PROYECTO 1

PROGRAMACIÓN DINÁMICA Y VORAZ

LAURA SOFÍA PEÑALOZA - 2259485-3743

LAURA TATIANA COICUE - 2276652-3743

SANTIAGO REYES RODRIGUEZ - 2259738-3743

PROFESOR

CARLOS ANDRES DELGADO

UNIVERSIDAD DEL VALLE SEDE TULUÁ

FACULTAD DE INGENIERÍA

PROGRAMA ACADÉMICO DE INGENIERÍA DE SISTEMAS

ANÁLISIS Y DISEÑO DE ALGORITMOS II

TULUÁ – VALLE DEL CAUCA

2024

El problema de la terminal inteligente

De Ingenioso a Ingeniero

- ➔ Avanzar I N G E N I
- ➔ Insertar E
- ➔ Insertar R
- ➔ Borrar S
- ➔ Avanzar O

Costo total:

$6a + 2i + 1d = 6(1) + 2(2) + 1(2) = 6 + 4 + 2 = 12$

- ➔ Avanzar I N G E N I
- ➔ Reemplazar O por E
- ➔ Reemplazar S por R
- ➔ Avanzar O

Costo:

$6a + 2r + 1a = 6(1) + 2(3) + 1(1) = 6 + 6 + 1 = 13$

De Francesa a Ancestro

- ➔ Borrar F
- ➔ Avanzar R
- ➔ Reemplazar A por N
- ➔ Avanzar N
- ➔ Insertar C
- ➔ Insertar E
- ➔ Insertar S
- ➔ Insertar T
- ➔ Reemplazar A por R
- ➔ Insertar O

Costo total:

$1k + 2a + 2r + 5i = 1 + 2(1) + 2(3) + 5(2) = 1 + 2 + 6 + 10 = 19$

- ➔ Borrar F R A N C
- ➔ Insertar A N C E S T R O

Costo total:

$5k + 8i = 5(1) + 8(2) = 5 + 16 = 21$

Otros ejemplos:

De "camino" en "destino"

- ➔ Reemplazar C por D
- ➔ Reemplazar A por E
- ➔ Insertar S
- ➔ Reemplazar M por T
- ➔ Avanzar I N O

Costo total:

$3r + i + 3a = 3(3) + 2 + 3(1) = 9 + 2 + 3 = 14$

"camino" en "destino"

- ➔ Borrar
- ➔ Insertar C
- ➔ Insertar A
- ➔ Insertar M
- ➔ Insertar I
- ➔ Insertar N
- ➔ Insertar O

Costo total:

$k + 6i = 1 + 6(2) = 1 + 12 = 13$

"perro" en "gato"

- ➔ Reemplazar P por G
- ➔ Reemplazar E por A
- ➔ Reemplazar R por T
- ➔ Eliminar R
- ➔ Avanzar O

Costo total:

$3r + 1d = 3(3) + 2 = 11$

Para este problema se utilizó esta definición recursiva de la solución óptima

$$M[i][j] = \min \begin{cases} M[i-1][j-1] + \text{costo de avanzar o reemplazar} & \text{si } x[i-1] = y[j-1] \\ M[i-1][j-1] + r & \text{si } x[i-1] \neq y[j-1] \\ M[i-1][j] + d & \text{borrar } x[i-1] \\ M[i][j-1] + i & \text{insertar } y[j-1] \end{cases}$$

Lo que busca es avanzar en dado caso que los caracteres sean iguales, o en su defecto, si esto no sucede, aplicar la operación con menor costo, esto dependiendo el algoritmo que se esté aplicando para ese caso.

Para entender de mejor forma la definición es fundamental describir la forma en la que se desarrollaron los algoritmos, y porque estos justamente cumplen las condiciones para ajustarse a la mejor solución en cada uno de los casos expuestos.

En primer lugar se va a revisar el algoritmo por programación dinámica, en este enfoque se busca la transformación de las cadenas usando una Matriz M para poder almacenar las soluciones parciales a subproblemas más pequeños, que posteriormente se van a combinar para encontrar la solución óptima.

Estas son las condiciones para llenar la matriz:

- Si los caracteres coinciden (x[i−1]=y[j−1]), no hay un costo adicional, así que solo avanzamos.
- Si los caracteres no coinciden, se elige la operación con menor costo entre:
    - Reemplazar el carácter.
    - Borrar un carácter de x.
    - Insertar un carácter en y.

Al final, la solución estará en M[n][m], que contiene el costo mínimo para transformar x en y.

La matriz funciona de la siguiente forma, esta se completa desde la parte inferior derecha, esto debido a que se evalúan desde las subcadenas finales hasta las iniciales, aplicando las condiciones mencionadas anteriormente. Por último la solución del problema original se encuentra en la posición (0,0) de la matriz, conteniendo el costo mínimo para la transformación de la cadena.

En resumidas cuentas, la solución al problema completo se construye a partir de soluciones óptimas de subproblemas más pequeños (transformar subcadenas); además, el mismo subproblema (transformar las primeras i letras de x en las primeras j letras de y) se resuelve múltiples veces, y se almacena para evitar cálculos redundantes.

Ahora, al revisar el algoritmo voraz, donde su enfoque se basa en tomar decisiones localmente óptimas en cada paso, con la esperanza de que estas decisiones conduzcan a una solución global óptima.

El algoritmo evalúa cuál es la operación más barata en cada paso (avanzar, reemplazar, borrar, insertar o "kill") y ejecuta esta operación inmediatamente, sin considerar el impacto de esa decisión en el futuro. Este tipo de enfoque no construye una tabla como en el método dinámico; simplemente toma decisiones en tiempo real.

Al momento de realizar las pruebas el equipo probó intercambiando los valores de las operaciones más costosas, ya que en primera medida, no se tomaban en cuenta estas, debido a que como se mencionó anteriormente, toma la mejor decisión localmente, por ende va a preferir escoger una operación que cueste 1 a una que cueste 3, sin importar si esto va a afectar de forma significativa en la solución final.

Si el carácter actual de x es igual al de y, el algoritmo avanza. Si no son iguales, elige la operación más barata entre insertar, borrar, reemplazar o "kill". El algoritmo sigue este proceso hasta que las dos cadenas sean iguales o se hayan agotado las posibilidades.

- Ventaja: El enfoque voraz es rápido y sencillo de implementar. Su complejidad es menor que la de programación dinámica en muchos casos.
- Desventaja: No siempre garantiza una solución óptima, ya que el algoritmo toma decisiones basadas sólo en la información inmediata, sin tener en cuenta cómo afectarán estas decisiones a pasos futuros.

Para finalizar, evaluando el algoritmo por fuerza bruta se exploran todas las posibles formas de transformar la cadena x en y. En lugar de tratar de optimizar el proceso de búsqueda, evalúa cada posible secuencia de operaciones (avanzar, reemplazar, borrar, insertar o "kill") y calcula el costo total de cada una. Es un enfoque recursivo que, en cada paso, explora todas las posibilidades de operaciones disponibles, y luego sigue buscando hasta llegar al final de las cadenas. Al final, el algoritmo selecciona la secuencia con el menor costo.

- Ventaja: Encuentra la solución óptima porque evalúa todas las opciones posibles.
- Desventaja: Es extremadamente ineficiente debido a su crecimiento exponencial en el tiempo. Dado que explora todas las posibles secuencias de operaciones, el número de posibilidades crece exponencialmente con la longitud de las cadenas.

El problema de la subasta pública

1. Muestre dos asignaciones de las acciones para A = 1000, B = 100, n = 2, la oferta < 500, 100, 600 >, la oferta < 450, 400, 800 >, y la oferta del gobierno < 100, 0, 1000 >. Indique el valor vr para la solución.

- Calcular las asignaciones y el valor de vr:

Asignación 1: X = (600, 400, 0)

vr = 600 x 500 + 400 x 450 + 0 x 100 = 480000

Asignación 2: X = (200, 800, 0)

vr = 200 x 500 + 800 x 450 + 0 x 100 = 460000

- Comparar los valores obtenidos:

Asignación 1: vr(X) = 480000

Asignación 2: vr(X) = 460000

- Conclusión:

La asignación que maximiza el beneficio es aquella que nos da el mayor valor, en este caso la opción es la **asignación 1**, (600 , 400 , 0) con el **máximo beneficio** de 480000

2. Utilice el algoritmo anterior para la entrada A = 1000, B = 100, n = 4, < 500, 400, 600 >, < 450, 100, 400 >, < 400, 100, 400 >, < 200, 50, 200 >, la oferta del gobierno < 100, 0, 1000 >.

Estas son algunas ejemplos de combinaciones de asignaciones que genera el algoritmo, ya que este algoritmo toma todas las combinaciones y en base a esto toma el máximo beneficio y encuentra la solución óptima:

Asignación_1 : X = (500, 400, 100, 0, 0)

vr (X) = 500 x 500 + 400 x 450 +100 x 400 + 0 x 200 + 0 x 100 = 470000

Asignación_2 : X = (400, 100, 100, 200, 100)

vr (X) = 400 x 500 + 100 x 450 +100 x 400 + 200 x 200 + 100 x 100 = 335000

asignación_3 : X = (600, 400, 0, 0, 0,)

vr (X) = 600 x 500 + 400 x 450 + 0 x 400 + 0 x 200 + 0 x 100 = 480000

asignación_4 : X = (500, 100, 200, 200, 0)

vr (X) = 500 x 500 + 100 x 450 + 200 x 400 + 200 x 200 + 0 x 100 = 415000

asignacion_5 : X = (400, 400, 100, 100, 0,)

vr (X) = 400 x 500 + 400 x 450 +100 x 400 + 100 x 200 + 0 x 100 = 440000

asignación_6 : X = (400, 100, 300, 200, 0)

vr (X) = 400 x 500 + 100 x 450 + 300 x 400 + 200 x 200 + 0 x 100 = 405000

…

El algoritmo busca todas las combinaciones de asignaciones posibles que sumen a A, luego selecciona la que tenga el máximo beneficio.

Este algoritmo pertenece al tipo de algoritmos de fuerza bruta, los cuales examinan exhaustivamente todas las posibles soluciones, esto garantiza que siempre se encuentre la solución óptima.

Algoritmo de dinámica:

La relación de recurrencia que define el costo de la solución óptima es:

```
dp[i + 1, x:] = np.maximum(dp[i + 1, x:], dp[i, :-x] + x * precios[i])
```

Donde:

- i es el índice de la oferta actual.
- x es el número de acciones asignadas en el subproblema actual.
- a es el número de acciones asignadas a la oferta actual iii, limitado por el mínimo y máximo de la oferta.

Esto significa que para cada oferta i, tienes dos opciones:

1. **No asignar acciones a la oferta i**: en este caso, el valor de dp[i+1,x:] será el mismo que el de dp[i + 1, x:].
2. **Asignar a acciones a la oferta i**: en este caso, el valor de dp[i+1,x:] será el valor que tendrías sin esa a acciones dp[i, :-x] + x * precios[i].

Esto representa la transición de un subproblema más pequeño (con menos ofertas) a uno más grande (considerando más ofertas), maximizando el valor en cada paso.

Al igual que el algoritmo dinámico del problema anterior, este algoritmo resuelve el problema dividiéndolo en subproblemas más pequeños y guardando los resultados intermedios para evitar cálculos redundantes. En resumidas cuentas, realiza estos pasos:

1. Definición de subproblemas: Se crea una matriz dp donde cada celda dp[i][x] representa el valor máximo que se puede obtener usando las primeras i ofertas y

asignando exactamente x acciones en total. Esto descompone el problema en soluciones parciales más manejables.

2. Recurrencia: Para cada oferta iii, si el precio de la oferta i cumple con el umbral B, se actualiza el valor en la matriz dp para cada cantidad posible de acciones asignadas. La fórmula de recurrencia utilizada es:

```
dp[i + 1, x:] = np.maximum(dp[i + 1, x:], dp[i, :-x] + x * precios[i])
```

Esto significa que el valor máximo al asignar x acciones en la oferta i se calcula comparando el valor de no asignar acciones adicionales (manteniendo el valor anterior) con el valor de asignar k acciones a la oferta actual.

3. Construcción de la solución: Después de llenar la matriz dp, el algoritmo reconstruye la solución óptima retrocediendo a través de las decisiones tomadas. Esto implica revisar cuántas acciones se asignaron en cada oferta, comenzando desde la última hasta la primera.
4. Asignación final: Si quedan acciones sin asignar (es decir, no se pudieron asignar a ninguna oferta con precio mayor o igual a B), se asignan a la oferta de referencia (por ejemplo, una oferta gubernamental que acepta cualquier cantidad de acciones).

Ahora, el algoritmo voraz adopta un enfoque diferente, tomando decisiones locales en cada paso con la esperanza de que esas decisiones lleven a una solución global óptima. En el contexto de la subasta, el enfoque voraz asigna tantas acciones como sea posible a las ofertas con el mayor precio por acción, en el siguiente orden:

1. Filtrado de ofertas: El algoritmo primero filtra las ofertas que cumplen con el umbral de precio B, es decir, sólo considera aquellas cuyo precio es mayor o igual a B. El resto de las ofertas se ignoran.
2. Ordenación de las ofertas: Luego, se ordenan las ofertas válidas de acuerdo al precio, de mayor a menor. La idea es maximizar el valor de la asignación asignando acciones primero a las ofertas más valiosas.
3. Asignación de acciones: Para cada oferta, el algoritmo asigna la mayor cantidad posible de acciones, respetando los mínimos y máximos de cada oferta, y siempre y cuando queden acciones por asignar.
4. Finalización: El algoritmo sigue asignando acciones hasta que no quedan más acciones disponibles o todas las ofertas han sido evaluadas. Una vez que no quedan más acciones, se devuelve la asignación resultante y el valor total.

Por último, el algoritmo de fuerza bruta que es el enfoque más básico, pero también el menos eficiente. El algoritmo genera todas las combinaciones posibles de asignaciones de acciones entre las ofertas. Su funcionamiento se puede detallar de la siguiente manera:

1. Generación de combinaciones: El algoritmo recorre todas las ofertas una por una. Para cada oferta i, genera todas las combinaciones posibles de asignación de acciones, que deben estar entre los valores mínimos y máximos permitidos para esa oferta.

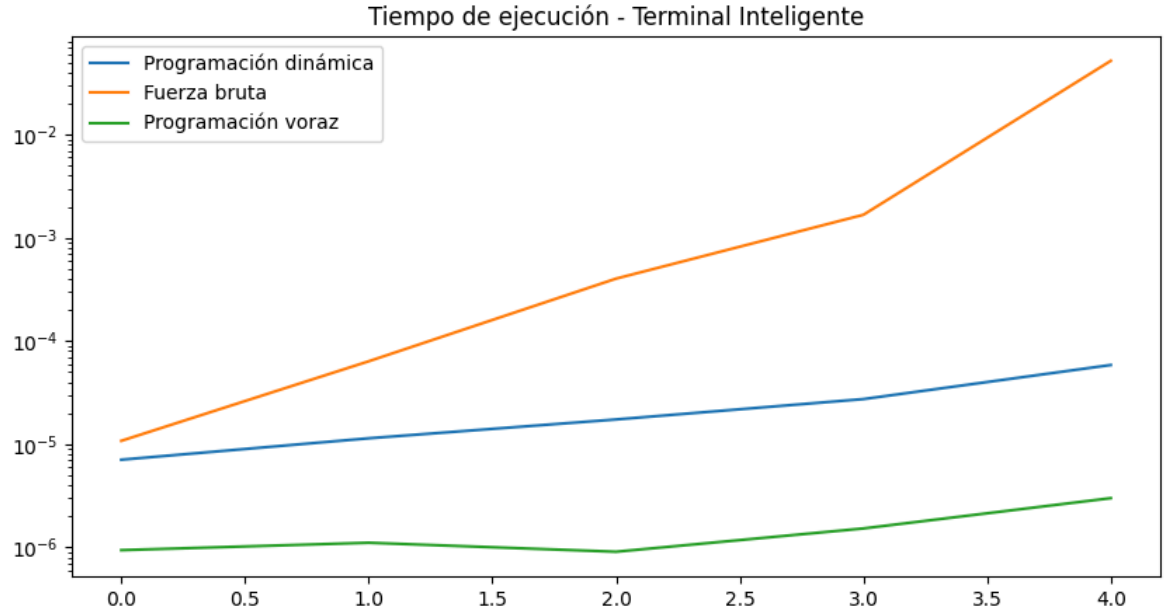
2. Exploración recursiva: El algoritmo sigue un enfoque recursivo. En cada paso, decide cuántas acciones asignar a la oferta actual y continúa con el resto de las ofertas con las acciones restantes. Esto se repite hasta que se han asignado todas las acciones, o hasta que se han recorrido todas las ofertas.
3. Evaluación de soluciones: Después de generar una combinación de asignaciones, el algoritmo calcula el valor total de esa combinación sumando el valor de las acciones asignadas a cada oferta.
4. Selección de la mejor solución: Se guarda la mejor solución (la que produce el mayor valor total) y, al final del proceso, se devuelve esa solución como la asignación óptima.

Tiempos de ejecución de las problemas

Terminal Inteligente: La gráfica compara los tiempos de ejecución de los tres algoritmos:

- Fuerza bruta: Es el más ineficiente, con un crecimiento rápido en tiempo de ejecución, indicando alta complejidad, probablemente exponencial o cuadrática.
- Programación dinámica: Es más eficiente, con un crecimiento moderado en el tiempo de ejecución, lo que sugiere una mejor gestión de la complejidad, pero aún sube con el tamaño del problema.
- Programación voraz: Es la más eficiente, con tiempos casi constantes y bajos, ideal para problemas grandes donde la optimalidad no es crítica.

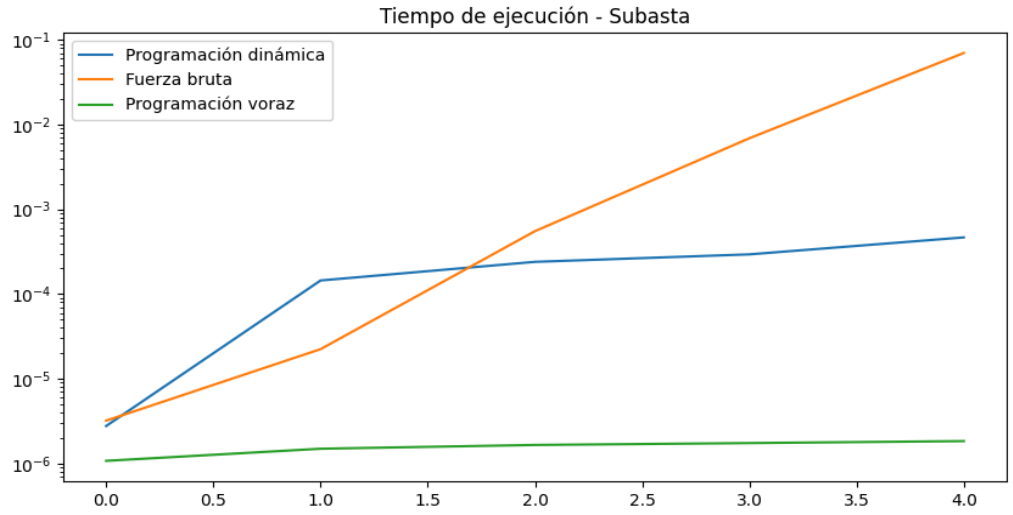
En resumen, voraz es el mejor en eficiencia, dinámica es intermedia, y fuerza bruta es la peor en cuanto a tiempos de ejecución.



Subasta pública: La gráfica compara los tiempos de ejecución de los tres algoritmos:

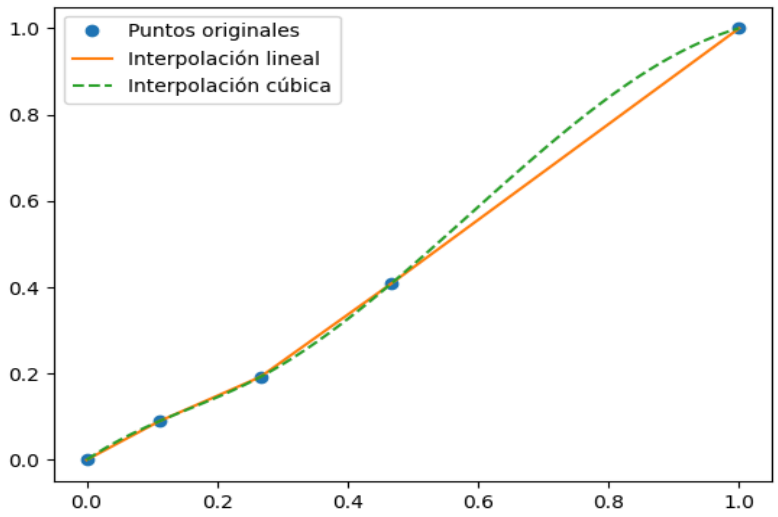
- Programación voraz: Es la más eficiente, con tiempos constantes y muy bajos.
- Programación dinámica: Tiene un crecimiento moderado, siendo una opción intermedia.
- Fuerza bruta: Es la menos eficiente, con tiempos que crecen rápidamente y son mucho mayores en problemas grandes.

En resumen, La programación voraz es ideal por su eficiencia, mientras que la fuerza bruta es ineficiente y debe evitarse en problemas grandes y la fuerza bruta es el peor caso, ya que es más costoso.

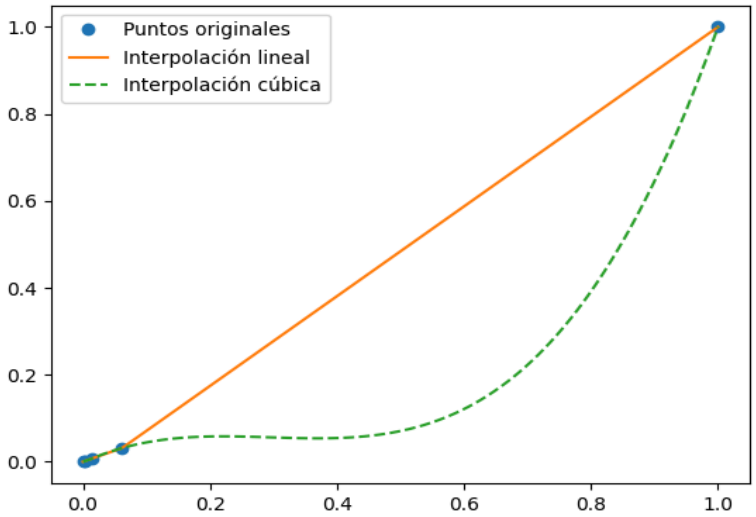


Comparación de los costos teóricos con los promedios de los tiempos

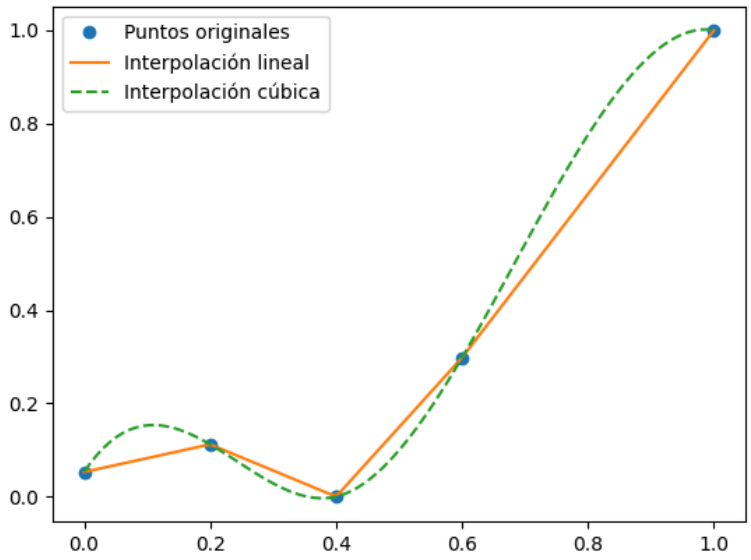
Terminal inteligente (Interpolacion_Terminal_dinamica): La interpolación lineal es más sencilla, pero menos precisa para capturar las curvas en los datos, mientras que la cúbica se adapta mejor a los cambios y ofrece una estimación más fiel, aunque con un costo computacional mayor.



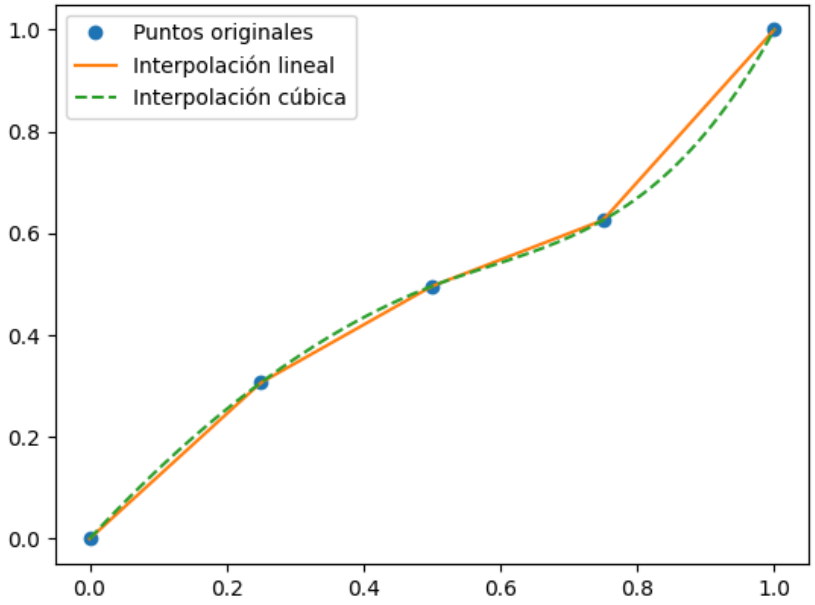
Terminal inteligente (Interpolacion_Terminal_Fuerza_bruta): La gráfica nos muestra que hay dos maneras de aproximar los valores intermedios entre los puntos de datos. La interpolación lineal es simple y directa, pero pierde detalles importantes si los datos tienen un comportamiento más complejo, mientras que la interpolación cúbica captura mejor las posibles variaciones. Esto significa que es importante elegir el método de interpolación adecuado según el contexto, para no perder información relevante o hacer suposiciones incorrectas sobre el comportamiento de los datos.



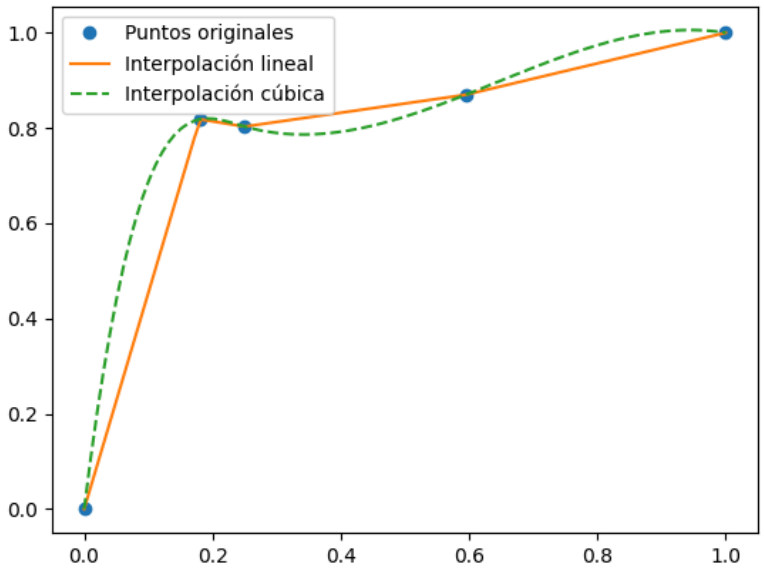
Terminal inteligente (Interpolacion_Terminal_Voraz): La interpolación lineal ofrece una visión simple y directa, mostrando cambios bruscos entre puntos, mientras que la interpolación cúbica captura mejor las fluctuaciones y variaciones suaves del sistema, sugiriendo un comportamiento más complejo entre los puntos.



Subasta_publica (interpolacion_subasta_dinamica): La interpolación lineal conecta estos puntos con líneas rectas, lo que genera transiciones abruptas entre los valores, dando valores no tan precisos. En cambio, la interpolación cúbica genera una curva más suave y un poco más detallada, ajustándose mejor a posibles cambios graduales en los valores.



Subasta_publica (interpolacion_subasta_fuerza_bruta): La interpolación cúbica muestra una mayor diferencia respecto a la lineal en la región intermedia, lo que sugiere que los datos tienen una variación más compleja en esa área, tal vez reflejando cambios más significativos en la subasta en esa fase, sugiriendose que en este caso la interpolación cúbica puede arrojarnos valores mucho más certeros y precisos.



Subasta_publica (interpolacion_subasta_Voraz): La interpolación cúbica presenta un pico alto, indicando una variación brusca en los datos de la subasta, mientras que la interpolación lineal permanece plana. Esto sugiere que los datos en este caso varían drásticamente y la cúbica capta mejor estas fluctuaciones, mientras que la lineal no refleja los cambios.