Secuencias Calculadas

Estructuras de Iteración y Recursividad

Prof. Hilda Contreras
Programación 1
hildac.programacion1@gmail.com

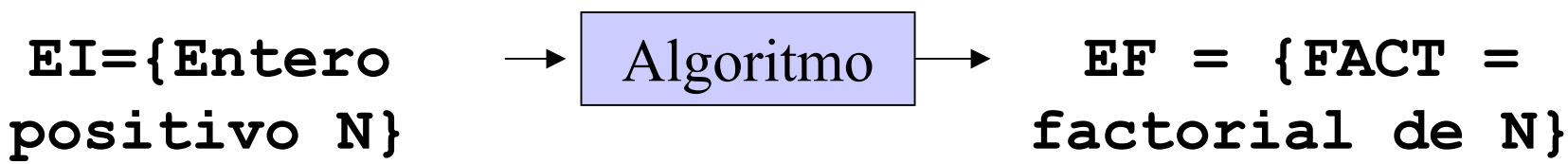
Secuencias Calculadas

Se aplican a los problemas donde los elementos de la secuencia son números que pueden ser generados por un patrón de calculo conocido.

Ejemplo 14:
Factorial de un número dado

Enunciado: Dado un número N entero positivo se quiere calcular el factorial de dicho número. Se define como:

$$N! = \begin{cases} 1 & \text{si } N=0 \\ Nx(N-1)x(N-2)...x1 & \text{si } N>= 1 \end{cases}$$



Solución: 1era versión

Estrategia: Debe calcularse la secuencia N, N-1, N-2, N-3,... ,1. Se usa una variable acumulador FACT (inicializada en 1) sobre el que se hará el cálculo mediante multiplicaciones sucesivas.

```
LEXICO:
Entero FACT: acumulador de las
    multiplicaciones de la secuencia de 1
    a N
```

Solución: 1era versión

```
INICIO
    EI = {Entero positivo N}
    Leer N
    FACT ← 1; EC ← N
    MIENTRAS (EC != 0) HACER
         INV = {FACT es el producto de los elementos
        anteriores a EC}
        FACT ← FACT * EC
        EC ← EC - 1
    FINMIENTRAS
    EF = {FACT = factorial de N}
    Escribir FACT
FIN
```

Recursividad

Es una forma matemática de definir un cálculo en términos de una versión reducida propia.

Por ejemplo: Factorial de un número

1. 6! = 6x5x4x3x2x1 = 6x5!
2. En general N! = N x (N-1)!

El factorial se define en términos de un factorial menor

Recursividad

En computación: función recursiva

Elementos comunes:

1. **Caso base**: caso sencillo, resultado conocido, se devuelve
2. **Caso recursivo**: caso complicado. La función se divide en 2 problemas: primera parte que sabe resolver y otra segunda que no sabe.

Recursividad

Caso recursivo: La segunda parte debe replantear el problema original (mas pequeño y sencillo), implica una llamada recursiva. Ambas partes se combinan y se retornan al programa original que lo llamó

Recursividad

Control: el caso recursivo se ejecuta mientras no haya terminado, las llamadas al problema cada vez mas pequeño coincide en algún momento con el caso base. En ese punto se da una secuencia de retornos consecutivos (orden inverso a la llamada) que termina devolviendo el resultado

Ejemplo 14
Solución: 2da versión

Estrategia: Se usa una función factorial que recibe como parámetro un número entero y que tiene un caso base y un caso recursivo.

Factorial(N) se define como:

Caso base: SI N <= 1 ENTONCES retorna 1

Caso recursivo: SINO retorna N * factorial(N-1)

Solución: 2da versión

```
Funcion Factorial ( N ) : entero
    Si N <= 1 Entonces
      F <- 1
    Sino
      F <- N * Factorial(N-1)
    FinSi
   Retornar F
FinFuncion
Algoritmo FactorialVersion1
    Leer N
    Escribir Factorial(N)
FinAlgoritmo
```

Llamadas Recursividad

[5] valor final = 120

5! [4] Devuelve 5! = 5 * 24 = 120

5 * 4! [3] Devuelve 4! = 4 * 6 = 24

4 * 3! [2] Devuelve 3! = 3 * 2 = 6

3 * 2! [2] Devuelve 2! = 2 * 1 = 2

2 * 1! [1] Devuelve 1

1

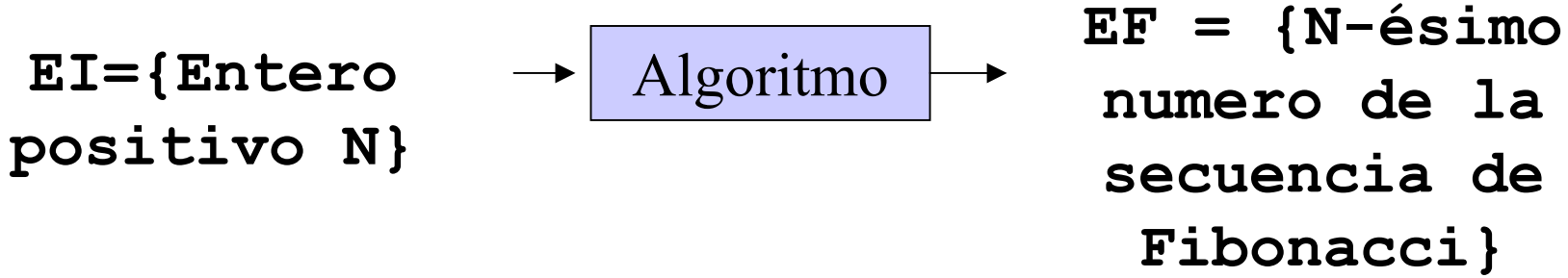
Ejemplo 15:
Números de Fibonacci

Enunciado: Dado un número N entero positivo se quiere calcular el N-ésimo número de la secuencia de Fibonacci. La sucesión de Fibonacci se define como:

F(0) = 0

F(1) = 1

F(N) = F(N-1) + F(N-2) para N >= 2



Solución: 1era versión

Estrategia: Para la secuencia de índice de 0,1,2,... n.
El tratamiento de cada elemento de la secuencia consiste en general el número de Fibonacci respectivo, para lo cual se necesitan los números anteriores n-1 y n-2.

LEXICO:

Entero FN: el Nésimo número de Fibonacci.

Solución: 1era versión

```
Algoritmo FibonacciVersion1
    Leer N
    Seleccion N
      0:FN <- 0
      1:FN <- 1
      De Otro Modo:
        FN2 <- 0; FN1 <- 1
        EC <- 2
        Mientras EC <= N Hacer
            FN <- FN2 + FN1
            FN2 <- FN1
            FN1 <- FN
            EC + 1
        Fin Mientras
    Fin Seleccion
    Escribir FN
FinAlgoritmo
```

Ejemplo 15
Solución: 2da versión

Estrategia: Se usa una función Fibonacci recursiva que recibe como parámetro un número entero y que tiene un caso base y un caso recursivo.

Fibonacci(N) se define como:

C. Base: SI N =1 o N = 0 ENTONCES retorna N

C. recursivo: SINO retorna Fibonacci(N-1) + Fibonacci(N-2)

Solución: 2da versión

```
Funcion Fibonacci ( N ): entero
    Si (N = 0) o (N = 1) Entonces
      retornar N
    Sino
      retornar Fibonacci(N-1) + Fibonacci(N-2)
    FinSi
FinFuncion

Algoritmo FibonacciVersion2
    Leer N
    Escribir Fibonacci(N)
FinAlgoritmo
```

Ejecución

```
Fibonacci (0) = 0
Fibonacci (1) = 1
Fibonacci (2) = 1
Fibonacci (3) = 2
Fibonacci (4) = 3
Fibonacci (5) = 5
Fibonacci (6) = 8
Fibonacci (10) = 55
Fibonacci (20) = 6765
Fibonacci (30) = 832040
```

El vigésimo número de Fibonacci requiere 2^{20} llamadas, cerca de un millón. Complejidad exponencial lo cual implica muchos recursos

Desventajas de la Recursividad

Sobrecarga de llamadas a la función (costo en tiempo y memoria). Cada llamada recursiva genera una copia de los datos y otra información para ejecutar la función

¿Por qué elegir la Recursividad?

Cuando el método recursivo refleja de manera mas natural el problema y genera un programa que es más fácil de entender y corregir. Una solución iterativa no es obvia

Evitar la recursividad en situaciones de rendimiento

Iteración vs. Recursividad

	Iteración	Recursividad
Estructura de Control	Estructura de Repetición	Llamadas repetidas a funciones
Prueba de terminación	Condición	Caso base
Contador	Datos que cambian en cada repetición que se evalúa en la condición	Se cambia el problema original hasta alcanzar el caso base
Pueden ser infinitas	Condición de terminación nunca es falsa	Si el paso recursivo no reduce el problema hasta el caso base


Para entender la recursividad,
había que saber
lo qué era la recursividad


NO VAYA A SER QUE POR BUSCAR SALIDAS NOS QUEDEMOS SIN ENTRADAS, ¿EH?

"Para entender la ***recursividad****, primero hay que entender lo qué es la* ***recursividad****"*

-Un programador anónimo