Creación de algoritmos sólidos: pruebas, análisis y comentarios

Valerie Jireth Hernández Pardo
Matrícula: 20241135037

18 de septiembre de 2025

Índice

1. Definición de Pseudocódigo

El pseudocódigo es una herramienta que permite representar de manera estructurada y ordenada los pasos que conforman un algoritmo. Se escribe en un lenguaje cercano al natural, pero con una sintaxis especial que facilita la comprensión del procedimiento lógico sin necesidad de utilizar directamente un lenguaje de programación. Su principal objetivo es servir como puente entre el planteamiento de un problema y su implementación en un lenguaje formal de programación.

Para garantizar claridad, el pseudocódigo utiliza ciertas convenciones: todos los algoritmos inician con la palabra **INICIO** y finalizan con **FIN**; las palabras reservadas como **LEER**, **ESCRIBIR** o **SI** se escriben en mayúsculas; y se emplea sangría o tabulación para resaltar la jerarquía de las instrucciones. Asimismo, las variables deben declararse indicando su nombre y tipo de dato, por ejemplo, `contador: ENTERO`. Estas reglas no son tan estrictas como en un lenguaje de programación, pero permiten mantener orden y coherencia en la descripción del algoritmo.

Un ejemplo básico de pseudocódigo sería el siguiente:

```
INICIO
  ESCRIBIR "Ingresar la altura del polígono"
  LEER altura
FIN
```

En este caso, se observa la interacción mínima entre el usuario y el programa mediante la lectura y escritura de datos.

2. Definición de Pruebas de Escritorio

Las pruebas de escritorio son un método de validación que se utiliza para verificar el correcto funcionamiento de un algoritmo antes de implementarlo en un lenguaje de programación. Consisten en simular manualmente la ejecución de un algoritmo, registrando los valores que van tomando las variables en cada paso o iteración del proceso. De esta manera, es posible identificar errores lógicos, condiciones mal planteadas o salidas inesperadas sin necesidad de ejecutar código en un computador.

La prueba se organiza generalmente en una tabla que contiene las variables de entrada, las variables de proceso y los resultados obtenidos en cada iteración. Esto no solo ayuda a confirmar que el algoritmo cumple con el objetivo propuesto, sino que también permite visualizar el flujo de datos y comprender mejor la lógica aplicada.

Por ejemplo, en el problema de determinar si un número es positivo o negativo, la prueba de escritorio incluiría casos con diferentes valores: si se ingresa 5, la salida debe ser “el número es positivo”; si se ingresa -29, la salida debe ser “el número es negativo”; y si se ingresa 0, el algoritmo debería solicitar nuevamente un número. Gracias a este procedimiento, se asegura que el algoritmo responde adecuadamente a distintas condiciones.

3. Análisis y Comentario de Ejemplos

3.1. Ejemplo 1: Determinar si un número es positivo o negativo

Análisis: Este algoritmo recibe como entrada un número real distinto de cero y lo clasifica como positivo o negativo. Su estructura es sencilla, basada en una condición **SI...ENTONCES**, lo que garantiza eficiencia ya que solo requiere una comparación. Una posible mejora sería permitir la validación del número cero con un mensaje específico.

Comentario: Este ejemplo me ayudó a entender cómo decidir si un número es positivo o negativo. Aprendí lo importante que es organizar bien las instrucciones para que el resultado sea correcto.

3.2. Ejemplo 2: Obtener el mayor de dos números diferentes

Análisis: El algoritmo solicita dos números reales distintos y determina cuál de ellos es mayor. Su eficiencia es alta ya que basta una comparación, aunque se podría ampliar para contemplar el caso en que ambos números sean iguales.

Comentario: Al trabajar con este algoritmo entendí mejor cómo comparar dos datos y la importancia de pensar en todas las situaciones posibles para que el resultado sea correcto.

3.3. Ejemplo 3: Calcular el factorial de un número

Análisis: Este algoritmo calcula el factorial de un número entero no negativo utilizando un ciclo repetitivo. Su complejidad depende del valor de entrada ($O(n)$), y se puede mejorar empleando técnicas de recursividad o almacenando resultados intermedios.

Comentario: Este ejemplo muestra cómo aplicar estructuras iterativas en pseudocódigo. Entendí mejor cómo funcionan las variables de control y acumulación en la construcción de algoritmos.