Inducción y recursión

El principio del buen orden

Ya hemos visto en varios ejemplos el conjunto Z de los números enteros, y tenemos los elementos como para afirmar que si $x, y \in Z$ entonces $x+y, x.y, x-y \in Z$, con lo cual la suma, el producto y la resta son operaciones cerradas en Z. Sin embargo, la división no es cerrada en Z, ya que $2,3 \in Z$ pero $2/3 \notin Z$.

Si tratamos de representar el conjunto Z^+ (de los enteros positivos), mediante los símbolos de desigualdad $>$ y $\geq$, podemos escribir lo siguiente:

$$Z^+ = \{x \in Z \mid x > 0\} = \{x \in Z \mid x \geq 1\}$$

Note el lector que no es posible representar conjuntos como Q o R mediante el símbolo $\geq$.

Esto permite enunciar el principio del buen orden:

Cualquier subconjunto no vacío de Z^+ contiene un elemento mínimo.

Y diremos que el conjunto Z^+ es bien ordenado.

Inducción matemática

Sea $S(n)$ una proposición en la que aparece una o más veces la variable n, que representa un entero positivo.

Si se cumple que:

- $S(1)$ verdadera, y
- $S(k)$ verdadera $\rightarrow S(k+1)$ verdadera

Entonces $S(n)$ es verdadera para todo $n \in Z^+$.

Demostración

Sea $S(n)$ una proposición en las condiciones de la hipótesis, y sea $F = \{t \in Z^+ \mid S(t)$ es falsa $\}$.
Intentaremos demostrar que $F = \varnothing$ a partir de una reducción al absurdo, por lo que supondremos que $F \neq \varnothing$. Por el principio del buen orden sabemos que F tiene un mínimo m. Como $S(1)$ es verdadera, debe ser $m \neq 1$, por lo cual es $m > 1$ y podemos afirmar que $m-1 \in Z^+$. Como $m-1 \notin F$ (porque m es el mínimo de F), tenemos que $S(m-1)$ es verdadera. Luego $S((m-1)+1) = S(m)$ es verdadera, lo que contradice que $m \in F$ (debe pertenecer a F para ser su mínimo).

Note que utilizamos el principio del buen orden en la demostración de la inducción matemática.

Las condiciones del principio de inducción matemática normalmente se denominan paso base y paso inductivo.

Ejercicios

Demostrar que $\sum_{i=1}^{n} i = \frac{n.(n+1)}{2}$ y que $\sum_{i=1}^{n} (2i-1) = n^2$

Definiciones recursivas

Las definiciones recursivas no deberían representar algo nuevo para el lector. El factorial (operación unaria conocida por el lector) suele definirse de una manera informal como:

$$n! = 1.2 \ldots (n-1).n$$

Sin embargo, no cuesta trabajo ver que esa definición no sirve para $0!$ o para $1!$, y que confía en la intuición del que la pretenda interpretar.

Por otro lado sería muy sencillo (y es la forma usual de realizarlo) definir el factorial de la siguiente forma:

- $0! = 1$
- Para $n \geq 1$, $n! = n.(n-1)!$

Tenemos entonces que el factorial de un número natural se puede obtener a partir del factorial del anterior, recursivamente, hasta llegar a cero.

Las definiciones recursivas son útiles en muchos casos, y en otros (como el caso del factorial) simplemente son necesarias.

Una definición recursiva no tiene por qué tener un único caso base y un único paso recursivo.

Un ejemplo de definición recursiva con más de un caso base son los números de Fibonacci:

- $F(0) = 0$
- $F(1) = 1$
- $F(n) = F(n-1) + F(n-2)$ para $n \in N$ con $n \geq 2$

Un ejemplo de definición recursiva con más de un paso recursivo es el conjunto de las expresiones aritméticas (sobre los números enteros):

- $\forall x \in Z$, x es una expresión aritmética
- Si e es una expresión aritmética, $-e$ es una expresión aritmética
- Si e es una expresión aritmética, (e) es una expresión aritmética
- Si e_1, e_2 son expresiones aritméticas, $e_1 + e_2$ es una expresión aritmética
- Si e_1, e_2 son expresiones aritméticas, $e_1 - e_2$ es una expresión aritmética
- Si e_1, e_2 son expresiones aritméticas, $e_1 * e_2$ es una expresión aritmética
- Si e_1, e_2 son expresiones aritméticas, e_1 / e_2 es una expresión aritmética

Aplicaciones a la programación

Los casos como los números de Fibonacci o el factorial son muy fáciles de definir mediante una recursión. Muchos lenguajes de programación incluyen la posibilidad de recursión, esto es, que una función se llame a sí misma.

```
int factorial(int num) {
   if (num < 0) {
      printf("%s", "No está definido el factorial de un negativo\n");
      exit(-1);
   }
   if (num == 0)
      return 1;
   else
      return num * factorial(num - 1);
}
```

Note que es importante que los parámetros que se pasan a la función recursiva sean más cercanos al caso base en cada llamada, lo que asegura (en un conjunto bien ordenado como el de los naturales) la terminación del algoritmo.

```
int fibonacci(int num) {
   if (num < 0) {
      printf("%s", "No está definido el número de fibonacci de un negativo\n");
      exit(-1);
   }
   if (num == 0 || num == 1)
      return num;
   else
      return fibonacci(num - 1) + Fibonacci(num - 2);
}
```