PROYECTO 1 PROGRAMACIÓN DINÁMICA Y VORAZ

CARLOS ALBERTO CAMACHO CASTAÑO - 2160331 - 3743
JUAN DAVID VALENCIA MONTALVO - 2160103 - 3743

ADA II

MSC. CARLOS ANDRÉS DELGADO

UNIVERSIDAD DEL VALLE

EISC

TULUÁ, VALLE DEL CAUCA
2024

1. LA TERMINAL INTELIGENTE

1.1 Entender el problema Muestre dos soluciones que permitan transformar la cadena ingenioso en ingeniero. Indique el costo de cada solución. Muestre dos soluciones que permitan transformar la cadena francesa en ancestro. Indique el costo de cada solución.

Teniendo en cuenta los costos = {

```
    'avanzar': 1,
    'borrar': 2,
    'reemplazar': 3,
    'insertar': 2,
    'kill': 4
}
```

Transformar "ingenioso" en "ingeniero"

Solución 1:

- Operaciones:
    1. Avanzar 'i' == 'i' (Costo: 1)
    2. Avanzar 'n' == 'n' (Costo: 1)
    3. Avanzar 'g' == 'g' (Costo: 1)
    4. Reemplazar 'e' con 'i' (Costo: 3)
    5. Avanzar 'n' == 'n' (Costo: 1)
    6. Reemplazar 'i' con 'e' (Costo: 3)
    7. Reemplazar 'o' con 'r' (Costo: 3)
    8. Borrar 's' (Costo: 2)

    Costo total: 15

Solución 2:

- Operaciones:
    1. Avanzar 'i' == 'i' (Costo: 1)
    2. Avanzar 'n' == 'n' (Costo: 1)
    3. Reemplazar 'g' con 'g' (Costo: 3, aunque sin impacto visible)
    4. Insertar 'i' después de 'e' (Costo: 2)
    5. Avanzar 'n' == 'n' (Costo: 1)
    6. Avanzar 'i' == 'i' (Costo: 1)
    7. Reemplazar 's' con 'r' (Costo: 3)
    8. Reemplazar 'o' con 'e' (Costo: 3)

    Costo total: 15

Transformar "francesa" en "ancestro"

Solución 1:

- Operaciones:
    1. Borrar 'f' (Costo: 2)
    2. Reemplazar 'r' con 'a' (Costo: 3)
    3. Avanzar 'a' == 'a' (Costo: 1)
    4. Reemplazar 'n' con 'n' (Costo: 3, sin impacto visible)
    5. Reemplazar 'c' con 'c' (Costo: 3, sin impacto visible)
    6. Reemplazar 'e' con 's' (Costo: 3)
    7. Insertar 't' (Costo: 2)
    8. Reemplazar 'r' con 'o' (Costo: 3)

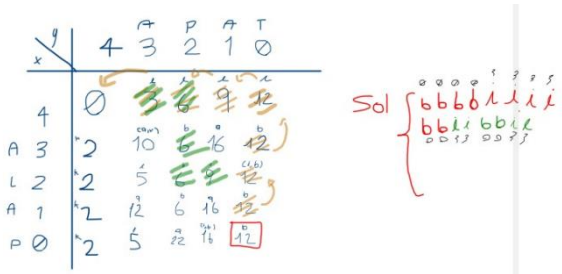
    Costo total: 20

Solución 2:

- Operaciones:
    1. Borrar 'f' (Costo: 2)
    2. Avanzar 'r' == 'r' (Costo: 1)
    3. Avanzar 'a' == 'a' (Costo: 1)
    4. Avanzar 'n' == 'n' (Costo: 1)
    5. Reemplazar 'c' con 's' (Costo: 3)
    6. Insertar 't' después de 'c' (Costo: 2)
    7. Reemplazar 'e' con 'o' (Costo: 3)
    8. Insertar 'r' al final (Costo: 2)

    Costo total: 15

1.2 Caracterizar la estructura de una solución óptima Sea x[1..n] la cadena a transformar en y[1..n], caracterice la estructura de una solución óptima, esto es, indique cómo la solución óptima está compuesta de otras soluciones optimas.



Basicamente las casillas mas a la orilla son las acciones de "Kill" e insert acumulado, la solucion general depende del minimo entre insert (matriz[i-1, j]), reemplazar (Matriz[i-1, j-1), borrar (matriz[i, j-1) en algunos casos al llegar a kill esto lo llevaria directamente a la posicion (0, 0) en la matriz dando fin a la solucion.

Es es una manera de ver como las solucion general depende de la solucion a los subrproblemas

1.3 Definir recursivamente el valor de una solución óptima Utilice la recurrencia. Explique con ejemplos cada uno de los valores que calcula para la matriz M.

```
Teniendo en cuenta los costos = {
    'avanzar': 1,
    'borrar': 2,
    'reemplazar': 3,
    'insertar': 2,
    'kill': 4
}
```

Para calcular el valor de M[i][j], debemos considerar tres posibles operaciones y elegir la que tenga el menor costo:

1. Si los caracteres coinciden (es decir, x[i-1] = y[j-1]):

Operación: Avanzar. No se necesita ninguna operación, solo se avanza en ambos prefijos.

Recurrencia: M[i][j] = M[i-1][j-1] + avanzar si x[i-1] = y[j-1]

2. Si los caracteres no coinciden:

Operación 1: Insertar. Se inserta el carácter y[j-1] en x. M[i][j] = M[i][j-1] + insertar

Operación 2: Reemplazar. Se reemplaza el carácter x[i-1] por y[j-1]. M[i][j] = M[i-1][j-1] + reemplazar

Operación 3: Borrar. Se elimina el carácter x[i-1]. M[i][j] = M[i-1][j] + borrar

Finalmente, seleccionamos la operación con el menor costo: M[i][j] = min(M[i][j-1] + insertar, M[i-1][j-1] + reemplazar, M[i-1][j] + borrar)

3. Condiciones de borde:

Si i = 0 (cadena x está vacía), debemos insertar todos los caracteres de y: M[0][j] = j * insertar

Si j = 0 (cadena y está vacía), debemos borrar todos los caracteres de x: M[i][0] = i * borrar

Si i = 0 y j = 0 (ambas cadenas están vacías), el costo es cero: M[0][0] = 0

1.4 Calcular el valor de una solución óptima Explique cuál es la forma correcta de completar la matriz M. Indique en qué posición de la matriz quedara la solución al problema original. Desarrolle un algoritmo para calcular el valor de la solución óptima. Indique su complejidad. Implemente el algoritmo.

El inicio de la cadena de entrada se encuentra en la esquina inferior derecha de la matriz (M), y el final de la cadena objetivo se encuentra en la esquina superior izquierda de la matriz.

En nuestro caso, por como está construida la matriz la solución se encuentra en la esquina inferior derecha. en la matriz se puede ver que se guarda la acción de menor valor que se hizo, a partir de esto al devolverse tiene las siguientes opciones:

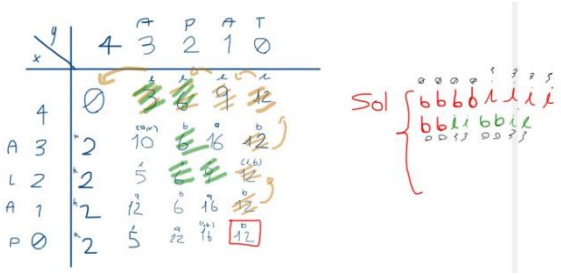
Avanzar: es i-1

borrar: j-1

reemplazar y avanzar: i-1,j-1

Kill: i=0, j=0

Teniendo en cuenta esto, el objetivo para encontrar la solución es moverse desde la esquina inferior derecha a la esquina superior izquierda



1.5 Construir una solución óptima Desarrolle un algoritmo que despliegue la secuencia de operaciones optima que permite transformar una cadena x[1..n] en y[1..n]. Indique su complejidad. Implemente el algoritmo. El programa debe permitir modificar los costos de las operaciones e ingresar las cadenas x y y.

Análisis del Costo de cada solución:

1. **Solución Ingenua:**

Costo computacional: La solución fuerza bruta sigue una estrategia de recursión profunda sin memorizar los subproblemas, lo que da lugar a una complejidad **exponencial**. En el peor caso, se pueden generar 3n+m3^{n+m}3n+m posibles combinaciones de operaciones (si consideramos 3 posibles decisiones en cada paso: insertar, borrar o reemplazar).

Notación: O(3^{n+m}) donde n y m son las longitudes de las cadenas x y y, respectivamente.

2. **Solución Dinámica:**

Costo computacional: La solución dinámica tiene una complejidad **polinómica** en función del tamaño de las cadenas, ya que la matriz tiene dimensiones (n+1)×(m+1), y cada celda de la matriz se calcula en tiempo constante. Por lo tanto, el tiempo total de ejecución es O(n×m) donde n y m son las longitudes de las cadenas x y y, respectivamente.

Notación: O(n.m), donde n y m son las longitudes de las cadenas x y y.

3. **Solución Voraz (Greedy):**

Costo computacional: La solución voraz tiene una complejidad **lineal** en términos de las longitudes de las cadenas, ya que itera sobre cada carácter de ambas cadenas una sola vez. Sin embargo, como no garantiza que se elijan las decisiones correctas globalmente, podría dar una solución no óptima en algunos casos.

Notación: O(n+m) donde n y m son las longitudes de las cadenas x y y.

Comparación de las soluciones:

Método	Complejidad Computacional	Costo	Argumentación
Ingenua	O(3^{n+m})	Exponencial	La solución recursiva examina todas las combinaciones posibles de operaciones sin memorizar subproblemas.
Dinámica	O(n.m)	Polinómica	La programación dinámica utiliza una matriz para almacenar los resultados intermedios, evitando recalcular subproblemas.
Voraz	O(n+m)	Lineal (No óptima)	El enfoque voraz toma decisiones locales sin considerar el impacto global, lo que no siempre produce la mejor solución.

2. EL PROBLEMA DE LA SUBASTA PUBLICA

2.1 Entender el problema 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.

Asignación 1:

x1 = 600, x2= 400, x3 = 0.

Vr(X)= 480.000

Asignación 2:

x1 = 500, x2 = 400, x3 = 100

Vr(X)= 440,000

2.2 Una primera aproximación Considere el algoritmo que lista las posibles asignaciones:

$$< y_1, y_2, ..., \sum_{i=1}^{n-1} y_i = A >$$

donde yi ∈ {0, Mi}, luego calcula el vr para cada asignación y selecciona la mejor.

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 >.

Indique si el algoritmo anterior siempre encuentra la solución óptima.

Utilizando nuestra solución dinámica, lo que tenemos es: (Explicación del algoritmo en el punto 2.5)

Valor máximo recibido: 480,000.

Asignación óptima de acciones:

Ofertante 1 recibe 600 acciones.

Ofertante 2 recibe 400 acciones.

Los otros ofertantes (3 y 4) no reciben ninguna acción.

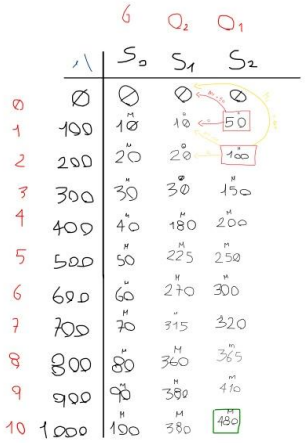
El gobierno recibe el resto de las acciones, es decir, 0 acciones en este caso ya que todas las acciones se asignaron a los ofertantes.

El valor total se calcula como la suma del valor recibido por las acciones asignadas a los ofertantes y el valor de las acciones no asignadas que se ofrecen a un precio de B=100B = 100B=100 por el gobierno.

El paso a paso de la solución es calcular las asignaciones óptimas y el valor recibido a partir de la matriz de costos, y reconstruir las asignaciones a partir de los valores de la matriz.

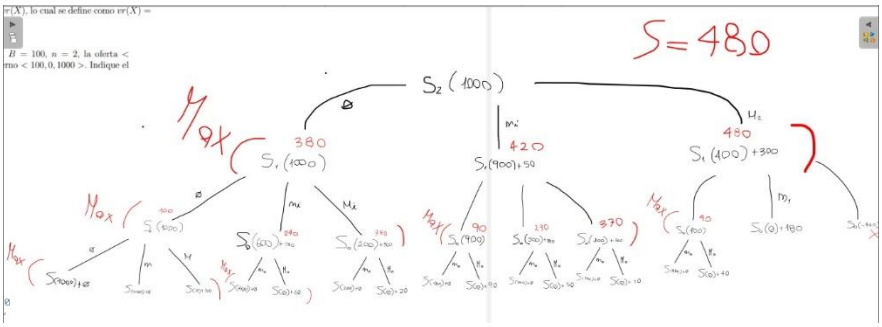
El resultado es la solución optima porque evalúa todas las posibles asignaciones de acciones a los ofertantes, eligiendo la opción que maximiza el valor total, teniendo en cuenta tanto los precios de las ofertas como las restricciones de cantidad. Al resolver subproblemas de manera óptima y almacenar resultados intermedios, garantiza que se obtiene la mejor solución global.

2.3 Caracterizar la estructura de una solución óptima Caracterice la estructura de una solución óptima, esto es, indique la forma de los subproblemas y cómo la solución óptima la componen.



La subestructura optima que utilizamos es muy similar al problema de la mochila, pero como se puede notar cada casilla depende directamente de la anterior más el beneficio sumado, el valor de una casilla depende del máximo de las casillas anteriores involucradas. La solución general se encuentra en la esquina inferior derecha

2.4 Definir recursivamente el valor de una solución ´optima Muestre la recurrencia que define el costo de la solución óptima.



La recursión se puede notar viendo el árbol de tres hijos cada uno, las opciones del árbol son:

- no comprar en la subasta

- comprar lo mínimo

- comprar lo máximo

Las opciones de comprar restan en las acciones actuales y se entrega el sobrante a la siguiente profundidad del árbol hasta el final de las subastas

Al devolverse se elige el máximo entre las tres acciones para comprar subastas.

2.5 Calcular el valor de una solución por medio de un algoritmo, muestre cómo se calcula el costo de la solución ´optima a través de subproblemas hasta llegar a dar la solución del problema original. Indique la complejidad del algoritmo. Implemente el algoritmo.

```python
def subastaDinamica(A, B, ofertantes):
    n = len(ofertantes)
    matriz_costo = np.zeros((A + 1, n + 1), dtype=np.int64)
    rastreo = np.zeros((A + 1, n + 1), dtype=np.int64)  # Rastreo para reconstruir asignaciones

    # Inicializar la matriz
    for i in range(0, A + 1):
        for j in range(0, n + 1):
            if i == 0:
                matriz_costo[i, j] = 0
            elif j == 0:
                matriz_costo[i, j] = i * B
            else:
                precio, minimo, maximo = ofertantes[j - 1]

                opciones = [(matriz_costo[i, j - 1], 0)]  # Opción: no comprar acciones

                # Comprar el mínimo de acciones si es posible
                if i >= minimo:
                    opciones.append(
                        (matriz_costo[i - minimo, j - 1] + precio * minimo, minimo)
                    )

                # Comprar el máximo de acciones si es posible
                if i >= maximo:
                    opciones.append(
                        (matriz_costo[i - maximo, j - 1] + precio * maximo, maximo)
                    )

                # Elegir la mejor opción
                matriz_costo[i, j], rastreo[i, j] = max(opciones, key=lambda x: x[0])
```

El algoritmo funciona de la siguiente manera:

Si el indice i es 0 entonces su solucion es 0 ya que es su caso trivial en el que ninguno de los ofertantes puede comprar ni siquiera lo minimo.

si el indice j es 0 entonces solamente se multiplica el numero acciones a comprar por parte del gobierno que en este caso son todas. este es el caso donde solo esta el gobierno para comprar

Si i es mayor al minimo entonces compra el minimo numero de acciones y lo ingresa en la matriz de costos. pasa lo mismo si i es mayor al máximo

2.6 Construir una solución optima Desarrolle un algoritmo que permita conocer la asignación de acciones X =< x1, x2, ..., xn, xn+1 > tal que vr(X) sea máximo. Indique su complejidad. Implemente el algoritmo. El programa debe permitir ingresar los valores para A, B, n y cada oferta < pi , mi , Mi >.

1. Solución Ingenua:

Si hay n ofertantes, cada ofertante tiene un rango de asignaciones entre un mínimo y un máximo. El número total de combinaciones posibles de asignaciones es el producto de las longitudes de los rangos de cada ofertante.

Por lo tanto, el número total de combinaciones es proporcional al producto de los rangos de asignación de cada ofertante, lo que da como resultado una **complejidad exponencial** en función del número de ofertantes.

$$O(\sum_{i=1}^{n}(Mi - mi + 1))$$

Donde:

N es el número de ofertantes.

Mi es el número máximo de acciones que el ofertante ipuede comprar.

mi es el número mínimo de acciones que el ofertante i puede comprar.

Notación Big-O:

O(2^n)en el peor caso, cuando los rangos de cada ofertante son grandes

2. Solución Dinamica:.

La matriz matriz_costo tiene dimensiones (A+1)×(n+1), donde A es el número total de acciones y n es el número de ofertantes.

Cada celda de la matriz se llena evaluando las opciones para asignar acciones, lo que implica iterar sobre todas las combinaciones de acciones (hasta A) y ofertantes (hasta n).

El tiempo de procesamiento por cada celda de la matriz es constante, pero como hay A×n, la complejidad temporal es proporcional a ese número.

O (A×n)

A es el número total de acciones.

n es el número de ofertantes.

Notación Big-O:

O (A×n)

3. Solución Voraz:

En primer lugar, se ordenan los ofertantes por su precio (en orden descendente). Este paso tiene una complejidad de O(nlogn), donde n es el número de ofertantes.

Después, en un bucle, se asignan las acciones a los ofertantes de acuerdo con el precio y los límites mínimo y máximo. El número de iteraciones es proporcional al número de ofertantes, por lo que este paso tiene una complejidad de O(n).

El paso de asignación de acciones es lineal, pero el paso de ordenación es lo que domina la complejidad.

O(nlogn)

n es el número de ofertantes.

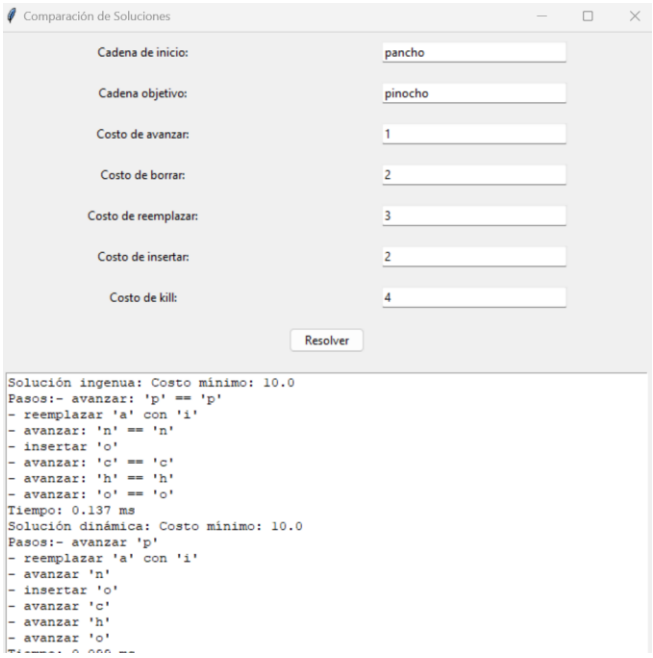
Esta es la complejidad más baja de las tres soluciones, ya que la complejidad de la ordenación es la que determina la eficiencia del algoritmo.

Notación Big-O:

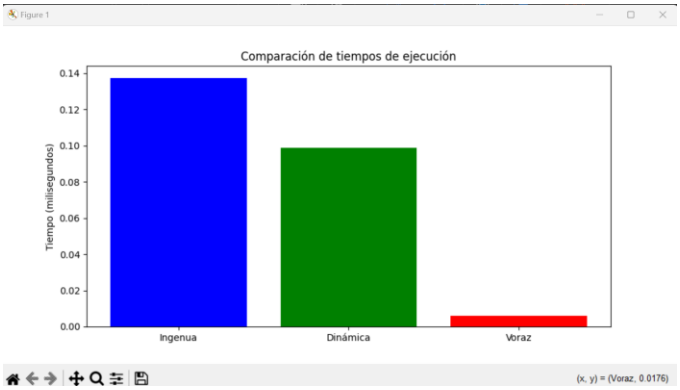
O(nlogn)

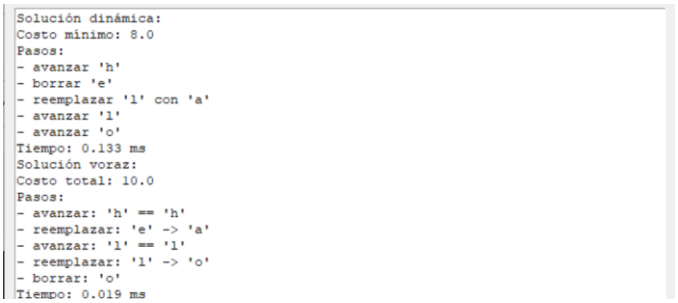
Pruebas: (Tiempos promedios a 50 repeticiones)

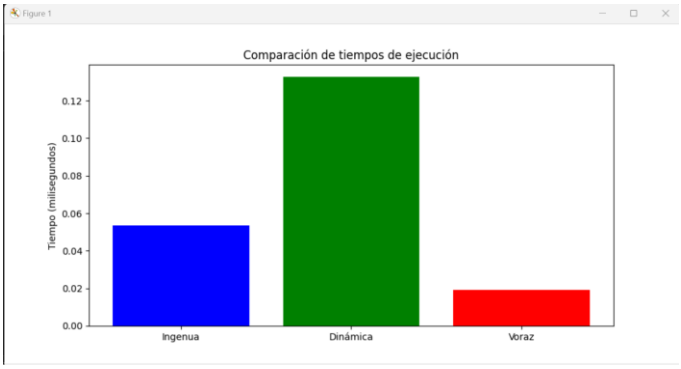
Consola:



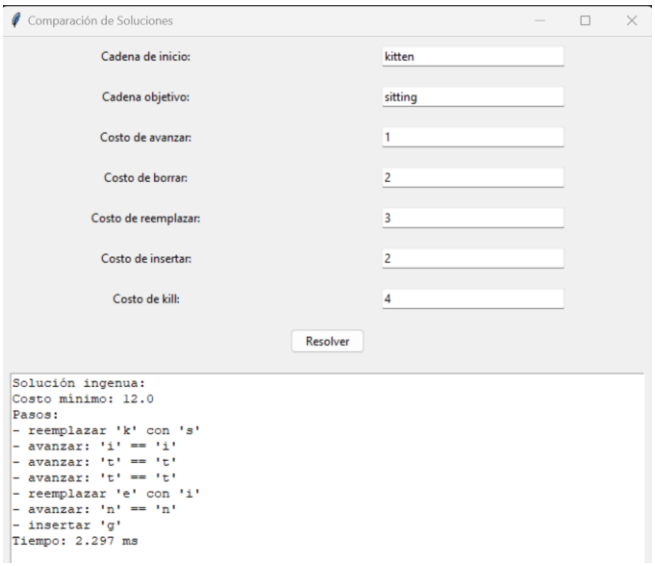
Los resultados muestran claras diferencias entre los enfoques. Tanto la solución ingenua como la dinámica llegaron al costo mínimo de 10.0, realizando los mismos pasos, pero la dinámica fue un poco más rápida (0.099 ms vs. 0.137 ms) gracias a su eficiencia. En cambio, la solución voraz fue mucho más rápida (0.006 ms), pero terminó con un costo más alto (16.0) porque toma decisiones inmediatas sin pensar en el mejor resultado global. Esto demuestra que, aunque el enfoque voraz es rápido, no siempre encuentra la solución óptima.

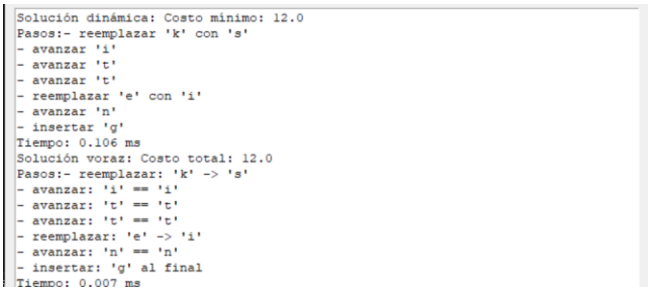






El tiempo de la solución dinámica fue mayor que el de la ingenua porque la implementación dinámica incluye la construcción y llenado de una tabla que almacena costos intermedios para todas las posibles combinaciones de subproblemas. Aunque esto permite garantizar una solución óptima y es eficiente en problemas más grandes, el costo adicional de gestionar esta tabla puede hacer que sea más lento en casos pequeños, como este.

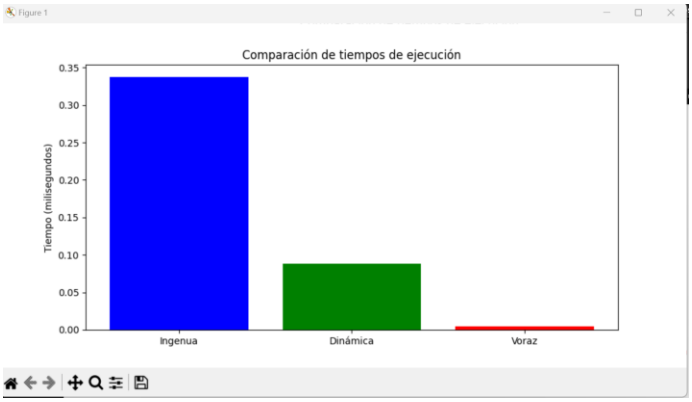


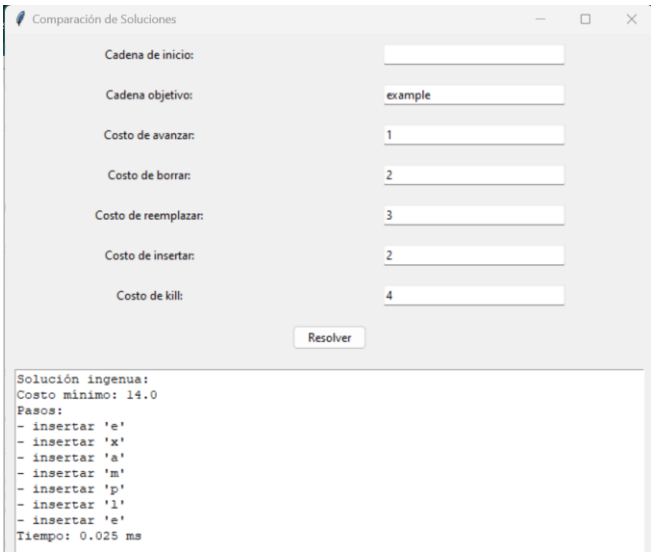


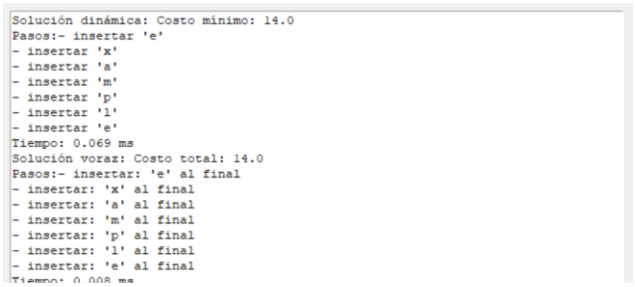
En este caso las tres soluciones encontraron la solución optima, sin embargo, el tiempo de ejecución varía mucho, la solución voraz fue muchísimo mas rápida con solo 0.006 ms, mientras que la dinámica, aunque un poco más lenta con 0.101 ms, sigue siendo muy eficiente. La ingenua, en cambio, tomó mucho más tiempo 2.441 ms por su estructura.

Universidad del Valle

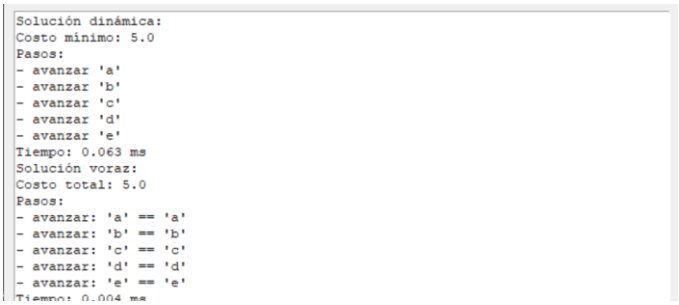

```
Solución dinámica:
Costo mínimo: 10.0
Pasos:- avanzar 'a'
- reemplazar 'b' con 'z'
- avanzar 'c'
- borrar 'd'
- avanzar 'e'
- insertar 'd'
Tiempo: 0.088 ms

Solución voraz:
Costo total: 11.0
Pasos:- avanzar: 'a' == 'a'
- reemplazar: 'b' -> 'z'
- avanzar: 'c' == 'c'
- reemplazar: 'd' -> 'e'
- reemplazar: 'e' -> 'd'
Tiempo: 0.004 ms
```



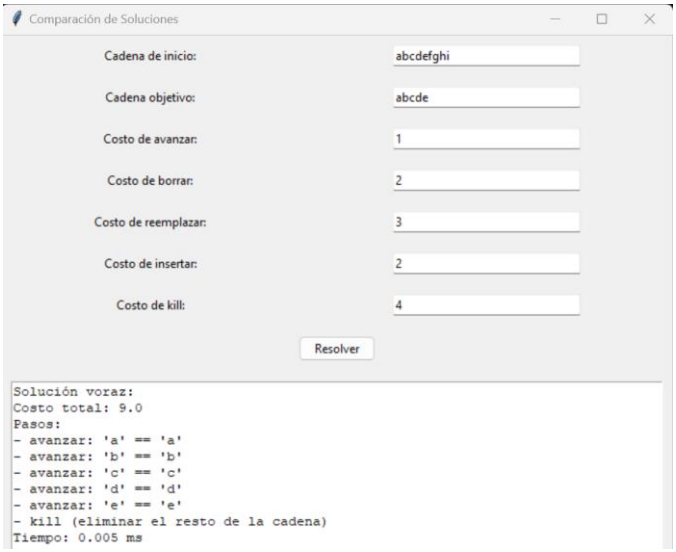




Probando insertar

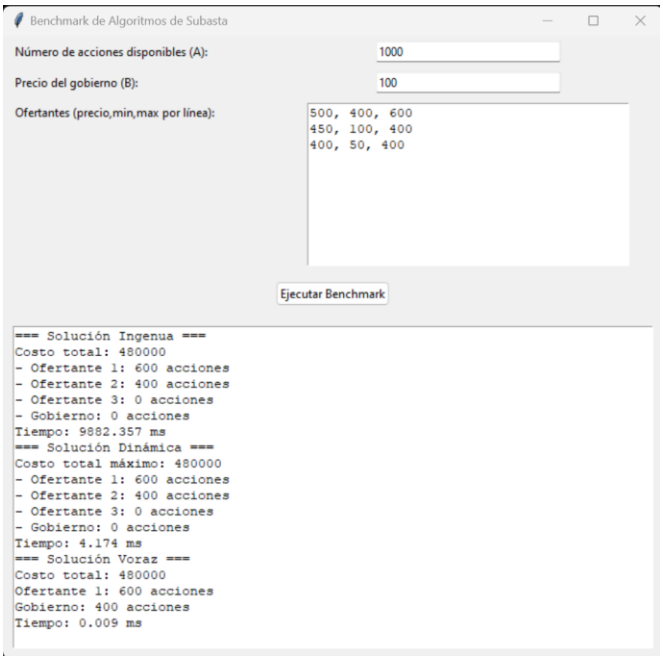


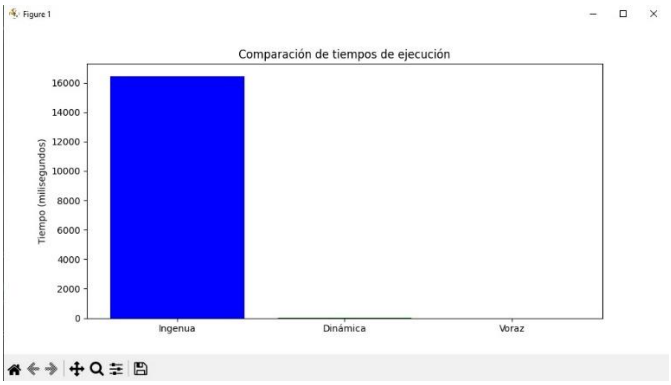
Probando avanzar



Probando Kill

Subasta:



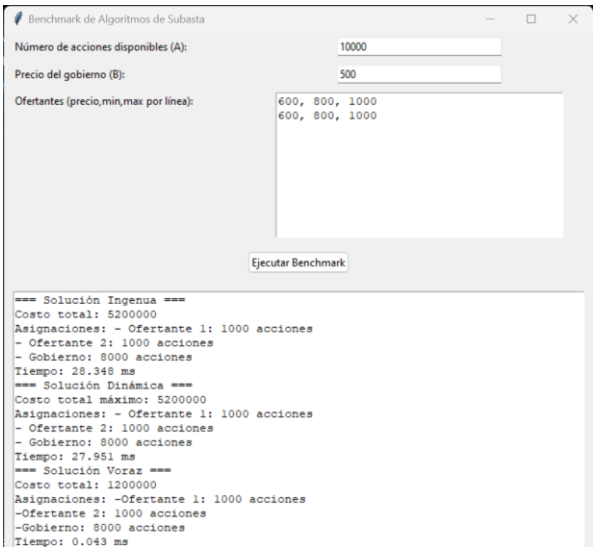


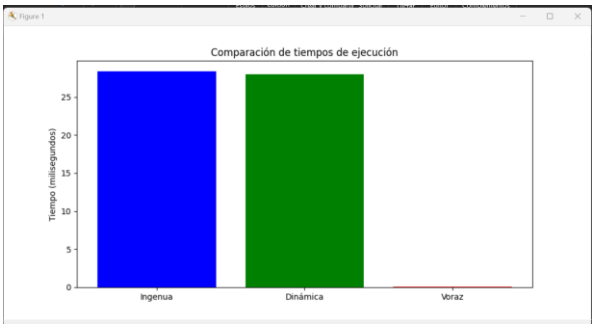
La ingenua a pesar de encontrar la solución óptima, se tarda demasiado este tiempo elevado refleja la ineficiencia del enfoque, ya que evalúa todas las posibles combinaciones de asignaciones posibles, lo que lo hace exponencial.

La solución dinámica aprovecha subestructuras óptimas y evita recomputaciones redundantes mediante el uso de una matriz de costos. Es adecuada para casos con un número moderado de ofertantes y acciones.

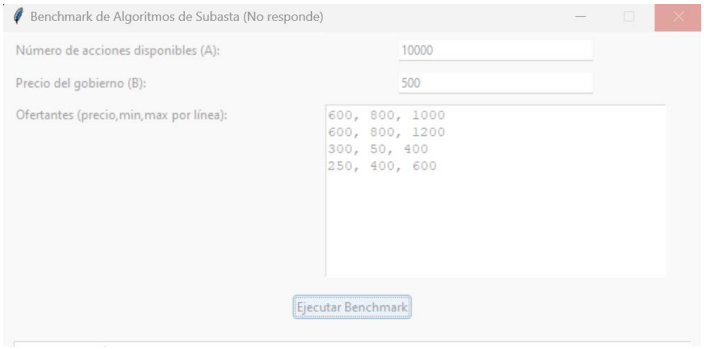
Es extremadamente eficiente, pero puede no ser confiable para problemas en los que no se cumple la propiedad de subestructura óptima global.

El programa recibe la misma oferta:





En un numero bajo de ofertantes la dinámica se comporta similar a la ingenua ya que no ha resuelto varios subproblemas.



A un numero mayor de ofertantes es bastante tardado ya que la solución ingenua tiene que calcular demasiadas combinaciones.

Numero bajo de ofertantes, numero alto de combinaciones: