => c = 2, r = 8	Entradas diferentes a la condición de borde
4	a = 34, b = 1 => c = 34, r = 0	Condición de borde en b

Prueba de algoritmos

- ⊙ $D = \mathbb{N} \times \mathbb{N}^+$, $R = \mathbb{N} \times \mathbb{N}$, $f: D \rightarrow R$ que asocia a cada par (a, b) de $\mathbb{N} \times \mathbb{N}^+$ un par (c, r) de $\mathbb{N} \times \mathbb{N}$ / $(a = bc + r) \wedge (r < b)$
- ⊙ La prueba de f se divide en 2 partes:
 - **Prueba de corrección parcial:** demuestra que `divisionEntera` termina y que el resultado es correcto
 - **Prueba de parada:** demuestra que `divisionEntera` termina para todo $\mathbb{N} \times \mathbb{N}^+$

Prueba de algoritmos

⊙ Prueba de corrección parcial

$E \{A\} S$ donde E, S son condiciones y A es una secuencia de sentencias. $\langle E, A, S \rangle$ tripleta Hoare.

E : precondición

S : postcondición

E, S son aserciones o condiciones que pueden tomar valores lógicos ligados con operadores lógicos $\{\neg, \wedge, \vee, \rightarrow, \sim\}$

Implica: $a \rightarrow b \sim \neg a \vee b$

Aserciones: son sentencias lógicas sobre el estado de un programa que se consideran verdaderas

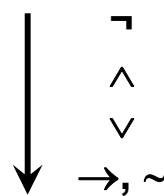
	a		b
implica	V	F	
V	V	F	
F	V	V	

Prueba de algoritmos

Equivalencia: $(a \sim b) \sim (a \rightarrow b) \wedge (b \rightarrow a)$

		b	
		V	F
a	equivalencia	V	F
	V	V	F
	F	F	V

⊙ Jerarquía de precedencia



⊙ Axiomas y reglas para las pruebas de corrección parcial

- Axiomas y teoremas son de la forma $E \{A\} S$
 1. Regla de la precondition: si $E \{A\} S$ y $E' \rightarrow E \Rightarrow E' \{A\} S$
 2. Regla de la postcondición: si $E \{A\} S$ y $S' \rightarrow S \Rightarrow E \{A\} S'$
 3. Regla del y-lógico (\wedge): si $E \{A\} S$ y $E \{A\} S' \Rightarrow E \{A\} S \wedge S'$
 4. Regla del o-lógico (\vee): si $E \{A\} S$ y $E' \{A\} S \Rightarrow E \vee E' \{A\} S$
 5. Regla de la composición de las sentencias (;): si $E \{A\} F$ y $F \{B\} S \Rightarrow E \{A; B\} S$

\Rightarrow Se lee entonces

Prueba de algoritmos

⊙ Axioma de la asignación:

- $x = \text{exp}$ y una postcondición S se tiene el axioma $E \{x = \text{exp}\}S$, donde E se obtiene a partir de S por substitución de todas las ocurrencias de x por la exp

⊙ E es la condición que deben satisfacer las variables antes de la ejecución de la asignación para que S sea verdadera

Ejemplos:

$$(x \ y \geq 0) \{z = x * y\} (z \geq 0)$$

$$(q + 1 \geq 0) \{q = q + 1\} (q \geq 0)$$

$$((x + y)^2 = y) \{x = x + y\} (x^2 = y)$$

Se puede encontrar E dado S , pero la inversa no es posible

Prueba de algoritmos

⊙ Dado $E = S (x/exp)$

$$S(x/si a entonces e_1 sino e_2) = (a \wedge S(x/e_1)) \vee (\neg a \wedge S(x/e_2))$$

Probar que

$$(z \geq 0) \{ z = si b > 0 entonces z + b sino z - b \} (z \geq 0)$$

⊙ Regla de las decisiones

1. Si $E \wedge B \{ A \} S$ y $E \wedge \neg B \rightarrow S \Rightarrow E \{ si B entonces A \} S$
2. Si $E \wedge B \{ A \} S$ y $E \wedge \neg B \{ P \} S \Rightarrow E \{ si B entonces A sino P \} S$

La evaluación de B no modifica los valores de las variables del bloque A

Prueba de algoritmos

- Regla del lazo (repita mientras)

(condición)[bloque de sentencias a repetir]

Si $E \wedge B\{A\}E \Rightarrow E\{\text{mientras } B \text{ haga } A\} E \wedge \neg B$

La condición E se denomina la invariante del lazo

No hay recetas para encontrar buenas invariantes

- Regla de la sentencia compuesta

Si $E \{A\} S \Rightarrow E\{\text{inicio } A \text{ fin}\} S$

Prueba de algoritmos

⊙ Prueba de la corrección parcial del algoritmo divisionEntera

Se quiere demostrar que

$$(a=A) \wedge (b=B) \wedge (A \geq 0) \wedge (B > 0) \{ \text{divisionEntera} \} (A=Bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < B)$$

Se tiene que $(a=A) \wedge (b=B) \wedge (A \geq 0) \wedge (B > 0) \rightarrow (a=A) \wedge (b=B) \wedge (a \geq 0) \wedge (b > 0)$

$$\Upsilon (a=A) \wedge (b=B) \wedge (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < b) \rightarrow (A=Bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < B)$$

Por las reglas de la pre y postcondición es suficiente demostrar que

$$(a=A) \wedge (b=B) \wedge (A \geq 0) \wedge (B > 0) \{ \text{divisionEntera} \} (a=A) \wedge (b=B) \wedge (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$$

Demostración en dos etapas

$$1) (a=A) \wedge (b=B) \{ \text{divisionEntera} \} (a=A) \wedge (b=B)$$

$$2) (a \geq 0) \wedge (b > 0) \{ \text{divisionEntera} \} (a=bc+r) \wedge (c \geq 0) \wedge (r > 0) \wedge (r < b)$$

1) Aplicando las reglas sucesivamente se demuestra que $(a=A) \wedge (b=B) \{ \text{divisionEntera} \} (a=A) \wedge (b=B)$, pues divisionEntera no altera las entradas

Prueba de algoritmos

2) Se quiere demostrar que

$$(a \geq 0) \wedge (b > 0) \{ \text{divisionEntera} \} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < b)$$

Invariante del lazo $(a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$

Por la regla de la asignación

$$(a=b(c+1)+r) \wedge (c+1 \geq 0) \wedge (r \geq 0) \{ c=c+1 \} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \quad \text{y}$$

$$(a=b(c+1)+r-b) \wedge (c+1 \geq 0) \wedge (r-b \geq 0) \{ r=r-b \} (a=b(c+1)+r) \wedge (c+1 \geq 0) \wedge (r \geq 0)$$

Por la regla de la composición de sentencias se tiene el

enunciado 1:

$$(a=b(c+1)+r-b) \wedge (c+1 \geq 0) \wedge (r-b \geq 0) \{ r=r-b; c=c+1 \} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$$

Prueba de algoritmos

Estudiando la precondition obtenida

$$(a=b(c+1)+r-b) \wedge (c+1 \geq 0) \wedge (r-b \geq 0) \sim (a=bc+r-b) \wedge (c \geq -1) \wedge (r \geq b)$$

Ya que $(c \geq 0) \rightarrow (c \geq -1)$ y $(r \geq 0) \wedge (r \geq b) \rightarrow (r \geq b)$

Se tiene que $(a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r \geq b) \rightarrow (a=bc+r-b) \wedge (c \geq -1) \wedge (r \geq b)$

El enunciado 1 pasa a ser (por la regla de la precondition)

$$(a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r \geq b) \{r=r-b ; c=c+1\} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$$

Se deduce el enunciado 2, por la regla del lazo

$$(a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \{\text{mientras } r \geq b \text{ haga inicio } r=r-b ; c=c+1 \text{ fin}\}$$
$$(a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < b)$$

Prueba de algoritmos

Se usa la invariante del lazo como postcondición para las primeras sentencias de la secuencia, por la regla de la asignación

$(a=b \cdot 0+r) \wedge (0 \geq 0) \wedge (r \geq 0) \{c=0\} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$ y
 $(a=b \cdot 0+r) \wedge (0 \geq 0) \wedge (r \geq 0) \sim (a=r) \wedge (r \geq 0)$

Se aplica de nuevo la regla de la asignación

$(a=a) \wedge (a \geq 0) \{r=a\} (a=r) \wedge (r \geq 0)$ y $(a=a) \wedge (a \geq 0) \sim (a \geq 0)$

Por la regla de la composición, se tiene el enunciado 3

$(a \geq 0) \{r=a ; c=0\} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0)$

Por la regla de la composición aplicada a los enunciados 2 y 3, se tiene

$(a \geq 0) \{divisionEntera\} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < b)$

Y la regla de la precondition permite concluir que

$(a \geq 0) \wedge (b > 0) \{divisionEntera\} (a=bc+r) \wedge (c \geq 0) \wedge (r \geq 0) \wedge (r < b)$

Prueba de algoritmos

⊙ Prueba de parada

Un algoritmo sin recursividad y ni bifurcaciones incondicionales, no puede producir secuencias de cálculo infinitas, a menos que ejecute infinitas veces el cuerpo del lazo

Probar que tal algoritmo termina para toda instancia del problema con $d \in D$, es probar que cada lazo del algoritmo no puede ser ejecutado sino un número finito de veces para todo $d \in D$.

Sea un lazo repita mientras, X_1, X_2, \dots, X_n las variables del algoritmo y W_C el conjunto de los valores del vector $w = [X_1, X_2, \dots, X_n]$ tal que dichas variables verifican una condición C

Sea E una condición invariante para el lazo considerado y satisfecha antes de su ejecución

Prueba de algoritmos

Sea w_1 que pertenece a W_E , el valor de w antes de la ejecución del lazo.

Si w_1 pertenece a $W_{E \wedge \neg B}$, el lazo termina.

Sino w_1 pertenece a $W_{E \wedge B}$ y luego de la ejecución de A , w tiene un valor w_2 .

Si w_2 pertenece a $W_{E \wedge \neg B}$, el lazo termina.

Sino luego de la ejecución de A , w tiene un valor w_3 , etc.

Para probar la terminación del lazo, es necesario demostrar que toda secuencia w_1, w_2, \dots, w_n , es finita

Un método para demostrarlo consiste en definir una aplicación m , de $W_{E \wedge B}$ en \aleph , tal que se pueda mostrar que

Prueba de algoritmos

$$E \wedge B \wedge (m(w) = m_0) \{A\} \neg B \vee (m(w) < m_0) \quad \forall m_0 \in \aleph$$

Esto indica que si w tienen un valor w_i antes de la ejecución de A , luego de la ejecución de A , o bien el lazo termina o bien w tiene un valor w_{i+1} tal que $m(w_{i+1}) < m(w_i)$

Para toda secuencia w_1, w_2, \dots, w_n la secuencia $m(w_1), m(w_2), \dots$ es estrictamente decreciente en \aleph , y por ello termina

Lo que implica que w_1, w_2, \dots es finita y que el lazo termina para todo valor w que satisfaga E

Prueba de algoritmos

⊙ Prueba de parada para divisionEntera

Tiene un solo lazo $(r \geq b) [r, c = r - b, c + 1]$

La condición $(r \geq 0)$ está en la invariante del lazo

Se considera el conjunto $W_1 = W_{(r \geq 0) \wedge (r \geq b)}$

Sea la aplicación $m : W_1 \rightarrow \mathbb{N}$ tal que
 $m(w) = r$

Falta probar que la variable r tomó valores estrictamente decrecientes

Prueba de algoritmos

Enunciado 1:

$$(r \geq 0) \wedge (r \geq b) \wedge (r=m_0) \{r=r-b ; c=c+1\} (r < b) \vee (r < m_0) \forall m_0 \in \mathbb{N}$$

Por la regla de la asignación

$$(r-b < m_0) \{r=r-b ; c=c+1\} (r < m_0) \forall m_0 \in \mathbb{N}$$

Que conduce con la aplicación de las reglas de la precondition y de la postcondición al enunciado 2

$$(r \geq 0) \wedge (r \geq b) \wedge (r=m_0) \wedge (r-b < m_0) \{r=r-b ; c=c+1\} (r < b) \vee (r < m_0) \forall m_0 \in \mathbb{N}$$

$$\begin{aligned} \text{O: } \quad (r=m_0) \wedge (r-b < m_0) &\sim (r=m_0) \wedge (m_0-b < m_0) \\ &\sim (r=m_0) \wedge (b > 0) \end{aligned}$$

Prueba de algoritmos

La condición $b > 0$ es siempre verdadera, pues $(a, b) \in D = \mathbb{N} \times \mathbb{N}^+$, entonces
 $(r=m_0) \wedge (r-b < m_0) \sim (r=m_0)$

Lo que permite deducir directamente el enunciado 1 del enunciado 2 y se prueba la terminación del algoritmo `divisionEntera`

- ⊙ Esta prueba no se puede aplicar si $L = \mathbb{Z} \times \mathbb{Z}$, ya que `divisionEntera` entra en un lazo infinito si $B = 0$

Algoritmos iterativos

- Prueba de corrección cuando el algoritmo contiene una estructura de repetición “repita-mientras”
(condición)[bloque de sentencias a repetir]

Ejemplo 1: determinar la suma de los elementos de un vector A [a..b]

Marzo/05		
sumaElementos(Arreglo[a..b]DeTipoEle: A, Entero: a, b):TipoEle		{ pos: sum = $\sum_{j=a} A_j$ }
{ pre: a ≤ b + 1 }		
1 2 3	i, sum = a, 0 (i ≠ b+1) [sum, i = sum + A _i , i + 1] regrese sum	a, b: Entero. Valores inicial y final de los subíndices del vector A. sum: TipoEle. Acumulador de la sumatoria i: Entero. Iterador.
1 2	A(), a = 0, b = 0 => sum = NuloTipoEle A(2, 3, 4), a = 1, b = 3 => sum = 9	Condición de borde Fuera de la condición de borde

Algoritmos iterativos

- ⦿ Tanto la pre como la postcondición deben ser verdaderas
- ⦿ Por definición, $A[a..a-1]$ denota un vector vacío cuya suma es Nulo del TipoEle y el algoritmo lo calcula correctamente
- ⦿ Etapa clave de la prueba de corrección: encontrar la invariante del lazo, que es verdadera al inicio y al final del mismo
- ⦿ El inicio del lazo es el momento justo antes de evaluar la condición del “repita-mientras”
- ⦿ Pueden haber varias invariantes del lazo, pero la útil es aquella que captura la relación entre las variables que cambian de valor a medida que el lazo progresa.

Algoritmos iterativos

- ◉ La invariante del lazo para sumaElementos es $sum = \sum_{j=a}^{i-1} A_j$, que expresa la relación entre las variables i y sum
- ◉ Teorema: Al inicio de la k -ésima iteración del algoritmo sumaElementos, la condición $sum = \sum_{j=a}^{i-1} A_j$ se cumple
 - Demostración por inducción sobre k
 - **Etapa base:** $k = 1$. Al inicio de la primera iteración, se inician las variables, $i = a$ y $sum = 0$, por lo cual $0 = \sum_{j=a}^{i-1} A_j$ la condición se cumple
 - **Etapa inductiva:** hipótesis inductiva: la condición $sum = \sum_{j=a}^{i-1} A_j$ se cumple al inicio de la k -ésima iteración. Se probará que dicha condición aún se cumple luego de una iteración mas, se asume que el lazo no se termina, $i \neq b+1$.

Algoritmos iterativos

- Sean i' y sum' los valores de i y sum al inicio de la $(k+1)$ -ésima iteración. Se debe mostrar que $sum' = \sum_{j=a}^{i'-1} A_j$. Como:

$$sum' = sum + A_i$$

y

$$i' = i + 1$$

se tiene

$$sum' = sum + A_i$$

$$sum' = \sum_{j=a}^{i-1} A_j + A_i$$

$$sum' = \sum_{j=a}^i A_j$$

$$sum' = \sum_{j=a}^{i'-1} A_j$$

y la condición se cumple al inicio de la $(k+1)$ -ésima iteración.

- Establecer la invariante del lazo es siempre la parte más difícil de la prueba.

Algoritmos iterativos

⊙ La postcondición debe cumplirse al final.

⊙ Al final del lazo se tiene $sum = \sum_{j=a}^{i-1} A_j \wedge i = b+1$

se cumple, pero

$$sum = \sum_{j=a}^{i-1} A_j \wedge i = b+1 \Rightarrow sum = \sum_{j=a}^b A_j$$

que es la postcondición

⊙ Antes de terminar la ejecución del lazo (B) [...], con la invariante del lazo I, la condición $I \wedge \neg B$ se cumple, por lo que es necesario probar que implica postcondición (pos)

Algoritmos iterativos

- ⦿ La etapa final de la prueba de corrección es probar que no hay un lazo infinito.
- ⦿ El método consiste en identificar algún entero que crezca (o decrezca) estrictamente de una iteración a la siguiente y mostrar que cuando dicho entero sea suficientemente grande (o pequeño) el lazo termina

En el algoritmo sumaElementos, i es estrictamente creciente y cuando llega a $b+1$ el lazo termina

La precondition debe ser cierta al entrar al algoritmo para garantizar el fin del lazo

Algoritmos iterativos

- ⊙ Resumen, para probar que un algoritmo iterativo de la forma $(C)[X_1, \dots, X_n = E_1, \dots, E_n]$ es correcto, se siguen los pasos siguientes :
 1. Adivinar una condición I
 2. Probar por inducción que I es la invariante del lazo
 3. Probar que $I \wedge \neg C \Rightarrow \text{pos}$
 4. Probar que el lazo termina
- ⊙ La invariante del lazo involucra a todas las variables cuyos valores cambian dentro de él, pero dicha invariante expresa una relación que no cambia entre las variables.
- ⊙ La invariante del lazo contiene una información completa de lo que el lazo calcula

Algoritmos iterativos

- ◉ Guía para encontrar la invariante del lazo: como $I \wedge \neg B \Rightarrow \text{pos}$, y B y pos son conocidos, se generaliza pos para encontrar una posible invariante.

Ejemplo 2: Prueba para el algoritmo de búsqueda binaria

La invariante del lazo debe contener la condición $x \in A[a..b] \Leftrightarrow x \in A[i..j]$, tomando en cuenta las variables i y j

El problema es incorporar a enc y m en la invariante, por lo cual se agrega la pos , $\text{enc} \Rightarrow (a \leq m \leq b \wedge x = A_m)$

Quedando la invariante como $(x \in A[a..b] \Leftrightarrow x \in A[i..j]) \wedge (\text{enc} \Rightarrow (a \leq m \leq b \wedge x = A_m))$

- ◉ Ejercicio: Hacer la prueba

Algoritmos iterativos

Marzo/05 busquedaBinaria(Arreglo[a..b]DeTipoEle: A, Entero: a, b, TipoEle: x):Lógico { pre: $a < b + 1 \wedge A_a \leq \dots \leq A_b$ } { pos: $enc \Rightarrow (a \leq m \leq b \wedge x = A_m)$ }		
1 i, j, enc = a, b, falso 2 ((i < j+1) \wedge \neg enc) [m = (i+j)/2 enc = Si (x = A _m) entonces verdadero fsi j, i = Si (x < A _m) entonces m - 1, i sino j, m + 1 fsi]	a, b: Entero. Valores inicial y final de los subíndices del vector A. enc: Lógico. Bandera que indica que el elemento x se encuentra en el vector A i, j, m: Entero. Subíndices.	
3 regrese enc		
1 A(), a = 0, b = -1, x = 'a' -> enc = falso 2 A(2, 3, 4), a = 1, b = 3, x = 3 -> enc = verdadero 3 A(6), a = 1, b = 1, x = 4 -> enc = falso	Arreglo vacío Elemento encontrado Elemento no encontrado	