Elementos de correctez de algoritmos

Pruebas basadas en invariantes

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

15 de marzo de 2021

Inducción matemática

Temas

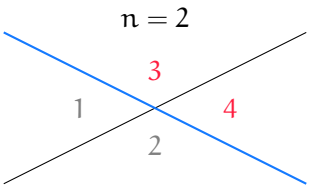
Inducción matemática

- Sea T un teorema a demostrar.
- Suponemos que T tiene como parámetro a $n \in \mathbb{N}$.
- Se debe probar que T es válido para todos los valores de n.
- Para ello se prueban las siguientes condiciones:
  1. Probar que T es válido para $n = 1$ *(base inductiva)*
  2. Para $n > 1$ *(hipótesis inductiva)*
     Probar que si T es válido para $(n-1) \Rightarrow T$ es válido para n.

Temas

Ejemplo

Llamememos a_n al número de regiones del plano que quedan determinadas por n rectas. Estas rectas deben ser tales que por cualquier punto del plano pasen a lo sumo dos de ellas; y tales que ninguna de ellas sea paralela a ninguna otra. *Fernández Gallardo y Fernández Pérez 2018*



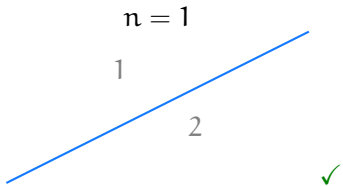
P.D. Dadas n líneas, el número de regiones en el plano son:

$$a_n = 1 + \frac{n(n+1)}{2} \tag{1}$$

Caso base $n = 1$:
La fórmula indica:

$$a_1 = 1 + \frac{1(1+1)}{2} = 2$$

Graficando, verificamos que esto es correcto:



✓

H.I. Suponemos que la fórmula es válida para n.
P.D. que es válida para $n+1$:

$$a_{n+1} = 1 + \frac{(n+1)(n+2)}{2}$$

Observamos entonces ¿cuántas regiones se agregan al incorporar una línea más?

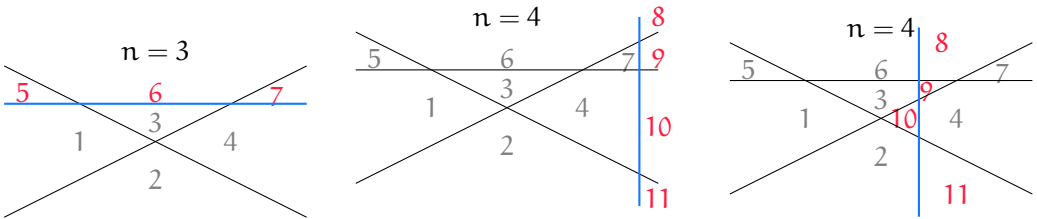


Figura: Cada recta nueva corta a las n rectas anteriores, agregando $n+1$ regiones.

Luego entonces, si por **H.I.** podemos calcular las regiones del paso anterior, sólo sumamos $n+1$ regiones del paso siguiente, dando un total de:

$$\begin{aligned} a_{n+1} &= a_n + (n+1) \\ a_{n+1} &= \underbrace{1 + \frac{n(n+1)}{2}}_{\text{H.I.}} + (n+1) \\ &= 1 + \frac{n(n+1) + 2(n+1)}{2} = 1 + \frac{(n+1)(n+2)}{2} \end{aligned}$$

Que es la fórmula que queríamos demostrar.

Temas

Principio de inducción

Débil:

Si una proposición P, *con parámetro* $n \in \mathbb{N}$, *es verdad para* $n = 1$ *y si* $\forall n, n > 1$, *la verdad de* P *para* $n - 1$ *implica la verdad para* n *entonces* P *es verdadero* $\forall n \in \mathbb{N}$

Fuerte:

Si una proposición P, *con parámetro* $n \in \mathbb{N}$, *es verdad para* $n = 1$ *y si* $\forall n, n > 1$, *la verdad de* P $\forall i, i \in \mathbb{N}, 1 \leqslant i < n$ *implica la verdad para* n *entonces* P *es verdadero* $\forall n \in \mathbb{N}$

Justificación de algoritmos

Temas

Semántica axiomática (Floyd & Hoare)

Para cada algoritmo se definen precondiciones y postcondiciones.

Precondición: Asume algún valor de verdad con el cual se debe iniciar el proceso. Condición que debe cumplir el ejemplar para que se le pueda aplicar el algoritmo.

Postcondición: Es un estado que garantiza una verdad sobre el resultado.

Algoritmos recursivos e iterativos

Consideraremos dos tipos de algoritmos:

Recursivos

Programa o definición que se refiere a sí misma directa o indirectamente.

- Un programa define un proceso.
- Una definición, un conjunto de objetos.

Iterativos

Una secuencia de instrucciones que se repite un número finito de veces.

Temas

Algoritmos recursivos

- Ofrecen una solución a un problema $P(n)$ con $n \in \mathbb{N}$, planteada en términos de sí misma.
- Sus elementos son:
  1. Caso base:
  $$P(0) \tag{2}$$
  2. Llamada recursiva:
  $$P(n) = f(P(n-1)) \tag{3}$$
  Donde f representa a una función (programa) que depende de la solución al problema para $n-1$.

Ejemplo 1: Factorial

Precondiciones: $n \geqslant 0$
Postcondiciones: FACT(n) devuelve $n!$

```
1: function FACT(n: entero)
2:     if n = 0 then return 1
3:     else
         return n * FACT(n-1)
4:     end if
5: end function
```

Demostración de correctez:

Teorema: `Fact(n)` regresa $n!$ $\forall n \geqslant 0$

Dem.: Inducción sobre n

Caso base: Si $n = 0$, `Fact(0)` regresa 1 ✓

Hip. Ind.: Para $n-1$ `Fact(n-1)` regresa $(n-1)!$

Ind. $n > 0$**:** Como $n \neq 0 \Rightarrow$ el algoritmo regresa $n *$ `Fact(n-1)`.

Por H.I. `Fact(n-1)` regresa $(n-1)! \Rightarrow$ el resultado de `Fact(n)` será $n \times (n-1)! = n!$. □

Ejemplo 2: Sumar los elementos de un arreglo recursivamente

Precondiciones: $b = \text{longitud}(A) - 1$
Postcondiciones: `suma(A)` devuelve $\sum_{i=0}^{b} A(i)$

function SUMA(A: arreglo de números, b: entero)
    if b < 0 **then return** 0
    else
        return A(b) + SUMA(A, b-1)
    end if
end function

```
1  private static
2    double suma(double[] a,
3                int last){
4    if(last == -1) return 0;
5    else return a[last] +
6         suma(a, last - 1);
7  }
8
9  public static
10   double suma(double[] s){
11   if(s == null)
12     throw new NullPointerException();
13   return suma(s, s.length - 1);
14 }
```

Demostración

Demostración de correctez:

Teorema: `Suma(A,b)` regresa $\sum_{i=0}^{b} A(i)\ \forall b = longitud(A) - 1, b \geqslant -1$

Dem.: Inducción sobre b

Caso base: Si $b = -1$, no quedan elementos por sumar en el arreglo (arreglo de longitud cero).
En `Suma(A, -1)`, $b < 0$ se cumple y el algoritmo regresa 0. ✓

Hip. Ind.: Para $b-1$ `Suma(A,b-1)` regresa $\sum_{i=0}^{b-1} A(i)$

Ind. $b > -1$: Como $b \geqslant 0 \Rightarrow$ el algoritmo regresa `A(b) + Suma(A, b-1)`.

Por H.I. `Suma(A, b-1)` regresa $\sum_{i=0}^{b-1} A(i) \Rightarrow$ el resultado de `Suma(A,b)` será $\sum_{i=0}^{b} A(i)$. □

Ejemplo 3: Búsqueda binaria

Dada una lista ordenada de n elementos, determinar si el elemento x está o no en la lista.

Ej: $x = 5$

a=0	1	2	3	**m=4**	5	6	7	8	*b=9*
-4	-1	0	5	$7 \neq 5$	200	301	1020	2000	5000

a=0	**m=1**	2	*b=3*
-4	$-1 \neq 5$	0	5

a=m=2	*b=3*
$0 \neq 5$	5

a=m=b=3
5✓

Devuelve: **Verdadero**

Ejemplo 3: Búsqueda binaria (algoritmo)

Precondiciones: $b \geqslant a$ & A está ordenado

Postcondiciones: Encontró $\in \mathbb{B} \leftarrow x \in A[a, b]$ y A no cambia

```
1:  function BB(A: arreglo de enteros, a: entero, b: entero, x: entero)
2:      if a > b then return False
3:      else
4:          m ← (a + b) div 2
5:          if A(m) = x then return True
6:          else if x < A(m) then return BB(A, a, m-1, x)
7:          else return BB(A, m+1, b, x)
8:          end if
9:      end if
10: end function
```

Ejemplo 3: Búsqueda binaria (dem)

Demostración de correctez: (Parte I).

Teorema: $\forall n, n \in \mathbb{N}, n = b - a + 1$ donde n es el número de elementos en el segmento del arreglo $A[a, b]$. BB(A,a,b,x) indica si $x \in A[a, b]$

Demostración: Inducción sobre n

Caso base: Si $n = 0$, x no puede estar en un segmento vacío.

P.D. que BB(A, a, b, x) con $b - a + 1 = 0$ regresa Falso.

Si $b - a + 1 = 0 \Rightarrow a = b + 1 \Rightarrow a > b \Rightarrow$ `If(a > b)` se cumple y el algoritmo devuelve Falso. ✓ □

Ejemplo 3: Búsqueda binaria (dem)

Demostración de correctez: (Parte II).

Hip. Ind.: $\forall j, 0 < j \leqslant n-1$ donde $j = b' - a' + 1$ es el número de elementos `BB(j)` regresa correctamente si $x \in A[a', b']$

Ind. $n > 0$**:** Se ejecuta el `else` y calcula $m = (a + b)$ **div** 2 con $b \geqslant a \Rightarrow m \geqslant a$.

Si $A(m) = x$, por la línea 5, el algoritmo regresa True ✓

Si $x < A(m)$, por la línea 6, el algoritmo ejecuta `BB(A,a,m-1,x)`, en el intervalo $[a, m-1]$ donde $n' = (m-1) - a + 1 \Rightarrow 0 \leqslant n' < n$.

Si $n' = 0$ es el caso base ✓.

Si no, por H.I. la respuesta es correcta ✓

Similarmente si $A(m) > x$ (línea 7). □

Diseño de algoritmos con inducción matemática

No es necesario diseñar todos los pasos requeridos para resolver el problema desde la "nada", es suficiente garantizar que:

1. *Caso base.* Es posible solucionar el problema para un ejemplar pequeño.
2. *Paso inductivo.* Una solución para cada ejemplar del problema puede ser construida por soluciones para ejemplares de menor tamaño.

Temas

Algoritmos iterativos

Técnica del invariante del ciclo.

El invariante del ciclo es una condición que se supone verdadera del inicio al final del ciclo.

Pueden existir muchos invariantes, pero nos interesa aquel que mantiene una relación entre las variables que modifican su valor durante la ejecución del ciclo.

Técnica del invariante del ciclo

1. Identificar el invariante *I* que nos permitirá alcanzar la postcondición.
2. Demostrar, usando inducción matemática, que el invariante es correcto.
3. Probar que $Invariante + \neg(\text{Condición del ciclo}) \Rightarrow Postcondición$.
4. Demostrar que el ciclo es finito
   - Identificar las variables que *afectan* a la condición de entrada al ciclo y verificar que en efecto, alcanzan el valor de *salida*.

Ejemplo 1: Sumar elementos de un arreglo

Precondiciones: $0 \leqslant a \leqslant b+1$ donde $\text{longitud}(A) = b+1$
Postcondiciones: Regresa $suma = \sum_{i=a}^{b} A(i)$ y A no cambia

1: **function** SUMA(A: arreglo de números, a: entero, b: entero)
2: $i \leftarrow a$
3: $suma \leftarrow 0$
4: **while** $i \neq b+1$ **do**
5: $suma \leftarrow suma + A(i)$
6: $i \leftarrow i+1$
7: **end while**
8: **return** $suma$
9: **end function**

Ejemplo 1: Suma (demostración del invariante)

Demostración del invariante (Parte I).

Teorema: Al inicio de la i-ésima iteración del algoritmo `Suma` la condición

$$suma = \sum_{j=a}^{i-1} A(j) \tag{4}$$

se satisface.

Demostración: Por inducción sobre el número de iteraciones.

Caso base: Si $n = 1$, al inicio de la 1er. iteración $suma = 0$ e $i = a$.
$\sum_{a}^{a-1} A(j) = 0 = suma$ ✓. □

Ejemplo 1: Suma (demostración del invariante)

Desmostración del invariante (Parte II).

Hip.Ind.: Suponemos que el teorema es válido para k.

Ind. Asumimos que $1 < i < b + 1$ para entrar al ciclo.

Sean $suma'$ e i' los valores de $suma$ e i al inicio de la (k+1)-ésima iteración.

P.D. $suma' = \sum_{j=a}^{i'-1} A(j)$

Por la línea 5 del algoritmo $suma' = suma + A(i)$.

Por la línea 6, $i' = i + 1 \Rightarrow i = i' - 1$.

Por H.I. $suma' = \sum_{j=a}^{i-1} A(j) + A(i) = \sum_{j=a}^{i'-2} A(j) + A(i' - 1) = \sum_{j=a}^{i'-1} A(j)$ □

Ejemplo 1: Demostración de la correctez del algoritmo

Postcondición.

¿Qué pasa cuando $i = b + 1$? i.e. cuando ya no se entra al ciclo:

$suma = \sum_{j=a}^{(b+1)-1} A(j) = \sum_{j=a}^{b} A(j)$

Respuesta: Se cumple la postcondición. □

El ciclo es finito.

Al inicio del ciclo $i = a$ con $a \leqslant b + 1$.

Si $a = b + 1$ la condición del ciclo no se cumple y el programa termina. ✓

Si $a < b + 1$ el ciclo se ejecuta y el valor de i es incrementado en 1 una sola vez por ciclo, generando una sucesión creciente acotada por $b + 1$: $a, a + 1, a + 2, ..., b + 1$ por lo que el número de iteraciones es finito. □

Bibliografía

Bibliografía I

- Cormen, Thomas H. y col. (2009). *Introduction to Algorithms*. 3rd. The MIT Press.
- Fernández Gallardo, Pablo y José Luis Fernández Pérez (2018). «El discreto encanto de la matemática». En: cap. Recurrencias. URL: http://verso.mat.uam.es/~pablo.fernandez/cap8-dic18.pdf.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual