Recursión e iteración

Correctez

Verónica E. Arriola-Rios

Facultad de Ciencias, UNAM

21 de noviembre de 2020

Diseño recursivo

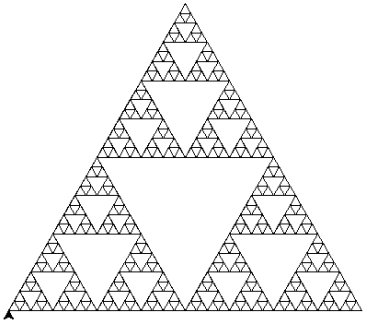
Diseño de algoritmos con inducción matemática

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.

De matemáticas a computación

- Un problema planeteado matemáticamente de forma recursiva puede requerir un poco más de procesamiento para programarlo.



Sierpinski

Algoritmo 1 Sierpinski.

```
1:  Inicia el lápiz en la esquina inferior izquierda, viendo hacia arriba.
2:  function SIERPINSKI(lápiz, longitud, nivel)
3:      if nivel = 0 then
4:          DERECHA(30), TRIÁNGULO(longitud), DERECHA(90)
5:      else
6:          SIERPINSKI(lápiz, longitud/2, nivel − 1)
7:          Coloca el lápiz en la punta superior, viendo hacia arriba.
8:          SIERPINSKI(lápiz, longitud/2, nivel − 1)
9:          Coloca el lápiz en la esquina derecha del 1er triángulo, viendo arriba.
10:         SIERPINSKI(lápiz, longitud/2, nivel − 1)
11:         Regresa el lápiz a la esquina inferior izquierda del 1er triángulo.
```

Correctez de algoritmos recursivos e iterativos

Temas

2. Correctez de algoritmos recursivos e iterativos
   - Inducción matemática
   - Correctez de un programa recursivo
   - Diseño de un algoritmo iterativo

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

2. Correctez de algoritmos recursivos e iterativos
   - Inducción matemática
   - Correctez de un programa recursivo
   - Diseño de un algoritmo iterativo

Ejemplo 1: Factorial

Algoritmo 2 Factorial.

Precondiciones: $n \geqslant 0$

Postcondiciones: FACT(n) devuelve $n!$

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

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 $n \times (n-1)! = n!$. □

Temas

2. Correctez de algoritmos recursivos e iterativos
   - Inducción matemática
   - Correctez de un programa recursivo
   - Diseño de un algoritmo iterativo

El invariante de ciclo

Definición (Invariante de ciclo)

Un *invariante de ciclo* es una condición que debe ser **verdadera** al comienzo de cada iteración.

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

Ejemplo: factorial

```java
public class Iteración {
  /** Calcula el factorial de un número recursivamente. */
  public static long factorial(long n) {
    if (n < 0) throw new IllegalArgumentException("Indefinido");
    long f = 1;
    while(n > 1) {
      f *= n;
      n--;
    }
    return f;
  }
}
```

Ejemplos de invariantes:

$$n > 0 \qquad\qquad n - 1 = \text{pasos faltantes}$$

$$f = \frac{n_0!}{n!}$$

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*.

Correctez de un algoritmo iterativo

```java
/** Precondición: n >= 0
  * Postcondición: devuelve n! */
public static long factorial(long n) {
  if (n < 0) throw // ...
  long f = 1;
  while(n > 1) {
    f *= n;
    n--;
  }
  return f;
}
```

Demostración del invariante

Teorema: Al inicio de cada iteración

$$f = \frac{n_0!}{n!}$$

con n_0 el valor original de n.

Demostración: Por inducción sobre el número de iteraciones.

Caso base: $i = 1 \Rightarrow n = n_0$, P.D. $f = \frac{n_0!}{n!} = 1$.

- Por la línea de código 4 $\Rightarrow n \geqslant 0$ lo que garantiza que el $n!$ está definido.
- Por 5, $f = 1$ y esto se mantiene hasta la línea 6, que es el inicio del ciclo y es el valor que buscábamos. ✓

Hip. Ind.: Suponemos que al inicio de la i-ésima iteración se cumple:

$$f = \frac{n_0!}{n!} \tag{1}$$

P.D.: Para $i' = i + 1$

$$f' = \frac{n_0!}{n'!}$$

donde f' es el valor almacenado en la variable f al inicio del ciclo i'.

Utilizando la hipótesis de inducción 1 y la línea de código 7 ahora:

$$f' = n * f \overset{\text{H.I.}}{\downarrow} = n * \frac{n_0!}{n!} = \frac{n_0!}{(n-1)!}$$

Por la línea de código 8

$$n' = n - 1$$

$$f' = \frac{n_0!}{n'!} \checkmark$$

Con esto queda demostrado que el invariante del ciclo es correcto.

Correctez del código factorial

P.D. El código regresa $n!$ con $n = n_0$ el valor inicial pasado como parámetro.

1. Se identificó el invariante:

$$f = \frac{n_0!}{n!}$$

2. Se demostró que es correcto usando inducción.
3. **Condición del ciclo:** $n > 1$ deja de cumplirse cuando $n = 1$, el valor del invariante en este momento es:

$$f = \frac{n_0!}{n!} = \frac{n_0!}{1!} = n_0!$$

que es justo el valor que queríamos calcular.

Demostración.

4. **Terminación:**
   1. Por la línea de código 4 $\Rightarrow n \geqslant 0$
   2. La única línea que modifica el valor de n es 8 y la decrementa.
   3. La condición del ciclo es $n > 1$, con n decreciente eventualmente dejará de cumplirse.

□

Bibliografía I

 Cormen, Thomas H. y col. (2009). *Introduction to Algorithms*. 3rd. The MIT Press.

Licencia

Creative Commons
Atribución-No Comercial-Compartir Igual