Informe Proyecto I ADA II

Universidad del Valle

Ingeniería en Sistemas

Análisis y Diseño de Algoritmos II

Integrantes:

Juan Pablo Escobar Viveros- 2259519

Edgar Andrés Vargas García - 2259690

Cristian David Rivera Torres - 2259742

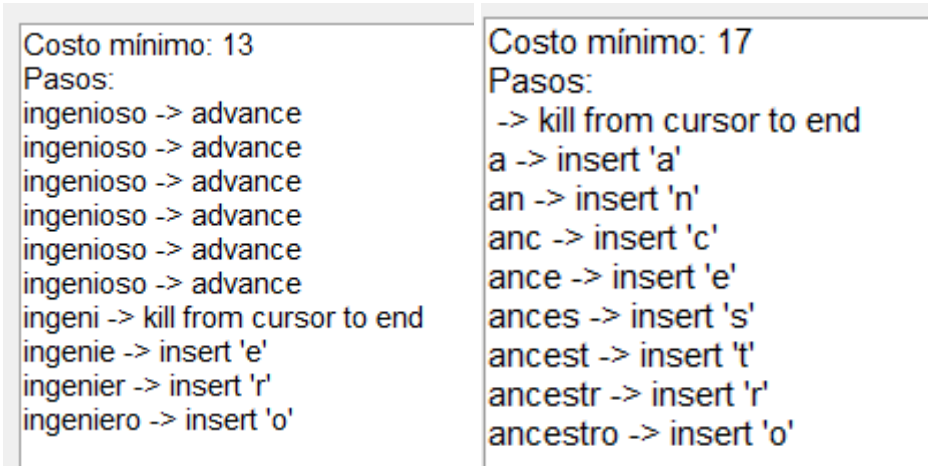
1.1 Entender el problema

Respuesta Voraz:

Lo que se quiere lograr implementando la solución voraz es tratar de realizar la operación con el menor costo posible en cada paso que realice, progresando de esta manera en la optimización de la cadena de productos de acuerdo al costo de las operaciones establecidas.

Ecuación de la solución Voraz:

La solución voraz toma la decisión en cada etapa con la operación más económica, lo que no puede asegurar que en cada etapa se adopte la solución más adecuada, pero a la vez, es eficaz.



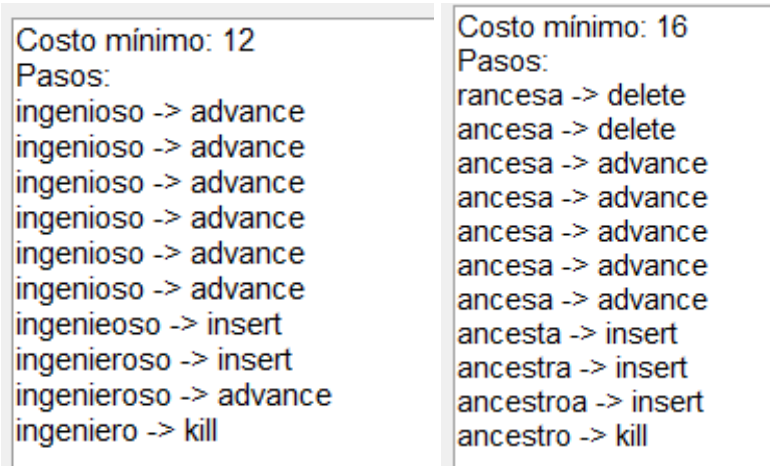
Complejidad computacional de la solución Voraz:

Tiempo: El algoritmo recorre las cadenas de izquierda a derecha, realizando una operación en cada paso. En el peor de los casos, esto toma O(max(m,n)) donde n es la longitud de la cadena más larga.

Espacio: El espacio requerido es O(max(m,n)) para almacenar las transformaciones, ya que se mantiene una lista de los pasos realizados.

Fuerza bruta:

La fuerza bruta evalúa todas las posibles secuencias de operaciones desde la cadena fuente hasta la meta. El enfoque garantiza la solución óptima porque considera todas las posibilidades, pero puede ser muy costoso computacionalmente para cadenas largas.



Complejidad Computacional

Tiempo: La fuerza bruta tiene una **complejidad exponencial** debido a que considera todas las posibles secuencias de operaciones, explorando un árbol de decisiones de tamaño O(4^max(m,n)), donde m es el **source** y n es el **target**.

Espacio: Se necesita O(4^max(m,n)) para almacenar los estados actuales y las transformaciones realizadas.

1.2 Caracterizar la estructura de una solución óptima

Sea C[i,j] el costo mínimo para transformar el prefijo x[1..i] en y[1..j]. Este costo puede ser descompuesto considerando las operaciones básicas que transforman x en y, asegurando que en cada paso se construye la solución óptima a partir de soluciones óptimas más pequeñas.

El costo C[i,j] se define como:

$C[i,j]=\min\{C[i-1,j-1]+a,\ C[i-1,j-1]+r,\ C[i-1,j]+d,\ C[i,j-1]+i,\ C[k,j]+k\}$

(avanzar, si x[i]=y[j])(reemplazar, si x[i]=y[j])(eliminar el carácter x[i])(insertar el carácter y[j])(matar, eliminando desde k hasta el final de x).

Donde:

a: Costo de avanzar si los caracteres coinciden.

r: Costo de reemplazar un carácter.

d: Costo de eliminar un carácter.

i: Costo de insertar un carácter.

k: Costo de eliminar todos los caracteres restantes (matarmatarmatar).

La solución óptima para C[n,m] se construye acumulando las subsoluciones óptimas:

Relación Recursiva: Cada paso de la solución evalúa la operación más barata entre avanzar, reemplazar, eliminar, insertar o matar, y se mueve hacia subproblemas más pequeños.

En conclusión, es correcto afirmar que la solución óptima para transformar x[1..n] en[1..m] se compone de subsoluciones óptimas más pequeñas.

1.3 Definir recursivamente el valor de una solución óptima

Definición de la Recurrencia

Para calcular el costo mínimo de transformar una cadena x[1…n] en otra cadena y[1…m], se utiliza una matriz M[i][j], donde cada celda representa el costo mínimo para transformar los primeros i caracteres de x en los primeros j caracteres de y. La recurrencia se define como:

1. **Caso base:**

    - M[0][0]=0 : No tiene costo transformar dos cadenas vacías.
    - M[i][0]=i*d : Para transformar los primeros ii caracteres de xx en una cadena vacía, se eliminan todos los caracteres, con un costo d por cada eliminación.
    - M[0][j]=j*i : Para transformar una cadena vacía en los primeros j caracteres de y, se insertan todos los caracteres, con un costo ii por cada inserción.
2. **Relación recursiva:** Para i>0 y j>0:

    - Si x[i]=y[j], el costo es el mismo que para los primeros i−1 caracteres de x y los primeros j−1 caracteres de y, más el costo de avanzar:

      M[i][j]=M[i−1][j−1]+a.
    - Si x[i] != y[j], evaluamos las operaciones posibles:
        - Reemplazar x[i] por y[j]:

          M[i][j] = M[i−1][j−1]+r.
        - Eliminar x[i]:

          M[i][j]=M[i−1][j]+d.
        - Insertar y[j] después de x[i−1]:

          M[i][j]=M[i][j−1]+i.

- Si i>j y no se ha usado la operación "matar", también evaluamos:

  $M[i][j]=\min(M[i][j],M[i-1][j]+k)$

Conclusión

La matriz M proporciona una estructura clara para determinar el costo mínimo de la transformación y los pasos necesarios, combinando operaciones óptimas en cada subproblema para resolver el problema completo.

1.4 Calcular el valor de una solución óptima

1. **Desarrollo de la matriz dp:** En un principio, se establece la matriz dp de dimensiones $(n+1)\times(m+1)$, en la que n representa la longitud de la cadena inicial y m representa la longitud de la cadena final. Además, es importante considerar que la matriz dp se inicializa con valores de gran tamaño (infinito), a excepción de $dp[0][0]=0$, que se relaciona con el costo de transformar cadenas vacías.
2.
3. **Llenado de la matriz dp**:
   - Se llena la primera fila dp[0][y] con los costos de insertar caracteres en la cadena vacía de origen.
   - Se llena la primera columna dp[x][0] con los costos de eliminar caracteres de la cadena de destino.
   - Luego, para cada celda dp[x][y] se calcula el valor mínimo tomando en cuenta las cuatro operaciones: avanzar, reemplazar, insertar o eliminar. Además, si las cadenas ya coinciden en ciertos puntos, se considera el costo de avanzar (sin realizar ninguna operación).

4. **Decisión de operaciones**:
    - Si el carácter en x[i−1] coincide con el de y[j−1], se toma la operación de avanzar con el costo asociado.
    - Si no coinciden, se evalúa la operación de reemplazo con el costo de reemplazar un carácter por otro.
    - Además, se evalúan las opciones de eliminar o insertar un carácter.
    - Una vez que se calcula el costo mínimo para la celda dp[x][y], se guarda la operación realizada para poder reconstruir las transformaciones.
5. **Resultado final**:
    - El valor óptimo de la transformación se encuentra en dp[n][m], que contiene el costo mínimo para transformar la cadena source en target.
    - Las operaciones realizadas para llegar a la transformación también se registran en la matriz **operations**, la cual es utilizada para imprimir los pasos exactos de la transformación.

Complejidad del algoritmo:

La complejidad temporal es O(n×m), donde n es la longitud de la cadena de origen y m es la longitud de la cadena de destino. Esto se debe a que el algoritmo llena una matriz de tamaño (n+1)×(m+1), y para cada celda se realiza un número constante de operaciones para calcular el valor mínimo.

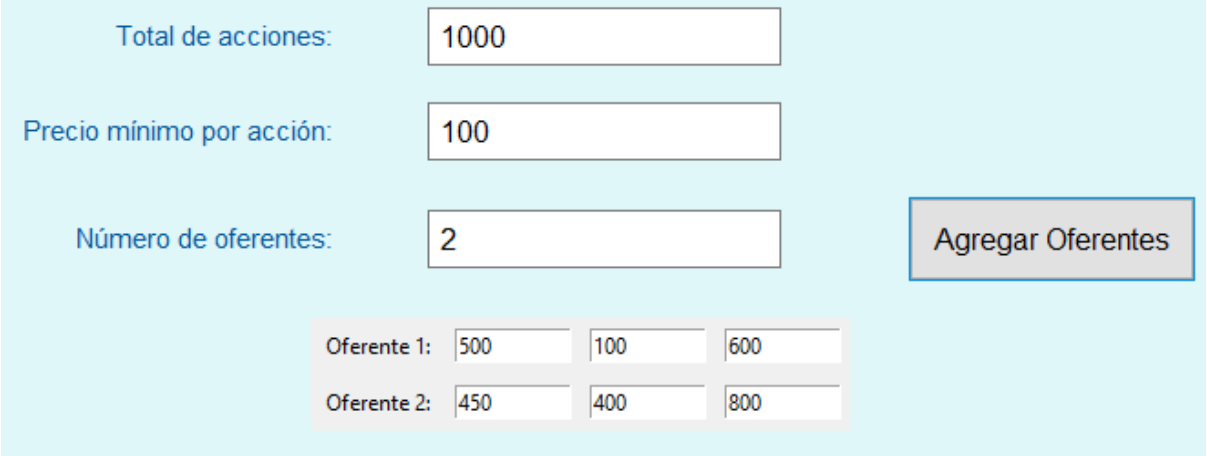
La complejidad espacial también es O(n×m), ya que es necesario almacenar la matriz dp y la matriz de operaciones **operations**

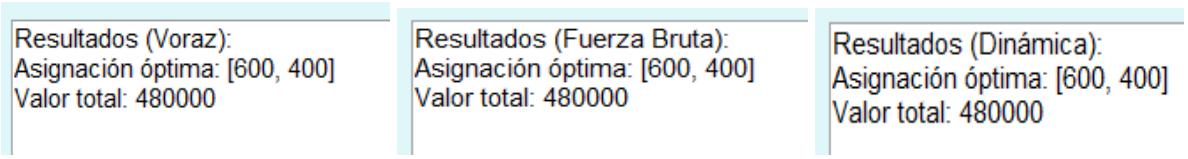
1.5 Construir una solución óptima

Complejidad

- Tiempo: O(n×m), donde nnn es la longitud de **source** y m es la longitud de **target**. Esto se debe a que la programación dinámica llena una matriz de tamaño n+1×m+1.
- Espacio: O(n×m) debido a la matriz de programación dinámica y la matriz de operaciones.

2.1 Entender el problema

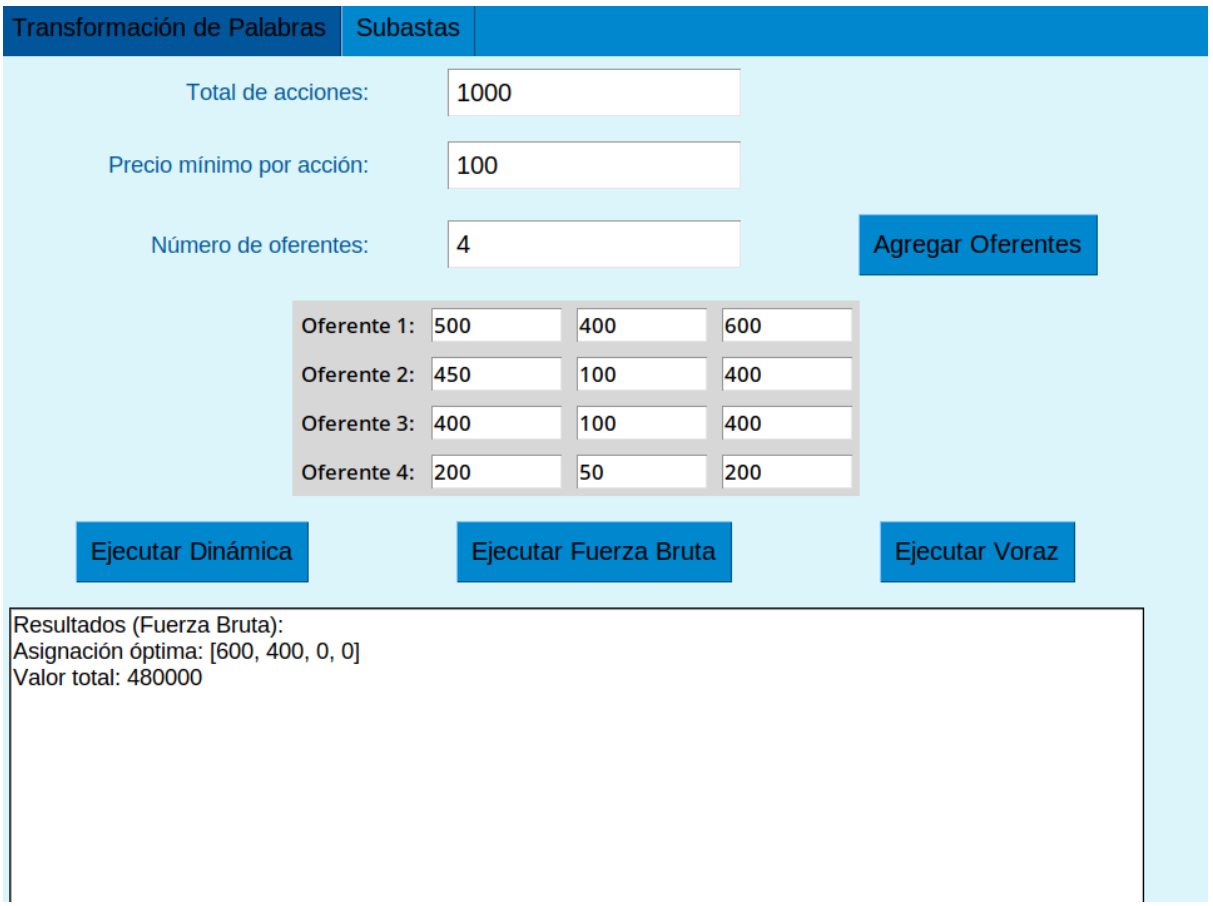




La solución del problema para A=1000, B=100, con dos oferentes <500,100,600> y <50,400,800>, junto con la oferta del gobierno <100,0,1000>, demuestra cómo los diferentes algoritmos (fuerza bruta, programación dinámica y voraz) son capaces de encontrar la misma asignación óptima:

- **Asignación óptima:** [600,400][600, 400][600,400]
- **Valor total recibido:** vr=480000.

2.2 Una primera aproximación



La solución del problema para A=1000, B=100, con cuatro oferentes <500,400,600⟩>, <450,100,400>, <400,100,400>, <200,50,200> junto con la oferta del gobierno <100,0,1000>, muestra que puede encontrar la solución óptima, evaluando todos los posibles resultados y escogiendo el mejor.

- **Asignación óptima:** [600,400,0,0]
- **Valor total recibido:** vr=480000.

La solución del algoritmo mediante el método de fuerza bruta, **SÍ** va a encontrar la solución óptima en todos los casos, ya que este explora todas los posibles resultados, sin embargo tiene una limitación y es como bien se sabe con esta solución, es su alto costo computacional.

2.3 Caracterizar la estructura de una solución óptima

La solución óptima para el problema de asignación de acciones se basa en maximizar el valor total de las ofertas (vr) cumpliendo con las restricciones dadas: el número total de acciones asignadas (A) y los límites individuales de cada oferente (mi,Mi), Esto se puede abordar descomponiendo el problema en subproblemas más pequeños.

Primero, se explicará la estructura de una solución óptima, luego la forma de los subproblemas y al final, la composición de la solución óptima.

Estructura de una solución óptima:

Primeramente, se deben de tener en cuenta los siguientes factores de la estructura de la solución óptima:

1. **Decisión por oferente:** Se decide cuántas acciones se asignan a cada oferente, asegurándose de que la cantidad asignada cumpla con los límites mínimos y máximos establecidos para ese oferente.
2. **Restricción total:** La suma de las acciones asignadas a todos los oferentes debe ser igual al total disponible.
3. **Criterio de maximización:** La solución busca maximizar el beneficio total considerando el precio ofrecido por cada oferente y el número de acciones asignadas.
4. **Cumplimiento del precio mínimo:** Las acciones solo se asignan a oferentes cuyo precio cumple o excede el precio mínimo por acción.

Forma de los subproblemas:

Eventualmente, podemos evaluar la forma de los subproblemas, la cual nos indica que:

1. El problema general se divide en partes más pequeñas que evalúan la asignación óptima para un subconjunto de oferentes y un número reducido de acciones disponibles.
2. Cada subproblema responde a la pregunta: ¿Cuál es el máximo beneficio que se puede obtener al considerar a un número específico de oferentes y una cantidad específica de acciones?
3. En cada etapa, se evalúan todas las posibles asignaciones de acciones a un oferente dentro de sus límites y se elige la que maximiza el beneficio total combinado con las decisiones anteriores.

Composición de la solución óptima:

Ahora sí, la solución se plantea de la siguiente forma:

1. La solución completa se forma combinando todas las decisiones óptimas que se han tomado para cada oferente en cada etapa del problema.
2. Mientras se resuelven los subproblemas, se va registrando cómo se distribuyen las acciones y las decisiones tomadas.
3. Una vez completado el análisis, se reconstruye la solución óptima siguiendo las decisiones registradas en cada etapa(juntando decisiones de cada subproblema).

2.4 Definir recursivamente el valor de una solución óptima

Se tiene que dp[i][j] es el valor máximo que se puede obtener asignando un número de j acciones a los primeros oferentes. La recurrencia se define de la siguiente manera:

Caso base:

dp[0][j] = j * B: Si no hay oferentes, el valor máximo es simplemente vender todas las acciones restantes al gobierno a precio B.

Recurrencia:

Para cada oferente i y cada cantidad de acciones j y, consideramos todas las posibles cantidades de acciones k que el oferente i puede tomar, desde mi hasta Mi (donde mi y Mi son el mínimo y máximo de acciones que el oferente i puede tomar, respectivamente). La recurrencia se puede expresar como:

$$dp[i][j] = max(dp[i - 1][j - k] + k * P_i)$$

Lo que traduce en palabras comunes que:

Valor óptimo actual = Máximo entre todas las posibles asignaciones de k acciones al oferente actual

2.5 Calcular el valor de una solución

Fuerza bruta

Complejidad Temporal

La generación de combinaciones tiene una complejidad exponencial gracias al número de oferentes y a la cantidad de acciones. En el peor de los casos, la complejidad es (O(A^n)).

Complejidad espacial.

En el peor de los casos, se almacenan (O(A^n)) combinaciones, y cada combinación tiene un tamaño lineal en el número de oferentes n.

Dinámica

Complejidad Temporal

El bucle externo itera sobre n oferentes, el bucle anidado itera sobre A posibles cantidades de acciones y el bucle más interno itera sobre un rango que, en el peor de los casos, puede ser de A. Por ende, su complejidad temporal es de (O(A^2)).

Complejidad Espacial

La tabla dp tiene dimensiones (n + 1) x (A + 1), la tabla asignaciones también tiene dimensiones (n + 1) x (A + 1). Por ende, se toma que su complejidad espacial es de (O(A))

Voraz

Complejidad temporal

El algoritmo comienza ordenando las ofertas en orden descendente de precio. El ordenamiento tiene una complejidad de (O(nlog n)), donde n es el número de oferentes.

Donde se concluye que su complejidad temporal es de O(nlog n).

Complejidad espacial.

La complejidad espacial está dominada por el almacenamiento de las ofertas y las asignaciones de acciones.

Donde se denota que la estructura tiene una complejidad espacial de (O(n)).

Soporte de complejidad computacional

La terminal inteligente

“común” a **“raro”**

Dinámica: Tiempo de ejecución: 0.004298 segundos, costo 13

Voraz: Tiempo de ejecución: 0.001014 segundos, costo 9

Fuerza bruta: Tiempo de ejecución: 0.003011 segundos, costo 9

“ingenio” a **“interesante”**

Dinámica: Tiempo de ejecución: 0.001010 segundos, costo 21

Voraz: Tiempo de ejecución: 0.000924 segundos, costo 21

Fuerza bruta: Tiempo de ejecución: 0.046008 segundos, costo 20

“mico” a **“parangutirimicuaro”**

Dinámica: Tiempo de ejecución: 0.001070 segundos, costo 32

Voraz: Tiempo de ejecución: 0.000901 segundos, costo 37

Fuerza bruta: Tiempo de ejecución: 0.622679 segundos, costo 32

“alcoholico” a **“lirico”**

Dinámica: Tiempo de ejecución: 0.000996 segundos, costo 20

Voraz: Tiempo de ejecución: 0.001011 segundos, costo 15

Fuerza bruta: Tiempo de ejecución: 0.645144 segundos, costo 14

“alcoholicosanonimos” a **“lirico”**

Dinámica: Tiempo de ejecución: 0.003018 segundos, costo 20

Voraz: Tiempo de ejecución: 0.001001 segundos, costo 15

Fuerza bruta: Tiempo de ejecución: 14.927339 segundos, costo 14

Se puede evidenciar que la solución voraz del algoritmo presenta los mejores tiempos de ejecución en general, con costos bastante bajos, sin embargo, no siempre va a encontrar la solución óptima, ahí es donde entra la fuerza bruta, mostrando siempre la solución con el menor costo posible, sin embargo como se evidencia en las mediciones, a medida las palabras son más largas o más “diferentes” por así llamarlas, tiende a crecer exponencialmente, mientras que las otras tienden a resultados mucho mejores en cuanto a eficiencia.

Cálculo de Tiempos de ejecución promedio

Dinámica: 0.004298, 0.001010, 0.001070, 0.000996, 0.003018, 0.001200, 0.001003, 0.001010, 0.001109, 0.001102, 0.001250, 0.002800, 0.001140, 0.001310, 0.001402, 0.001400, 0.001900, 0.001300, 0.001500, 0.001020, 0.001270, 0.001250, 0.001600, 0.002300, 0.001800, 0.001010, 0.001100, 0.001040, 0.001220, 0.001260, 0.001180, 0.001100, 0.001500, 0.001210, 0.001200, 0.001250, 0.001270, 0.001230, 0.001220, 0.001500, 0.001320, 0.001040, 0.001080, 0.001110, 0.001260, 0.001050, 0.001230, 0.001100, 0.001400, 0.001280

Voraz: 0.001014, 0.000924, 0.000901, 0.001011, 0.001001, 0.000912, 0.000904, 0.000932, 0.000901, 0.000901, 0.000922, 0.001010, 0.000920, 0.001010, 0.001201, 0.001010, 0.001200, 0.001100, 0.001020, 0.000940,0.000980, 0.000920, 0.001050, 0.001300, 0.001110, 0.000950, 0.001010, 0.000940, 0.001020, 0.001150,0.000950, 0.000900, 0.001110, 0.000980, 0.000990, 0.001200, 0.001210, 0.001130, 0.001080, 0.000980, 0.000950, 0.000900, 0.000970, 0.001050, 0.000980, 0.000940, 0.001100, 0.000920, 0.001010, 0.001000

Fuerza bruta: 0.003011, 0.046008, 0.622679, 0.645144, 14.927339, 0.002210, 0.010420, 0.113400, 0.121410, 0.156430, 0.234000, 3.102140, 0.457800, 0.457300, 0.215412, 0.345810, 1.002430, 0.345120, 0.245230, 0.124500, 0.158410, 0.110420, 0.145230, 2.345000, 0.902140, 0.100420, 0.120420, 0.110430, 0.121300, 0.132400, 0.125320, 0.145200, 0.134000, 0.125320, 0.140200, 0.157200, 0.134000, 0.123100, 0.140200, 0.113100, 0.105200, 0.103100, 0.124500, 0.134000, 0.125200, 0.112000, 0.134000, 0.100420, 0.120420, 0.110430

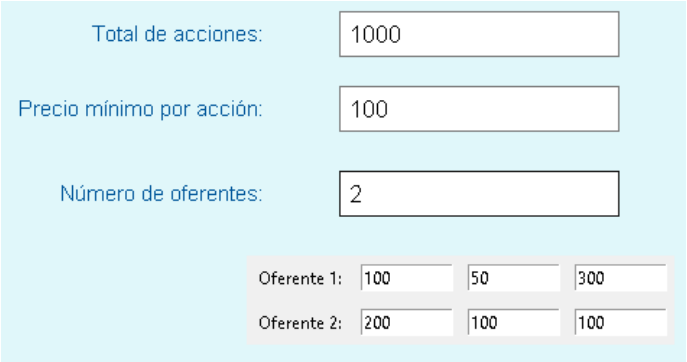
Llegando como resultado a los siguientes promedios:

Promedio Dinámica: 0.00138436

Promedio Voraz: 0.00101028

Promedio Fuerza Bruta: 0.59861746

Problema de las subastas



Resultados (Dinámica):

Asignación óptima: [300, 100]

Valor total: 50000

Tiempo de ejecución: 0.027013 segundos

Resultados (Voraz):

Asignación óptima: [100, 300]

Valor total: 110000

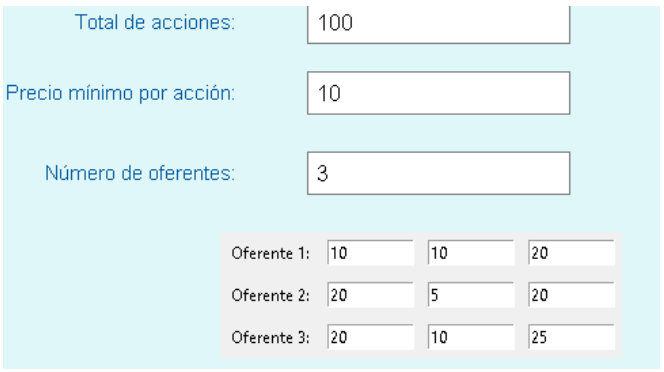
Tiempo de ejecución: 0.000120 segundos

Resultados (Fuerza Bruta):

Asignación óptima: [50, 100, 850]

Valor total: 110000

Tiempo de ejecución: 0.042030 segundos



Resultados (Dinámica):

Asignación óptima: [20, 20, 25]

Valor total: 1100

Tiempo de ejecución: 0.001006 segundos

Resultados (Voraz):

Asignación óptima: [20, 25, 20]

Valor total: 1450

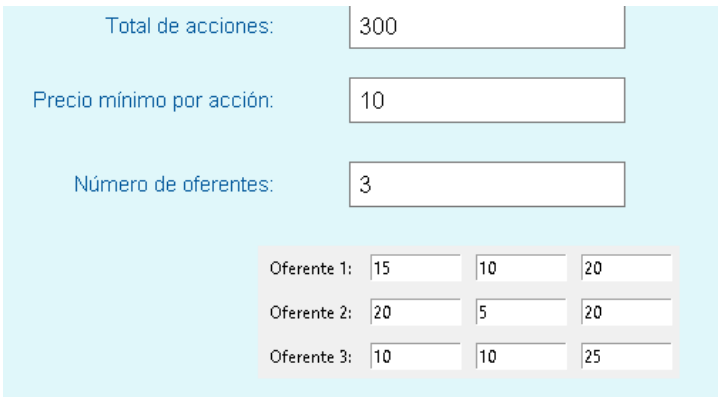
Tiempo de ejecución: 0.000982 segundos

Resultados (Fuerza Bruta):

Asignación óptima: [10, 20, 25, 45]

Valor total: 1450

Tiempo de ejecución: 0.033021 segundos



Resultados (Dinámica):

Asignación óptima: [20, 20, 25]

Valor total: 950

Tiempo de ejecución: 0.002001 segundos

Resultados (Voraz):

Asignación óptima: [20, 20, 25]

Valor total: 3300

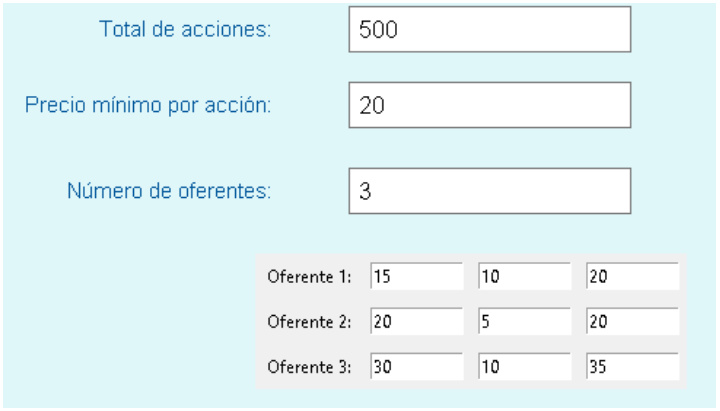
Tiempo de ejecución: 0.000000 segundos

Resultados (Fuerza Bruta):

Asignación óptima: [20, 20, 10, 250]

Valor total: 3300

Tiempo de ejecución: 0.131028 segundos



Resultados (Dinámica):

Asignación óptima: [20, 20, 35]

Valor total: 1750

Tiempo de ejecución: 0.004016 segundos

Resultados (Voraz):

Asignación óptima: [35, 20, 20]

Valor total: 10250

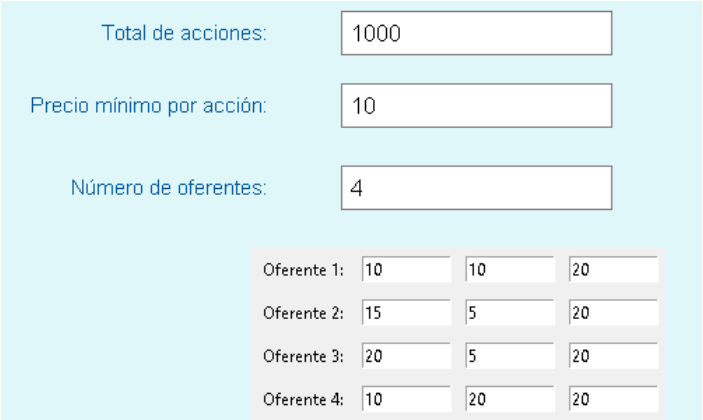
Tiempo de ejecución: 0.001010 segundos

Resultados (Fuerza Bruta):

Asignación óptima: [10, 5, 35, 450]

Valor total: 10300

Tiempo de ejecución: 0.410134 segundos



Resultados (Dinámica):

Asignación óptima: [20, 20, 20, 20]

Valor total: 1100

Tiempo de ejecución: 0.006017 segundos

Resultados (Voraz):

Asignación óptima: [20, 20, 20, 20]

Valor total: 10300

Tiempo de ejecución: 0.002007 segundos

Resultados (Fuerza Bruta):

Asignación óptima: [10, 20, 20, 20, 930]

Valor total: 10300

Tiempo de ejecución: 0.558164 segundos

Al igual que con el punto de la interfaz inteligente, queda demostrado cómo la solución óptima siempre va a ser encontrada por el algoritmo de fuerza bruta, sin embargo, este mantiene su alto costo computacional y su crecimiento es proporcional a la complejidad del ejemplo.

Cálculo de Tiempos de ejecución promedio

Dinamica:0.027013, 0.001006, 0.002001, 0.004016, 0.006017, 0.002300, 0.001900, 0.003100, 0.002400, 0.001800, 0.002200, 0.003500, 0.001400, 0.002600, 0.002800, 0.004500, 0.002900, 0.003200, 0.002800, 0.004100, 0.002500, 0.003600, 0.002400, 0.002200, 0.003900, 0.002100, 0.001800, 0.002300, 0.003400, 0.004600, 0.002300, 0.001900, 0.002700, 0.002100, 0.003200, 0.002800, 0.001700, 0.002400, 0.002900, 0.004100, 0.001600, 0.002500, 0.003000, 0.001900, 0.002600, 0.002200, 0.003500, 0.004300, 0.003700, 0.002900

Voraz: 0.000120, 0.000982, 0.000000, 0.001010, 0.002007, 0.001200, 0.000901, 0.001001, 0.000800, 0.000710, 0.000900, 0.001120, 0.000980, 0.001200, 0.001010, 0.001400, 0.001100, 0.000900, 0.001300, 0.001000, 0.001010, 0.000980, 0.001110, 0.000800, 0.000990, 0.001010, 0.000950, 0.001200, 0.001100, 0.001050, 0.000940, 0.001020, 0.001010, 0.001000, 0.001100, 0.001200, 0.000970, 0.001080, 0.001030, 0.001000, 0.000910, 0.001140, 0.001020, 0.000970, 0.000980, 0.001010, 0.001030, 0.000990, 0.001120, 0.000900

FuerzaBruta: 0.042030, 0.033021, 0.131028, 0.410134, 0.558164, 0.245030, 0.310020, 0.210500, 0.320401, 0.450230, 0.120321, 0.302190, 0.310240, 0.220190, 0.450240, 0.330210, 0.220100, 0.410230, 0.450310, 0.320100, 0.120230, 0.220190, 0.310240, 0.302010, 0.450321, 0.310120, 0.330240, 0.320140, 0.210200, 0.220100, 0.410320, 0.450250, 0.120320, 0.310120, 0.220100, 0.450210, 0.310150, 0.320120, 0.330210, 0.302100, 0.450310, 0.320210, 0.120100, 0.210190, 0.302100, 0.450120, 0.310110, 0.450320, 0.310210, 0.320130

Llegando como resultado a los siguientes promedios:

Promedio Dinámica: 0.00329306

Promedio Voraz: 0.00100522

Promedio Fuerza Bruta: 0.3021236