RECURSIVIDAD Y VERIFICACIÓN

Desarrollo de Software

Jhon James Angulo Panameño
Angel David Angulo Valencia

Conceptos • Ejemplos • Aplicación • Verificación

1. ¿Qué es la recursividad?

Una técnica para resolver problemas dividiéndolos en versiones más pequeñas del mismo problema.

La recursividad es una técnica de programación en la que una función se llama a sí misma para resolver un problema. Para que funcione correctamente debe existir un caso base que indique cuándo detener las llamadas.

- Ejemplos: factorial, recorrido de árboles, búsqueda y algoritmos de divide y vencerás.

2. Estructura de una función recursiva

CASO BASE

Una función recursiva comienza con un problema y lo va reduciendo en problemas más pequeños.

CASO RECURSIVO

Cada llamada debe acercarse al caso base, hasta llegar a una condición que permita detener el proceso.

Ejemplo: factorial

Un ejemplo clásico es el cálculo del factorial. Para calcular 4!, se puede dividir el problema en 4 × 3!, luego 3 × 2!, hasta llegar a 0!, que corresponde al caso base.

3. Ejemplo paso a paso: factorial de 4

4! = 4 × 3 × 2 × 1 = 24

1	**factorial(4)**	→	4 × factorial(3)
2	**factorial(3)**	→	3 × factorial(2)
3	**factorial(2)**	→	2 × factorial(1)
4	**factorial(1)**	→	1 × factorial(0)
5	**factorial(0)**	→	1 ← caso base

Al regresar de las llamadas: 1 → 1 → 2 → 6 → 24

4. Ventajas y desventajas

VENTAJAS

La recursividad puede hacer que algunos programas sean más fáciles de entender y escribir, especialmente cuando se trabajan estructuras como árboles, carpetas o problemas que pueden dividirse en partes similares.

DESVENTAJAS

Si una función recursiva está mal diseñada, puede realizar demasiadas llamadas y consumir mucha memoria. Por eso es importante definir correctamente el caso base y controlar cómo avanza cada llamada.

5. ¿Qué es la verificación de software?

Proceso para comprobar que el software fue construido correctamente según sus especificaciones.

- Busca detectar errores antes de que el software llegue al usuario.
- Comprueba que cada componente cumple los requisitos y reglas definidas.
- Puede incluir revisiones, inspecciones, pruebas y análisis del código.
- La verificación responde principalmente: “¿Estamos construyendo el producto correctamente?”

6. Verificación aplicada a la recursividad

CASO BASE

Cuando verificamos una función recursiva debemos revisar especialmente el caso base

PROGRESO

Comprobar que cada llamada se acerca al caso base.

RESULTADO

Probar entradas normales, límites y casos especiales.

Ejemplo: factorial(4) debe producir 24; factorial(0) debe detenerse y producir 1.

7. Verificación: casos de prueba

Para una función factorial: Los casos de prueba permiten comparar el resultado que esperamos con el resultado que realmente produce el programa. En una función factorial, por ejemplo, podemos probar valores como 0, 1, 4 y 5 para comprobar diferentes situaciones.

Entrada: 0

Esperado: 1
Caso base

Entrada: 1

Esperado: 1
Mínimo positivo

Entrada: 4

Esperado: 24
Caso normal

Entrada: 5

Esperado: 120
Caso normal

Una prueba ayuda a comprobar el comportamiento esperado, pero la verificación debe considerar también estructura, lógica y requisitos.

8. Relación entre recursividad y verificación

- La recursividad aporta una forma de dividir el problema en pasos más pequeños.
- La verificación comprueba que cada paso y condición funcionan como se espera.
- En una función recursiva hay que revisar especialmente el caso base y la condición de avance.
- Combinar ambas prácticas ayuda a producir código más confiable y fácil de mantener.

9. Conclusiones

La recursividad es una herramienta útil para solucionar determinados problemas de programación, pero debe utilizarse correctamente. La verificación ayuda a comprobar que la solución funciona como se espera y permite detectar posibles errores.

- En Desarrollo de Software, usar recursividad con una verificación adecuada mejora la confiabilidad del programa.

GRACIAS