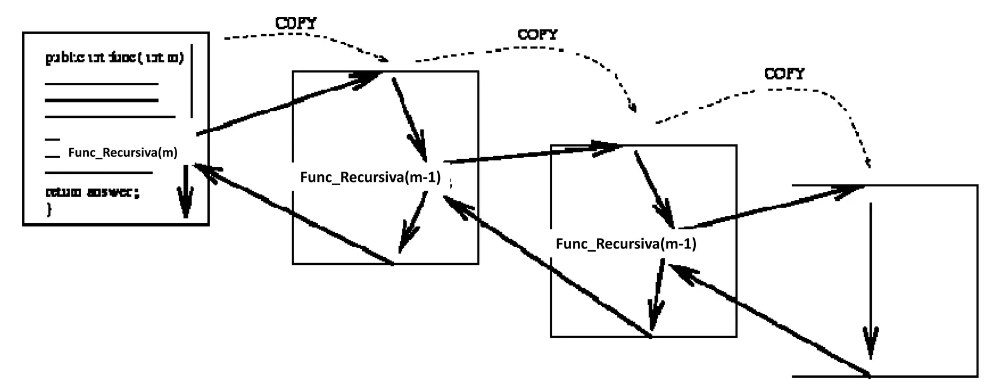
**Principios de los Algoritmos Recursivos**

**1. Introducción**

La recursividad es un concepto fundamental en ciencias de la computación que implica la definición de funciones o algoritmos que se llaman a sí mismos de manera iterativa hasta que se alcanza un caso base, se refiere a la capacidad de una función o procedimiento para llamarse a sí mismo.

Esta técnica se utiliza comúnmente en programación cuando un problema puede ser descompuesto en instancias más simples del mismo problema. La recursividad simplifica la implementación de ciertos algoritmos y puede conducir a soluciones más elegantes y compactas.



**Ilustración 1** Recursividad de una función

**2. Principios de los Algoritmos Recursivos**

A continuación, se detallan los principios fundamentales que guían la construcción y comprensión de los algoritmos recursivos:

**2.1. Caso Base**

Todo algoritmo recursivo debe tener al menos un caso base, que es la condición que detiene la recursión. Sin un caso base, la función recursiva continuaría llamándose a sí misma indefinidamente, lo que resultaría en un bucle infinito y un agotamiento de los recursos del sistema. Es crucial definir claramente este caso base para garantizar la terminación adecuada del algoritmo.

**2.2. Paso Recursivo**

El paso recursivo es el proceso mediante el cual se reduce el problema original a uno más pequeño y manejable. Después de verificar el caso base, el algoritmo debe realizar una llamada a sí mismo con una instancia más simple del problema. Este paso es fundamental para que la recursión avance hacia la resolución del problema original de manera progresiva.

**2.3. Acumulación de Resultados**

En muchos casos, especialmente cuando se trabaja con algoritmos recursivos que procesan estructuras de datos como árboles o listas, es necesario acumular resultados parciales a medida que se avanza en las llamadas recursivas. Esta acumulación puede lograrse mediante parámetros adicionales que se pasan a las llamadas recursivas o mediante el uso de variables globales. Es esencial gestionar correctamente esta acumulación para obtener el resultado correcto al final de la recursión.

**3. Ejemplos Prácticos**

Para ilustrar estos principios, consideremos el ejemplo clásico de la recursión: el cálculo del factorial de un número. El factorial de un número entero no negativo ***n***, denotado como ***n***, se define como el producto de todos los enteros positivos menores o iguales a n. La función factorial se define recursivamente de la siguiente manera:

scssCopy code

factorial(n): si n == 0: retornar 1 sino: retornar n \* factorial(n - 1)

En este ejemplo:

El caso base es cuando *n*=0, donde se retorna 1.

El paso recursivo reduce el problema al multiplicar n por el factorial de n−1.

No se necesita acumulación de resultados en este caso, ya que el valor del factorial se calcula directamente a partir de las llamadas recursivas.

**4. Conclusiones**

Los principios de los algoritmos recursivos proporcionan una base sólida para la comprensión y la implementación de funciones recursivas. Al entender y aplicar correctamente estos principios, los programadores pueden crear algoritmos recursivos eficientes y libres de errores.

**5. Referencias**

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2019). "Introduction to Algorithms." MIT Press.

Sedgewick, R., & Wayne, K. (2021). "Algorithms." Addison-Wesley.