Escuela de ingenieros Julio Garavito

Ingeniería de inteligencia artificial

Resumen semana 4

Programación Dinámica

Edward Alexander Francia Guzman

Diseño de datos y algoritmos

ID: 1000110781

Sergio Alejandro Ariza Ocampo

12/02/26

Programación dinámica.

Según Brassard y Bratley (2006), la programación dinámica, es una técnica ascendente. Normalmente, se empieza por los sub-casos más pequeños, y por tanto mas sencillos. Combinando sus soluciones, obtenemos las respuestas para sub-casos de tamaños cada vez mayores, hasta que finalmente llegamos a la solución del caso original (p. 291).

Una manera más didáctica de entender la programación dinámica es contrastarla con la técnica Divide y Vencerás, ya que ambas descomponen un problema en subproblemas, solucionan cada subproblema, y luego unen los problemas dando solución al problema original. Pero a diferencia de la técnica Divide y vencerás que suele plantearse de forma recursiva, la programación dinámica construye la solución de manera ascendente.

Esto lo hace a partir de la memoización, que en pocas palabras consiste en almacenar las soluciones de los subproblemas ya resueltos para reutilizarlas cuando vuelvan a necesitarse, reduciendo así el número de recomputaciones y mejorando la eficiencia del algoritmo.

Bien, ahora ¿Cómo sabemos que la mejor solución de un problema usar la programación dinámica?, Como dijimos anteriormente el algoritmo debe tener cierto contraste con la técnica Dividir y Conquistar, en el sentido de que el problema pueda subdividirse y que la solución de los subproblemas al unirse pueda dar la solución final, además el problema debe tener una característica muy importante, y es que los problemas estén solapados o superpuestos, lo que significa que algunos de los subproblemas compartan soluciones entre sí.

Veamos un ejemplo de todo lo que acabamos de decir, con la solución recursiva del algoritmo de la secuencia Fibonacci,

El código de este algoritmo es el siguiente:

```
Fibonacci(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return Fibonacci(n-1) + Fibonacci(n-2)
```

Estamos haciendo una llamada para un numero anterior y dos números anteriores al numero que nos interesa encontrar el valor Fibonacci sumando el resultado de las dos llamadas anteriores.

Lo que ocurre es esto gráficamente es:

Para n = 5:

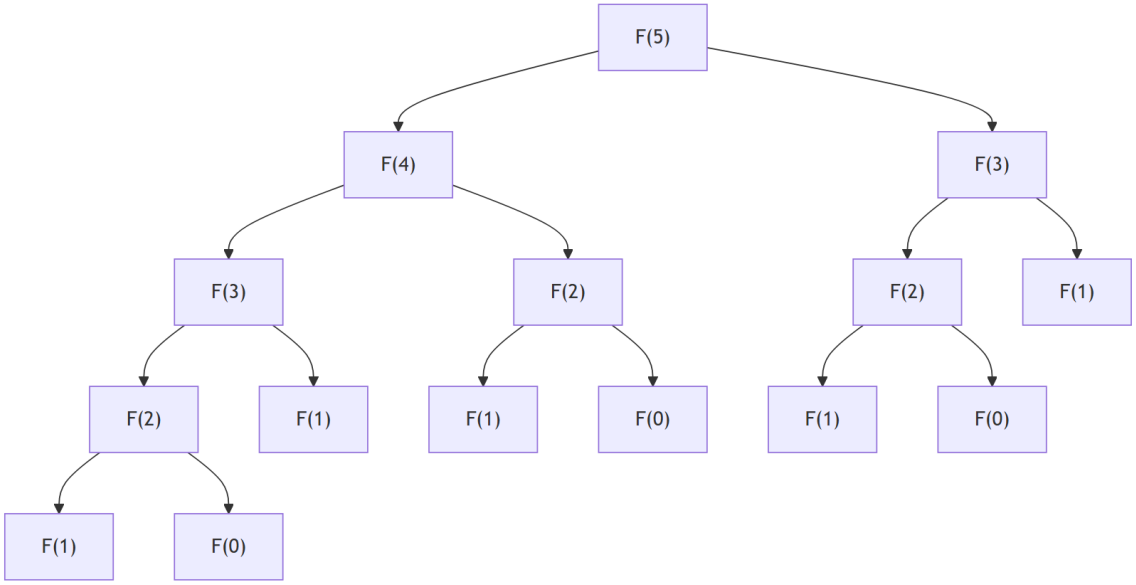

Figura 1. Representación gráfica del algoritmo Fibonacci

Nota. Elaboración propia.

Observamos que F(3) y F(2) se calculan varias veces

Complejidad:

La recurrencia es aproximadamente:

$$T(n) = T(n-1) + T(n-2) + O(1)$$

Por tanto, la complejidad de este algoritmo es 2^n, lo que resulta que para valores mayores a 35 un T(n) muy grande, el cálculo explota al punto en que podría demorar horas o días.

Sin embargo, si usamos la programación dinámica adaptando este algoritmo Fibonacci con memoizacion:

```
memo = array inicializado en -1

Fibonacci(n):
    if memo[n] != -1:
        return memo[n]
    if n == 0:
        memo[n] = 0
    else if n == 1:
        memo[n] = 1
```

```
    else:
        memo[n] = Fibonacci(n-1) + Fibonacci(n-2)
    return memo[n]
```

Lo que ocurre es esto gráficamente es:

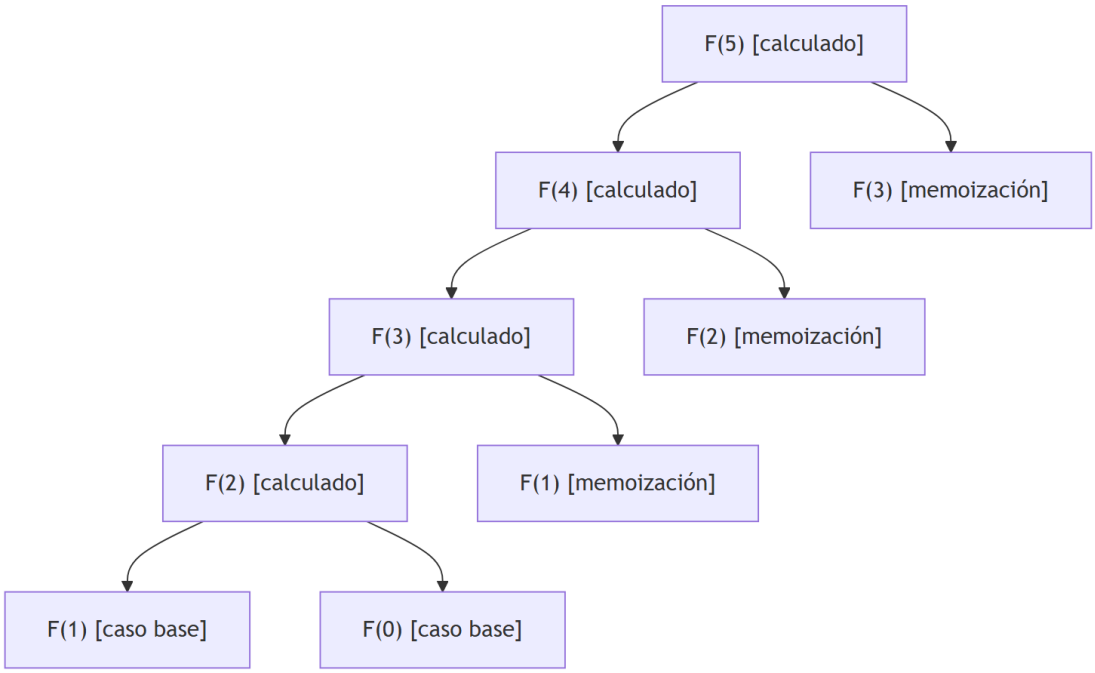

Figura 2. Grafica del algoritmo Fibonacci con memoizacion.

Nota. Elaboración propia.

Guardamos los resultados haciendo que cada valor se calcule una sola vez, por tanto, la complejidad de tiempo es:

$$T(n) = O(n)$$

Es un cambio muy significante, pasar de una complejidad exponencial a lineal.

Ejemplos:

El problema del corte de varillas consiste en determinar la forma óptima de dividir una varilla de longitud n, dado un conjunto de precios asociados a cada posible longitud, con el fin de maximizar el ingreso total (Cormen et al., 2009).

Historias de Usuario:

HU-01: Registro de datos del problema

Como administrador quiero ingresar la longitud total de la varilla (n) y la tabla que asocia cada longitud de varilla, el programa debe calcular el corte que tenga el mayor ingreso.

Criterios de aceptación:

- El sistema deja ingresar valores enteros positivos.
- La tabla de precios puede tener valores decimales.
- El sistema valida que no existan precios negativos.
- Se muestra un resumen de los datos antes de ejecutar el cálculo.

HU-02: Cálculo del ingreso máximo

Como administrador, quiero que el sistema calcule el ingreso máximo posible al cortar la varilla, por tanto, requiero que la solución sea la opción mas optima.

Criterios de aceptación:

- El sistema implementa programación dinámica.
- Se calcula el valor óptimo usando la recurrencia:
- El sistema muestra el valor máximo alcanzable.

HU-03: Reconstrucción del plan de cortes

Como administrador, quiero que el sistema me indique exactamente cómo cortar la varilla (por ejemplo: 2 + 2 + 3), para así poder tener las medidas de como cortar la varilla.

Criterios de aceptación:

- El sistema almacena las decisiones tomadas.
- Se muestra la lista completa de cortes que generan el máximo ingreso.
- El sistema puede manejar múltiples soluciones.

HU-04: Comparación con estrategia sin cortes

Como administrador, quiero comparar el ingreso obtenido con cortes respecto al ingreso de vender la varilla completa, para así verificar si realmente conviene cortar.

Criterios de aceptación:

- El sistema calcula automáticamente p_n.
- Muestra una comparación clara entre:
    - Vender completa
    - Cortar óptimamente
- Indica la diferencia en valor.

HU-05: Análisis de eficiencia

Como administrador, quiero conocer el tiempo de ejecución del algoritmo, para evaluar su viabilidad en longitudes grandes.

Criterios de aceptación:

- El sistema muestra la complejidad del algoritmo.
- El sistema puede mostrar el número de iteraciones realizadas.
- Se documenta el uso de memoria.

Requerimientos no funcionales

RNF-01: Eficiencia

El algoritmo debe ejecutarse en tiempo lineal.

RNF-02: Escalabilidad

El sistema debe poder resolver valores de n de al menos 10.000 sin que el tiempo de ejecución se multiplique.

RNF-03: Claridad de resultados

La salida debe mostrar:

- Ingreso máximo, mínimo y medio.
- Secuencia de cortes.

RNF-04: Correctitud matemática

El sistema debe garantizar que:

- Cada subproblema se resuelve una única vez en toda su ejecución.

Diagrama de flujo

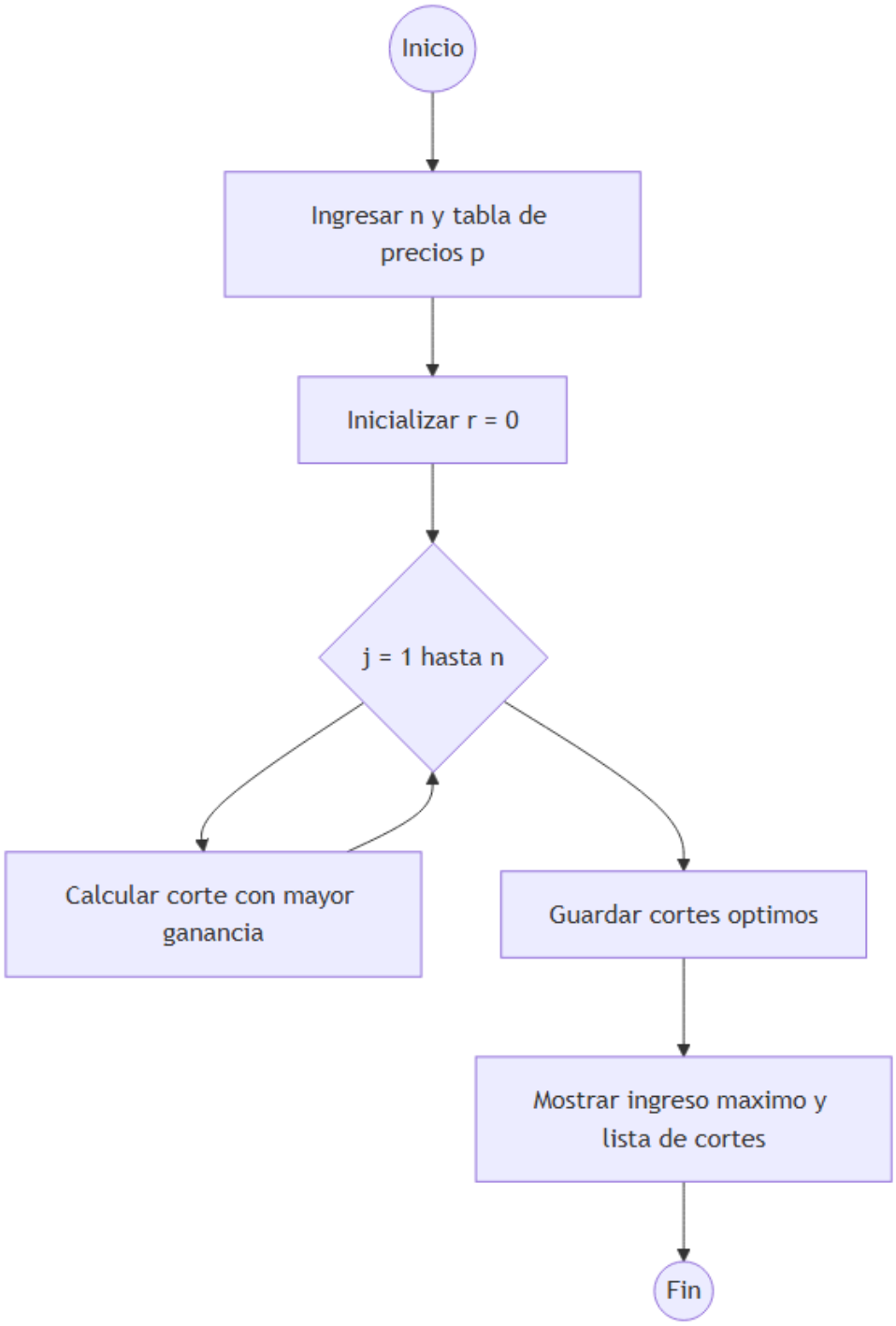

Figura 3. Diagrama de flujo del problema de las varillas.

Nota. Elaboración propia.

Diagrama de caso de uso.

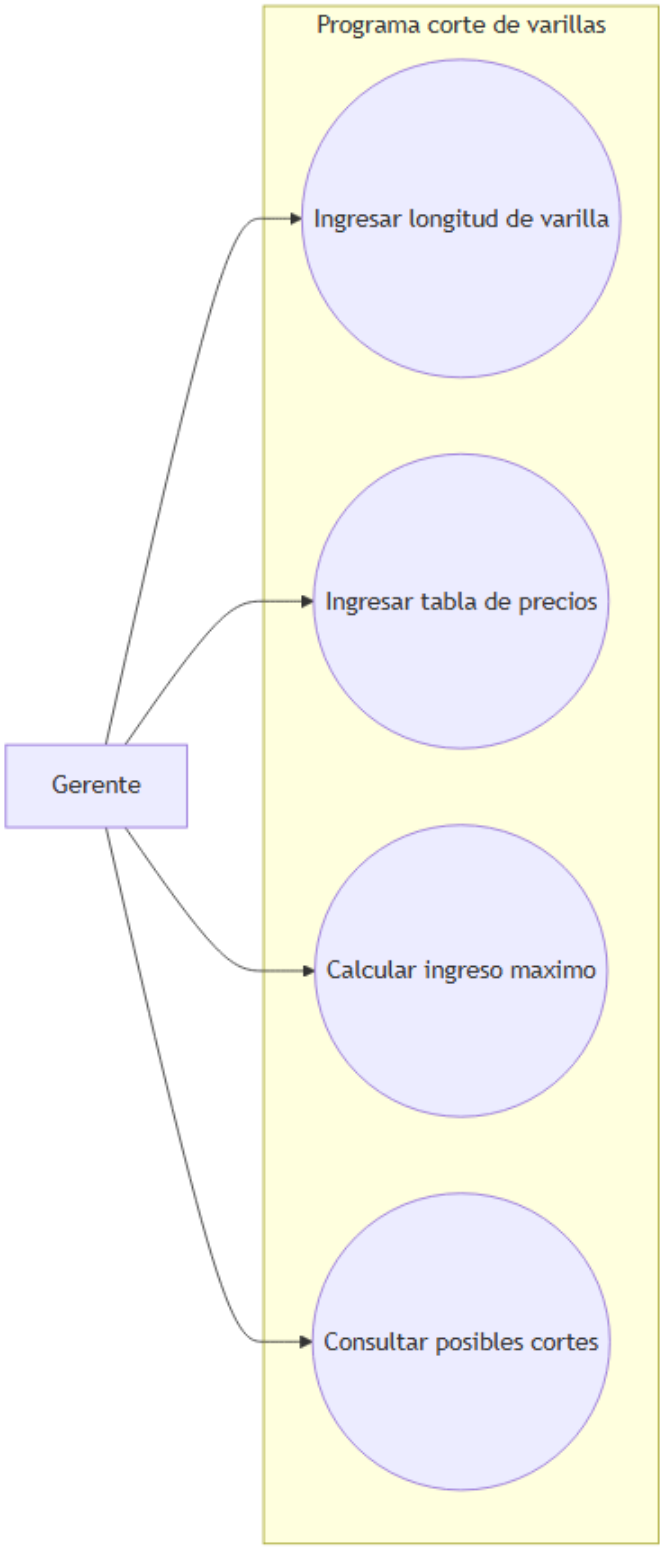

Figura 4. Diagrama de caso de uso del problema de las varillas

Nota. Elaboración propia.

Diagrama de secuencia.

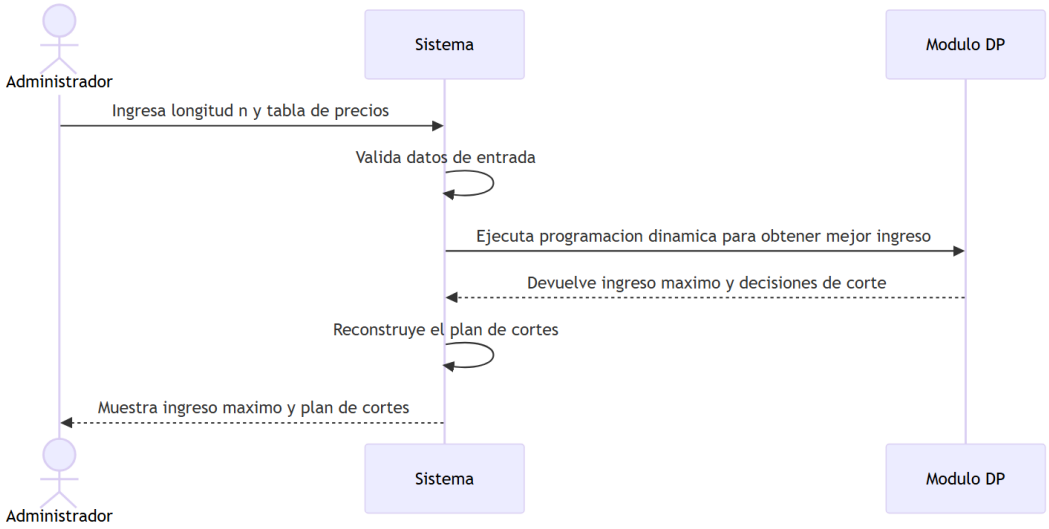

Figura 5. Diagrama de secuencia del problema de las varillas. Nota. Elaboración propia.

Análisis del problema

Este problema funciona muy bien con programación dinámica porque lo que buscamos es minimizar un costo total, y esa decisión grande depende de decisiones más pequeñas. Si al dividir la cadena en cierto punto esa división hace parte de la mejor solución, entonces las dos partes que quedan también deben estar resueltas de la mejor manera posible. Si una de esas partes pudiera hacerse más barato, entonces el resultado final también sería más barato, lo que significa que la solución anterior no era realmente la mejor. Por esa razón se dice que tiene subestructura óptima.

Además, cuando uno empieza a probar distintas formas de dividir la cadena, nos damos cuenta que muchas subcadenas se repiten varias veces. En vez de volver a calcular lo mismo cada vez, la programación dinámica guarda esos resultados y los reutiliza. Así se evita hacer fuerza bruta, que sería demasiado costoso porque las formas de agrupar las matrices crecen muchísimo. Al final, el algoritmo construye una tabla con los costos mínimos y también guarda dónde conviene dividir, para poder reconstruir el orden óptimo sin tener que probar todo otra vez.

Ejemplo 2:

El problema de la multiplicación óptima de una cadena de matrices consiste en determinar la forma más eficiente de parentizar el producto de varias matrices, de manera que se minimice el número total de multiplicaciones escalares necesarias (Cormen et al., 2009)

Historias de Usuario:

HU-01: Registro de dimensiones de las matrices

Como administrador quiero ingresar n matrices para calcular su multiplicación.

Criterios de aceptación:

- El sistema permite ingresar una lista de dimensiones enteras positivas.
- El sistema valida que las dimensiones sean compatibles para multiplicación.
- Se muestra un resumen indicando cada matriz con su tamaño correspondiente.

HU-02: Cálculo del costo mínimo

Como administrador, quiero que el sistema calcule el número mínimo de multiplicaciones escalares necesarias para multiplicar toda la cadena de matrices, por tanto, requiero que la solución sea la opción óptima.

Criterios de aceptación:

- El sistema implementa programación dinámica.
- El sistema evalúa todas las posibles particiones.
- Se muestra el costo mínimo total de multiplicaciones.

HU-03: Reconstrucción del orden óptimo

Como administrador, quiero que el sistema me indique el orden exacto en que se deben multiplicar las matrices (por ejemplo: (A1(A2A3))), para así ejecutar el cálculo de la forma más eficiente.

Criterios de aceptación:

- Se muestran los paréntesis de la formula.
- El sistema puede representar como estructura jerárquica las multiplicaciones.

HU-04: Comparación con orden simple

Como administrador, quiero comparar el costo obtenido con programación dinámica respecto a un orden simple de izquierda a derecha, para verificar si realmente conviene usar la optimización.

Criterios de aceptación:

- El sistema calcula el costo.
- Muestra una comparación clara entre:

- Multiplicación en secuencia.
- Multiplicación optima.

HU-05: Análisis de eficiencia

Como administrador, quiero conocer el comportamiento del algoritmo cuando aumenta el número de matrices, para evaluar su viabilidad en cadenas grandes.

Criterios de aceptación:

- El sistema muestra la complejidad del algoritmo.
- El sistema indica el uso de memoria.
- Se explica el crecimiento del tiempo de ejecución cuando aumenta la cantidad de matrices.
-

Requerimientos no funcionales

RNF-01: Eficiencia

El algoritmo debe ejecutarse en tiempo polinómico, evitando el crecimiento exponencial que tendría evaluar todas las posibles combinaciones.

RNF-02: Escalabilidad

El sistema debe poder resolver cadenas de al menos 200 matrices sin que el tiempo de ejecución sea un problema.

RNF-03: Claridad de resultados

La salida debe mostrar:

- Costo mínimo total de multiplicaciones.
- Orden óptimo de multiplicación.
- Comparación con orden simple.

RNF-04: Correctitud matemática

El sistema debe garantizar que:

- Cada subproblema se resuelve una única vez.
- Se cumple la propiedad de subestructura óptima.

Diagrama de flujo

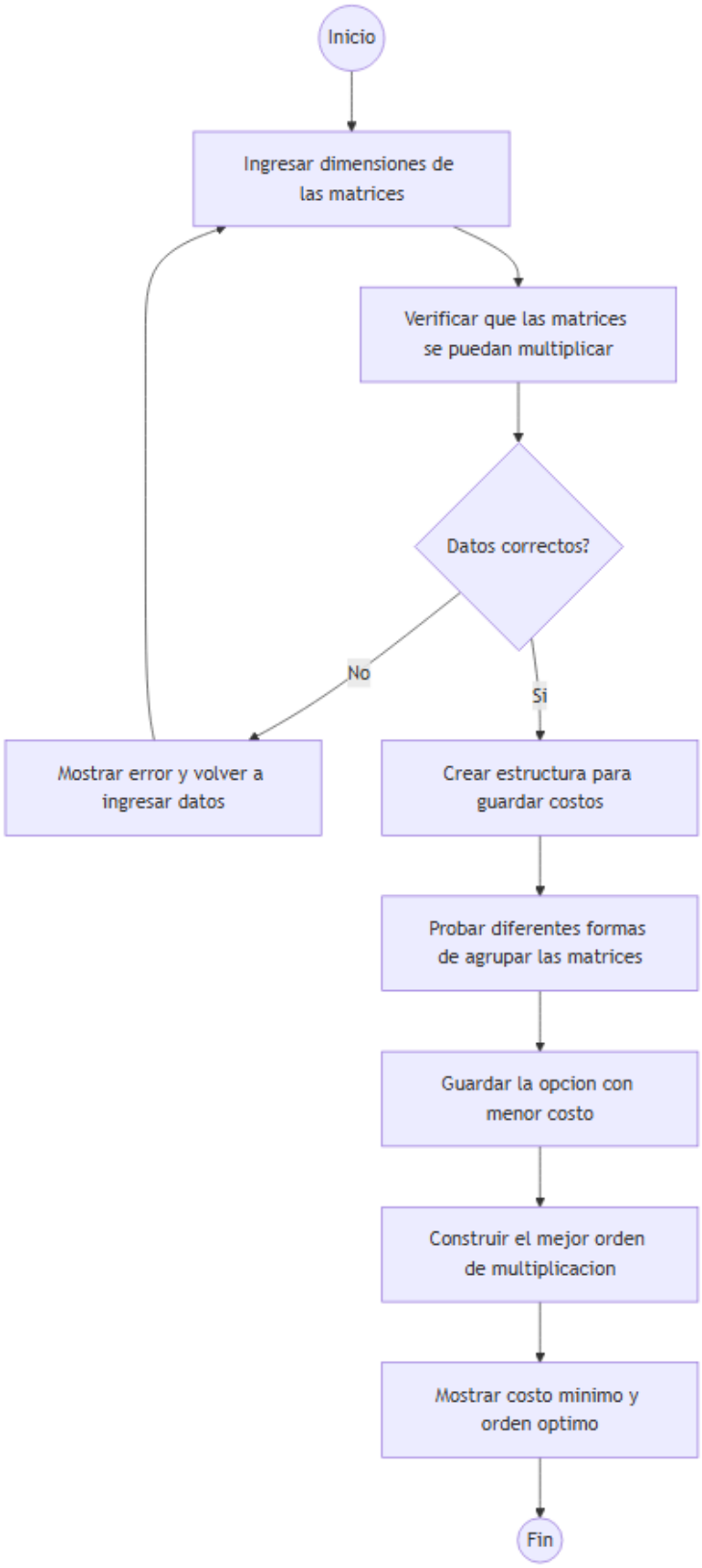

Figura 6. Diagrama de flujo del problema de la multiplicación óptima de matrices.

Nota. Elaboración propia.

Diagrama de caso de uso.

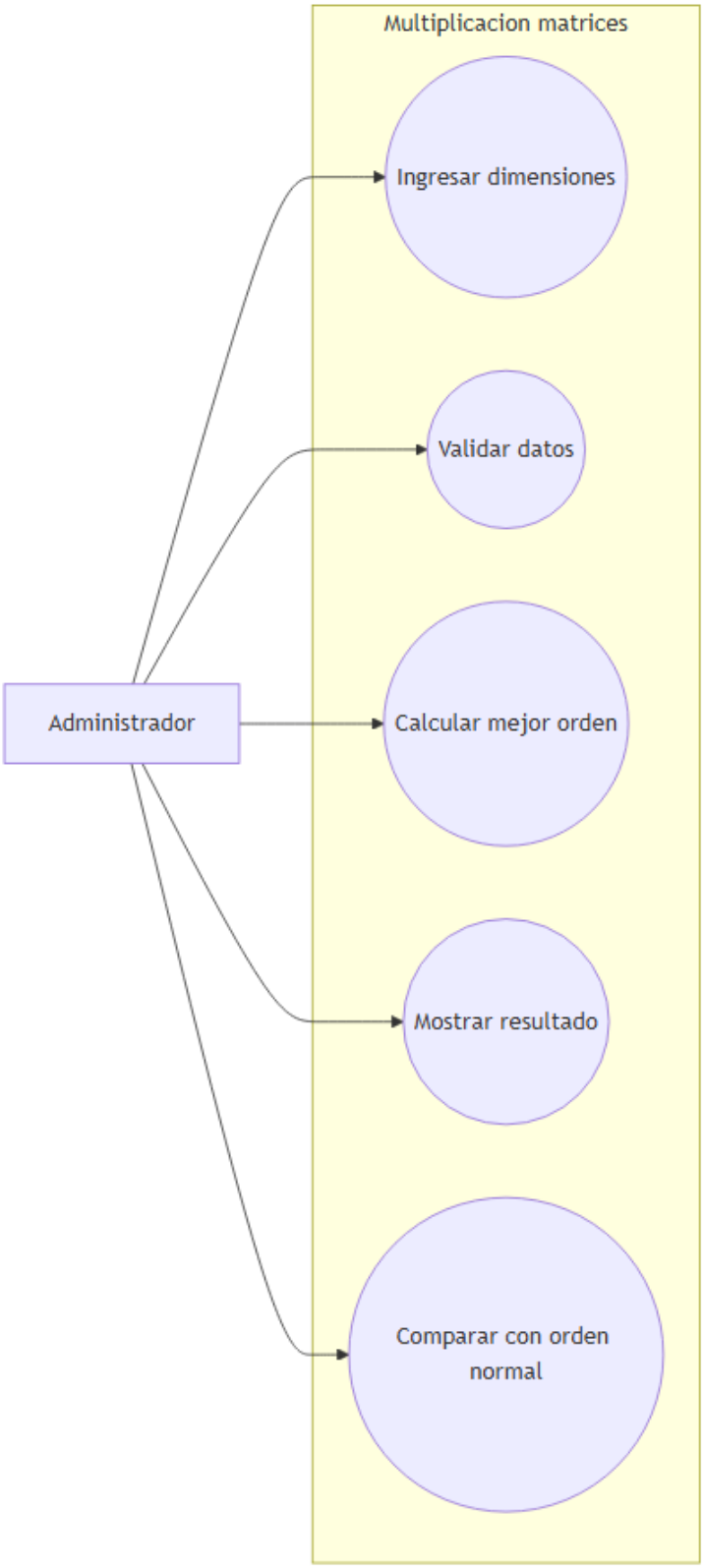

Figura 7. Diagrama de caso de uso del problema de la multiplicación óptima de matrices.

Nota. Elaboración propia.

Diagrama de secuencia.

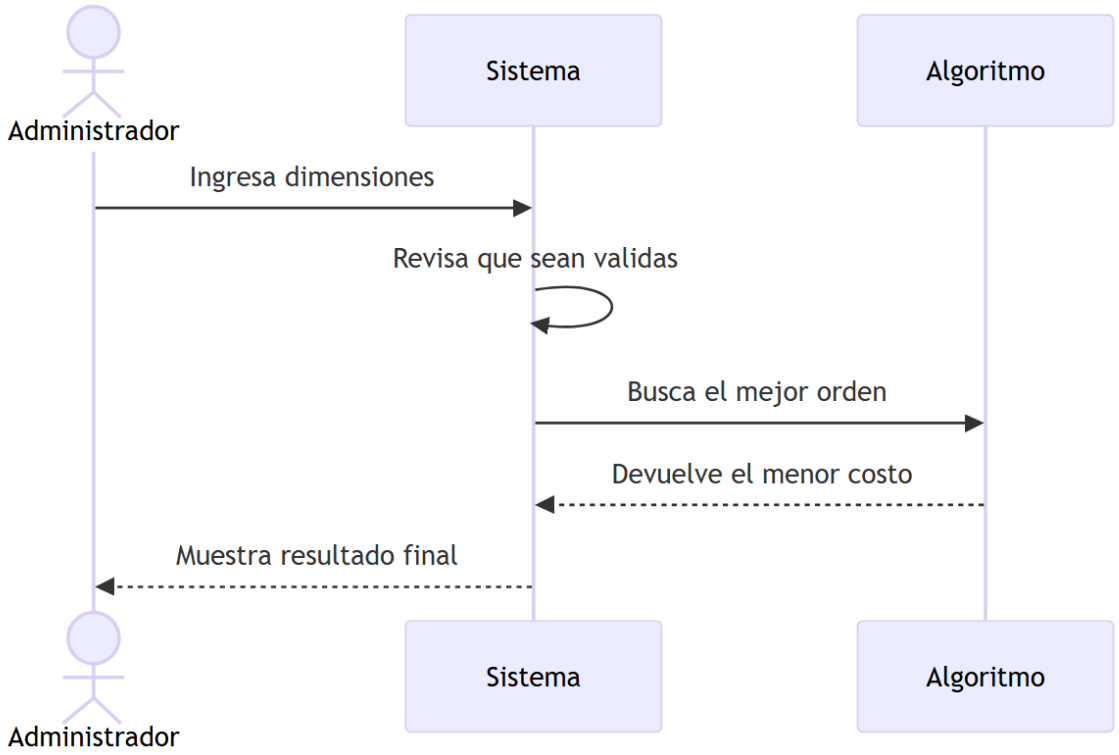

Figura 8. Diagrama de secuencia del problema de la multiplicación óptima de matrices.

Nota. Elaboración propia.

Análisis técnico formal

Este problema encaja bien con programación dinámica porque lo que se quiere es minimizar un costo total, y esa decisión grande depende de cómo se resuelvan partes más pequeñas. Si dividimos la cadena en cierto punto y esa división hace parte de la mejor solución, entonces las subcadenas también deben estar resueltas de forma óptima. Si alguna pudiera hacerse más barata, entonces el resultado final también mejoraría, lo que significa que la solución anterior no era realmente la mejor. Por eso se dice que tiene subestructura óptima.

Además, al probar distintas divisiones, muchas subcadenas se repiten varias veces. En lugar de recalcularlas siempre, la programación dinámica guarda esos resultados y los reutiliza. Así se evita la fuerza bruta, que sería inviable porque las formas de agrupar matrices crecen demasiado. Al final, el algoritmo construye una tabla con los costos mínimos y guarda dónde conviene dividir, para obtener el mejor orden de multiplicación de manera más eficiente.

Referencias

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009).

Introduction to Algorithms (3rd ed.). MIT Press.

Bohórquez Villamizar, J. A. (s. f.).

Diseño de algoritmos. Escuela de Ingenieros Julio Garavito.

Brassard, G., & Bratley, P. (2006).

Fundamentals of Algorithmics. Prentice Hall.